

UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Task Decomposition with Multi-Agent Systems

Permalink

<https://escholarship.org/uc/item/2z57p08m>

ISBN

9798189272196

Author

Lian, Long

Publication Date

2026-05-01

Peer reviewed|Thesis/dissertation

Task Decomposition with Multi-Agent Systems

by

Long Lian

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair

Assistant Professor Adam Yala

Professor Kurt Keutzer

Doctor Yin Cui

Spring 2026

Task Decomposition with Multi-Agent Systems

Copyright 2026
by
Long Lian

Abstract

Task Decomposition with Multi-Agent Systems

by

Long Lian

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Trevor Darrell, Chair

Modern foundation models have made rapid progress in language, vision, and reasoning. However, many real-world tasks remain difficult because they require more than mapping a single input to a single output: they require decomposing ambiguous goals, coordinating multiple specialized systems, grounding intermediate representations in the visual world, and allocating computation adaptively across subtasks. This thesis studies how task decomposition with multi-agent systems can make generative and reasoning models more controllable, scalable, and effective. I approach this question through two complementary settings. First, I show how large language models can serve as planners and controllers for visual generation, translating complex text prompts into structured spatial and temporal representations, using feedback to correct generation errors, and enabling finer control over images and videos. This visual part also includes Describe Anything, which produces detailed localized descriptions for image and video regions and can support high-quality synthetic data generation for controllable image and video synthesis. Second, I extend decomposition to language-model reasoning, introducing systems for adaptive parallel reasoning and efficient threaded inference. Across these contributions, a common pattern emerges: when complex tasks are represented as interacting subproblems, models can use language, visual grounding, and parallel computation to solve tasks that are difficult for monolithic generation. Together, the methods in this dissertation advance a view of AI systems as coordinated collections of agents and modules that can plan, verify, specialize, and reason in parallel.

To my family, friends, and mentors.

Contents

Contents	ii
List of Figures	vi
List of Tables	xviii
1 Introduction	1
1.1 Motivation	1
1.2 Summary of Contributions	2
1.3 Thesis Organization	3
I Language-Guided Visual Generation	4
2 LLM-grounded Diffusion: Enhancing Prompt Understanding of Text-to-Image Diffusion Models with Large Language Models	5
2.1 Introduction	5
2.2 Related Work	8
2.3 LLM-grounded Diffusion	9
2.4 Evaluation	16
2.5 Discussions	22
2.6 Summary	22
3 LLM-grounded Video Diffusion Models	24
3.1 Introduction	24
3.2 Related Work	26
3.3 Can LLMs Generate Spatiotemporal Dynamics?	27
3.4 DSL-grounded Video Generation	30
3.5 Evaluation	32
3.6 Discussion and Future Work	37
4 Self-correcting LLM-controlled Diffusion Models	38
4.1 Introduction	38

4.2	Related Work	41
4.3	Self-correcting LLM-controlled Diffusion	41
4.4	Experiments	48
4.5	Conclusion	51
5	Describe Anything: Detailed Localized Image and Video Captioning	52
5.1	Introduction	52
5.2	Related Work	55
5.3	DAM: Describe Anything Model	56
5.4	DLC-SDP: SSL-based Data Pipeline	59
5.5	DLC-Bench: Benchmark for DLC	61
5.6	Results	61
5.7	Discussions and Conclusion	64
6	Relevance of Language-Guided Visual Decomposition	72
6.1	Why Explicit Visual Structure Still Matters	72
6.2	A Continuing Role for LMD, LVD, and SLD	73
6.3	Why Describe Anything Remains Useful	74
6.4	Part I in Retrospect	75
	II Adaptive Parallel Reasoning	76
7	Learning Adaptive Parallel Reasoning with Language Models	77
7.1	Introduction	77
7.2	Related Work	79
7.3	Adaptive Parallel Reasoning	80
7.4	Experiments	84
7.5	Conclusions, Limitations, and Future Work	89
8	ThreadWeaver: Adaptive Threading for Efficient Parallel Reasoning in Language Models	92
8.1	Introduction	92
8.2	Related work	94
8.3	Model	95
8.4	Parallel Trajectory Curation for Supervised Fine-tuning	98
8.5	Reinforcement Learning	100
8.6	Experiments	102
8.7	Conclusion	106
9	Conclusions and Future Work	109
9.1	Conclusions	109
9.2	Future Directions	110

Bibliography	112
A LLM-grounded Diffusion	128
A.1 Preliminary introduction to latent diffusion models	128
A.2 Pseudo-code for layout-grounded image generation	131
A.3 Additional features and use cases from instruction-based scene specification	132
A.4 Additional ablation studies	132
A.5 Are the generated layouts distributed similarly to the in-context examples?	135
A.6 Additional visualizations	137
A.7 Benchmarking VisualChatGPT and GILL for multi-round instruction-based scene specification	137
A.8 Details for SDXL integration	137
A.9 Generating images from languages not supported by the underlying diffusion model	138
A.10 Details for text-to-image benchmarks	139
A.11 Our LLM prompt	139
B LLM-grounded Video Diffusion	145
B.1 Reasoning Statements from GPT-4	145
B.2 Additional Qualitative Visualizations	145
B.3 Our Prompts and In-context Examples for DSL Generation	147
B.4 Details about Our Setup and Benchmarks	148
B.5 Failure Cases	150
B.6 Additional qualitative results with higher video resolution	151
B.7 Additional ablation experiments	152
C Self-correcting LLM-controlled Diffusion	156
C.1 More Comparisons with Image Editing Methods	157
C.2 Comparison between LMMs and Detector-LLM Combinations	157
C.3 Comprehensive Image Generation Results	158
C.4 Our Prompts and Instructions to the LLM	158
D Describe Anything	165
D.1 Challenges in Generating Detailed Localized Descriptions with Off-the-Shelf VLMs	165
D.2 Evaluation Benchmarks	167
D.3 Discussions	169
D.4 Details for DLC-Bench	173
D.5 Additional Ablation Studies	177
D.6 Additional Quantitative Results	178
D.7 Additional Qualitative Results	180
D.8 Implementation Details	182

E	Adaptive Parallel Reasoning	192
E.1	Additional Related Works	192
E.2	Implementation Details	194
E.3	Additional Results with Larger Models	194
E.4	Additional Results with Pretrained Models	195
E.5	Extended Context Window Experiments	196
E.6	Symbolic Search Algorithm	196
E.7	SoS vs. SoS+	197
E.8	Effect of SoS+ Supervised Training Data	198
E.9	Effect of Temperature	199
E.10	Differences Between Sequential Tokens and Wall-Clock Time	199
F	ThreadWeaver	201
F.1	Preliminaries	201
F.2	Derivations for P-GRPO	203
F.3	Implementation Details	205
F.4	Dataset Statistics	208
F.5	Additional Results	209
F.6	Reasoning Trajectory Annotation Pipeline for ThreadWeaver	220
F.7	Discussion: Reward Hacking	235
F.8	Future Directions	235

List of Figures

2.1	(a) Text-to-image diffusion models such as SDXL [1] often struggles to accurately follow prompts that involve negation, numeracy, attribute binding, or spatial relationships. (b) Our method LMD achieves enhanced prompt understanding capabilities and accurately follows these types of prompts.	6
2.2	Our proposed LMD enhances prompt understanding in text-to-image diffusion models through a novel two-stage generation process: 1) An LLM layout generator takes a prompt from the user and outputs an image layout in the form of captioned bounding boxes. 2) A stable diffusion model guided by our layout-grounded controller generates the final image. Both stages utilize frozen pretrained models, which makes our method applicable to off-the-shelf LLMs and other diffusion models without grounding in their training objectives.	6
2.3	LMD naturally enables instruction-based multi-round scene specification and is able to adapt subsequent rounds of generation according to users' followup instructions and clarifications.	7
2.4	In stage 1, LMD generates an image layout from a user prompt. LMD embeds the user prompt into a template with instructions and in-context examples. An LLM is then queried for completion. Finally, the LLM completion is parsed to obtain a set of captioned bounding boxes, a background caption, and an optional negative prompt.	10
2.5	In stage 2, we introduce a novel layout-grounded controller that guides stable diffusion to generate images based on the layout obtained from the previous stage. Our layout-grounded image generation process consists of two steps: (a) generating masked latents for each box specified in the layout, with attention control ensuring that the object is placed in the designated box; and (b) composing the masked latents as priors to guide the image generation to adhere to the specified layout.	12

2.6	LMD and LMD+ support instruction-based scene specification, empowering the users to add/move/remove objects, modify object attributes, and clarify the prompt in multiple rounds of dialog. (a): the initial prompt for the scene; (b)-(i): eight subsequent instructions that <i>sequentially</i> modify the scene. By separating the generation of each foreground object as well as the background, LMD ensures consistent image generation when the same seed is used for image generation throughout the dialog.	14
2.7	LMD outperforms its base text-to-image diffusion model [1] in accurately following the prompts that require spatial and language reasoning. Best viewed when zoomed in.	15
2.8	We qualitatively compare with VisualChatGPT [2] and GILL [3] that also leverage LLMs in the image generation pipelines. Both baselines lack the ability to accurately follow the prompts for three out of four tasks that our method can solve in Figure 2.1.	17
2.9	A failure case occurs when our method, shown in (c) , generates objects in unintentional viewpoints and sizes due to the ambiguity in the generated layout. The LLM-generated layout (b) is suitable for close-up top-down view of a small table, but the layout-to-image model assumes a side view and thus fails to generate a feasible image. Nevertheless, our method still provides more interpretability through the intermediate layout (b) compared to baseline SDXL (a) . With an additional request for the side view and correct object sizes, the LLM adjusted the layout in (d) and the final generation (e) is aligned with the text prompt.	22
3.1	<i>Left:</i> Existing text-to-video diffusion models such as [7] often encounter challenges in generating high-quality videos that align with complex prompts. <i>Right:</i> Our training-free method LVD, when <i>applied on the same model</i> , allows the generation of realistic videos that closely align with the input text prompt.	24
3.2	Our method LVD improves text-to-video diffusion models by turning the text-to-video generation into a two-stage pipeline. In stage 1, we introduce an LLM as the spatiotemporal planner that creates plans for video generation in the form of a dynamic scene layout (DSL). A DSL includes objects bounding boxes that are linked across the frames. In stage 2, we condition the video generation on the text and the DSL with a DSL-grounded video generator. Both stages are training-free: <i>LLMs and diffusion models are used off-the-shelf without updating their parameters</i> . By using DSL as an intermediate representation for text-to-video generation, LVD generates videos that align much better with the input prompts compared to its vanilla text-to-video model counterpart.	27

3.3	In-context examples. We propose to prompt the LLMs with <i>only one in-context example per key desirable property</i> . Example (a)/(b)/(c) demonstrates gravity/elasticity/perspective projection, respectively. In Section 3.3, we show LLMs are able to generate DSLs aligned with the query prompts with only these in-context examples, empowering downstream applications such as video generation.	29
3.4	DSLs generated by the LLM from input text prompts with only in-context examples in Figure 3.3. Remarkably, despite <i>not having any textual explanations on the applicability of each physical property</i> , the LLM discerns and applies them appropriately. (a,b) The LLM selectively applies the elasticity rule to the ball but not the rock, even though the attributes of a rock are never mentioned in the example. (c) The LLM infers world context from the examples and factors in air friction, which is not mentioned either, when plotting the paper airplane’s trajectory. (d-f) reveal the LLM’s inherent grasp of the role of viewpoint in influencing object dynamics, without explicit instructions in the examples. These discoveries suggest that <i>the LLM’s innate knowledge, embedded in its weights, drives this adaptability, rather than solely relying on the provided examples</i>	30
3.5	Videos generated by our method LVD from the DSLs in Figure 3.4. Our approach generates videos that correctly align with input text prompts. (a-c) show various objects thrown from the right to the left. (d-f) depict objects of the same type viewed from different viewpoints.	30
3.6	Our DSL-grounded video generator generates videos from DSLs using existing text-to-video diffusion models augmented with appropriate DSL guidance. Our method alternates between DSL guidance steps and denoising steps. While the denoising steps gradually transform the random video-shaped Gaussian noise into clean videos, our DSL guidance steps ensure that the denoised videos satisfy our DSL constraints. We designed an energy function minimized when the text-to-video cross-attention maps have high values within the box area and low values outside the box w.r.t. the layout box name. In the DSL guidance step, our method minimizes the energy function E by performing gradient descent on the video, steering the generation to follow the constraints.	31
3.7	Despite not aiming for each individual aspect of the quality improvement, LVD generates videos that align much better with the input prompts in several tasks that require numerical, temporal, and 3D spatial reasoning. Best viewed when zoomed in.	36
4.1	Our proposed Self-correcting LLM-controlled Diffusion (SLD) enhances text-to-image alignment through an iterative self-correction process. It begins with LLM-driven object detection (Section 4.3), and subsequently performs LLM-controlled analysis and correction (Section 4.3). The entire pipeline is outlined in Algorithm 1.	40

4.2	Our self-correction pipeline is driven by two distinct LLMs: (a) The LLM parser analyzes user prompts P to extract a list of key object information S , which is then passed to the open-vocabulary detector. (b) The LLM controller, taking both the user prompt P and currently detected bounding boxes B_{curr} as input, outputs suggested new bounding boxes B_{next} . These are subsequently transformed into a set of latent space operations Ops for image manipulation.	42
4.3	Our latent operations can be summarized into two key concepts: (1) latent in removed regions are re-initialized to Gaussian noise, and latent of newly added or modified objects are composited onto the canvas. (2) Latent composition is confined to the initial steps, followed by “unfrozen” steps for a standard forward diffusion process, enhancing visual quality and avoiding artificial copy-and-paste effects.	44
4.4	SLD enhances text-to-image alignment across diverse diffusion-based generative models such as SDXL, LMD+, and DALL-E 3. Notably, as highlighted by the red boxes in the first row, SLD precisely positions a blue bicycle in relation to a bench and a palm tree, while maintaining the accurate count of palm trees and seagulls. The second row further demonstrates SLD’s robustness in complex, cluttered scenes, effectively managing object collision through our training-free latent operations.	47
4.5	When instructed to perform object-level image editing. InstructPix2Pix [37] completely fails to accomplish the task, and DiffEdit [91] falls short, as highlighted in the green box of the image. Conversely, our method demonstrates a significantly better performance in executing these object-level edits.	49
4.6	SLD can handle a diverse array of image editing tasks guided by natural, human-like instructions. Its capabilities span from adjusting object counts to altering object attributes, positions, and sizes.	50
4.7	SLD struggles with objects of complex shapes, as the SAM module may unintentionally segment adjacent parts during the process.	51
5.1	Top: Prior regional captioners derive regional features from global image representations, leading to vague descriptions. Bottom: Zooming in (cropping the image region) enhances detail but loses contextual cues, degrading recognition. This underscores the need for a design that encodes detail-rich regional features while preserving context for improved DLC performance.	54

5.2	Architecture of the Describe Anything Model (DAM). DAM employs a <i>focal prompt</i> to encode user-specified regions with high token density while preserving context for detailed understanding. Focal cropping is applied to the image and its corresponding mask, retaining surrounding areas for local context. Both the full image and the focal crop are the inputs into the <i>localized vision backbone</i> , where images and binary masks are embedded in a spatially aligned fashion. Global context from the full image is leveraged to help understand the focal crop through gated cross-attention. The resulting visual features and prompt tokens are fed into a large language model to generate detailed, context-aware descriptions. * indicates initialized to output zeros.	67
5.3	We propose DLC-Bench, a benchmark tailored to detailed localized captioning. In DLC-Bench, a captioning model is prompted to describe a specified image region (a). The generated description (b) is then evaluated by querying an LLM Judge (c). Points are assigned or deducted based on the LLM’s response (d). The question we show in (c) is an example of positive questions.	68
5.4	DAM generates detailed and localized descriptions, whereas prior works generate descriptions that are less precise. Green: correct description. Red: factual error or mislocalization.	69
5.5	DAM accurately describes user-specified regions in <i>videos</i> and <i>multi-view scenes</i> under challenging conditions.	70
5.6	DAM offers multi-granular localized descriptions.	71
6.1	Current image generators are still not guaranteed to preserve user-specified spatial and counting constraints. Given the prompt “generate an image: six apples arranged in an N-shape,” GPT-image-2 produces a plausible image but does not reliably satisfy the requested N-shaped arrangement, while Nano Banana 2 acknowledges that the result should contain six apples in an N-shape but generates the wrong number of apples. These examples illustrate that current systems can recognize the intended constraint while still failing to enforce it reliably in the visual output.	73

7.1	Serialized search [4] (top) vs Adaptive Parallel Reasoning (bottom) illustrated on an example of the Countdown task, with a target number of 27 and input numbers of {22, 26, 31, 53}. Each box represents a node in the search representing the value of the intermediate expression (x). Edges are annotated with explored arithmetic operations relative to the parent-node x using remaining input numbers. In serialized search, the context window of the single inference thread is exhausted before a solution is found. In Adaptive Parallel Reasoning, the parent thread (blue) spawns two child threads (orange), which are executed in parallel. Child threads have access only to a limited context passed to them by the parent thread and return a summary of their execution to the parent thread. The parent thread can then continue to decode with access to these summaries. This parallel distribution of computation prevents context window exhaustion while reducing latency.	78
7.2	Overview of Adaptive Parallel Reasoning (APR). While previous chain-of-thought methods directly linearize the reasoning tree, Adaptive Parallel Reasoning alternates between the parent thread and parallel child threads to traverse the reasoning tree more efficiently.	81
7.3	An example trajectory of our APR method solving the Countdown task. The model starts with the parent thread, solving the task through reasoning in language, and generates spawn commands to instantiate two child threads for parallel reasoning, which are joined back when finished, after which the main thread continues decoding. Orange denotes prefix input tokens, while Green denotes LM-generated tokens. Bold parts correspond to the spawn and join operations.	83
7.4	Scaling performance of SoS+ vs APR. (a) APR achieves higher accuracy with increasing compute budget compared to SoS+ with Best-of-N sampling. (b) APR more effectively utilizes fixed context windows across different thread configurations. The cumulative accuracy measures the total accuracy considering only the outputs that fall within the context window constraint—that is, only outputs no longer than the set constraint are counted as correct.	85
7.5	Comparison of model performance and statistics before and after reinforcement learning (RL). RL significantly improves performance by optimizing the APR policy beyond what can be learned from supervised imitation learning. Post-RL models strategically increase sequence length (total tokens and avg token per seq) and, more substantially, child thread count (the number of spawned child threads), demonstrating an advantage of broadening rather than deepening search in reasoning. This optimization results in substantially higher accuracy (from 75.5% to 83.4%).	87
7.6	Efficiency comparison between APR and SoS+. Left: accuracy vs. sequential token usage. Right: accuracy vs. wall clock latency. APR consistently achieves higher accuracy with fewer tokens and lower latency, demonstrating superior efficiency.	88

7.7	RL provides a larger performance gain to APR (7.9%) compared to SoS+ (2.7%), resulting in substantial advantages of APR over SoS+ after RL (83.4% vs 60.0%). Ablations with the number of child threads enforced show that such performance gain mainly comes from teaching the model how to allocate compute resources instead of making more optimal decisions with the same resources.	88
8.1	For complex reasoning problems, a sequential reasoner generates tokens one at a time, so latency scales with the trajectory length and is difficult to reduce with additional compute. ThreadWeaver instead adaptively spawns ($\diamond$) and joins ($\blacklozenge$) concurrent threads that work on different parts of the solution, using extra compute to shorten the critical reasoning path.	93
8.2	Format of our parallel reasoning trajectories. <code><Parallel></code> encloses a fork-join block with <code><Outlines></code> and multiple <code><Thread></code> 's; content inside each <code><Thread></code> is generated <i>concurrently</i> , while the other spans are generated sequentially.	96
8.3	Inference-time request sequence for a parallel reasoning trajectory. For a given user prompt, Timestep 1 decodes the prefix and the <code><Outlines></code> block sequentially up to <code></Outlines></code> . Timestep 2 launches one completion request per outline in parallel, each seeded with its corresponding " <code><Thread> i:</code> " prefix and stopped at <code></Thread></code> . Timestep 3 resumes sequential decoding over the joined context until [EOS] or the next <code></Outlines></code> . All steps are executed on a standard LLM inference engine.	98
8.4	Our trie-based training sequence construction has three steps: 1) extract context and completion segments that the inference state machine will encounter in order to produce this trajectory; 2) insert these segments into a token-level trie (prefix-tree); 3) traverse the trie to produce a flat sequence with an ancestor-only attention mask and standard incremental positions counting from the root node. * denotes text spans with loss applied. Although we only show one branching in this example, ThreadWeaver supports branching and joining multiple times in a trajectory.	99
8.5	Per-problem speedup distributions on math benchmarks. The speedup is the ratio between the token latency of the sequential reasoning baseline and the token latency of ThreadWeaver. A vertical reference line at $1.0\times$ marks parity with the sequential baseline. Parentheses show mean speedup per dataset.	105
A.1	Additional features and use cases enabled by instruction-based scene specification.	133

- A.2 **The generated layouts are not necessarily similar to the in-context examples in terms of the spatial distribution of boxes.** We present the LLM with *only one in-context example* and query it with a prompt that is similar to the example. **Top:** While the query and the example shares a similar structure (only one object), the LLM generates a box for “an apple” that is very different from “a panda” in terms of the size and position. **Bottom:** The LLM does not simply copy boxes for the three skiers in the in-context example to generate the boxes for three bears. 136
- A.3 **We also generate images with the same text prompts as Figure 2.1 with SDv1.5 and LMD on SDv1.5.** We observe similar results which show that while Stable Diffusion [13] (a) often struggles to accurately follow several types of complex prompts, our method LMD (b) achieves enhanced prompt understanding capabilities and accurately follows these types of prompts. 137
- A.4 VisualChatGPT [2] and GILL [3] generally cannot handle more than one round of image generation requests, with the generated image degraded starting from the second request. In contrast, our method is able to handle several rounds of sequential requests on image generation without generation degradation, shown in Figure 2.6. 141
- A.5 **LMD can be easily applied on the latest stable diffusion model SDXL [1].** We compare generated images from text prompt “A realistic photo of a **gray cat** and an **orange dog** on the grass”. (a) directly generates the image from the text prompt. SDXL does not accurately generate the image from the prompt, showing that *simply scaling the diffusion model does not necessarily lead to improved prompt following ability*. (b) Thanks to our method being training-free, our method can be directly applied on SDXL without additional training. (c) An alternative way to integrate our method with SDXL is to use our method to generate low-resolution images with SD and then refine the image in high-resolution in SDXL. Since most denoising is completed in low-resolution latents, this approach is more efficient. 142
- A.6 By asking the LLM to always output layouts in English, **LMD is naturally able to generate images from prompts in languages not supported by the underlying diffusion model.** 142

B.1	Additional visualizations of LVD. (a) Given a prompt that has ambiguity (the walking direction of the dog is not given), LLM is still able to make a reasonable guess and often includes the assumption in the reasoning statement, leading to reasonable generation. (b) Our DSL-to-video stage is able to process highly overlapping boxes (outlined in the dashed boxes): the box of the raccoon overlaps about half of the barrel, yet the generation of the raccoon on the barrel is still realistic and not abrupt. (c) Although our base video diffusion model ModelScope often generates static background, which is inherited by our training-free method, our model still works when asked to generate background that changes significantly such as lightning and raining. Note that the model generates raining when only lightning is mentioned in the prompt, which is a reasonable assumption.	148
B.2	Failure case of LLM-generated DSL.	150
B.3	One failure case: <i>A cartoon of an apple moving from the left to the right.</i> Since LVD is training-free, it does not improve the quality on domains/styles that the base model is not good at. Although LVD correctly generates a video that is aligned with the prompt, neither the baseline or LVD generates background that fits the apple in the foreground.	151
B.4	Another failure case: <i>A wooden barrel is drifting on a river.</i> Since LVD controls the cross-attention map between the image and the text, the DSL guidance sometimes leads to undesirable generations during the energy minimization process. In this failure case, the model attempts to hide the barrel on the left with leaves and pop a barrel up from the water to reach the desired energy minimization effect, while the DSL means to move the barrel to the right.	151
C.1	We demonstrate that current diffusion-based image editing methods, such as InstructPix2Pix and DiffEdit, face challenges in fundamental operations like object deletion (top example) and addition (bottom example). In contrast, our proposed SLD pipeline can easily handle these tasks. As highlighted in the red box, despite DiffEdit’s ability to identify the object for removal, it falls short in generating a plausible background.	161
C.2	Leveraging the advanced localization abilities of OWL-ViT v2 open-vocabulary detectors, we accurately identify all seagulls in the image, enabling selective removal to align with the user’s prompt.	162

D.1	Existing Vision-Language Models (VLMs) do not perform well in generating localized descriptions. (a) to (c) demonstrate several ways to prompt existing VLMs, but none achieves satisfactory performance, leading to the need for a new method that is capable of providing detailed and localized descriptions. In (d), we propose a model that accepts the condition in a separate form of input, making it easy for users to specify the object of interest and for the models to accurately localize the referred object. Note that our focal prompt, proposed in Section 5.3, is considered part of the Describe Anything Model and is not shown in the figure for simplicity.	166
D.2	Caveats for using boxes to indicate region of interests. Top: Using a box to indicate the region of interest leads to ambiguity. Middle and Bottom: Switching to a mask representation leads to more specific referring and correct descriptions.	170
D.3	The pitfall of using reference captions for caption evaluation. Evaluation benchmarks based on reference captions may incorrectly treat correct details in the predicted caption as hallucination. Since the GPT evaluator relies solely on the ground truth caption without viewing the video, it mistakenly flags gestures and expressions as hallucinations, resulting in a low score. However, the evaluation is invalid since the predicted details are correct.	171
D.4	Failure cases for our proposed DAM.	172
D.5	An example from DLC-Bench for detailed localized captioning. (a) The process begins by prompting a model to describe a specified region within the image. The resulting description is then evaluated using a text-only LLM as a judge that rates each response by answering positive and negative questions. (b) shows an example description to be evaluated. (c) Positive questions are designed to test whether the model correctly identifies specific details within the described region. The model receives points for accurate details and is penalized for factual errors. The bold option (option C) indicates that the LLM judge believes that option C is applicable, allowing the model to get a point for this example positive question. (d) Negative questions ensure the model refrains from mentioning irrelevant or nonexistent details. Mislocalization or hallucination results in penalties to prevent false positives. The bold option (option B) indicates that the LLM judge believes that option B is applicable, allowing the model to get a point for this negative question.	174
D.6	Emerging zero-shot QA capabilities. DAM could answer questions about regions in an image, showcasing capabilities such as object recognition and property identification.	179

D.7	Additional results from LVIS [161] demonstrating DAM’s detailed localized image captioning capabilities. Our model exhibits robust region understanding and localization across diverse scenarios. It produces precise descriptions of objects within masked regions while successfully identifying challenging details like the roller blind in the second example through effective use of contextual cues.	187
D.8	Additional results from DAM on detailed localized video captioning (Part 1). Our model is able to accurately describe small objects in complex scenes that involve large object motion and camera motion. Our model also correctly identifies stationary objects captured by a non-stationary camera by saying they are stationary. Videos visualized in this figure are from SA-V [159] dataset. .	188
D.9	Additional results from DAM on detailed localized video captioning (Part 2). Our model is able to accurately describe objects that are partially occluded and is able to perceive and describe slight motion. Videos visualized in this figure are from MOSE [244] dataset.	189
D.10	Qualitative comparisons demonstrate the superior localized image understanding capabilities of our model compared to GPT-4o [100] and VP-SPHINX [110], our strongest open-weight baseline. GPT-4o struggles to recognize objects in masked regions accurately, offering only vague descriptions. In the left image, VP-SPHINX incorrectly describes a group of seals when the masked region contains only one seal. In the right image, VP-SPHINX identifies the towel but provides minimal detail, missing key attributes like its color. In contrast, our model delivers precise, detailed descriptions and captures the seal’s sleek elongated body, dark brown coloration with lighter patches, and the towel’s light gray color, wrinkled texture, and darker edge pattern. This superior performance stems from our model’s architecture that effectively fuses object-specific details with broader contextual understanding.	190
D.11	Our proposed DAM demonstrates superior localized video understanding compared to GPT-4o [100], QwenVL-2.5 [174], and VideoRefer [120]. Top figure: DAM accurately captures the cow’s forward movement with comprehensive details, whereas GPT-4o and QwenVL-2.5 mistakenly perceive the cow as stationary. Compared to VideoRefer, DAM provides richer descriptions of both motion and appearance. Bottom figure: DAM correctly recognizes the animal as a sheep and accurately describes its graceful movement, while GPT-4o erroneously identifies it as transforming into other animals, and QwenVL-2.5 incorrectly perceives that only the sheep’s head is moving. VideoRefer provides limited appearance details, while DAM offers extensive, accurate descriptions. These cases highlight DAM’s precise understanding of motion and appearance throughout video sequences.	191

E.1	Accuracy vs. average total compute (in thousands of tokens) for 200M and 600M parameter models. APR consistently outperforms SoS+ and scales more strongly with model size.	195
E.2	Accuracy vs. context window size on the five-number Countdown task for SoS+ and APR.	196
E.3	Ablation of APR and SoS+ vs. total compute under different temperature settings. Across all settings, APR demonstrates more reliable and efficient scaling behavior.	199
F.1	Trie-based sequence merging with more than one parallel block. We still build a prefix-tree (trie) based on the <context, completion> pairs from the training example and then flatten the tree into one packed training sequence with an ancestor-only attention mask.	206
F.2	Per-problem speedup distributions on math benchmarks without filtering for correctness. The speedup is the ratio between the token latency of the autoregressive baseline and the token latency of ThreadWeaver. A vertical reference line at 1.0× marks parity with the sequential baseline. In contrast to Figure 8.5, these histograms include all rollouts rather than restricting to samples with correct final answers. However, taking both sets of histograms into account, we show that ThreadWeaver achieves substantial speedup on most problems no matter whether correctness filtering is performed.	218

List of Tables

2.1	With guidance from the LLM-based layout generator and our novel layout-grounded controller, our LMD significantly outperforms the Stable Diffusion model (SD) that we use under the hood in four tasks benchmarking prompt-following abilities. LMD denotes our method directly applied on SD. LMD+ denotes additionally integrating pretrained GLIGEN [5] adapters into our controller.	17
2.2	Ablations on layout-to-image methods as stage 2 with our LLM layout generator as stage 1. Our proposed layout-grounded controller performs the best among them. Our controller could also be applied on top of training-based GLIGEN [5], denoted as LMD+, for additional improvements. Finally, the LLM-generated layouts almost always align with the prompt, highlighting that the bottleneck is the layout-grounded image generation. The scores for negation task are high because we pass the negative prompts generated by the LLM to the underlying diffusion model, which does not depend on the stage 2 implementation.	18
2.3	LMD achieves comparable gains when adapted to Stable Diffusion v2.1 without any hyperparameter tuning or model training. This shows a promising signal that the gains from our method could carry along with the enhancement of diffusion models. The performance of our method could potentially be improved further with additional hyperparameter tuning.	20
2.4	Ablations on using SAM vs using simple attention thresholding in stage 2. While removing SAM leads to a slight degradation in LMD, removing SAM leads to even better performance in LMD+.	20
2.5	Ablations on the number of in-context examples (“shots”) given to the LLM. While GPT-3.5 benefits from more in-context examples, GPT-4 already excels in layout generation even with only one example.	21
2.6	Our method surpasses the base diffusion models SDv1 and SDv2 on T2I-CompBench [6] on all four tasks without additional training.	21
3.1	Evaluation of the generated DSLs. Our LLM spatiotemporal planner is able to generate layouts that align well with the spatiotemporal dynamics requirements in several types of prompts.	33

3.2	Detection-based evaluation of the generated videos. LVD follows the spatiotemporal keywords in the prompts better than ModelScope, the diffusion model that it uses under the hood. Generated videos have a resolution of 256×256 .	33
3.3	Ablations on the energy function used to condition the video diffusion model on the layouts. We use the same set of layouts generated by the LLM (GPT-4) in the ablation. Neither E_{ratio} proposed in [15] nor E_{corner} proposed in Boxdiff [16] help improve the text-video alignment, whereas simply using E_{topk} achieves good text-video alignment. Adding E_{CoM} can further boost the text-video alignment for sequential tasks. Generated videos are of 256×256 .	34
3.4	Video quality evaluation. Lower FVD scores are better. [†] indicates our replication of the results. LVD is able to generate videos with higher quality than ModelScope on both benchmarks.	34
3.5	Assessment based on human evaluators. Despite sharing <i>the same diffusion weights</i> as the baseline model [7], our method is preferred significantly more often.	35
4.1	Our method can be applied on various image generation methods and improves the generation accuracy by a large margin.	46
4.2	While the majority of errors are typically rectified in the first round, multi-round correction consistently outperforms a single-round approach.	50
5.1	Advantages of our proposed model DAM, our SSL data pipeline DLC-SDP, and our benchmark DLC-Bench to previous practices.	55
5.2	LVIS [161] and PACO [162] open-class keyword-level captioning benchmarks. DAM excels particularly in the challenging PACO benchmark that requires distinguishing between objects and parts.	59
5.3	Zero-shot evaluation on phrase-level dataset Flickr30k Entities [163]. Our model achieves 12.3% average relative improvement against previous best.	59
5.4	Zero-shot evaluation on the detailed captioning dataset Ref-L4 [164]. Our method achieves 33.4% and 13.1% average relative improvement on the short/long language-based captioning metrics, respectively.	59
5.5	Accuracies on detailed localized captioning in our proposed DLC-Bench. DAM outperforms previous API-only models, open-source models, and region-specific VLMs on detailed localized captioning. <u>Underlined</u> : the second-best method. †: models with thinking mode.	63
5.6	Detailed localized <i>video</i> captioning on HC-STVG [169].	64
5.7	Performance on detailed localized video description on VideoRefer-Bench-D [120]. †: We provide analysis on hallucination scores (HD). *: trained on in-domain VideoRefer-700k with regard to VideoRefer-Bench, both sourcing videos from Panda-70M [170].	65

5.8	Ablation studies across different visual prompts, cross-attention settings, and the number of image tokens. Local crop denotes cropping without surrounding context. XAttn: cross-attention. #IT: number of image tokens. * indicates the full image and the crop are concatenated on the sequence dimension. Bold: Our proposed focal prompt	65
5.9	Our model benefits from diverse datasets generated by DLC-SDP. Scaling the dataset in size and diversity significantly improves model performance, and SSL further enhances performance using widely available unannotated images.	66
8.1	Comparison between the sequential GRPO baseline and ThreadWeaver on Qwen3-8B across six math benchmarks. ThreadWeaver matches the baseline’s accuracy (71.9% vs. 72.2%) while reducing mean critical-path length from 15.1k to 13.2k tokens and achieving up to 1.53× average speedup and up to 3.56× max speedup . Speedups are computed as ratios of the token latencies of the sequential reasoning baseline and ThreadWeaver.	103
8.2	Comparison with other adaptive parallel reasoning approaches on AIME24. Self-parallelism speedup is the ratio between the total number of tokens and the critical path length. Activation ratio is defined as the percentage of samples with parallelization in the inference trajectories. ThreadWeaver achieves higher accuracy and stronger self-parallelism than Multiverse and Parallel-R1. Results from Yang et al. [8] and Zheng et al. [9].	104
8.3	Wall-clock latency on 50 MATH problems (averaged over 8 runs per problem).	105
8.4	Effect of std-dev normalization on AIME 24 (Avg@32).	106
8.5	Ablation on parallel rollout and self-training (AIME 24).	106
A.1	Ablations on different LLMs in stage 1. Although proprietary models such as GPT-3.5 and GPT-4 perform the best, the ability to generate high-quality layouts is also present in open-source models <i>Stable beluga models, 2023</i> [10], <i>Mixtral of experts</i> [11], and <i>Llama 2: Open foundation and fine-tuned chat models, 2023</i> [12]. The image accuracy is benchmarked using LMD+ as stage 2.	134
A.2	Ablations on the LLM model size on StableBeluga Models [10] based on Llama-2 [12] for layout generation (stage 1 only). Larger LLMs offer more accurate layout generation compared to smaller LLMs.	134
A.3	Ablations on hyperparameter ω and λ. Our method is relatively stable in terms of hyperparameter values ω and λ . While we did not perform hyperparameter search, our default hyperparameter $\omega = 4$ allows optimal performance for both LMD and LMD+. For the hyperparameter λ , we found that setting $\lambda = 3$ leads to better performance compared to our default hyperparameter setting with $\lambda = 2$, which indicates that the performance of our method can be further improved through hyperparameter tuning. <u>Underlined numbers</u> indicate performance with our default hyperparameter selection ($\omega = 4, \lambda = 2$). Bold numbers indicate the best performance among all the hyperparameters ablated.	135

A.4	Our method surpasses the base diffusion model SD as well as several variants of our method that combines our stage 1 with previous layout-to-image methods as stage 2 on T2I-CompBench [6].	136
A.5	Our full prompt to the LLM for layout generation. LLM starts completion from “Objects:”.	143
A.6	Our in-context examples. We use fixed in-context examples for layout generation.	144
B.1	Reasoning statements from GPT-4.	146
B.2	Additional results and ablations on videos of higher resolution. Generated videos are of 512×512 . † indicates unnormed corner energy function.	152
B.3	Ablations on the number of in-context examples. We found that 3 in-context examples are sufficient for stable performance. †: We observe instabilities when we only use one example with GPT-3.5, where two of the three runs could not finish due to invalid DSLs generated.	153
B.4	Ablations on the number of DSL guidance steps between two denoising steps in LVD. We use 5 DSL guidance steps as the performance saturates with more DSL guidance steps.	153
B.5	Our prompt for DSL generation. When using API calls, line 1-3 presented to GPT-4 with role “system”. Each caption line in the in-context examples and user query are assigned with role “user”. The example outputs are assigned with role “assistant”. These lines are merged into one message when querying GPT-4 through the ChatGPT web interface.	154
B.6	Our in-context examples for DSL generation. We use these three in-context examples that we use throughout our work unless stated otherwise.	155
C.1	Our method can be applied on various image generation methods and improves the generation accuracy by a large margin.	157
C.2	Our full prompt for the LLM parser.	159
C.3	Our full prompt for the LLM controller.	160
C.4	Self-correction in-context examples for the LLM controller.	163
C.5	Image editing in-context examples for the LLM controller.	164
D.1	Ablations on architecture design compared to our strongest baseline VP-SPHINX [110]. We trained a model with VP-SPHINX [110] architecture on our curated DLC data from various segmentation datasets. The results on DLC-Bench indicate the advantages of our model architecture that allows detailed localized features to be presented to the LLM for DLC.	176

D.2	Comparison of performance of DAM with and without prompt augmentation. Prompt augmentation has minimal effect on DAM’s performance on DLC-Bench. While descriptions generated by the model may occasionally be less detailed, leading to a slight decrease in the performance on positive questions, we observed that prompt augmentation enhances instruction following when prompts include specific guidelines, such as length constraints. We use the model without prompt augmentation with our benchmark, including ablations, by default.	176
D.3	Comparison of performance of our image-only DAM and DAM trained with both localized image description task and localized video description task. Joint training benefits generating high-quality localized image descriptions compared to image-only training.	176
D.4	Additional results with existing general VLMs using Set-of-Mark (SoM) prompting [144]. The results are accuracies on detailed localized captioning in DLC-Bench. Compared with results in Table 5.5 which are obtained with the same prompt engineering as we use in the stage 1 of our data pipeline, SoM leads to degraded quality. In this comparison, the advantages of our method, compared with prior baselines, are much larger. Negative numbers are due to penalties from factual errors. Note that region-specific VLMs, including our proposed DAM, have predefined ways of inputting regions, and thus SoM prompting is not applicable to these models. †: models with thinking mode.	178
D.5	Dataset statistics across stages with total images and regions for training detailed localized image captioning. In stage 1, we annotated 684k regions across 226k images from existing instance and semantic segmentation datasets. In stage 2, we perform SSL on 10% of SA-1B images without using the masks provided by the datasets, resulting in 774k regions across 593k images. In total, we annotated 1.46M regions across 819k images with detailed localized descriptions. This diverse and high-quality dataset is the key to our model’s performance. Note that due to filtering the number of instances and images are lower than the number of instances and images in the original dataset.	181
D.6	Dataset statistics across stages with total videos and regions for training detailed localized video captioning. We label 94k regions across 37k videos from SA-V dataset [159] for detailed localized video captioning. Note that each region indicates an instance across multiple frames in the video.	182
D.7	Our prompt for data annotation in stage 1.	183
E.1	Accuracy of SoS+ and APR when fine-tuning on a pretrained Qwen2.5 1.5 B model versus a Llama 2 200 M model.	195
E.2	Accuracy comparison between SoS and SoS+ across different temperature settings. SoS+ consistently achieves higher performance, especially at lower temperatures.	197
E.3	Accuracy of SoS+ trained with different supervision strategies.	199
E.4	Accuracy of different search strategies across sampling temperatures. Each base method is followed by its variant trained with RL.	200

F.1	Statistics for the three training trajectory datasets. “Sequential” are raw Qwen3-8B trajectories subsampled from the 36k trajectories with the correct answers; “First-stage” are GPT-5-rewritten parallel traces; “Self-train” are trajectories generated by the model trained with data in the first stage and filtered as in Section 8.4. Parallel and acceleration ratios are computed per trajectory then averaged.	208
F.2	Stagewise performance on AIME 24. <i>Format correctness</i> checks proper closure of all opened <code>Parallel</code> blocks and well-formed thread markers (a proxy for structural stability). Common reasons for format error are truncation under token budgets and early stopping without closing the parallel block. Correctness of the final answer depends only on the last <code>\boxed{}</code>	210
F.3	Ablation of sequential-parallel cold-start mixing strategies during SFT on AIME 24. “No sequential mixture cold start” uses only parallel data in SFT. “Multiverse-style mixing recipe” follows a non-gradual schedule that switches between mostly sequential, mixed, and mostly parallel phases. “Linear mixing recipe” uses an eight-stage curriculum that linearly increases the proportion of parallel data across epochs. Neither strategy improves over the parallel-only SFT setting, suggesting that additional sequential data does not help Qwen3-8B transition to parallel reasoning once it already has strong sequential capabilities.	210
F.4	Comparison of base model performance on AIME 24 (Avg@32). AR SFT uses sequential Qwen3-8B trajectories; Our SFT uses GPT-5-rewritten parallel trajectories.	212
F.5	Two notions of efficiency. Speedup vs. sequential baseline reflects switching from sequential to parallel reasoning. Self-parallelism speedup reflects internal parallel breadth. Speedups are computed per-problem and averaged.	213
F.6	SFT data comparison on AIME 24. Our data yields stronger performance. *Results from [9] (Avg@16); others use Avg@32.	214

Acknowledgments

I am deeply grateful to the mentors, collaborators, friends, and family who supported me throughout this journey.

First, I would like to thank my advisors, Trevor Darrell and Adam Yala. Trevor, thank you for creating an environment where ambitious ideas in vision, language, and learning could take shape, and for giving me the freedom to explore directions that crossed boundaries between generation, perception, and reasoning. Adam, thank you for your guidance, clarity, and trust as my research moved toward multi-agent systems, reinforcement learning, and parallel reasoning. I am grateful to both of you for believing in me, for pushing me to think bigger, and for helping me turn scattered ideas into a coherent research agenda. I also thank Kurt Keutzer and Yin Cui for serving on my committee and for their thoughtful feedback on this dissertation.

I am especially grateful to Xudong Wang, Victoria Lin, and Yin Cui for their mentorship. Their guidance shaped not only the technical directions of my work, but also how I learned to ask research questions, evaluate evidence, and communicate ideas clearly.

I have been fortunate to work with many collaborators who shaped the work in this thesis. Thank you to Letian (Max) Fu, Baifeng Shi, Jiayi Pan, Xiuyu Li, Yifei Zhou, Boyi Li, Tsung-Han Wu, Charlie Snell, Felix Juefei-Xu, Tsu-Jui Fu, Sida Wang, Yuandong Tian, and so many others. The papers in this dissertation are collective work, and I learned from each of you through the many discussions, experiments, revisions, and late-stage pushes that made these projects possible.

I am also grateful to the broader research communities that shaped my path. BAIR and Berkeley EECS provided an extraordinary place to grow as a researcher, surrounded by people who cared deeply about both technical rigor and the future of AI. I am thankful to Stella Yu for advising my undergraduate research at Berkeley and helping me begin this path, and to the mentors and teams I worked with during internships at Baidu, NVIDIA Research, and Meta MSL.

I am especially thankful for the labmates and friends who made Berkeley feel like home: Amil Dravid, Brent Yi, Chenfeng Xu, Chung Min Kim, Dacheng Li, Grace Luo, Hang Gao, Haozhi Qi, Kaihua Hou, Konpat Preechakul, Lisa Dunlap, Ruilong Li, Ruiqi Zhong, Shiyi Cao, Sizhe Chen, Toru Lin, Wenjie Ma, Xinyan Hu, Yossi Gandelsman, Zhongyuan Liang, and many others. Thank you for the conversations, meals, advice, shared deadlines, and the everyday companionship that made the Ph.D. much less solitary.

I would also like to thank the people at Thinking Machines Lab, including Alexander Kirillov, Alexander H. Liu, Andrea Madotto, Brandon Trabucco, Giovanni D’Antonio, Guangxuan Xiao, Horace He, Jacob van Gogh, John Schulman, Kevin Lu, Lili Yu, Lilian Weng, Mianna Chen, Piotr Dollár, Rowan Zellers, Shaojie Bai, Songlin Yang, Tim Li, Tyler Griggs, Yong Lin, Ziqiao (Martin) Ma, and so many others. Although my primary work there is not included in this dissertation, they helped me greatly in both work and life during the final stage of my Ph.D. and while I was writing this dissertation. I am grateful for their encouragement, conversations, and suggestions.

Finally, I want to thank the support of my family and partner. To my mom and dad: thank you for your unconditional love, patience, and support, and for giving me the confidence to keep going through the uncertain parts of this process. To my partner, Julie Zhu: thank you for being beside me through the ordinary days and the difficult ones, for your encouragement when experiments failed, and for reminding me that life is larger than any deadline.

Chapter 1

Introduction

1.1 Motivation

Modern foundation models have transformed how computers process language, images, videos, and long chains of reasoning. A single model can now write code, answer questions, generate images from text, describe visual scenes, and solve mathematical problems. Yet many tasks that matter in practice remain difficult for monolithic generation. Complex prompts often contain multiple objects, relations, constraints, and temporal events. Visual understanding often requires focusing on a specific region while retaining global context. Hard reasoning problems often admit independent subproblems that should be explored in parallel rather than forced through one serial chain of thought. In each case, the bottleneck is not only model capacity, but also the structure through which a task is represented and executed.

This dissertation studies task decomposition with multi-agent systems as a way to address this bottleneck. The central idea is simple: complex tasks become more tractable when they are broken into interacting subproblems with explicit intermediate representations. A language model can translate a text prompt into a spatial layout; a generation model can ground visual synthesis on that layout; a verifier or controller can detect failures and propose corrections; a captioning model can combine local and global visual evidence; a reasoning system can fork into parallel threads and later join the results. These systems are “multi-agent” in a broad sense: they coordinate multiple roles, modules, calls, or reasoning threads, each responsible for a different part of the computation.

Large language models make this decomposition especially powerful. Because they are trained on broad linguistic and procedural knowledge, they can act as planners, parsers, controllers, annotators, and reasoning agents without requiring task-specific supervision for every new decomposition. At the same time, language alone is not enough. The decompositions must be grounded in visual representations, connected to controllable generative models, and implemented in ways that improve accuracy or efficiency. This thesis therefore asks how language-mediated decomposition can be made concrete across visual generation, visual data

generation, and reasoning.

1.2 Summary of Contributions

This dissertation presents six research contributions organized around the central theme of task decomposition with multi-agent systems.

Chapter 2 — LLM-grounded Diffusion. Chapter 2 introduces LLM-grounded Diffusion, a training-free method for improving prompt following in text-to-image diffusion models. The method decomposes image generation into two stages: an LLM first parses a complex prompt into a structured layout with objects, bounding boxes, background descriptions, and negative prompts; a layout-grounded diffusion controller then uses this structure to guide image synthesis. This decomposition allows the system to handle prompts involving counting, negation, attribute binding, and spatial relations more reliably than direct text-to-image generation.

Chapter 3 — LLM-grounded Video Diffusion. Chapter 3 extends language-guided decomposition from images to videos. LLM-grounded Video Diffusion asks an LLM to generate dynamic scene layouts that describe objects and their bounding boxes over time, then uses these layouts to guide text-to-video diffusion. This makes temporal and spatial constraints explicit, allowing the system to better follow prompts involving motion, object interactions, and sequential events. The chapter shows that LLMs can provide useful spatiotemporal structure even without being trained as video generators.

Chapter 4 — Self-correcting LLM-controlled Diffusion. Chapter 4 studies decomposition as an iterative correction process. Self-correcting LLM-controlled Diffusion uses external perception and LLM control to analyze a generated image, identify mismatches with the prompt, and propose latent-space operations that correct the image. Rather than relying on a single generation pass, the system decomposes synthesis into generation, detection, diagnosis, and repair. This enables training-free self-correction for images that fail on object counts, attributes, or spatial relations.

Chapter 5 — Describe Anything. Chapter 5 introduces Describe Anything, a detailed localized captioning system for user-specified regions in images and videos. Although the method is a vision-language model, it belongs to the visual generation part of this thesis because rich localized descriptions can serve as synthetic supervision for systems such as LLM-grounded Diffusion and LLM-grounded Video Diffusion. The method decomposes visual evidence into local focal crops and global context, then combines them through a localized vision backbone. This design preserves fine-grained regional detail while avoiding the context loss that occurs when a region is cropped in isolation. Beyond synthetic data generation

for controllable image and video synthesis, this capability also supports applications such as accessibility for visually impaired people. The chapter also introduces DLC-Bench, a benchmark for evaluating detailed localized captioning without relying on incomplete reference captions.

Chapter 6 — Relevance of Language-Guided Visual Decomposition. Chapter 6 reflects on the continuing relevance of the visual decomposition methods in Part I. It argues that although modern image and video generators have become much stronger, explicit spatial, temporal, and localized descriptive structure remains useful for prompts and workflows that require correctness, controllability, inspection, and data generation rather than only plausible visual synthesis. The chapter also discusses why detailed localized captioning remains a valuable standalone task and a useful source of synthetic supervision for controllable visual generation.

Chapter 7 — Adaptive Parallel Reasoning. Chapter 7 studies decomposition in language-model reasoning. Adaptive Parallel Reasoning trains models to use explicit `spawn` and `join` operations, allowing a solution process to branch into parallel sub-searches and later combine their results. This chapter shows that parallel reasoning can improve the accuracy-latency tradeoff by allocating computation to multiple reasoning paths when a problem admits useful decomposition, rather than committing to one serial trajectory.

Chapter 8 — ThreadWeaver. Chapter 8 develops ThreadWeaver, a framework showing that adaptive parallel reasoning can work with strong reasoning language models. Building on the fork-join decomposition studied in Chapter 7, ThreadWeaver trains existing sequential reasoning LLMs to dynamically decide when to spawn concurrent threads and when to merge their results. The chapter shows that this kind of parallel reasoning can match the accuracy of comparable sequential thinking models while reducing critical-path token latency, making decomposition useful not only as an algorithmic search strategy but also as a practical route to faster reasoning inference.

1.3 Thesis Organization

The rest of this thesis is organized as follows. Part I (Chapters 2–6) addresses language-guided visual generation, presenting methods that use LLMs to decompose and control image and video synthesis, a localized captioning model that can generate rich visual descriptions for downstream data creation, and a synthesis chapter on the continuing relevance of these ideas. Part II (Chapters 7–8) addresses adaptive parallel reasoning, presenting methods that decompose language-model inference into concurrent reasoning threads. Chapter 9 concludes with a summary of findings and directions for future work.

Part I

Language-Guided Visual Generation

Chapter 2

LLM-grounded Diffusion: Enhancing Prompt Understanding of Text-to-Image Diffusion Models with Large Language Models

This chapter is based on the following publication:

Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. “LLM-grounded Diffusion: Enhancing Prompt Understanding of Text-to-Image Diffusion Models with Large Language Models.” *Transactions on Machine Learning Research (TMLR)*, 2024.

2.1 Introduction

The field of text-to-image generation has witnessed significant advancements, particularly with the emergence of diffusion models. These models have showcased remarkable capabilities in generating realistic and diverse images in response to textual prompts. However, despite the impressive results, diffusion models often struggle to accurately follow complex prompts that require specific capabilities to understand. Figure 2.1 shows that Stable Diffusion [13], even the latest SDXL [1], often could not generate a certain number of objects or understand negation in the prompt. It also struggles with spatial reasoning or associating attributes correctly with objects.

One potential solution to address this issue is of course to gather a comprehensive multi-modal dataset comprising intricate captions and train a text-to-image diffusion model for enhanced prompt understanding. Nonetheless, this approach presents notable drawbacks. It requires considerable time and resources to curate a diverse and high-quality multi-modal dataset, not to mention the challenges associated with training or fine-tuning a diffusion model on such extensive data.

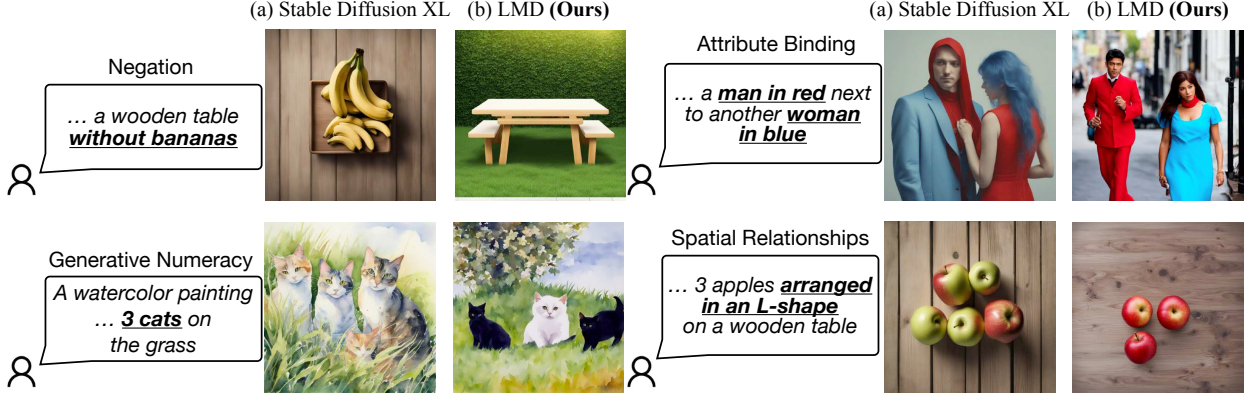


Figure 2.1: (a) Text-to-image diffusion models such as SDXL [1] often struggles to accurately follow prompts that involve negation, numeracy, attribute binding, or spatial relationships. (b) Our method LMD achieves enhanced prompt understanding capabilities and accurately follows these types of prompts.

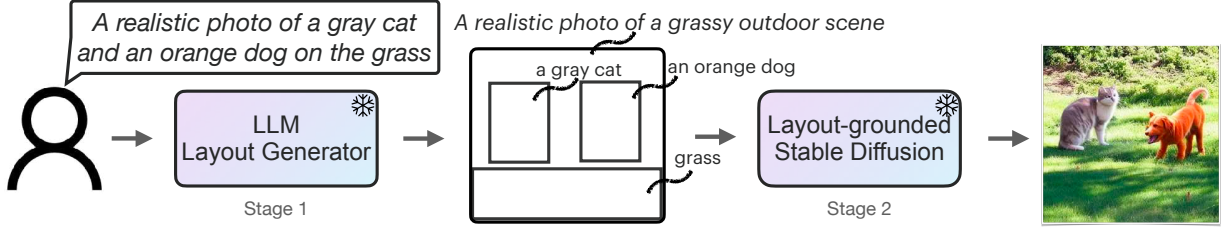


Figure 2.2: **Our proposed LMD enhances prompt understanding in text-to-image diffusion models through a novel two-stage generation process:** 1) An LLM layout generator takes a prompt from the user and outputs an image layout in the form of captioned bounding boxes. 2) A stable diffusion model guided by our layout-grounded controller generates the final image. Both stages utilize frozen pretrained models, which makes our method applicable to off-the-shelf LLMs and other diffusion models without grounding in their training objectives.

In contrast, we propose a novel *training-free* method that equips the diffusion model with an LLM that provides grounding for enhanced prompt understanding. Our method **LLM-grounded Diffusion (LMD)** consists of a two-stage generation process as shown in Figure 2.2.

In the first stage of our method, we adapt an LLM to be a text-grounded layout generator through in-context learning. Given a prompt describing the desired image, the LLM generates scene layouts in the form of captioned bounding boxes, with a background caption and a negative prompt for what to avoid in generation.

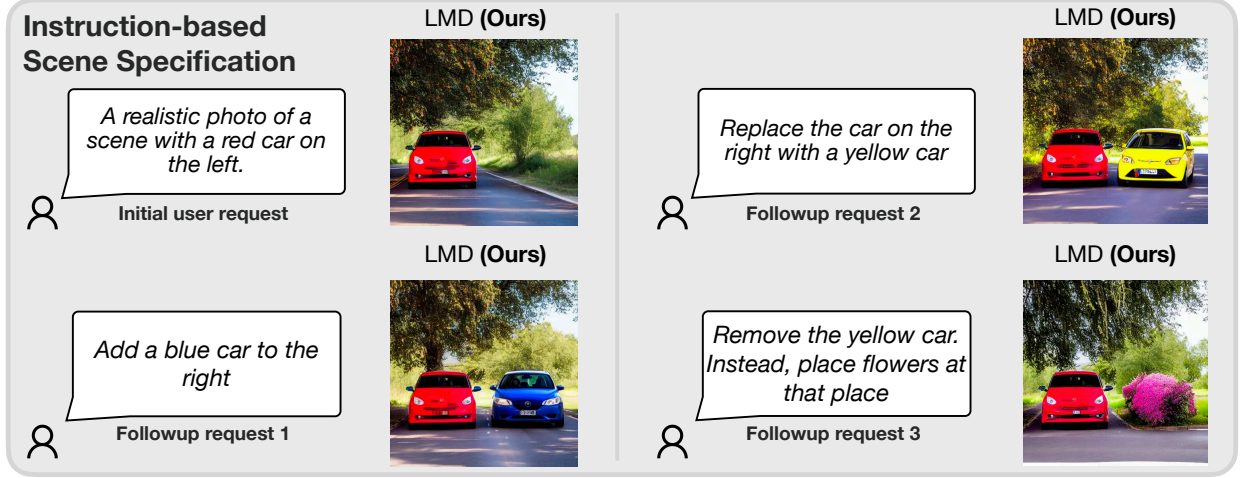


Figure 2.3: **LMD naturally enables instruction-based multi-round scene specification** and is able to adapt subsequent rounds of generation according to users’ followup instructions and clarifications.

In the second stage, we introduce a novel controller that guides an existing diffusion model without grounding in its training objective (e.g., Stable Diffusion) to follow the layout grounding generated in the first stage. In contrast to previous and concurrent works on region control (e.g., *Multidiffusion: Fusing diffusion paths for controlled image generation* [14], *Training-free layout control with cross-attention guidance* [15], and *Boxdiff: Text-to-image synthesis with training-free box-constrained diffusion* [16]) that apply *semantic* control to certain spatial regions, our approach allows precise control over object *instances* in designated regions.

Notably, both stages utilize frozen pretrained models *off-the-shelf*, making our method applicable to LLMs and diffusion models trained independently *without any LLM or diffusion model parameter optimization*.

In addition to enhanced prompt understanding, our method also naturally enables instruction-based scene specification with multiple rounds of user requests (Figure 2.3) and image generation from prompts in languages not supported by the base diffusion model (Figure A.6) without additional training.

Shown in Figure 2.1, LMD provides a unified solution to several caveats in prompt understanding *at once* and enables accurate and high-quality image generation from complex prompts. We demonstrate that a diffusion model grounded with LLM-generated layouts outperforms its base diffusion model and several recent baselines, *doubling* the average generation accuracy across four tasks. **Our primary contributions include:**

1. We propose a training-free two-stage generation pipeline that introduces LLMs to improve the prompt understanding ability of text-to-image diffusion models.

2. We introduce layout-grounded Stable Diffusion, a novel controller that steers an off-the-shelf diffusion model to generate images grounded on instance-level box layouts from the LLM.
3. LMD enables instruction-based scene specification and allows broader language support in the prompts.
4. We propose a benchmark to assess the prompt understanding ability of a text-to-image model and demonstrate the superior performance of LMD over recent baselines.

We expect LMD to empower users with more precise control of text-to-image diffusion models. Our code, demo, and benchmark are publicly available.

2.2 Related Work

Text-to-image diffusion models. High-quality image generation from textual descriptions with diffusion models has been popular recently [17, 18, 13, 1]. Despite the impressive visual quality, these models still tend to exhibit unsatisfactory performance when it comes to complex prompts that involve skills such as binding attributes to objects and spatial reasoning [17].

LLMs for visual grounding. Many multi-modal models benefit from integrating LLMs for grounding vision models. BLIP-2 [19] bootstraps vision-language pre-training from a frozen image encoder and an LLM. Flamingo [20] tackles tasks such as few-shot visual question-answering and captioning tasks. *Layouttransformer: Layout generation and completion with self-attention* [21] uses Transformer [22] for layout prediction but focuses on generating layouts for a limited closed set of object classes in the annotated training set and thus is not able to generate layouts for objects not in the training set. *Visual chatgpt: Talking, drawing and editing with visual foundation models* [2] and *Generating images with multimodal language models* [3] also involve LLMs in conditional image generation. However, these methods still rely on CLIP text embeddings to convey the information to the diffusion model. Therefore, they often exhibit insufficient control compared to our method, which explicitly asks the LLM to reason about the spatial composition of different objects and poses direct spatial control. Concurrent to our work, LayoutGPT [23] proposes prompting an LLM for layout generation in a CSS structure. While LayoutGPT depends on a dataset annotated with boxes and captions to retrieve relevant in-context examples for the LLM, our method demonstrates that the ability for generating high-quality layouts is already present in pretrained LLM weights and can be prompted with a fixed set of in-context examples without external annotations.

Spatially-conditioned image generation methods. These methods create images based on given priors such as poses, segmentation maps, strokes, and layouts. Prior to the popularity of diffusion models, SPADE [24], BlobGAN [25], and Layout2Im [26] synthesize photorealistic images from a given layout. *Scene graph generation by iterative message passing* [27], *Image generation from scene graphs* [28], and *Learning canonical representations for scene graph*

to image generation [29] generate images with scene graphs. ControlNet [30], SpaText [31], LayoutDiffuse [32], LayoutDiffusion, [33], GLIGEN [5] and ReCo [34] propose training-based adaptation on the diffusion models for spatially-conditioned image generation, with *Gligen: Open-set grounded text-to-image generation* [5] and *Reco: Region-controlled text-to-image generation* [34] supporting open-vocabulary labels for layout boxes. However, these methods rely on annotated external datasets such as COCO [35] to supply images with annotations such as boxes and captions. Furthermore, training-based adaptation makes the model incompatible to add-ons such as LoRA weights [36] and renders it difficult to train a new LoRA model from a training set without box annotations. In contrast, we propose a training-free generation controller that steers *existing* text-to-image diffusion models that are *not specifically trained* for layout-grounded image generation and does *not* require external datasets. Furthermore, our method can also integrate with training-based methods for further improvements.

Very recently, *Multidiffusion: Fusing diffusion paths for controlled image generation* [14], *Training-free layout control with cross-attention guidance* [15], and *Boxdiff: Text-to-image synthesis with training-free box-constrained diffusion* [16] allow training-free region control in image generation and share a similar task formulation to our layout-to-image stage. However, these works ground the image generation on the region *semantics* and pose little control over the number of object instances inside each semantic region, whereas our method focuses on grounding generation on *instances*.

Similar to our instruction-based scene specification, *Instructpix2pix: Learning to follow image editing instructions* [37] recently proposed instruction-based image editing. *Visual chatgpt: Talking, drawing and editing with visual foundation models* [2] and *Visual programming: Compositional visual reasoning without training* [38] also allow using external image editing models in an LLM-driven dialog. Different from these methods, we aim to edit the *scene layout* rather than the *image pixels*, which easily allows support for a greater set of instructions such as swapping/moving objects.

2.3 LLM-grounded Diffusion

In this section, we introduce our method **LLM-grounded Diffusion** (LMD). LMD focuses on the text-to-image generation setting, which involves generating image $\mathbf{x}_0$ given text prompt $\mathbf{y}$. Our method generates an image in two stages: text-grounded layout generation (Section 2.3) and layout-grounded image generation (Section 2.3). The layout-to-image stage of our method LMD builds upon the latent diffusion framework [13], for which we refer readers to Section A.1 for preliminaries.

LLM-based Layout Generation

To generate the layout of an image, our method embeds the input text prompt $\mathbf{y}$ into a template and queries an LLM for completion (Figure 2.4).

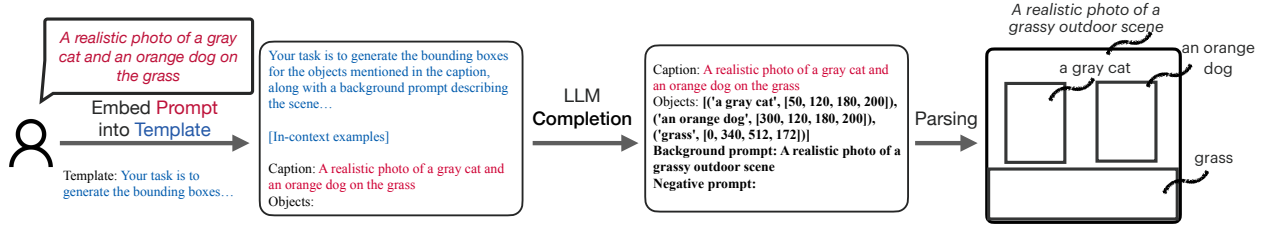


Figure 2.4: **In stage 1, LMD generates an image layout from a user **prompt**.** LMD embeds the user **prompt** into a **template** with instructions and in-context examples. An LLM is then queried for completion. Finally, the LLM completion is parsed to obtain a set of captioned bounding boxes, a background caption, and an optional negative prompt.

Layout representation. LMD’s layout representation comprises two components: **1)** a captioned bounding box for each foreground object, with coordinates specified in the $(x, y, width, height)$ format, and **2)** a simple and concise caption describing the image background along with an optional negative prompt indicating what should not appear in a generated image. The negative prompt is an empty string when the layout does not impose restrictions on what should not appear.

Instructions. Our text instructions to the LLM consist of two parts:

1. Task specification:

Your task is to generate the bounding boxes for the objects mentioned in the caption, along with a background prompt describing the scene.

2. Supporting details:

The images are of size 512×512... Each bounding box should be in the format of ... If needed, you can make reasonable guesses.

In-context learning. Similar to *Instructpix2pix: Learning to follow image editing instructions* [37], we provide the LLM with manually curated examples after the task description. Through these examples, we clarify the layout representation and provide preferences to disperse ambiguity. An example is shown as follows:

*Caption: A watercolor painting of a wooden table in the living room with an apple on it
Objects: [(\'a wooden table\', [65, 243, 344, 206]), (\'an apple\', [206, 306, 81, 69])]
Background prompt: A watercolor painting of a living room
Negative prompt:*

To ensure precise layout control, we adhere to two key principles in our example design:

- 1) Each object instance is represented by a single bounding box. For instance, if the prompt

mentions four apples, we include four boxes with “an apple” in each caption. **2)** We leave no foreground objects specified in the boxes to the background caption to ensure all foreground objects are controlled by our layout-grounded image generator (Section 2.3). These principles allow for accurate and instance-controlled layout generation.

LLM completion. After providing the in-context examples, we query the LLM for completion:

Caption: [input prompt from the user]
Objects: [start of LLM completion]

The resulting layout from the LLM completion is then parsed and used for the subsequent image generation process. We refer readers to Section A.11 for our complete prompt.

Layout-grounded Stable Diffusion

In this stage, we introduce a controller to ground the image generation on the LLM-generated layout. While previous training-free region control methods [14, 15, 16] apply *semantic* guidance through regional denoising or attention manipulation, these methods lack the ability to control the number of objects within a semantic region. This limitation arises as the different instances are often indistinguishable in either the latent space or the attention map, hindering instance-level control.

In contrast, LMD enables *instance*-level grounding by first generating masked latents for each individual bounding box and then composing the masked latents as priors to guide the overall image generation. This allows for precise placement and attribute binding for each object instance.

Per-box masked latents. While diffusion models lack inherent instance-level distinction in their latent space or attention maps for fine-grained control, we observe that they are often able to generate images with one specified instance. Hence, we process one foreground box at a time for instance-level grounding.

As depicted in Figure 2.5(a), for each foreground object i , we first generate an image with a single instance by denoising from $\mathbf{z}_T^{(i)}$ to $\mathbf{z}_0^{(i)}$, where $\mathbf{z}_t^{(i)}$ refers to the latents of object i at denoising timestep t .¹ In this denoising process, we use “[background prompt] with [box caption]” (e.g., “a realistic image of an indoor scene with a gray cat”) as the text prompt for denoising. The initial noise latent is shared for all boxes to ensure globally coherent viewpoint, style, and lighting (i.e., $\mathbf{z}_T^{(i)} = \mathbf{z}_T, \forall i$).

To ensure the object aligns with the bounding box, we manipulate the cross-attention maps $\mathbf{A}^{(i)}$ of the noise-prediction network.² Each map describes the affinity from pixels to

¹We refer readers to Section A.1 for definitions of the terms, such as latents, that are introduced in the latent diffusion framework.

²The cross-attention layer index is omitted for simplicity. We sum the energy values for all selected layers during optimization.

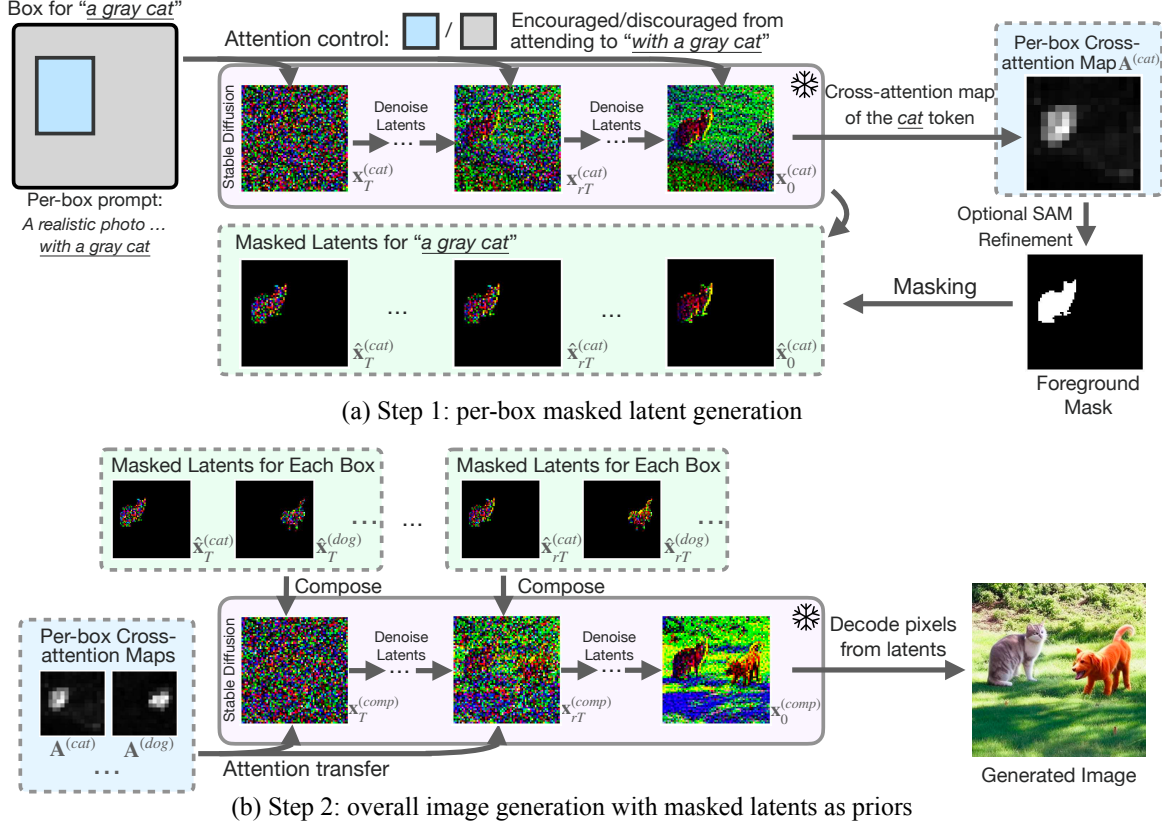


Figure 2.5: **In stage 2, we introduce a novel layout-grounded controller that guides stable diffusion to generate images based on the layout obtained from the previous stage.** Our layout-grounded image generation process consists of two steps: **(a)** generating masked latents for each box specified in the layout, with attention control ensuring that the object is placed in the designated box; and **(b)** composing the masked latents as priors to guide the image generation to adhere to the specified layout.

text tokens:

$$\mathbf{A}_{uv}^{(i)} = \text{Softmax}(\mathbf{q}_u^T \mathbf{k}_v) \quad (2.1)$$

where $\mathbf{q}_u$ and $\mathbf{k}_v$ are linearly transformed image feature at spatial location u and text feature at token index v in the prompt, respectively.

Following *Training-free layout control with cross-attention guidance* [15] and *Boxdiff: Text-to-image synthesis with training-free box-constrained diffusion* [16], we strengthen the cross-attention from pixels inside the box to tokens associated with the box caption while attenuating the cross-attention from pixels outside the box. To achieve this, we define a simple energy function:

$$E(\mathbf{A}^{(i)}, i, v) = -\text{Topk}_u(\mathbf{A}_{uv} \cdot \mathbf{b}^{(i)}) + \omega \text{Topk}_u(\mathbf{A}_{uv} \cdot (1 - \mathbf{b}^{(i)})) \quad (2.2)$$

where $\cdot$ is element-wise multiplication, $\mathbf{b}^{(i)}$ is a rectangular binary mask of the box i with the region in the box set to 1, Topk_u takes the average of top-k values across the spatial dimension u , and $\omega = 4.0$. The energy function is minimized by updating the latent before each denoising step:

$$\mathbf{z}_t^{(i)} \leftarrow \mathbf{z}_t^{(i)} - \eta \nabla_{\mathbf{z}_t^{(i)}} \sum_{v \in V_i} E(\mathbf{A}^{(i)}, i, v) \quad (2.3)$$

$$\mathbf{z}_{t-1}^{(i)} \leftarrow \text{Denoise}(\mathbf{z}_t^{(i)}) \quad (2.4)$$

where η is the guidance strength; the set V_i contains the token indices for the box caption in the prompt for box i (e.g., while generating the masked latents for a box i with caption “*a gray cat*”, V_i indicates the indices of tokens that correspond to the box caption in the per-box denoising text prompt “*[background prompt] with a gray cat*”). $\text{Denoise}(\cdot)$ denotes one denoising step in the latent diffusion framework.

After generation, we obtain the cross-attention map that corresponds to the box caption, which serves as a saliency mask for the object. We optionally use SAM [39] to refine the quality of the mask. This can be done by querying either with the pixel location that has the highest saliency or with the layout box. The functionality of SAM can also be replaced by a simple thresholding, as experimented in Section 2.4. With the refined mask for exactly one foreground instance, denoted as $\mathbf{m}^{(i)}$, we perform element-wise multiplication between the mask and the latent at each denoising step to create a sequence of *masked* instance latents $(\hat{\mathbf{z}}_t^{(i)})_{t=0}^T$:

$$\hat{\mathbf{z}}_t^{(i)} = \mathbf{z}_t^{(i)} \otimes \mathbf{m}^{(i)} \quad (2.5)$$

Masked latents as priors for instance-level control. The masked instance latents $(\hat{\mathbf{z}}_t^{(i)})_{t=0}^T$ are then leveraged to provide instance-level hints to the diffusion model for the overall image generation. As illustrated in Figure 2.5(b), during each denoising time step in the early denoising process, we place each masked foreground latents $\hat{\mathbf{z}}_t^{(i)}$ onto the composed latents $\mathbf{z}_t^{(\text{comp})}$:

$$\mathbf{z}_t^{(\text{comp})} \leftarrow \text{LatentCompose}(\mathbf{z}_t^{(\text{comp})}, \hat{\mathbf{z}}_t^{(i)}, \mathbf{m}^{(i)}) \quad \forall i \quad (2.6)$$

where $\mathbf{z}_T^{(\text{comp})}$ is initialized from $\mathbf{z}_T$ for foreground generation for consistency. The operator LatentCompose simply puts the masked foreground latents $\hat{\mathbf{z}}_t^{(i)}$ onto the corresponding location on $\mathbf{z}_t^{(\text{comp})}$.

Since diffusion models tend to generate the object placement in the initial denoising steps and then object details in later steps [14], we only compose the latents from timestep T to rT^3 , where $r \in [0, 1]$ balances instance control and image coherency. By primarily intervening during the steps for object placement, our method merely provides instance-level layout hints

³In the notation of this work, the denoising process starts from time step T to step 0.

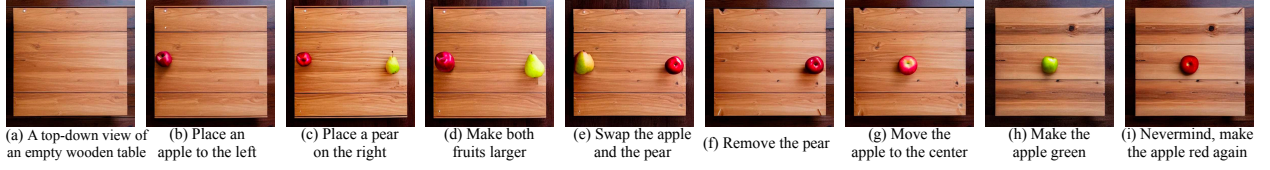


Figure 2.6: **LMD and LMD+ support instruction-based scene specification, empowering the users to add/move/remove objects, modify object attributes, and clarify the prompt in multiple rounds of dialog.** (a): the initial prompt for the scene; (b)-(i): eight subsequent instructions that *sequentially* modify the scene. By separating the generation of each foreground object as well as the background, LMD ensures consistent image generation when the same seed is used for image generation throughout the dialog.

rather than forcing each masked region of the resulting generation to look the same as the per-box generation.

To make our guidance more robust, we further transfer the cross-attention maps from per-box generation to the corresponding regions in the composed generation by adapting the energy function:

$$E^{(\text{comp})}(\mathbf{A}^{(\text{comp})}, \mathbf{A}^{(i)}, i, v) = E(\mathbf{A}^{(\text{comp})}, i, v) + \lambda \sum_{u \in V'_i} \mathbf{A}_{uv}^{(\text{comp})} - \mathbf{A}_{uv}^{(i)} \quad (2.7)$$

where $\lambda = 2.0$ and the energy value of each box i is summed up for optimization. V'_i denotes the indices of tokens that correspond to the box caption in the text prompt for the overall denoising process, similar to the definition of V_i in Equation (2.3).

In this way, our controller conditions the diffusion model to generate one instance at each masked location, with the final generation natural and coherent in terms of foreground-background composition.

Finally, we decode latents $\mathbf{z}_0^{(\text{comp})}$ to pixels $\mathbf{x}_0$ via the diffusion image decoder.

Integration with training-based methods. Our training-free controller can also be applied along with training-based methods such as GLIGEN [5] to leverage instance-annotated external datasets when available. Since GLIGEN trains adapter layers taking box inputs, the integration with GLIGEN, denoted as LMD+, involves adopting its adapter weights and passing the layout guidance to the adapter layers. Note that LMD+ uses adapters along with the instance-level guidance introduced above, which greatly surpasses only using GLIGEN adapters, as shown in Table 2.2. We achieve further enhanced instance and attribute control without additional training through this integration.

Additional Capabilities of LMD

Our LLM-grounded generation pipeline allows for two additional capabilities without additional training.

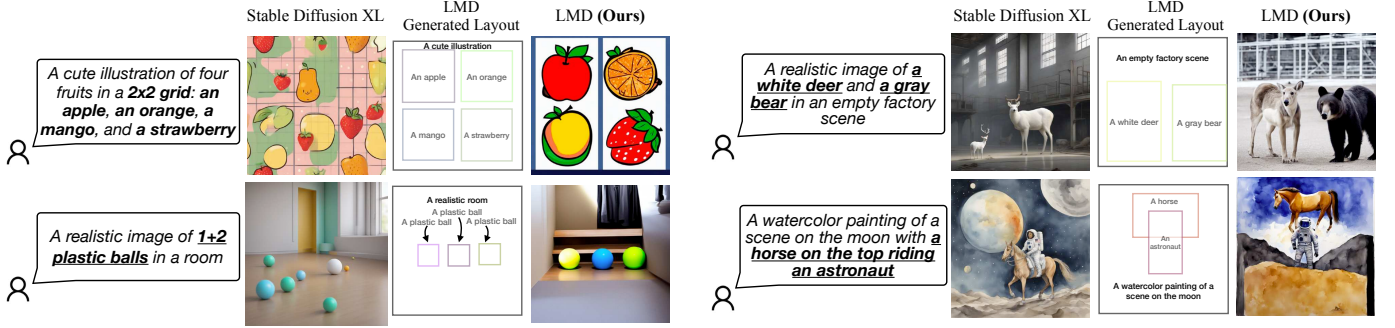


Figure 2.7: LMD outperforms its base text-to-image diffusion model [1] in accurately following the prompts that require spatial and language reasoning. Best viewed when zoomed in.

Instruction-based scene specification. Leveraging an LLM that supports multi-round dialog (e.g., GPT-3.5/4), LMD empowers the users to specify the desired image with multiple instructions following an initial prompt (Figure 2.3). Specifically, after the initial image generation, a user can simply give clarifications or additional requests to the LLM. With the updated layout from the LLM, we can leverage LMD again to generate images with the updated layout. Updating the layout rather than the raw image gives LMD several advantages, as demonstrated in Figure 2.6: **1)** Our generation remains consistent after multiple rounds of requests instead of gradually drifting away from the initial image. **2)** LMD can handle requests that involve spatial reasoning, which are the limitations of previous instruction-based image editing method *Instructpix2pix: Learning to follow image editing instructions* [37]. In contrast, we demonstrate that VisualChatGPT [2], which equips ChatGPT with tools such as *Instructpix2pix: Learning to follow image editing instructions* [37], is not able to follow the instructions in Figure 2.6, especially for spatial instructions over multiple iterations of dialog. This capability applies to both LMD and LMD+. Our LMD can handle requests for open-ended scene adjustments, offer suggestions for the current scene, understand user requests within the dialog context, and allow the users to try out different detailed adjustments while preserving the overall image style and layout, facilitating fine-grained content creation.

Supporting more languages. By giving an in-content example of a non-English user prompt and an English layout output⁴, the LLM layout generator accepts non-English user prompts and outputs layouts with English captions. This allows generation from prompts in languages not supported by the underlying diffusion model *without additional training* (Figure A.6). We refer readers to Section A.9 for additional details.

⁴We simply translate the example prompt input in our last example to the desired language, while keeping its layout in English.

2.4 Evaluation

Qualitative Comparison

Setup. We qualitatively compare our approach with Stable Diffusion (SD, *High-resolution image synthesis with latent diffusion models* [13] and *Sdxl: improving latent diffusion models for high-resolution image synthesis* [1]). SD family is also chosen as our underlying base model for layout-grounded image generation given its strong capabilities and widespread adoption in text-to-image generation research. Thanks to the training-free nature of our work, our method is applicable to various diffusion models without additional training. Therefore, for Figure 2.1, 2.7, and 2.9, we use the largest Stable Diffusion model SDXL as the base model of LMD and compare against SDXL as a baseline. For all other settings, we use Stable Diffusion v1.5 as the base model unless stated otherwise. We use `gpt-4` [40] for layout generation for all qualitative comparisons. **Results.** In Figure 2.1 and 2.7, we observe that our two-stage text-to-image generation approach greatly enhances prompt following ability compared to our base model by generating images that align with the layouts from the LLM.

Comparing with other LLM-based image generators. VisualChatGPT [2] and GILL [3] also leverage LLMs as a part of the image generation pipelines. Both works leverage SD as the underlying image generation model. VisualChatGPT treats SD as a module that can be used by the LLM and passes text caption to it, and GILL outputs a embedding in place of the text embedding for SD. Since both methods utilize LLMs to only provide conditions to SD in the form of text embeddings, these methods still inherit the problems of insufficient control of text embeddings from the base SD model. In contrast, our method asks the LLM to explicitly reason about the spatial relationships and applies direct spatial control on our underlying diffusion model, thereby bypassing the bottleneck of the text embedding representation that does not accurately convey spatial information. As shown in Figure 2.8, neither method accurately follows text prompts of several categories that our method is able to correctly generate. Furthermore, although the involvement of LLM in VisualChatGPT and GILL also potentially allows multi-round instruction-based scene specification (Section 2.3), we empirically observe that the generated images quickly deviate from the scene of “a wooden table” starting from the second iteration, with the final generation being incomprehensible.

Quantitative evaluation

Proposed benchmark

We propose a text-to-image evaluation benchmark that includes four tasks: negation, generative numeracy, attribute binding, and spatial reasoning. Negation and generative numeracy involve generating a specific number of objects or not generating specific objects. Attribute binding involves assigning the right attribute to the right object with multiple objects in the prompt. Spatial reasoning involves understanding words that describe the relative locations of objects. For each task, we programmatically compose 100 prompts and query each model for text-to-image generation, with 400 prompts in total. `gpt-3.5-turbo` [41]

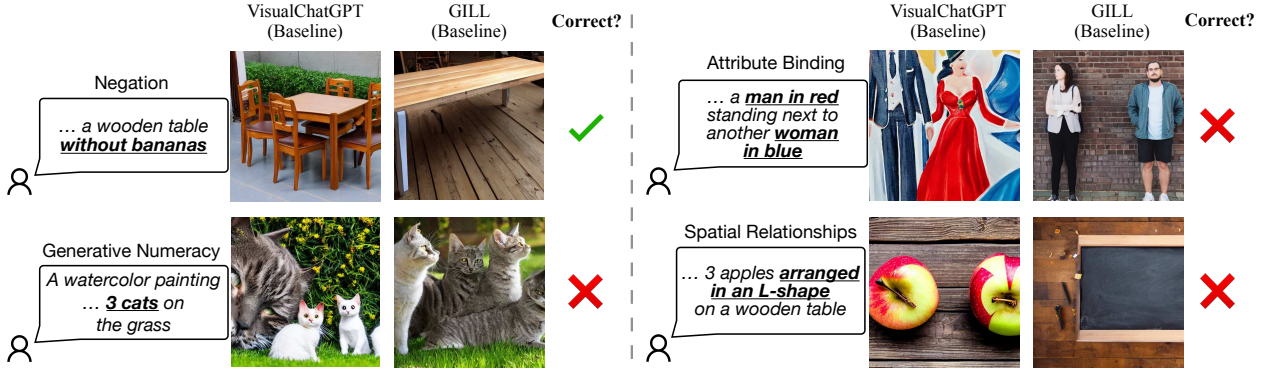


Figure 2.8: We qualitatively compare with VisualChatGPT [2] and GILL [3] that also leverage LLMs in the image generation pipelines. **Both baselines lack the ability to accurately follow the prompts for three out of four tasks that our method can solve in Figure 2.1.**

Tasks	SD	Accuracy	
		LMD	LMD+
Negation	28%	100% (3.6×)	100% (3.6×)
Generative Numeracy	39%	62% (1.6×)	86% (2.2×)
Attribute Binding	52%	65% (1.3×)	69% (1.3×)
Spatial Relationships	28%	79% (2.8×)	67% (2.4×)
Average	37%	77% (2.1×)	81% (2.2×)

Table 2.1: With guidance from the LLM-based layout generator and our novel layout-grounded controller, **our LMD significantly outperforms the Stable Diffusion model (SD) that we use under the hood in four tasks benchmarking prompt-following abilities.** LMD denotes our method directly applied on SD. LMD+ denotes additionally integrating pretrained GLIGEN [5] adapters into our controller.

is used in LMD for the benchmarks. We also implemented LMD+, a LMD variant that integrate pretrained GLIGEN [5] adapters into our controller without further training.

Detection-based evaluation. We use an open-vocabulary object detector, OWL-ViT [42], to obtain bounding boxes for the objects of interest. We then check whether each generated image satisfies the requirements in the prompt. The accuracy of each task is computed by calculating the proportion of the image generations that match their corresponding prompts over all generations.

Results. As presented in Table 2.1, our model shows significant improvements in generation accuracy, ranging from 1.3× to 3.6× compared to SD across four tasks and *doubling* the

Stage 1/Stage 2	Accuracy				
	Negation	Numeracy	Attribute	Spatial	Average
<i>Training-free methods:</i>					
LMD/MultiDiffusion [14]	100%	30%	42%	36%	52.0%
LMD/Backward Guidance [15]	100%	42%	36%	61%	59.8%
LMD/BoxDiff [16]	100%	32%	55%	62%	62.3%
LMD/LMD (Ours)	100%	62%	65%	79%	76.5% (+ 14.2)
<i>Training-based methods:</i>					
LMD/GLIGEN [5]	100%	57%	57%	45%	64.8%
LMD/LMD+ (Ours)	100%	86%	69%	67%	80.5% (+ 15.7)
LMD/LMD+ (Ours, GPT-4)		84%	79%	82%	86.3% (+ 21.5)
<i>Evaluating generated layouts only (upper bound for image generation):</i>					
LMD/-	100%	97%	100%	99%	99.0%

Table 2.2: **Ablations on layout-to-image methods as stage 2 with our LLM layout generator as stage 1. Our proposed layout-grounded controller performs the best among them.** Our controller could also be applied on top of training-based GLIGEN [5], denoted as LMD+, for additional improvements. Finally, the LLM-generated layouts almost always align with the prompt, highlighting that the bottleneck is the layout-grounded image generation. The scores for negation task are high because we pass the negative prompts generated by the LLM to the underlying diffusion model, which does not depend on the stage 2 implementation.

accuracy on average. Notably, LMD achieves image generation accuracy that is *more than twice* of the SD accuracy for the spatial relationships and the negation task. This highlights the utility of the grounding image generation on the LLM layout generator. Furthermore, when additionally integrating GLIGEN to our pipeline to leverage in-domain instance-annotated data, our method, denoted as LMD+, achieves additional improvements.

Ablation Study

Layout-to-image stage. Comparing with other layout-to-image methods. As shown in Table 2.2, compared with training-free layout-to-image generation methods that perform *semantic-level* grounding, our proposed layout-grounded controller provides much better *instance-level* grounding. This is justified by the fact that our training-free controller even *surpasses training-based method* GLIGEN [5] in the generative numeracy task, despite not trained with any instance-level annotation. Furthermore, our controller also significantly surpasses training-based method GLIGEN [5] in attribute binding and spatial reasoning task. When integrated with GLIGEN to leverage instance-annotated datasets, our integration,

denoted as LMD+, allows for further improvements without the need for additional training. ***Switching the base diffusion model without hyperparameter tuning.*** As shown in Table 2.3, thanks to our training-free nature, LMD maintains the gains to the base model (around $2\times$ performance boost) when we switch the base diffusion model from SDv1.5 to SDv2.1 *without tuning any hyperparameters*, including λ and ω that are introduced by our method.⁵ This showcases the potential of integrating LMD with future diffusion models. ***Using SAM vs a simple attention threshold to obtain the per-box mask.*** Instead of using SAM to obtain the mask for each box, we also explored an approach that does not require an additional segmentation module. Alternatively, we sort the pixels in each box according to their attention value with respect to the box caption and pick the top 75% pixels in each box with the highest attention as the mask for the box. As shown in Table 2.4, the impact of SAM is different for LMD/LMD+. In LMD, since the attention-based guidance is less spatially accurate with respect to the layout boxes, SAM helps to obtain the right mask that covers the object. Therefore, removing SAM leads to a slight degradation in LMD. In LMD+, since the guidance is more spatially accurate, SAM is no longer necessary most of the time. Instead, SAM sometimes picks a region that includes the background, causing confusion and reduced performance. Therefore, removing SAM slightly improves the results in LMD+. We make SAM an optional choice (as described in Figure 2.2) but still recommend it for LMD and enable it by default.

Text-to-layout stage. Ablating in-context examples. In addition to using the seven fixed in-context examples by default, we also vary the number of in-context examples given to the LLM (i.e., “shots”). We show in Table 2.5 that while GPT-3.5 benefits from more in-context examples, GPT-4 is able to successfully generate all the layouts even when given only one in-context example. Note that we also observe GPT-4 to still be able to generate layouts *without any in-context examples* (i.e., given only the text instructions). However, since no examples are offered as references in this zero-shot setting, the format of LLM outputs are observed to differ in different runs, making it hard to parse with a program. Since it is much easier to convey the format through an example than through language instructions, we recommend having at least one example. Our observation shows that LLMs *already learn the ability to generate object boxes during pretraining* and do not need us to convey through many in-context examples. ***Varying the model types and the sizes of the LLMs.*** We also ablate the LLMs used for text-to-layout generation, including using self-hosted LLMs with public weights [10, 12, 43, 11]. The results show that the capability to generate high-quality layouts are *not limited to proprietary LLMs*, and larger LLMs offer much better layout generation capabilities.

⁵The hyperparameters inherited from the latent diffusion framework, such as the number of denoising steps, are also unchanged.

Method	Image Accuracy
	Average of 4 tasks
SD v1.5 (Default)	37%
LMD (on SDv1.5) (Ours , default)	77% (2.1 $\times$)
SD v2.1	38%
LMD (on SDv2.1) (Ours)	77% (2.0 $\times$)

Table 2.3: **LMD achieves comparable gains when adapted to Stable Diffusion v2.1 without any hyperparameter tuning or model training.** This shows a promising signal that the gains from our method could carry along with the enhancement of diffusion models. The performance of our method could potentially be improved further with additional hyperparameter tuning.

Method	Image Accuracy
	Average of 4 tasks
LMD (w/o SAM)	72.8%
LMD (with SAM)	76.5%
LMD+ (w/o SAM)	82.8%
LMD+ (with SAM)	80.5%

Table 2.4: **Ablations on using SAM vs using simple attention thresholding in stage 2.** While removing SAM leads to a slight degradation in LMD, removing SAM leads to even better performance in LMD+.

T2I-CompBench

In addition to our proposed benchmark with detection-based evaluation, we evaluate our method on T2I-CompBench [6] that additionally uses visual question answering (VQA) models for generation evaluation. The color, shape, and texture tasks employ BLIP [44] in a VQA setting, while the spatial task uses UniDet [45] for evaluation. As shown in Table 2.6, our method LMD, when applied on either SDv1 or SDv2, improves the performance on all four tasks.

Evaluator-based Assessment

Setting. We also assess the prompt following ability of our method and vanilla SD, the base diffusion model that our method uses under the hood. We randomly selected 10 text prompts from our proposed benchmark and generated a pair of images per text prompt, one with our

# shots	Layout Accuracy (4 tasks)	
	gpt-3.5-turbo	gpt-4
1 Shot	89.8%	100.0%
4 Shots	96.3%	100.0%
7 Shots	99.0%	100.0%

Table 2.5: **Ablations on the number of in-context examples (“shots”) given to the LLM.** While GPT-3.5 benefits from more in-context examples, GPT-4 already excels in layout generation even with only one example.

	Color	Shape	Texture	Spatial
SDv1	0.3765	0.3576	0.4156	0.1246
LMD (on SDv1)	0.5495	0.5462	0.5241	0.2570
SDv2	0.5065	0.4221	0.4922	0.1342
LMD (on SDv2)	0.5736	0.5334	0.5227	0.2704

Table 2.6: **Our method surpasses the base diffusion models SDv1 and SDv2 on T2I-CompBench [6] on all four tasks without additional training.**

LMD+ and one with the base model SD.⁶ We then invited 11 evaluators to compare each image pair and answer two questions:

1. Question 1: Which image aligns better with the text prompt?
2. Question 2: Which image has a more natural and coherent foreground-background composition?

In addition to an option for preferring each image, a “similar” option is also provided for each pair.

Results. We average the scores across 110 responses. The results show that our method LMD+ got **88.18%** (vs 10.90% for SD) for the first question and **35.45%** (vs 31.81% for SD) for the second question. This indicates that our method generates images that accurately align with the prompt compared to the baseline SD without degradation of naturalness or coherency.

⁶The base model of LMD+ is technically GLIGEN, which itself is built on SD. However, in our setting where only text is used as input, the conditioning scheme of GLIGEN simply becomes degenerate. Therefore, we compare with SD directly instead.

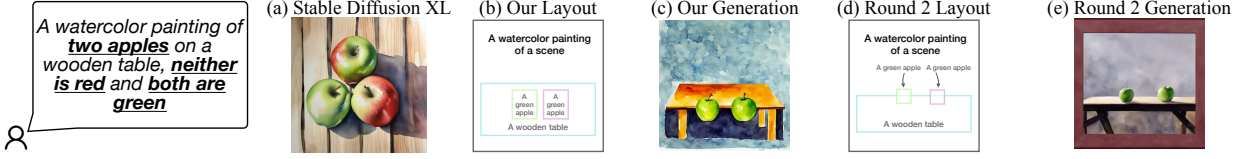


Figure 2.9: **A failure case** occurs when our method, shown in (c), generates objects in unintentional viewpoints and sizes due to the ambiguity in the generated layout. The LLM-generated layout (b) is suitable for close-up top-down view of a small table, but the layout-to-image model assumes a side view and thus fails to generate a feasible image. Nevertheless, our method still provides more interpretability through the intermediate layout (b) compared to baseline SDXL (a). With an additional request for the side view and correct object sizes, the LLM adjusted the layout in (d) and the final generation (e) is aligned with the text prompt.

2.5 Discussions

Since we use models off-the-shelf, the LLM may generate layouts that are ambiguous to the diffusion model. For example, the layout in Figure 2.9(b) is feasible for a top-down close-up image, but the diffusion model generates an image viewing from the side. This makes the apples not on the table in Figure 2.9(c). Prompting or fine-tuning the LLM to be more explicit about its assumptions in the layouts (e.g., viewpoints) may alleviate this problem. The intermediate layout in our two-stage generation allows for more interpretability compared to our base model stable diffusion. After diagnosing the point of failure, we give an additional request for the side view and correct object sizes to the LLM. The LLM adjusted the subsequent layout generation, which allows generating images that align with the input prompt in round 2, as shown in Figure 2.9(d,e). Our method also inherits biases from the base diffusion model [46]. Moreover, although our method can handle objects not mentioned in the in-context examples (e.g., the bear and the deer in Figure 2.7), the LLM may still generate better layouts for objects mentioned in the in-context examples by referencing layout examples. Our method could also be distilled into a one-stage text-to-image diffusion model to improve its prompt understanding abilities without leveraging LLMs at inference time for the ease of deployment.

2.6 Summary

In this paper, we enhance the prompt understanding capabilities of text-to-image diffusion models. We present a novel training-free two-stage generation process that incorporates LLM-based text-grounded layout generation and layout-grounded image generation. Our method also enables instruction-based scene specification and generation from prompts in languages unsupported by the base diffusion model. Our method outperforms strong baselines

in accurately following the prompts in text-to-image generation.

Acknowledgements. The authors would like to thank Aleksander Holynski for the helpful discussions.

Chapter 3

LLM-grounded Video Diffusion Models

This chapter is based on the following publication:

Long Lian, Baifeng Shi, Adam Yala, Trevor Darrell, and Boyi Li. “LLM-grounded Video Diffusion Models.” *International Conference on Learning Representations (ICLR)*, 2024.

3.1 Introduction

Text-to-image generation has made significant progress in recent years [18, 17]. In particular, diffusion models [47, 48, 49, 50, 51, 13] have demonstrated their impressive



Figure 3.1: *Left*: Existing text-to-video diffusion models such as [7] often encounter challenges in generating high-quality videos that align with complex prompts. *Right*: Our training-free method LVD, when *applied on the same model*, allows the generation of realistic videos that closely align with the input text prompt.

ability to generate high-quality visual contents. Text-to-video generation, however, is more challenging, due to the complexities associated with intricate spatial-temporal dynamics. Recent works [52, 53, 54, 7] have proposed text-to-video models that specifically aim to capture spatiotemporal dynamics in the input text prompts. However, these methods still struggle to produce realistic spatial layouts or temporal dynamics that align well with the provided prompts, as illustrated in Figure 3.1.

Despite the enormous challenge for a diffusion model to generate complex dynamics directly from text prompts, one possible workaround is to first generate explicit spatiotemporal layouts from the prompts and then use the layouts to control the diffusion model. In fact, recent work [55, 23, 56] on text-to-image generation proposes to use Large Language Models (LLMs) [57, 58, 40] to generate spatial arrangement and use it to condition text-to-image models. These studies demonstrate that LLMs have the surprising capability of generating detailed and accurate coordinates of spatial bounding boxes of each object based on the text prompt, and the bounding boxes can then be utilized to control diffusion models, enhancing the generation of images with coherent spatial relationships. However, it has not been demonstrated if LLMs can generate *dynamic* scene layouts for videos on both spatial and temporal dimensions. The generation of such layouts is a much harder problem, since an object’s motion patterns often depend on both the physical properties (e.g., gravity) and the object’s attributes (e.g., elasticity vs rigidity).

In this paper, we investigate whether LLMs can generate spatiotemporal bounding boxes that are coherent with a given text prompt. Once generated, these box sequences, termed Dynamic Scene Layouts (DSLs), can serve as an intermediate representation bridging the gap between the text prompt and the video. With a simple yet effective attention-guidance algorithm, these LLM-generated DSLs are leveraged to control the generation of object-level spatial relations and temporal dynamics in a training-free manner. The whole method, referred to as LLM-grounded Video Diffusion (LVD), is illustrated in Figure 3.2. As shown in Figure 3.1, LVD generates videos with the specified temporal dynamics, object attributes, and spatial relationships, thereby substantially enhancing the alignment between the input prompt and the generated content.

To evaluate LVD’s ability to generate spatial layouts and temporal dynamics that align with the prompts, we propose a benchmark with five tasks, each requiring the understanding and generation of different spatial and temporal properties in the prompts. We show that LVD significantly improves the text-video alignment compared to several strong baseline models. We also evaluate LVD on common datasets such as UCF-101 [59] and MSR-VTT [60] and conducted an evaluator-based assessment, where LVD shows consistent improvements over the base diffusion model that it uses under the hood.

Contributions. **1)** We show that text-only LLMs are able to generate dynamic scene layouts that generalize to previously unseen spatiotemporal sequences. **2)** We propose LLM-grounded Video Diffusion (LVD), the first training-free pipeline that leverages LLM-generated dynamic scene layouts for enhanced ability to generate videos from intricate text prompts. **3)** We introduce a benchmark for evaluating the alignment between input prompts and the videos generated by text-to-video models.

3.2 Related Work

Controllable diffusion models. Diffusion models have made a huge success in content creation [17, 61, 49, 62, 63, 50, 64, 65, 66]. ControlNet [30] proposes an architectural design for incorporating spatial conditioning controls into large, pre-trained text-to-image diffusion models using neural networks. GLIGEN [5] introduces gated attention adapters to take in additional grounding information for image generation. Shape-guided Diffusion [67] adapts pretrained diffusion models to respond to shape input provided by a user or inferred automatically from the text. Control-A-Video [68] train models to generate videos conditioned on a sequence of control signals, such as edge or depth maps. Despite the impressive progress made by these works, the challenge of generating videos with complex dynamics based solely on text prompts remains unresolved.

Text-to-video generation. Although there is a rich literature on video generation [69, 70, 71, 72, 73, 74, 75], text-to-video generation remains challenging since it requires the model to synthesize the video dynamics based only on text. Make-A-Video [52] breaks down the entire temporal U-Net [76] and attention tensors, approximating them in both spatial and temporal domains, and establishes a pipeline for producing high-resolution videos. Imagen Video [77] creates high-definition videos through a foundational video generation model and spatial-temporal video super-resolution models. Video LDM [53] transforms the image generator into a video generator by adding a temporal dimension to the latent space diffusion model. Text2Video-Zero [54] introduces two post-processing techniques for ensuring temporal consistency: encoding motion dynamics in latent codes and reprogramming frame-level self-attention with cross-frame attention mechanisms. However, these models still easily fail in generating reasonable video dynamics due to the lack of large-scale paired text-video training data that can cover diverse motion patterns and object attributes, let alone the demanding computational cost to train with such large-scale datasets.

Grounding and reasoning from large language models. Several recent text-to-image models propose to feed the input text prompt into an LLM to obtain reasonable spatial bounding boxes and generate high-quality images conditioned on the boxes. LMD [55] proposes a training-free approach to guide a diffusion model using an innovative controller to produce images based on LLM-generated layouts. LayoutGPT [23] proposes a program-guided method to adopt LLMs for layout-oriented visual planning across various domains. Attention refocusing [56] introduces two innovative loss functions to realign attention maps in accordance with a specified layout. Collectively, these methods provide empirical evidence that the bounding boxes generated by the LLMs are both accurate and practical for controlling text-to-image generation. In light of these findings, our goal is to enhance the ability of text-to-video models to generate from prompts that entail complex dynamics. We aim to do this by exploring and harnessing the potential of LLMs in generating spatial and temporal video dynamics. A concurrent work VideoDirectorGPT [78] proposes using an LLM in multi-scene video generation, which pursues a different focus and technical route. It explores training diffusion adapters for additional conditioning, while LVD focuses on inference-time guidance and in-depth analysis on the generalization of the LLMs.

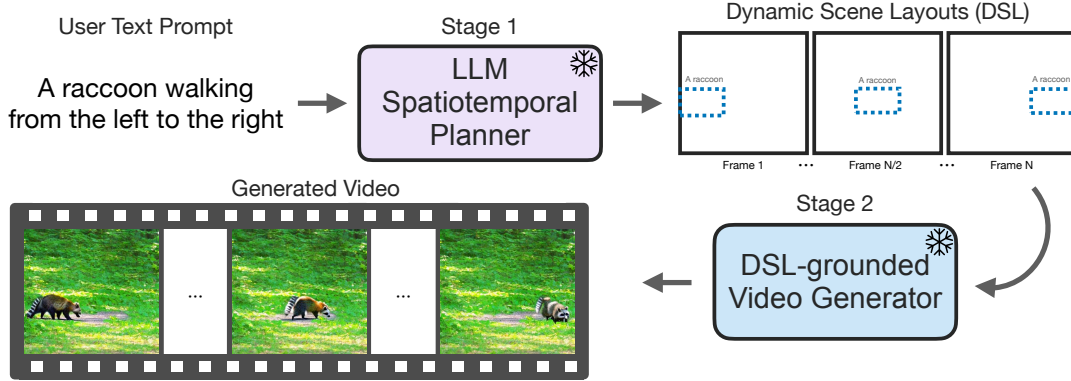


Figure 3.2: Our method LVD improves text-to-video diffusion models by turning the text-to-video generation into a two-stage pipeline. In stage 1, we introduce an LLM as the spatiotemporal planner that creates plans for video generation in the form of a dynamic scene layout (DSL). A DSL includes objects bounding boxes that are linked across the frames. In stage 2, we condition the video generation on the text and the DSL with a DSL-grounded video generator. Both stages are training-free: *LLMs and diffusion models are used off-the-shelf without updating their parameters*. By using DSL as an intermediate representation for text-to-video generation, LVD generates videos that align much better with the input prompts compared to its vanilla text-to-video model counterpart.

3.3 Can LLMs Generate Spatiotemporal Dynamics?

In this section, we explore the extent to which LLMs are able to produce spatiotemporal dynamics that correspond to a specified text prompt. **We aim to resolve three questions in this investigation:**

1. Can LLMs generate realistic dynamic scene layouts (DSLs) aligned with text prompts and discern when to apply specific physical properties?
2. Does the LLM’s knowledge of these properties come from its weights, or does the understanding of these concepts develop during inference on the fly?
3. Can LLMs generalize to broader world concepts and relevant properties based on the given examples that only entail limited key properties?

To prompt the LLM for dynamics generation, we query the LLM with a prompt that consists of two parts: task instructions in text and a few examples to illustrate the desired output and rules to follow. Following the prompt, we query the LLM to perform completion, hereby generating the dynamics.

Task instructions. We ask the LLM to act as a “video bounding box generator”. In the task setup, we outline task requirements like the coordinate system, canvas size, number of frames, and frame speed.

DSL representation. We ask the LLM to represent the dynamics for each video using a Dynamic Scene Layout (DSL), which is a set of bounding boxes linked across the frames. LLMs are prompted to sequentially generate the boxes visible in each frame. Each box has a representation that includes the location and size of the box in numerical coordinates, as well as a phrase describing the enclosed object. Each box is also assigned a numerical identifier starting from 0, and boxes across frames are matched using their assigned IDs without employing a box tracker.

In-context examples. LLMs might not understand our need for real-world dynamics such as gravity in the DSL generation, which can lead to assumptions that may not align with our specific goals. To guide them correctly, we provide examples that highlight the desired physical properties for generating natural spatiotemporal layouts. A key question arises: How many examples do LLMs need in order to generate realistic layouts, given they are not designed for real-world dynamics? One might assume they need numerous prompts to grasp the complex math behind box coordinates and real-world dynamics. Contrary to such intuition, we suggest presenting the LLM with a few examples demonstrating key physical properties and using *only one example for each desired property*. Surprisingly, this often suffices for LLMs to comprehend various aspects including physical properties and camera motions, empowering downstream applications such as video generation (Section 3.4). We also find that LLMs can extrapolate from provided examples to infer related properties that are not explicitly mentioned. This means we do not need to list every desired property but just highlight a few main ones. Our in-context examples are visualized in Figure 3.3.

Reason before generation. To improve the interpretability of the LLM output, we ask the LLM to output a brief statement of its reasoning before starting the box generation for each frame. We also include a reasoning statement in each in-context example to match the output format.

Investigation setup. With the potential downstream of video generation, we categorize the desired physical properties into world properties and camera properties. Specifically, we examine gravity and object elasticity as world properties and perspective projection as a camera property. However, the surprising generalization capabilities indicate promising opportunities for LLMs to generalize to other *uninvestigated properties*. As displayed in Figure 3.3, we provide the LLM with one example for each of the three properties. We then query the LLM with a few text prompts for both world properties and camera properties (see Figure 3.4). We use GPT-4 [40] for investigation in this section. Additional models are explored in Section 3.5, confirming they exhibit similar capabilities.

Discoveries. We discovered that for various query scenarios, either seen or unseen, LLMs can create realistic DSLs aligned with the prompt. Notably, this ability is rooted in the LLM’s inherent knowledge, rather than solely relying on provided examples. For instance, given the ball example that shows the presence of elasticity in our desired world for LLM to simulate (Figure 3.3(b)), LLM understands that the ball, when thrown, will hit the ground and bounce

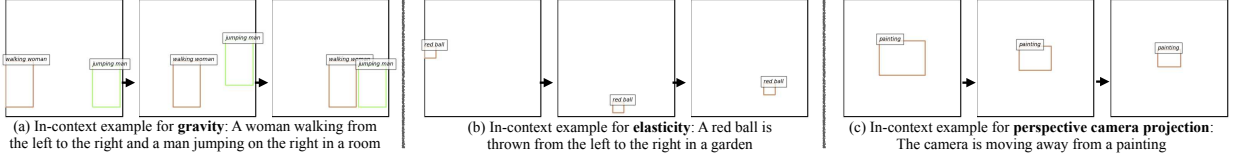


Figure 3.3: **In-context examples.** We propose to prompt the LLMs with *only one in-context example per key desirable property*. Example (a)/(b)/(c) demonstrates gravity/elasticity/perspective projection, respectively. In Section 3.3, we show LLMs are able to generate DSLs aligned with the query prompts with only these in-context examples, empowering downstream applications such as video generation.

back (Figure 3.4(a)), no matter whether it was thrown from the left (Figure 3.3(b)) or right. Furthermore, LLM is able to generalize to objects with different elasticity. For example, when replacing the ball with a rock without updating in-context examples (Figure 3.4(b)), LLM recognizes that rocks are not elastic, hence they do not bounce, but they still obey gravity and fall. This understanding is reached *without explicit textual cues about the object’s nature in either our instructions or our examples*. Since this ability is not derived from the input prompt, it must originate from the LLM’s weights. To illustrate our point further, we show the LLM that the “imagined camera” follows the principle of perspective geometry in Figure 3.3(c): as the camera moves away from a painting, the painting appears smaller. We then test the LLM with a scenario not directly taught by our examples, as shown in Figure 3.4(e,f). Even without explicit explanations about how perspective geometry varies with camera viewpoint and object motion, the LLM can grasp the concept. If the car’s forward movement is perpendicular to the direction that the camera is pointing at, perspective geometry does not apply.

More compellingly, by setting the right context of the imagined world, *LLMs can make logical extensions to properties not specified in the instructions or examples*. As shown in Figure 3.4(d), the LLM recognizes that the principles of perspective geometry applies not just when a camera moves away from a static object (introduced in Figure 3.3(c)), but also when objects move away from a stationary camera. This understanding extends beyond our given examples. As seen in Figure 3.4(c), even without explicit mention of air friction, the LLM takes it into account for a paper airplane and generates a DSL in which the paper airplane is shown to fall slower and glide farther than other queried objects. This indicates that our examples help establish an imagined world for the LLM to simulate, removing the need for exhaustive prompts for every possible scenario. LLMs also demonstrate understanding of buoyancy and animals’ moving patterns to generate realistic DSLs for Figure 3.1 in the reasoning statements.

These observations indicate that LLMs are able to successfully generate DSLs aligned with complex prompts given only the three in-context examples in Figure 3.3. We therefore use these in-context examples and present further quantitative and qualitative analysis in

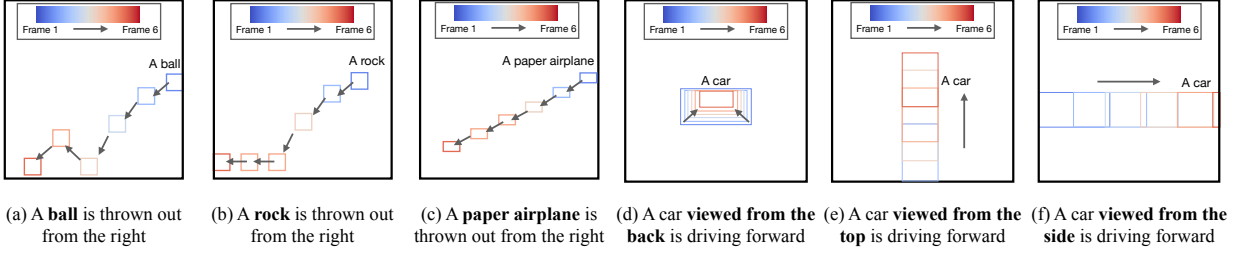


Figure 3.4: **DSLs generated by the LLM from input text prompts** with only in-context examples in Figure 3.3. Remarkably, despite *not having any textual explanations on the applicability of each physical property*, the LLM discerns and applies them appropriately. **(a,b)** The LLM selectively applies the elasticity rule to the ball but not the rock, even though the attributes of a rock are never mentioned in the example. **(c)** The LLM infers world context from the examples and factors in air friction, which is not mentioned either, when plotting the paper airplane’s trajectory. **(d-f)** reveal the LLM’s inherent grasp of the role of viewpoint in influencing object dynamics, without explicit instructions in the examples. These discoveries suggest that *the LLM’s innate knowledge, embedded in its weights, drives this adaptability, rather than solely relying on the provided examples.*

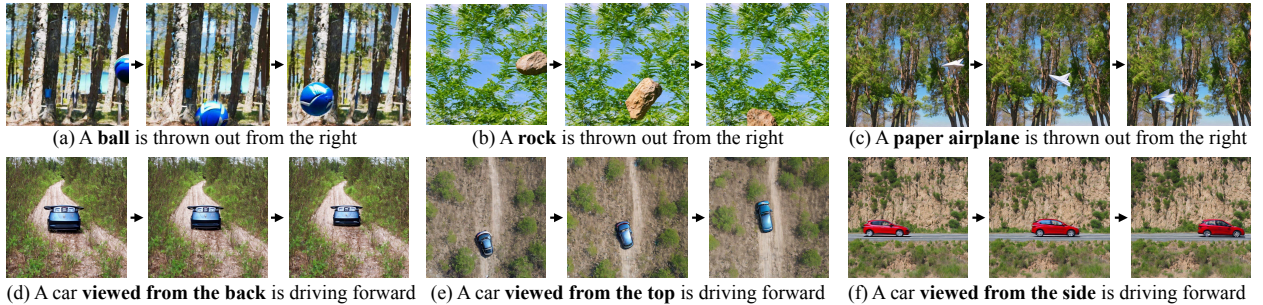


Figure 3.5: **Videos generated by our method LVD** from the DSLs in Figure 3.4. Our approach generates videos that correctly align with input text prompts. **(a-c)** show various objects thrown from the right to the left. **(d-f)** depict objects of the same type viewed from different viewpoints.

Section 3.5.

3.4 DSL-grounded Video Generation

Leveraging the capability of LLMs to generate DSLs from text prompts, we introduce an algorithm for directing video synthesis based on these DSLs. Our DSL-grounded video generator directs an off-the-shelf text-to-video diffusion model to generate videos consistent

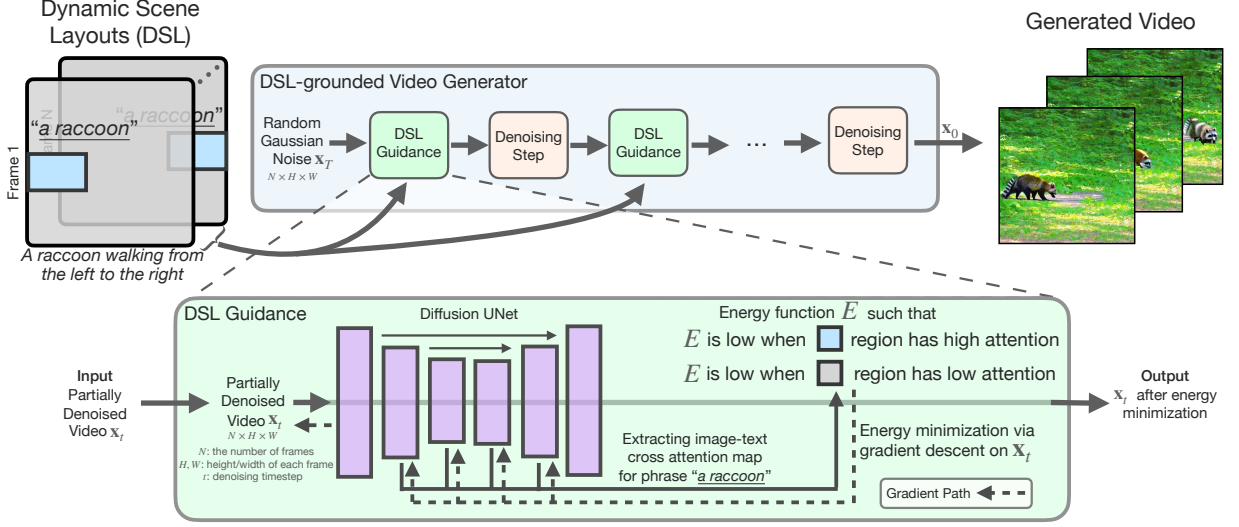


Figure 3.6: **Our DSL-grounded video generator** generates videos from DSLs using existing text-to-video diffusion models augmented with appropriate DSL guidance. Our method alternates between DSL guidance steps and denoising steps. While the denoising steps gradually transform the random video-shaped Gaussian noise into clean videos, our DSL guidance steps ensure that the denoised videos satisfy our DSL constraints. We designed an energy function minimized when the text-to-video cross-attention maps have high values within the box area and low values outside the box w.r.t. the layout box name. In the DSL guidance step, our method minimizes the energy function E by performing gradient descent on the video, steering the generation to follow the constraints.

with both the text prompt and the given DSL. Our method does not require fine-tuning, which relieves it from the need for instance-annotated images or videos and the potential catastrophic forgetting incurred by training.

A text-to-video diffusion model (e.g., [7]) generates videos by taking in a random Gaussian noise of shape $N \times H \times W \times C$, where N is the number of frames and C is the number of channels. The backbone network (typically a U-Net [76]) iteratively denoises the input based on the output from the previous time step as well as the text prompt, and the text-conditioning is achieved with cross-attention layers that integrate text information into latent features. For each frame, we denote the cross-attention map from a latent layer to the object name in the text prompt by $\mathbf{A} \in \mathbb{R}^{H \times W}$, where H and W are the height and width of the latent feature map. For brevity, we ignore the indices for diffusion steps, latent layers, frames, and object indices. The cross-attention map typically highlights positions where objects are generated.

As illustrated in Figure 3.6, to make sure an object is generated in and moves with the given DSL, we encourage the cross-attention map to concentrate within the corresponding bounding box. To achieve this, we turn each box of the DSL into a mask $\mathbf{M} \in \mathbb{R}^{H \times W}$, where

the values are ones inside the bounding box and zeros outside the box. We then define an energy function $E_{\text{topk}} = -\text{Topk}(\mathbf{A} \cdot \mathbf{M}) + \text{Topk}(\mathbf{A} \cdot (1 - \mathbf{M}))$, where $\cdot$ is element-wise multiplication, and Topk takes the average of top- k values in a matrix. Intuitively, minimizing this energy encourages at least k high attention values inside the bounding box, while having only low values outside the box.

However, if the layout boxes in consecutive frames largely overlap (i.e., the object is moving slowly), E_{topk} may result in imprecise position control. This is because stationary objects in the overlapped region can also minimize the energy function, which the text-to-video diffusion model often prefers to generate. To further ensure that the temporal dynamics of the generated objects are consistent with the DSL guidance, we propose Center-of-Mass (CoM) energy function E_{CoM} which encourages CoM of $\mathbf{A}$ and $\mathbf{M}$ to have similar positions and velocities. Denoting the CoM of t -th frame’s attention map and box mask by $\mathbf{p}_t^A$ and $\mathbf{p}_t^M$, the $E_{\text{CoM}} = \|\mathbf{p}_t^A - \mathbf{p}_t^M\|_2^2 + \|(\mathbf{p}_{t+1}^A - \mathbf{p}_t^A) - (\mathbf{p}_{t+1}^M - \mathbf{p}_t^M)\|_2^2$. Overall energy E is a weighted sum of E_{topk} and E_{CoM} .

To minimize the energy E , we add a DSL guidance step before each diffusion step where we calculate the gradient of the energy w.r.t. the partially denoised video for each frame and each object, and update the partially denoised video by gradient descent with classifier guidance [48, 16, 15, 55, 56, 79].

With this simple yet effective training-free conditioning stage, we are able to control video generation with DSL and complete the LVD pipeline of text prompt $\rightarrow$ DSL $\rightarrow$ video. Furthermore, the two stages of our LVD pipeline are implementation-agnostic and can be easily swapped with their counterparts, meaning that the generation quality and text-video alignment will continue to improve with more capable LLMs and layout-conditioned video generators. In Figure 3.5, we show that our DSL-grounded video generator can generate realistic videos represented by the DSLs in Figure 3.4.

3.5 Evaluation

Setup. We select ModelScope [7] as our base video diffusion model and apply LVD on top of it. In addition to comparing LVD with the ModelScope baseline without DSL guidance, we also evaluate LVD against a retrieval-based baseline. Given a text prompt, this baseline performs nearest neighbor retrieval from a large-scale video captioning dataset WebVid-2M based on the similarity between the prompt and the videos with Frozen-In-Time [80], and runs GLIP [81] to extract the bounding boxes for the entities in the captions of the retrieved videos and tracks the bounding boxes with SORT [82] to get the DSLs.

Benchmarks. To evaluate the alignment of the generated DSLs and videos with text prompts, we propose a benchmark comprising five tasks: generative numeracy, attribute binding, visibility, spatial dynamics, and sequential actions. Each task contains 100 programmatically generated prompts that can be verified given DSL generations or detected bounding boxes using a rule-based metric, with 500 prompts in total. We generate two videos for each prompt with different random seeds, leading to 1000 video generations per evaluation. We

Table 3.1: **Evaluation of the generated DSLs.** Our LLM spatiotemporal planner is able to generate layouts that align well with the spatiotemporal dynamics requirements in several types of prompts.

Method	Numeracy	Attribution	Visibility	Dynamics	Sequential	<i>Average</i>
Retrieval-based	20%	12%	28%	8%	0%	<i>14%</i>
LVD (GPT-3.5, ours)	100%	100%	100%	71%	16%	<i>77%</i>
LVD (GPT-4, ours)	100%	100%	100%	100%	88%	98%
+ retrieval examples	98%	100%	100%	96%	85%	<i>96%</i>

Table 3.2: **Detection-based evaluation of the generated videos.** LVD follows the spatiotemporal keywords in the prompts better than ModelScope, the diffusion model that it uses under the hood. Generated videos have a resolution of 256×256 .

Method	LLM-grounded	Numeracy	Attribution	Visibility	Dynamics	Sequential	<i>Average</i>
ModelScope	✗	32%	54%	8%	21%	0%	<i>23.0%</i>
Retrieval-based	✓	15%	15%	11%	9%	0%	<i>9.7%</i>
LVD (GPT-3.5, ours)	✓	52%	79%	64%	37%	2%	<i>46.4%</i>
LVD (GPT-4, ours)	✓	41%	64%	55%	51%	38%	49.4%

then report the success rate of how many videos follow the text prompts. Furthermore, we adopt the benchmarks (UCF-101 [59] and MSR-VTT [60]) as referenced in literature [53, 52] to evaluate the quality of video generation. We also conduct an evaluator-based assessment as well. We refer readers to Section B.4 for details about our evaluation setup and proposed benchmarks.

Evaluation of Generated DSLs

We evaluate the alignment between the DSLs generated by an LLM and the text prompts without the final video generation. The results are presented in Table 3.1. We first compare with a baseline model that directly outputs the layout of the closest retrieved video as the DSL without an LLM involved. Its low performance indicates that nearest neighbor retrieval based on text-video similarity often fails to locate a video with matching spatiotemporal layouts. In contrast, our method generates DSLs that align much better with the query prompts, where using GPT-3.5 gives an average accuracy of 77% and using GPT-4 can achieve a remarkable accuracy of 98%. These results highlight that current text-only LLMs can generate reasonable spatiotemporal layouts based solely on a text prompt, which is hard to achieve with a retrieval-based solution. Notably, only three in-context examples in Figure 3.3 are used throughout the evaluation. To determine if incorporating examples from real videos enhances the performance, we experimented with adding the samples from

Table 3.3: **Ablations on the energy function used to condition the video diffusion model on the layouts.** We use the same set of layouts generated by the LLM (GPT-4) in the ablation. Neither E_{ratio} proposed in [15] nor E_{corner} proposed in Boxdiff [16] help improve the text-video alignment, whereas simply using E_{topk} achieves good text-video alignment. Adding E_{CoM} can further boost the text-video alignment for sequential tasks. Generated videos are of 256×256 .

E_{topk}	E_{ratio}	E_{corner}	E_{CoM}	Numeracy	Attribution	Visibility	Dynamics	Sequential	Average
✗	✓	✗	✗	44%	53%	11%	54%	21%	36.4%
✓	✗	✓	✗	38%	60%	53%	48%	31%	45.6%
✓	✗	✗	✗	39%	65%	54%	52%	23%	46.3%
✓	✗	✗	✓	41%	64%	55%	51%	38%	49.4%

Table 3.4: **Video quality evaluation.** Lower FVD scores are better. [†] indicates our replication of the results. LVD is able to generate videos with higher quality than ModelScope on both benchmarks.

Method	FVD@UCF-101 (↓)	FVD@MSR-VTT (↓)
CogVideo (Chinese) [73]	751	-
CogVideo (English) [73]	702	1294
MagicVideo [83]	699	1290
Make-A-Video [52]	367	-
VideoLDM [53]	551	-
ModelScope [7]	-	550
ModelScope [†]	984	590
ModelScope w/ LVD (ours)	828	565

retrieval. However, these additional examples lead to similar performance, underscoring the LLM’s few-shot generalization capabilities. We also noticed that GPT-3.5 struggles with sequential tasks. While it typically begins correctly, it either skips one of the two actions or executes them simultaneously.

Evaluation of Generated Videos

Evaluation of video-text alignment. We now assess whether the generated videos adhere to the spatial layouts and temporal dynamics described in the text prompts. For evaluation, we also use the rule-based method, with boxes from detector OWL-ViT [42]. With results presented in Table 3.2, ModelScope underperforms in most of the scenarios and is barely able to generate videos in two of the five tasks. In addition, videos from the

Table 3.5: **Assessment based on human evaluators.** Despite sharing *the same diffusion weights* as the baseline model [7], our method is preferred significantly more often.

Question	Baseline	LVD	Similar
<i>Which video aligns better with the text prompt?</i>	3.2%	85.8%	12.6%
<i>Which video has better quality?</i>	14.5%	66.0%	19.5%

retrieved layouts have worse text-video alignment performance when compared to the baseline method. On the other hand, LVD significantly outperforms both baselines by a large margin, suggesting the improved video-text alignment of the proposed LVD pipeline. Interestingly, the layouts generated by GPT-3.5 lead to better text-video alignment for three tasks, as they have less motion in general and are easier to generate videos from. The gap between the results in Table 3.1 and Table 3.2 still indicates that the bottleneck is in the DSL-grounded video generation stage.

Ablations on the energy function. *E.* Boxdiff [16] proposes a corner term E_{corner} with E_{topk} , and [15] proposes to use a ratio-based energy function E_{ratio} . As shown in Table 3.3, we found that using only E_{topk} achieves a good text-video alignment already. We also recommend adding E_{CoM} when sequential tasks are entailed, as it offers significant boosts.

Evaluation of video quality. Since our primary objective is to generate videos that align with intricate text prompts without additional training, improving video quality is not the key objective to demonstrate the effectiveness of our method. We evaluate the video quality through FVD score on UCF-101 and MSR-VTT. The FVD score measures if the generated videos have the same distribution as the dataset. Both benchmarks feature videos of complex human actions or real-world motion. Since our method is agnostic to the underlying models and applicable to different text-to-video diffusion models, we mostly compare with ModelScope [7], the diffusion model that we use under the hood. Both LVD and our replicated ModelScope generate 256×256 videos. As shown in Table 3.4, LVD improves the generation quality compared to the ModelScope baseline on both datasets, suggesting our method can produce actions and motions that are visually more similar to real-world videos. ModelScope and LVD surpass models such as [83] on MSR-VTT, while being worse on UCF-101, possibly due to different training data.

Visualizations. In Figure 3.7, we show our video generation results along with baseline Modelscope. LVD consistently outperforms the base model in terms of generating the correct number of objects, managing the introduction of objects in the middle of the video, and following the sequential directions specified in the prompt. In addition, LVD is also able to handle descriptions of multiple objects well. For example, in Figure 3.7(b), while ModelScope assigns the color word for the dog to a ball, LVD precisely captures both the blue dog and the brown ball in the generation.

Evaluator-based assessment. To further validate the quality of videos generated with LVD, we conducted an evaluator-based assessment. We asked two questions to 10

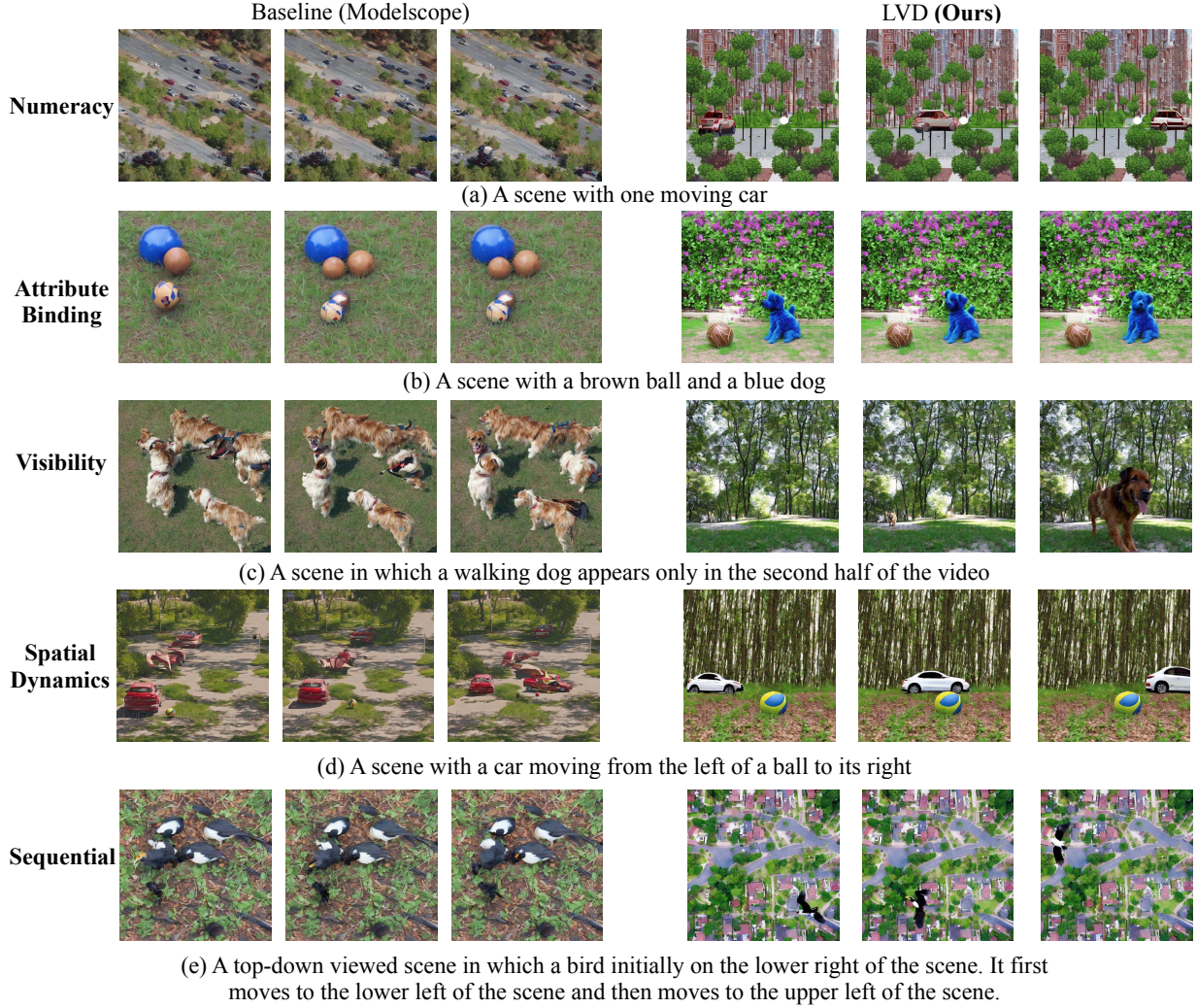


Figure 3.7: Despite not aiming for each individual aspect of the quality improvement, **LVD generates videos that align much better with the input prompts** in several tasks that require numerical, temporal, and 3D spatial reasoning. Best viewed when zoomed in.

participants about each of the 20 pairs of videos randomly selected from our benchmark: “Which video aligns better with the text prompt?” and “Which video has better quality?”. Each set is composed of two videos, one generated by baseline and the other by LVD. We randomly shuffled the videos within each set. For each question, the options are “Video #1”, “Video #2”, and “Similar”. We show the averaged score in Table 3.5. LVD holds a substantial edge over the baselines. In 96.8% of alignment cases and 85.5% of visual quality cases, the evaluators indicate that LVD outperforms or is at least on par with the baseline model. This highlights that LVD brings about a significant enhancement in the performance of baseline

video diffusion models in both the alignment and visual quality.

3.6 Discussion and Future Work

In this work, we propose LLM-grounded Video Diffusion (LVD) to enhance the capabilities of text-to-video diffusion models to handle complex prompts without any LLM or diffusion model parameter updates. Although LVD shows substantial enhancement over prior methods, there is still potential for further improvement. For example, LVD still inherits limitations from the base model when it comes to synthesizing objects or artistic styles it struggles with. We anticipate that techniques like LoRA fine-tuning [36] could help mitigate such challenges and leave this exploration for future research. Our aim for this work is **1)** to demonstrate LLM’s surprising spatiotemporal modeling capability and **2)** to show that LLM-grounded representation, such as DSLs, can be leveraged to improve the alignment for text-to-video generation, with potential extensions to control finer-grained properties, including human poses, to model complex dynamics. Our DSL-to-video stage is still simple, and thus a more advanced DSL-conditioned video generator may further improve the precision of control and video quality in future work.

Reproducibility statement

We mainly apply and evaluate our method on open-sourced model ModelScope [7], allowing the reproduction of results in this work. We will also release code and benchmarks for reproducibility and future research.

Chapter 4

Self-correcting LLM-controlled Diffusion Models

This chapter is based on the following publication:

Tsung-Han Wu*, Long Lian*, Joseph E. Gonzalez, Boyi Li, and Trevor Darrell.
 “Self-correcting LLM-controlled Diffusion Models.” *IEEE/CVF Conference on
 Computer Vision and Pattern Recognition (CVPR)*, 2024. (*Equal contribution.)

4.1 Introduction

Text-to-image generation has made remarkable advancements, especially with the advent of diffusion models. However, these models often struggle with interpreting complex input text prompts, particularly those that require skills such as understanding the concept of numeracy, spatial relationships, and attribute binding with multiple objects. Despite the astonishing scaling of model sizes and training data, these challenges, as illustrated in Figure 4.1, are still present in state-of-the-art open-source and proprietary diffusion models.

Several research and engineering efforts aim to overcome these limitations. For instance, methods such as DALL-E 3 [84] focus on the diffusion training process and incorporate high-quality captions into the training data at a massive scale. However, this approach not only incurs substantial costs but also frequently falls short in generating accurate images from complicated user prompts, as shown in Figure 4.1. Other work harnesses the power of external models for a better understanding of the prompt in the inference process before the actual image generation. For example, [85, 23] leverages Large Language Models (LLMs) to pre-process textual prompts into structured image layouts and thus ensures the preliminary design aligns with the user’s directives. However, such integration does not resolve the inaccuracies produced by the downstream diffusion models, particularly in images with complex scenarios like multiple objects, cluttered arrangements, or detailed attributes.

Drawing inspiration from the from the process of a human painting and a diffusion model in generating images, we observe a key distinction in their approach to creation. Consider

a human artist tasked with painting a scene featuring two cats. Throughout the painting process, the artist remains cognizant of this requirement, ensuring that two cats are indeed present before considering the work complete. Should the artist find only one cat depicted, an additional one would be added to meet the prompt’s criteria. This contrasts sharply with current text-to-image diffusion models, which operate on an *open-loop* basis. These models generate images through a predetermined number of diffusion steps and present the output to the user, regardless of its alignment with the initial user prompt. Such a process, irrespective of scaling training data or LLM pre-generation conditioning, lacks a robust mechanism to ensure the final image aligns with the user’s expectations.

In light of this, we propose our method **Self-correcting LLM-controlled Diffusion (SLD)** that performs *self-checks* to confidently offer users guarantees of the alignment between the prompt and the generated images. Departing from conventional single-round generation methods, SLD is a novel *closed-loop* approach that equips diffusion models with the ability to iteratively identify and rectify errors. Our SLD framework, illustrated in Figure 4.1, contains two main components: LLM-driven object detection as well as LLM-controlled assessment and correction.

The SLD pipeline follows a standard text-to-image generation setting. Given a textual prompt that outlines the desired image, SLD begins with calling an image generation module (e.g., the aforementioned open-loop text-to-image diffusion models) to generate an image in a best-effort fashion. Given that these open-loop generators do not guarantee an output that aligns perfectly with the prompt, SLD then conducts a thorough evaluation of the produced image against the prompt, with an LLM parsing key phrases for an open-vocabulary detector to check. Subsequently, an LLM controller takes the detected bounding boxes and the initial prompt as input, checks for potential mismatches between the detection results and the prompt requirements, suggesting appropriate self-correction operations, such as adding, moving, and removing objects. Finally, utilizing a base diffusion model (e.g., Stable Diffusion [13]), SLD employs latent space composition to implement these adjustments, thereby ensuring that the final image accurately reflects the user’s initial text prompt.

Notably, our pipeline does not pose restrictions on the source of the initial generation and thus is applicable to images generated by proprietary models, such as DALL-E 3 [84], through APIs, as shown in Figure 4.1. Furthermore, none of the self-correction operations require any additional training on our base diffusion model [13], which easily allows our method to be applied to various diffusion models without the costs of external human annotation or training.

We demonstrate that our SLD framework can achieve significant improvement over current diffusion-based methods on complex prompts with the LMD benchmark [85]. The results show that our method is able to surpass LMD+, which is a strong baseline that already leverages LLM in the image process generation, by 9.0%. More importantly, with DALL-E 3 for initial generation, the generated images from our method achieve 26.5% performance gains compared to ones before self-correction.

Finally, since the SLD pipeline is agnostic to the initially generated image, it can easily be transformed into an image editing pipeline by simply changing the prompts to the LLM.

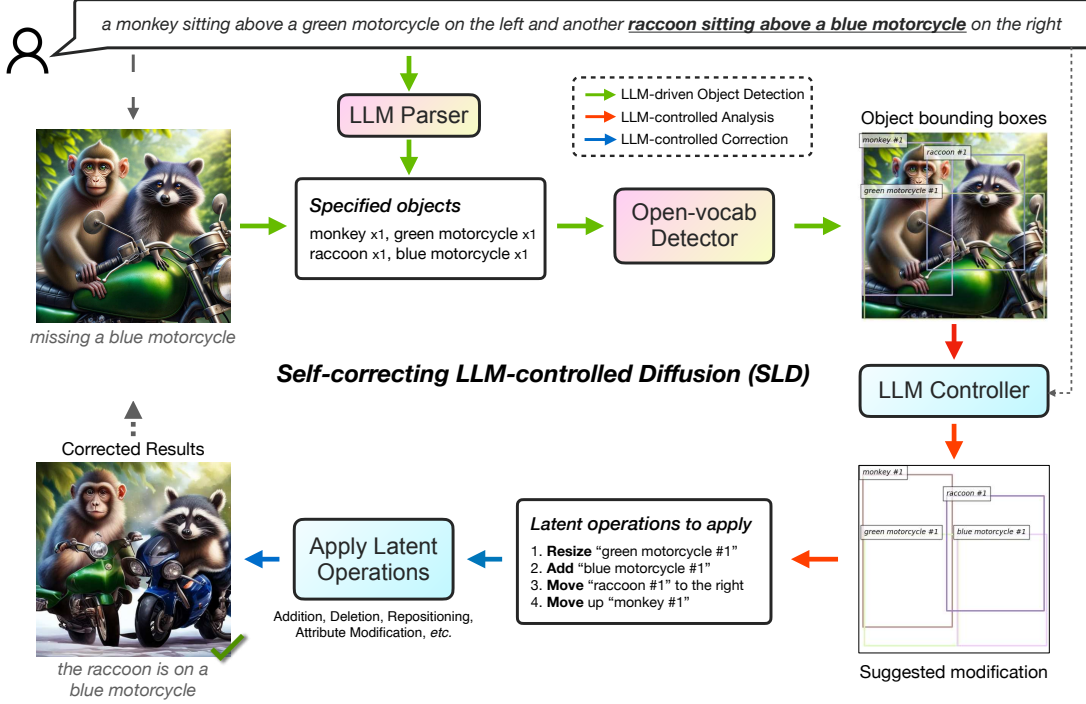


Figure 4.1: Our proposed Self-correcting LLM-controlled Diffusion (SLD) enhances text-to-image alignment through an iterative self-correction process. It begins with LLM-driven object detection (Section 4.3), and subsequently performs LLM-controlled analysis and correction (Section 4.3). The entire pipeline is outlined in Algorithm 1.

While text-to-image generation and image editing are often treated as distinct tasks by the generative modeling community, our SLD is able to perform these two tasks with a unified pipeline. **We list our key contributions below:**

1. SLD is the first to integrate a detector and an LLM to *self-correct* generative models, ensuring accurate generation without extra training or external data.
2. SLD offers a unified solution for both image generation and editing, enabling enhanced text-to-image alignment for any image generator (e.g., DALL-E 3) and object-level editing on any images.
3. Experimental results show that our approach can correct a majority of incorrect generations, particularly in aspects of numeracy, attribute binding, and spatial relationships.

We will release our code for future research and applications.

4.2 Related Work

Text-to-Image Diffusion Models

Diffusion-based text-to-image generation has advanced significantly. Initial studies [13, 17, 18] showed diffusion models’ ability to create high-quality images, but they struggle with complex prompts. Subsequent research [30, 5, 34, 15, 86, 87] has incorporated additional inputs such as keypoints and bounding boxes to control the diffusion generation process.

Recent advancements have incorporated LLMs to control the generation of diffusion models, bypassing the need for additional complementary information as inputs [85, 55, 23, 78, 88]. In these approaches, LLMs play a central role in directly interpreting user textual prompts and managing the initial layout configuration. Despite some progress, these models often operate in an open-loop fashion, producing images in one iteration that cannot guarantee the generated images align with user prompts.

Unlike prior work, SLD is the first closed-loop diffusion-based generation method. Integrating advanced object detectors and LLMs, SLD performs iterative self-checking and correction, significantly enhancing text-to-image alignment. This improvement spans numeracy, attribute binding to multiple objects, and spatial reasoning, applicable to various models, including those like DALL-E 3.

Diffusion-based Image Editing

Recent advancements in text-to-image diffusion models have significantly expanded their applications in image editing, encompassing both global and local editing tasks. Techniques like Prompt-2-prompt [89] and InstructPix2Pix [37] specialize in global edits, such as style transformations. Conversely, methods like SDEdit [90], DiffEdit [91], and Plug-and-Play [92] focus on local edits, targeting specific areas within images. Despite their progress, these methods often struggle with precise object-level manipulation and tasks that require spatial reasoning, such as resizing or repositioning objects. While recent approaches like Self-Guidance [93] offer fine-grained operations, they still necessitate user inputs for specific coordinates when moving or repositioning objects.

Unlike these methods only focusing on diffusion models, SLD introduces the combination of detectors and LLMs in the loop of editing, enabling fine-grained editing with only user prompts. Also, SLD excels in a variety of object-level editing tasks, including adding, replacing, moving, and modifying attributes, swapping, and so on, demonstrating a notable improvement in both ease of use and editing capabilities.

4.3 Self-correcting LLM-controlled Diffusion

In this section, we introduce our **Self-correcting LLM-controlled Diffusion (SLD)** framework. SLD consists of two main components: LLM-driven object detection (Section 4.3) as

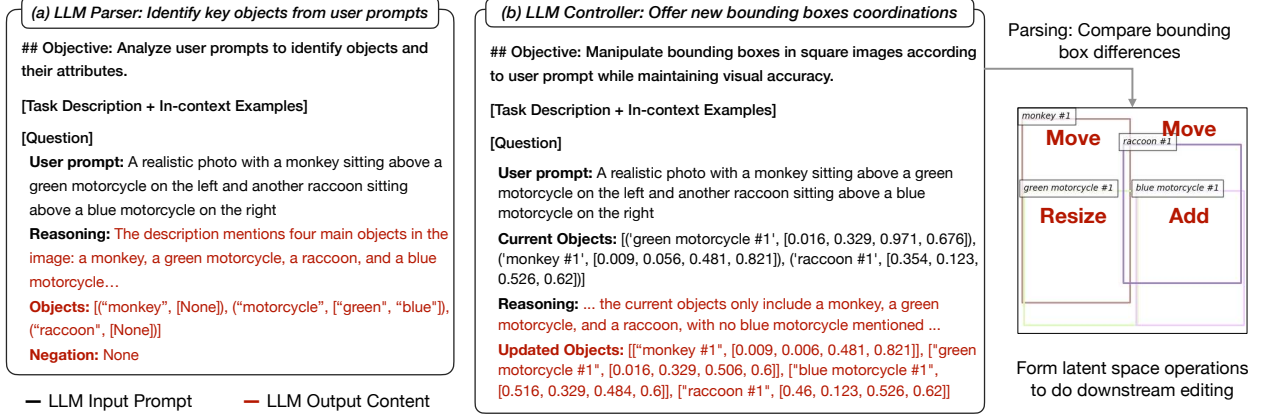


Figure 4.2: Our self-correction pipeline is driven by two distinct LLMs: **(a)** The LLM parser analyzes user prompts P to extract a list of key object information S , which is then passed to the open-vocabulary detector. **(b)** The LLM controller, taking both the user prompt P and currently detected bounding boxes B_{curr} as input, outputs suggested new bounding boxes B_{next} . These are subsequently transformed into a set of latent space operations Ops for image manipulation.

well as LLM-controlled assessment and correction (Section 4.3). Moreover, with a simple change of the LLM instructions, we show that SLD is applicable to image editing, unifying text-to-image generation and editing as discussed in Section 4.3. The complete pipeline is shown in Algorithm 1.

LLM-driven Object Detection

Our SLD framework starts with LLM-driven object detection, which extracts information necessary for downstream assessment and correction. As shown with green arrows in Figure 4.1, the LLM-driven object detection includes two steps: **1)** We leverage an LLM as a parser that parses the user prompt and outputs key phrases that are potentially relevant to image assessment. **2)** The phrases are then passed into an open-vocabulary object detector. The detected boxes are supposed to contain information that supports the assessment of whether the image is aligned with the specifications in the user prompt.

In the initial step, an LLM parser is directed to extract a list of key object details, denoted as S , from the user-provided text prompt P . This parser, aided by text instructions and in-context examples, can easily accomplish this as shown in Figure 4.2 (a). For a user prompt that includes phrases like “a green motorcycle” and “a blue motorcycle,” the LLM is expected to identify and output “green” and “blue” as attributes associated with “motorcycle.” When the prompt references objects without specific quantities or attributes, such as “a monkey” and “a raccoon,” these descriptors are appropriately left blank. Importantly, the LLM’s role

is not limited to merely identifying object nouns; it also entails identifying any associated quantities or attributes.

In the second step, an open-vocabulary detector processes the list of key object information, S , parsed in the first step, to detect and localize objects within the image. We prompt the open-vocabulary object detector with queries formatted as `image of a/an [attribute] [object name]`, where the “attribute” and “object name” are sourced from the parser’s output. The resulting bounding boxes, B_{curr} , are then organized into a list format like `[("[attribute] [object name] [#object ID]", [x, y, w, h])]` for further processing. A special case is when the prompt poses constraints on the object quantity. For cases where attributed objects (e.g., “blue dog”) fall short compared to the required quantities, a supplementary count of non-attributed objects (e.g., “dog”) is provided to provide context for the subsequent LLM controller deciding whether to add more “blue dogs” or simply alter the color of existing dogs to blue. We will explain these operations, including object addition and attribute modification, in greater detail in Section 4.3.

LLM-controlled Analysis and Correction

We use an LLM controller for image analysis and the subsequent correction. The controller, given the user prompt P and detected boxes B_{curr} , is asked to analyze whether the image, represented by objects bounding boxes, aligns with the description of the user prompt and offer a list of corrected bounding boxes B_{next} , as shown in Figure 4.2 (b).

SLD then programmatically analyzes the inconsistencies between the refined and original bounding boxes to output a set of editing operations Ops , which includes addition, deletion, repositioning, and attribute modification. However, a simple set-of-boxes representation does not carry correspondence information, which does not allow an easy way to compare the input and the output layout of the LLM controller when multiple boxes share the same object name. For example, when there are two cat boxes in both the model input and the model output, whether one cat box corresponds to which cat box in the output layout is unclear. Rather than introducing another algorithm to guess the correspondence, we propose to let the LLM output correspondence with a very simple edit: we give an object ID to each bounding box, with the number increasing within each object type, as a suffix added after the object name. In the in-context examples, we demonstrate to the LLM that the object should have the same name and object ID before and after the proposed correction.

Latent Operations for Training-Free Image Correction

The LLM controller outputs a list of correction operations to apply. For each operation, we first transform the original image into latent features. Our approach then executes a series of operations Ops , such as addition, deletion, repositioning, and attribute modification, applied to these latent layers. We explain how each operation is performed below.

Addition. Inspired by [85], the addition process entails two phases: pre-generating an object and integrating its latent representation into the original image’s latent space. Initially, we

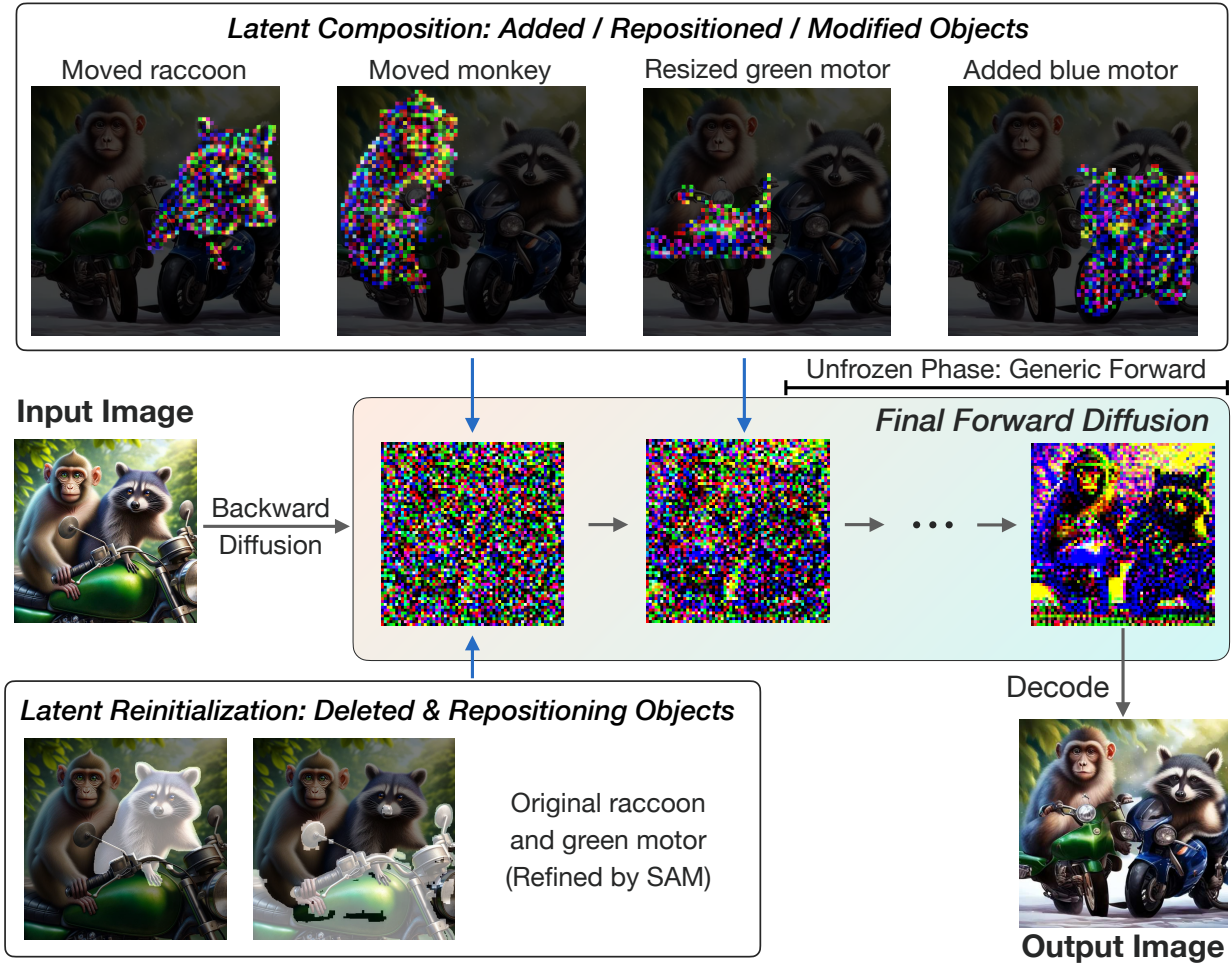


Figure 4.3: Our latent operations can be summarized into two key concepts: (1) latent in removed regions are re-initialized to Gaussian noise, and latent of newly added or modified objects are composited onto the canvas. (2) Latent composition is confined to the initial steps, followed by “unfrozen” steps for a standard forward diffusion process, enhancing visual quality and avoiding artificial copy-and-paste effects.

use a diffusion model to create an object within a designated bounding box, followed by precise segmentation using models (e.g., SAM [94]). This object is then processed through a backward diffusion sequence with our base diffusion model, yielding masked latent layers corresponding to the object, which are later merged with the original canvas.

Deletion operation begins with SAM refining the boundary of the object within its bounding box. The latent layers associated with these specified regions are then removed and reset with Gaussian noise. This necessitates a complete regeneration of these areas during the following forward diffusion process.

Algorithm 1 Self-correction for Image Generation.

Input: User prompt P , Initial generated image I , Maximum number of self-correction round K .

```

1: for  $k \leftarrow 1$  to  $K$  do
2:    $S \leftarrow \text{LLM-Parser}(P)$ 
3:    $B_{curr} \leftarrow \text{Detector}(S)$ 
4:    $B_{next} \leftarrow \text{LLM-Analysis}(P, B_{curr})$ 
5:    $Ops \leftarrow \text{Diff}(B_{curr}, B_{next})$ 
6:   if  $Ops \neq \emptyset$  then (i.e.,  $B_{next} \neq B_{curr}$ )
7:      $I = \text{Correction}(Ops, B_{next}, B_{curr})$ 
8:   else
9:     break
10:  end if
11: end for

```

Output: Image I .

Repositioning involves modifying the original image to align objects with new bounding boxes, taking care to preserve their original aspect ratios. The initial steps include shifting and resizing the bounding box in the image space. Following this, SAM refines the object boundary, succeeded by a backward diffusion process to generate its relevant latent layers, similar to the approach in the addition operation. Latent layers corresponding to the excised parts are replaced with Gaussian noise, while the newly added sections are integrated into the final image composition. An important consideration in repositioning is conducting object resizing in the image space rather than the latent space to maintain high-quality results.

Attribute modification starts with SAM refining the object boundary within the bounding box, followed by applying attribute modifications such as DiffEdit [91]. The base diffusion model then reverses the image, producing a series of masked latent layers ready for final composition.

After editing operations on each object, we proceed to the recomposition phase as shown in Figure 4.3. In this phase, while latents for removed or repositioned regions are reinitialized with Gaussian noise, the latents for added or modified latents are updated accordingly. For regions with multiple overlapping objects, we place the larger masks first to ensure the visibility of the smaller objects.

The stack of modified latent then undergoes a final forward diffusion process, which begins with steps in which regions not reinitialized with Gaussian noise are frozen (i.e., forced to align with the unmodified latent at the same step). This is crucial for the accurate formation of updated objects while maintaining background consistency at the same time. The procedure finishes with several steps where everything is allowed to change, resulting in a visually coherent and correct image.

Method	Accuracy				
	Negation	Numeracy	Attribute	Spatial	Average
MultiDiffusion [14]	29%	28%	26%	39%	30.5%
Backward Guidance [15]	22%	37%	26%	67%	38.0%
BoxDiff [16]	22%	30%	37%	71%	40.0%
LayoutGPT + GLIGEN [23, 5]	36%	65%	26%	78%	51.3%
DALL-E 3 [84]	25%	38%	74%	71%	52.0%
+ 1-round SLD (Ours)	90%	61%	80%	83%	78.5% (+26.5)
LMD+ [85]	100%	82%	49%	86%	79.3%
+ 1-round SLD (Ours)	100%	98%	63%	92%	88.3% (+ 9.0)

Table 4.1: Our method can be applied on various image generation methods and improves the generation accuracy by a large margin.

Termination of the Self-Correction Process

Even though we observe that one round of generation is often enough for a majority of the cases that we encountered, subsequent rounds could still benefit the performance in terms of correctness further, making our self-correction an iterative process.

Determining the optimal number of self-correction rounds is critical for balancing efficiency and accuracy. As outlined in Algorithm 1, our method sets a maximum number of attempts on the correction rounds to ensure the process finishes within a reasonable amount of time.

The process completes when the LLM outputs the same layout as the input (i.e., if the bounding boxes suggested by the LLM controller (B_{next}) align with the current detected bounding boxes (B_{curr})), or when the maximum rounds of generation are reached, which indicates that the method is unable to make a correct generation for the prompt. This iterative process provides guarantees on the correctness of the image, up to the accuracy of the detector and the LLM controller, ensuring it aligns closely with the initial text prompt. We explore the efficacy of multi-round corrections in Section 4.4.

Unified text-to-image generation and editing

In addition to self-correcting image generation models, our SLD framework is readily adaptable for image editing applications, requiring only minimal modifications. A key distinction is in the format of user input prompts. Unlike image generation, where users provide scene descriptions, image editing requires users to detail both the original image and the desired changes. For instance, to edit an image with two apples and a banana by replacing the banana with an orange, the prompt could be: "Replace the banana with an orange, while keeping the two apples unchanged."



Figure 4.4: SLD enhances text-to-image alignment across diverse diffusion-based generative models such as SDXL, LMD+, and DALL-E 3. Notably, as highlighted by the red boxes in the first row, SLD precisely positions a blue bicycle in relation to a bench and a palm tree, while maintaining the accurate count of palm trees and seagulls. The second row further demonstrates SLD’s robustness in complex, cluttered scenes, effectively managing object collision through our training-free latent operations.

The editing process is similar to our self-correction mechanism. The LLM parser extracts key objects from the user’s prompt. These objects are then identified by the open-vocabulary detector, establishing a list of current bounding boxes. The editing-focused LLM controller, equipped with specific task objectives, guidelines, and in-context examples, analyzes these inputs. It proposes updated bounding boxes and corresponding latent space operations for precise image manipulation.

SLD’s ability to perform detailed, object-level editing distinguishes it from existing diffusion-based methods like InstructPix2Pix [37] and prompt2prompt [89], which mainly address global image style changes. Also, SLD outperforms tools like DiffEdit [91] and SDEdit [90], which are restricted to object replacement or attribute adjustment, by enabling comprehensive object repositioning, addition, and deletion with exact control. Our comparative analysis in Section 4.4 will further highlight SLD’s superior editing capabilities over existing methods.

4.4 Experiments

Comparison with Image Generation Methods

Setup. We evaluate the performance of the SLD framework with the LMD benchmark [85], which is specifically designed to evaluate generation methods on complex tasks such as handling negation, numeracy, accurate attribute binding to multiple objects, and spatial reasoning. For each task, 100 programmatically generated prompts are fed into various text-to-image generation methods to produce corresponding images. We evaluate the images generated by our method and the baselines with open-vocabulary detector OWL-ViT v2 [95] for a robust quantitative evaluation of the alignment between the input prompts and the generated images. We compared SLD with several leading text-to-image diffusion methods, such as Multidiffusion [14], BoxDiff [16], LayoutGPT [23], LMD+ [55], and DALL-E 3 [84]. To ensure fair comparisons, all models incorporating LLMs used the same GPT-4 model. For our SLD implementation, we utilized LMD+ as the base model for latent space operations and OWL-ViT v2 for the open-vocabulary object detector.

Results. As shown in Table 4.1, applying the SLD method to both open-source (LMD+) and proprietary models (DALL-E 3) significantly enhances their performance in terms of generation correctness. **For negation tasks**, as LMD+ converts user prompts containing “without” information into negative prompts, which already achieves a remarkable 100% accuracy without SLD integration. In contrast, even though DALL-E 3 also uses an LLM to rewrite the prompt, it still fails for some negation cases, likely because the LLM simply puts the negation keyword (e.g., “without”) into the rewritten prompt. In this case, our SLD method can automatically rectify most of these errors. **For numeracy tasks**, integrating SLD with LMD+ results in a significant improvement, with up to 98% accuracy. We noted that DALL-E 3 often struggles to generate an image with the correct number of objects. However, this issue is substantially mitigated by SLD, which enhances performance by over 20%. **For attribute binding tasks**, SLD improves the performance of both DALL-E 3 and LMD+ by 6% and 14%, respectively. Notably, DALL-E 3 initially outperforms LMD+ in this task, likely due to its training on high-quality image caption datasets. Finally, **for spatial reasoning tasks**, the integration of SLD with both LMD+ and DALL-E 3 demonstrates enhanced performance by 12% and 6%, respectively.

Application to Image Editing

As discussed in Section 4.3, SLD excels in fine-grained image editing over existing methods. As demonstrated in Figure 4.5, our integration of an open-language detector with LLMs enables precise modifications within localized latent space regions. SLD adeptly performs specific edits, like seamlessly replacing an apple with a pumpkin, while preserving the integrity of surrounding objects. In contrast, methods like InstructPix2Pix [37] are confined to global transformations, and DiffEdit [91] often fails to accurately locate objects for modification, leading to undesired results.

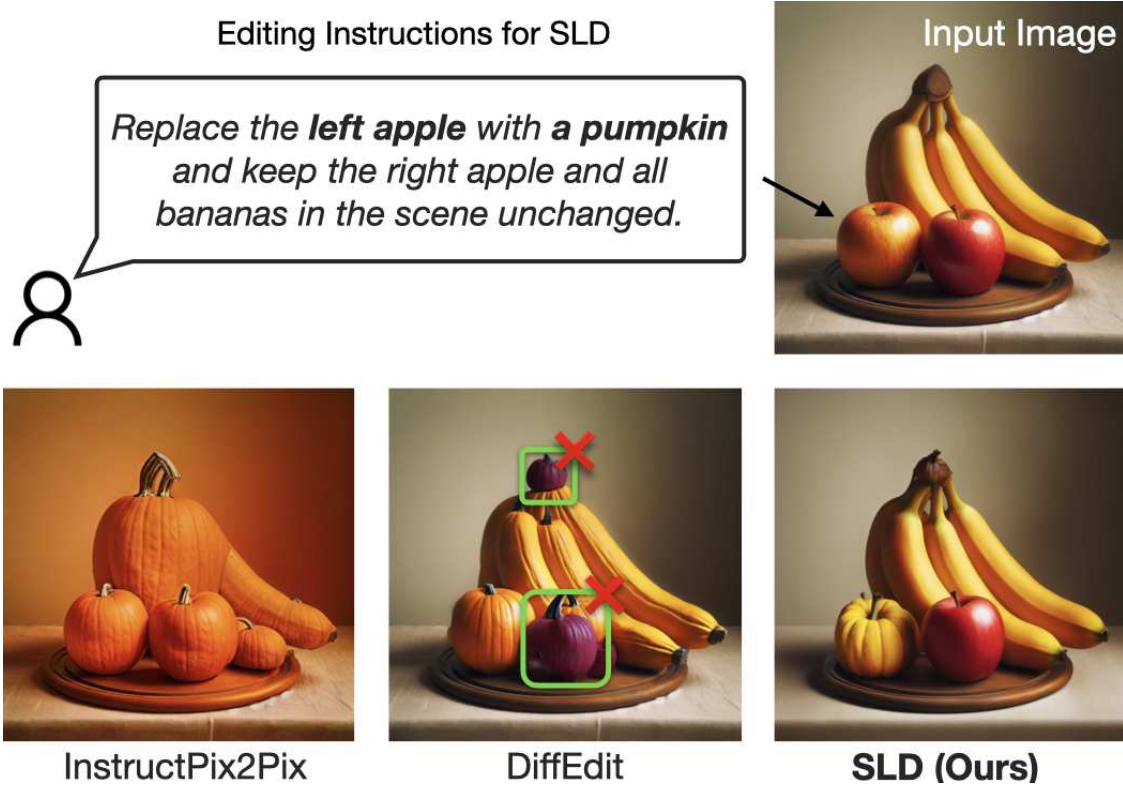


Figure 4.5: When instructed to perform object-level image editing. InstructPix2Pix [37] completely fails to accomplish the task, and DiffEdit [91] falls short, as highlighted in the green box of the image. Conversely, our method demonstrates a significantly better performance in executing these object-level edits.

Furthermore, as exemplified in Figure 4.6, SLD supports a wide array of editing instructions, including counting control (such as adding, deleting, or replacing objects), attribute modification (like altering colors or materials), and intricate location control (encompassing object swapping, resizing, and moving). A standout example is featured in the “Object Resize” column of the first row, where SLD precisely enlarges the cup on the table by an exact factor of $1.25\times$. We encourage readers to verify this with a ruler for a clear demonstration of our method’s precision. This level of precision stems from the detector’s exact object localization coupled with the LLMs’ ability in reasoning and suggestions for new placements. Such detailed control over spatial adjustments is unmatched by any previous method, highlighting SLD’s contributions to fine-grained image editing.



Figure 4.6: SLD can handle a diverse array of image editing tasks guided by natural, human-like instructions. Its capabilities span from adjusting object counts to altering object attributes, positions, and sizes.

Method	Accuracy				
	Negation	Numeracy	Attribute	Spatial	Average
SD [13]	19%	38%	24%	33%	28.5%
+ 1-round SLD	69%	55%	25%	69%	54.5% (+26.0)
+ 2-round SLD	73%	61%	31%	75%	60.0% (+31.5)

Table 4.2: While the majority of errors are typically rectified in the first round, multi-round correction consistently outperforms a single-round approach.

Discussion

Multi-round self-correction. Our analysis in Table 4.2 highlights the benefits of multi-round self-correction and the fact that the first round correction is always the most effective one and has a marginal effect. The first round of corrections substantially mitigates issues inherent in Stable Diffusion [13]. Then, a second round of correction still yields significant improvements across all four tasks.

Limitations and future work. A limitation of our method is illustrated in Figure 4.7, where SLD fails to accurately remove a person’s hair. In this instance, despite the successful



Figure 4.7: SLD struggles with objects of complex shapes, as the SAM module may unintentionally segment adjacent parts during the process.

identification and localization of the hair, the complex nature of its shape poses a challenge to the SAM module used for region selection, resulting in the unintended removal of the person’s face in addition to the hair. However, since the person’s cloth is not removed, the base diffusion model fails to generate a natural composition. This suggests that a better region selection method is needed for further improvements in the generation and editing quality.

4.5 Conclusion

We introduce the Self-correcting Language-Driven (SLD) framework, a pioneering self-correction system using detectors and LLMs to significantly enhance text-to-image alignment. This method not only sets a new SOTA in image generation benchmark but is also compatible with various generative models, including DALL-E 3. Also, SLD extends its utility to image editing applications, offering fine-grained object-level manipulation that surpasses existing methods.

Chapter 5

Describe Anything: Detailed Localized Image and Video Captioning

This chapter is based on the following publication:

Long Lian, Yifan Ding, Yunhao Ge, Sifei Liu, Hanzi Mao, Boyi Li, Marco Pavone, Ming-Yu Liu, Trevor Darrell, Adam Yala, and Yin Cui. “Describe Anything: Detailed Localized Image and Video Captioning.” *International Conference on Computer Vision (ICCV)*, 2025.

5.1 Introduction

Image captioning has been a longstanding challenge in computer vision and natural language processing [96], as it involves understanding and describing visual content in natural language. While recent Vision-Language Models (VLMs) have achieved impressive results in image-level captioning, generating detailed and accurate captions for specific regions within an image remains an open problem. This challenge intensifies with videos, where models must additionally capture dynamic visual content, such as human actions, object motions, and human-object interactions. If resolved, it would open new doors for fine-grained grounded image/video understanding [97, 98] and generation [99, 5].

Most existing VLMs (e.g., GPT-4o [100]) lack mechanisms for precise localization. Recent approaches that empower VLMs to take 2D localization cues such as bounding boxes [101, 102, 103, 104] often yield brief phrases rather than detailed descriptions. While there are methods [105, 106, 107, 108, 109, 110] that produce longer captions, they provide minimal detail or include unrelated content from other regions, as shown in Figure 5.4. This raises the question: *What makes detailed localized captioning (DLC) so challenging?*

We identify three key obstacles to DLC:

1. **Loss of Region Details:** As shown in Figure 5.1, prior methods extract local features from global image representations, often leading to loss of fine-grained de-

tails—particularly for small objects in complex scenes. By the time the LLM processes the visual features, crucial details necessary for generating precise captions are already lost. Cropping the region of interest may enhance detail but risks losing essential contextual cues.

2. **Scarcity of High-Quality Datasets:** Datasets such as RefCOCOs [111, 112] and Visual Genome [113] typically offer only short phrases that do not suffice for training models to generate rich, detailed captions. Recent synthetic data approaches [107, 110] are based on bounding boxes that could not precisely convey the exact region of interests, while methods that rely on global captions [114] may have difficulty capturing non-salient regions.
3. **Limitations in Benchmarks:** Prior localized captioning benchmarks compare generated captions against reference captions using language-based image captioning metrics [115, 116, 117, 118, 119] or LLM-based scoring [108, 120]. However, such techniques are not very applicable to DLC. Since the reference captions provided in the benchmarks often lack comprehensive details of the region, DLC models are often unfairly penalized for correct details not explicitly mentioned in the reference.

We propose the following solutions to these challenges:

To tackle the loss of details in regional features, we propose the **Describe Anything Model** (DAM), which preserves both local detail and global context. DAM achieves this through two key innovations (Figure 5.2): **1)** the *focal prompt*, which encodes the region of interest with high token density, and **2)** the *localized vision backbone*, which ensures precise localization while integrating global context. These components enable DAM to generate detailed and accurate captions, even for small objects in complex scenes.

To overcome the scarcity of high-quality DLC datasets, we introduce **Semi-supervised learning (SSL)-based Data Pipeline** (DLC-SDP) to generate high-quality localized captions in two stages. First, leveraging high-quality masks and keywords (e.g., class names, part names, or entities) from human-annotated segmentation datasets, we query a VLM to expand each keyword into a detailed caption given each mask-referred region. Second, inspired by self-training-based SSL in image classification [121, 122, 123, 124, 125], DLC-SDP performs self-training with web images as an unlabeled dataset and segmentation datasets as labeled data. An LLM further summarizes descriptions into multiple granularities, yielding a diverse dataset with high-quality localized captions, enabling our model to outperform strong API-only baselines such as GPT-4o [100] and o1 [126].

To mitigate the limitations of current benchmarks, we introduce **DLC-Bench**, which evaluates detailed localized captions based on a set of predefined positive and negative attributes for each region, eliminating the reliance on comprehensive reference captions. This approach provides a more flexible and accurate evaluation, encouraging models to generate informative and precise descriptions.

We summarize our contributions as follows:

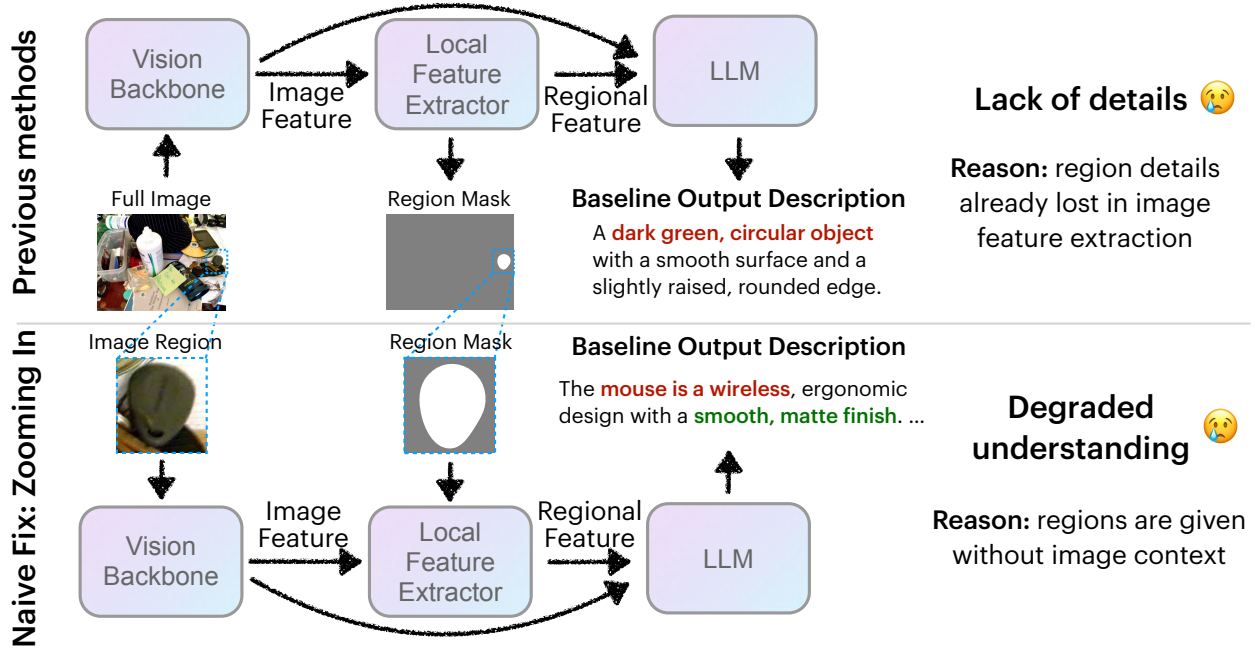


Figure 5.1: **Top:** Prior regional captioners derive regional features from global image representations, leading to vague descriptions. **Bottom:** Zooming in (cropping the image region) enhances detail but loses contextual cues, degrading recognition. This underscores the need for a design that **encodes detail-rich regional features while preserving context for improved DLC performance**.

1. **Describe Anything Model (DAM):** A novel architecture with a focal prompt and a localized vision backbone for multi-granular regional image and video captioning.
2. **SSL Data Pipeline (DLC-SDP):** A semi-supervised data pipeline that leverages high-quality segmentation annotations and unlabeled web images for scalable and diverse data curation.
3. **DLC-Bench:** A benchmark designed to evaluate DLC without reference captions.

Unlike generalist models, we focus on *localized image and video captioning* across *multiple granularities*, **achieving SOTA performance on 7 benchmarks** across keyword, phrase, and detailed multi-sentence captioning. Advantages of DAM, DLC-SDP, and DLC-Bench to prior practices are presented in Table 5.1. We release our code, models, data, and benchmark at describe-anything.github.io.

Component	Previous Practice	Problem	Our Solution	Advantages
DAM	Extracting regional features from global image features	Regional details already lost in image feature extraction and not provided to the LLM	Providing focal prompt to proposed localized vision backbone	Detail-rich contextul features allowing for accurate, multi-granular localized descriptions
DLC-SDP	Query a data curation VLM with referring boxes and global image captions	Imprecise referring to data curation model	Reframe the query into a mask-referred keyword expansion question	Leverage high-quality precise human annotated regional masks and keywords
DLC-SDP	Fully supervised learning	Limited data with high annotation quality	Semi-supervised learning	Scalable to diverse web-scale unlabeled datasets
Benchmark (DLC-Bench)	Pred caption + reference GT caption → language-based similarity metrics or LLM scorer	Incorrect hallucination penalty for correct details not present in the reference caption	Pred caption + query for positive/negative attributes → LLM scorer	Accurate detail and hallucination assessment without relying on reference captions

Table 5.1: Advantages of our proposed model DAM, our SSL data pipeline DLC-SDP, and our benchmark DLC-Bench to previous practices.

5.2 Related Work

Vision-Language Models (VLMs). VLMs integrate visual and textual inputs for multi-modal understanding and are broadly classified into BLIP-style [44, 20, 127, 19, 128, 129, 130] and LLaVA-style [131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142]. However, these models lack precise localization capabilities, limiting their ability to generate regional

descriptions.

Localized Image Captioning. While general VLMs generate image-level captions, localized captioning requires fine-grained regional descriptions. SoM [143, 144] augments VLMs with visual markers, but these markers may blend with the background. Region-aware VLMs [105, 108, 106, 145, 146, 147, 148, 114, 110, 109, 103, 101, 104] introduce regional referring inputs. Recent efforts such as Merlin, Artemis, and VideoRefer [149, 150, 120] extend region-based captioning to videos. However, these methods still struggle with capturing intricate details in the referring regions, as shown by the examples in Figure 5.4. This is because prior models either extract localized features from global image embeddings or simply encode the referring condition as referring tokens, which leads to insufficient regional details for the LLM, especially for small objects. We address this via focal prompting and a localized vision backbone, balancing local detail with global context.

Another limitation is the scarcity of high-quality datasets. Datasets like RefCOCO [111, 112] and VG [113] provide only short phrases. Recent approaches [107, 110, 114, 147, 108, 120] use bounding-box-based VLM queries, sometimes augmented with global captions, for synthetic generation, which can introduce caveats around regional specificity. We propose an SSL data pipeline that uses human-annotated and unlabeled data for richer regional descriptions.

Benchmarking Localized Captioning. [107, 151, 108, 114, 147, 109, 104, 103, 150] evaluate localized captioning by computing language-based image captioning metrics [115, 116, 117, 118, 119] between predicted captions and reference captions. However, these metrics focus on textual matching and may not correlate well with the factual correctness or quality for detailed descriptions. [108, 120] use Sentence-BERT [152] and text-only LLMs to score the predictions against reference captions. However, reference captions often lack comprehensive details about the region of interest, which penalizes models for correct details not explicitly mentioned in the reference by treating them as hallucinations. Our DLC-Bench resolves this issue by eliminating the need for reference captions.

Vision Models with Focus. Prior works enhance attention to salient regions using focal self-attention [153], ToSA [154], ToMe [155], DW-ViT [156], Quadformer [157], and V* [158]. These methods allocate resources dynamically to salient regions defined by the model. In contrast, our focal prompt explicitly prioritizes user-specified regions, ensuring accurate and detailed captions even for non-salient objects.

5.3 DAM: Describe Anything Model

Describe Anything Model (DAM) generates detailed localized descriptions of user-specified regions within images and videos. DAM effectively balances local detail and contextual information through our proposed *focal prompt* and *localized vision backbone*.

Task Formulation

The task of detailed localized captioning involves generating comprehensive textual descriptions focused exclusively on specified regions within images or videos. Formally, given N input frames $I^{(i)} \in \mathbb{R}^{H \times W \times 3}$ and corresponding binary masks $M^{(i)} \in \{0, 1\}^{H \times W}$ indicating the region of interest in each frame, the objective is to produce a detailed description T of the content within the region through a captioning model:

$$T = \text{CaptioningModel}(\{I^{(i)}, M^{(i)}\}_{i=1}^N) \quad (5.1)$$

We focus on using binary masks $M^{(i)}$ as the localization input, since other forms of localization (e.g., points, scribbles, boxes, or masks on an image or a subset of frames in a video) can be transformed into masks via segmentation models such as SAM [94] and SAM 2 [159]. For simplicity, we first introduce our method for localized image captioning, omitting the frame index i , and later extend it to videos in Section 5.3.

Model Architecture

As shown in Figure 5.2, DAM consists of two key components: the *focal prompt* and the *localized vision backbone*.

Focal Prompt

To provide a detailed representation of the region of interest within its context, we introduce the *focal prompt*, which includes both the full image and a focal crop centered around the specified area, along with their corresponding masks.

We first extract the bounding box B of the mask M and expand it by a factor α in both the horizontal and vertical directions to include additional surrounding context:

$$B' = \text{ExpandBox}(B, \alpha). \quad (5.2)$$

For instance, setting $\alpha = 3$ results in a region that can be up to 9 times as large as the original bounding box, subject to clipping at the image boundaries. If either the height or width of the expanded box is less than 48 pixels, we enforce a minimum size of 48 pixels in that dimension to ensure sufficient context for very small regions.

The focal crop of the image and mask are then:

$$I' = I|_{B'}, \quad M' = M|_{B'}, \quad (5.3)$$

where $|_{B'}$ denotes cropping to B' . The focal prompt thus consists of 1) the full image I and its mask M and 2) the focal crop I' and its mask M' . By including both the full image and the focal crop, along with their masks, the focal prompt contains both global context and a detailed view of the region of interest.

Localized Vision Backbone

Effectively processing all four components of the focal prompt with a VLM is non-trivial, as naively concatenating the full image and the focal crop leads to a loss in performance (Table 5.8). We propose the *localized vision backbone*, which 1) achieves localized understanding by encoding the masks in a spatially aligned manner and 2) integrates global context into the region of interest through gated cross-attention.

Handling Localization Inputs. Similar to how an image is encoded by a linear patch embedding layer in vision transformers (ViTs) [160], we integrate the mask M into its corresponding full image I through another patch embedding layer that takes in 2D inputs with one channel.

Specifically, the full image I and its mask M are processed through patch embedding layers, followed by the global vision encoder $f_G(\cdot)$ to obtain global visual features $\mathbf{z}$. The focal crop I' and its mask M' undergo a similar process with the regional vision encoder $f_R(\cdot)$, except that $f_R(\cdot)$ also takes $\mathbf{z}$ as a context to obtain the final fused visual features $\mathbf{z}'$. Specifically, we have:

$$\mathbf{x} = E_I(I) + E_M(M) + P, \quad \mathbf{z} = f_G(\mathbf{x}), \quad (5.4)$$

$$\mathbf{x}' = E_I(I') + E_M(M') + P, \quad \mathbf{z}' = f_R(\mathbf{x}', \mathbf{z}), \quad (5.5)$$

where $E_I(\cdot)$ and $E_M(\cdot)$ are the image and mask patch embedding layer, respectively, $\mathbf{x}$ and $\mathbf{x}'$ are global and focal embedded inputs with information for both the image and the mask, and P denotes the positional encoding.

The newly added mask embedding layer E_M is initialized to output zeros, ensuring that the VLM’s initial behavior is unaffected prior to fine-tuning.

Regional Feature Encoding with Gated Cross-Attention Adapters. To integrate global context into the focal prompt, we insert gated cross-attention adapters [20, 44] into each transformer block of the regional vision encoder f_R . After the self-attention and feed-forward layers, we add a gated cross-attention mechanism that allows local features to attend to global features:

$$\mathbf{h}^{(l)'} = \mathbf{h}^{(l)} + \tanh(\gamma^{(l)}) \cdot \text{CrossAttn}(\mathbf{h}^{(l)}, \mathbf{z}), \quad (5.6)$$

$$\mathbf{h}_{\text{Adapter}}^{(l)} = \mathbf{h}^{(l)'} + \tanh(\beta^{(l)}) \cdot \text{FFN}(\mathbf{h}^{(l)'}), \quad (5.7)$$

where $\mathbf{h}^{(l)}$ is the output of the l -th self-attention block in f_R , $\gamma^{(l)}$ and $\beta^{(l)}$ are learnable scaling parameters initialized to zero, and CrossAttn denotes cross-attention with queries from $\mathbf{h}^{(l)}$ and keys and values from the global features $\mathbf{z}$, similar to how cross-attention is employed in encoder-decoder Transformers [22]. $\mathbf{h}_{\text{Adapter}}^{(l)}$ is used in place of $\mathbf{h}^{(l)}$ in the next Transformer block. To reduce the number of parameters, f_R shares self-attention block weights with f_G .

By initializing $\gamma^{(l)}$ and $\beta^{(l)}$ to zero, we ensure that the initial behavior of the model remains identical to the original VLM prior to fine-tuning. During training, the model learns to leverage the global context to enhance local feature representations, facilitating detailed and contextually accurate descriptions.

Method	LVIS (%)				PACO (%)			
	Sem.	Sim.	(↑)	Sem. IoU (↑)	Sem.	Sim.	(↑)	Sem. IoU (↑)
LLaVA-7B [131]	49.0	19.8		42.2	14.6			
Shikra-7B [105]	49.7	19.8		43.6	11.4			
GPT4RoL-7B [106]	51.3	12.0		48.0	12.1			
Osgpt-7B [108]	65.2	38.2		73.1	<u>22.1</u>			
Ferret-7B [107]	65.0	37.8		-	-			
VP-SPHINX-7B [110]	86.0	61.2		74.2	49.9			
VP-LLAVA-8B [110]	<u>86.7</u>	<u>61.5</u>		<u>75.7</u>	50.0			
DAM-8B (Ours)	89.0	77.7		84.2	73.2			

Table 5.2: LVIS [161] and Table 5.3: Zero-shot evaluation on **phrase-level keyword-level** captioning dataset Flickr30k Entity benchmarks. DAM excels ties [163]. Our model particularly in the challenging achieves 12.3% average relative improvement against PACO benchmark that requires distinguishing between previous best objects and parts.

Method	BLEU	METEOR	ROUGE-L	CIDEr	SPICE
Shikra-7B [105]	18.2	15.3	25.2	49.8	22.0
GPT4RoL-7B [106]	19.7	17.7	29.9	61.7	24.0
Ferret-7B [107]	11.1	8.8	22.7	38.1	17.5
VP-SPHINX-13B [110]	15.2	15.6	27.2	67.4	24.0
RegionGPT-7B [114]	16.1	16.7	27.4	54.6	20.5
DAM-8B (Ours)	22.6	17.8	31.2	74.7	25.5

Method	Short Captioning Metrics					Long Cap. Metrics	
	BLEU	METEOR	ROUGE-L	CIDEr	SPICE	CLAIR	
Shikra-7B [105]	29.5	11.1	23.9	42.7	9.0	34.5	
GPT4RoL-7B [106]	27.1	11.6	26.8	50.9	11.1	43.9	
Ferret-7B [107]	24.6	10.7	22.3	39.7	8.2	45.2	
GLaMM-7B [147]	23.2	10.1	23.8	51.1	8.7	43.8	
VP-SPHINX-13B [110]	22.6	10.7	22.6	32.4	7.6	51.2	
RegionGPT-7B [114]	25.4	12.2	25.3	42.0	8.1	37.2	
DAM-8B (Ours)	38.7	19.4	37.1	70.0	16.9	57.9	

Table 5.4: Zero-shot evaluation on the **detailed captioning** dataset Ref-L4 [164]. Our method achieves 33.4% and 13.1% average relative improvement on the short-/long language-based captioning metrics, respectively.

Generating Detailed Localized Descriptions. The visual features from both the global and regional vision encoders are combined and fed into the large language model to generate detailed, context-aware descriptions T :

$$T = \text{LLM}(\mathbf{t}, \mathbf{z}'), \quad (5.8)$$

where $\mathbf{t}$ denotes textual prompt tokens.

Notably, the proposed components do *not* increase the sequence length of the vision tokens, ensuring that DAM remains efficient. By initializing new modules (mask embedding E_M and scaling parameters $\gamma^{(l)}$ and $\beta^{(l)}$) to zeros, we preserve the pre-trained capabilities of the VLM prior to fine-tuning, *allowing for smooth adaptation of an off-the-shelf VLM without rerunning pre-training*. Thanks to this design, our model requires way less training data ($\sim 1.5\text{M}$ samples) than prior works that involve VLM pretraining.

Extension to Videos

Since images can be considered as videos with a single frame, the model naturally extends to handling videos by processing sequences of frames and their corresponding masks. The visual features from all frames are concatenated in the sequence dimension and fed into the language model to generate detailed localized descriptions across the video frames, compatible with how VLMs are pretrained to handle videos. We leverage SAM 2 [159] to turn sparse localizations into a mask for each frame.

5.4 DLC-SDP: SSL-based Data Pipeline

The effectiveness of DAM depends critically on the availability of high-quality training data for detailed localized descriptions. To this end, we propose the **Semi-supervised**

learning (SSL)-based Data Pipeline (DLC-SDP), a two-stage approach that enables us to build a large and diverse dataset with high-quality localized descriptions.

Stage 1: Leveraging Existing Annotations

The first stage of DLC-SDP reframes the data generation problem into a *vision-grounded description expansion task*. We observed that although current VLMs often struggle to generate detailed localized descriptions when given a referring mask, they can effectively expand short localized descriptions into detailed ones. Inspired by this observation, we leverage high-quality human-annotated masks and keywords (object class names, part names, entities, etc.) from existing segmentation datasets, reframing the VLM query to expand each regional keyword into a detailed caption given the referring masks.

Importantly, our model is trained to predict these high-quality descriptions *without* taking the initial keywords as inputs. Since there are no class labels provided at inference time—and existing VLMs perform poorly without them—our approach ensures superior data quality compared to direct VLM prompting for distillation.

Stage 2: SSL with Unlabeled Data

Since it is not scalable to rely on high-quality manual annotations, the second stage of DLC-SDP employs self-training-based semi-supervised learning techniques inspired by successful approaches in image classification [165, 121, 122, 123]. Our self-training approach involves four steps:

1. **Mask Generation.** We use open-vocabulary segmentation models [140, 104] to extract object masks from unlabeled web images.
2. **Description Generation.** Our DAM, initially trained on the DLC dataset based on annotated segmentation datasets, generates detailed localized descriptions for these regions.
3. **Confidence-based Filtering.** We apply CLIP-based confidence filtering to keep only high-quality samples, following SSL literature.
4. **Data Expansion.** The newly generated (image, mask, description) triplets are added to our training dataset.

This semi-supervised approach dramatically expands the range of object categories and increases data diversity beyond the initial supervised dataset. Furthermore, to support the capability of DAM for multi-granular captioning, we leverage an LLM to summarize the detailed descriptions into shorter forms (e.g., phrases or short sentences), enabling DAM to flexibly generate captions ranging from succinct phrases to multi-sentence narratives.

By systematically curating our training data through this two-stage SSL pipeline, DAM achieves significant performance improvements. Notably, our model trained on data obtained with DLC-SDP outperforms GPT-4o [100] and o1 [126], two strong closed-sourced baselines, showing the effectiveness of DLC-SDP.

5.5 DLC-Bench: Benchmark for DLC

We introduce **DLC-Bench**, a benchmark designed for DLC to eliminate the need for comprehensive reference captions. The core intuition behind DLC-Bench is that an ideal description should be rich in relevant details while strictly avoiding factual errors or information for irrelevant regions. Therefore, we assess predictions based on a set of predefined positive and negative attributes for each region.

As illustrated in Figure 5.3, the evaluation process for a model like DAM has two steps:

1. The model is prompted to generate a detailed description for each masked region in the benchmark dataset.
2. An LLM serves as a judge, assessing the generated description by responding to a set of manually curated positive and negative questions about region details.

DLC-Bench employs two categories of questions for each annotated instance:

- **Positive questions** focus on specific attributes of object parts that *should* be present in the description. The model earns a point if the description accurately includes the specified detail; omissions receive no points, while factual errors incur a penalty.
- **Negative questions** focus on details that *should not* be present—either attributes typical of similar objects but absent in the target instance, or descriptions irrelevant to the specified region. A point is awarded if the model correctly omits such details; conversely, including them results in a penalty. To avoid getting high scores for captions that are completely off, point could only be awarded if the caption has the correct recognition of the object.

This approach provides a more flexible and accurate evaluation, encouraging models to generate informative and precise descriptions without constraints from incomplete reference captions.

Our DLC-Bench comprises a total of 892 manually verified questions covering a wide range of attributes and potential cases for hallucinations.

5.6 Results

DAM excels at *localized image and video captioning* across *multiple granularities* including keyword, phrase, and detailed captions, **achieving SOTA on 7 in-domain and zero-shot benchmarks** (Tables 5.2 to 5.7). We evaluate across image, video, and multi-view detailed localized captioning benchmarks.

Quantitative Results

Open-class keyword-level localized captioning task requires the model to output keywords containing the object and part entities to describe the region. Tested on object-level LVIS [161] and part-level PACO [162] datasets in Table 5.2, our method achieves state-of-the-art performance. In the PACO benchmark, a challenging benchmark that includes both full objects and parts in complex scenes and requires the model to decide whether the region is an object or a part, our method achieves 73.2% semantic IoU and 84.2% semantic similarity, outperforming the previous best by 23.2% and 8.5% respectively.

Phrase-level localized captioning task requires the model to output a phrase containing a brief description for each region that includes object identification and attributes typically within a few words. Tested zero-shot on Flickr30k Entities [163], our model achieves strong performance, outperforming the previous best by 12.3% relative improvement on an average of 5 metrics.

Detailed localized captioning task requires the model to output a detailed description for each the region with the length spanning from a long sentence to multiple sentences. An ideal description includes the description of the object in the region, its parts, as well as their attributes and relationships. We benchmark this capability on the challenging Ref-L4 [164] benchmark and our proposed DLC-Bench. On the Ref-L4 benchmark in Table 5.4, our method achieves 33.4% relative improvement on average over the previous best on short language-based captioning metrics [115, 116, 117, 118, 119], and 13.1% relative improvement on the long language-based captioning metrics [166].

We also benchmark various regional captioning models on **our proposed DLC-Bench**, which does not suffer from the limitations of requiring reference captions in previous benchmarks. As shown in Table 5.5, our Describe Anything Model (DAM) significantly outperforms existing general and region-specific VLMs, achieving state-of-the-art positive and negative accuracy and demonstrating its ability to produce detailed and accurate descriptions. Remarkably, DAM surpasses GPT-4o [100] and o1 [126], two strong API-only baselines. DAM also surpasses models with thinking mode enabled [126, 167, 168].

Detailed localized video captioning requires the model to output a detailed description for each region in a video. We benchmark this capability on the challenging HC-STVG [169] benchmark and the detailed captioning benchmark proposed by VideoRefer [120]. In Table 5.6, our proposed DAM achieves 19.8% relative improvement over the previous best on HC-STVG, including concurrent work VideoRefer [120]. In Table 5.7, the benchmark proposed by concurrent work VideoRefer [120], our proposed DAM surpasses the previous best in *both zero-shot and in-domain settings*, where zero-shot indicates not being trained on in-domain datasets derived from Panda-70M [170], which the benchmark also sources videos from.

Finally, we analyzed the performance of DAM in HD (hallucination detection) sub-task and found that DAM often predicts correct details not present in the reference caption. This indicates that the lower zero-shot performance on this sub-task is *not necessarily due to the hallucination of our model* but rather due to the missing details in the reference caption. This further illustrates the limitations of reference-caption-based evaluation.

Method	#Params	Pos (%)	Neg (%)	Avg (%)
<i>General VLMs:</i>				
GPT-4o [100]	-	43.4	<u>79.6</u>	61.5
o1 [126] [†]	-	<u>46.3</u>	78.8	<u>62.5</u>
Claude 3.7 Sonnet [167] [†]	-	21.8	50.4	36.1
Gemini 2.5 Pro [171, 168] [†]	-	36.5	75.2	55.8
Llama-3.2 Vision [130]	11B	30.7	63.8	47.3
VILA1.5-Llama-3 [140]	8B	22.5	61.0	41.8
InternVL2.5 [139, 172, 173]	8B	15.9	42.0	28.9
LLaVA v1.6 [131, 132, 133]	7B	15.4	55.0	35.2
Qwen2.5-VL [138, 174]	7B	20.3	62.2	41.2
VILA1.5 [140]	3B	16.0	50.0	33.0
<i>Region-specific VLMs (full / cropped input):</i>				
GPT4RoI [106]	7B	6.5/3.5	46.2/52.0	26.3/27.7
Shikra [105]	7B	2.7/8.0	41.8/51.4	22.2/29.7
Ferret [107]	7B	6.4/14.2	38.4/46.8	22.4/30.5
RegionGPT [114]	7B	13.0/10.6	41.4/46.4	27.2/28.5
ControlCap [103]	0.3B	18.3/ 3.6	75.6/53.6	47.0/28.6
SCA [104]	3B	3.4/ 0.1	44.6/18.4	24.0/ 9.3
OMG-LLaVA [109]	7B	0.9/ 5.6	16.0/32.6	8.5/19.1
VP-SPHINX [110]	13B	11.7/26.3	33.2/71.6	22.5/49.0
DAM (Ours)	3B	52.3	82.2	67.3

Table 5.5: **Accuracies on detailed localized captioning in our proposed DLC-Bench.** DAM outperforms previous API-only models, open-source models, and region-specific VLMs on detailed localized captioning. Underlined: the second-best method. †: models with thinking mode.

Qualitative Results

Qualitative comparisons in Figure 5.4 show that DAM excels in both accuracy and the level of details.

Detailed Localized Video Captioning. DAM describes user-specified objects in videos with localization from any frame. As shown in Figure 5.5 (a), DAM effectively describes objects under challenging conditions, such as motion and occlusion.

Method	BLEU@4	METEOR	ROUGE-L	CIDEr	SPICE
Osprey-7B [108]	0.7	12.0	18.0	1.2	15.6
Ferret-13B [107]	0.5	10.2	17.0	1.2	11.2
Shikra-7B [105]	1.3	11.5	19.3	3.1	13.6
Merlin-7B [149]	3.3	11.3	26.0	10.5	20.1
Artemis-7B [150]	15.5	18.0	40.8	53.2	25.4
VideoRefer-7B [120]	16.5	18.7	42.4	68.6	28.3
DAM-8B (Ours)	19.8	21.0	45.9	91.0	31.4

Table 5.6: **Detailed localized *video* captioning** on HC-STVG [169].

Controlling Description Granularity. As shown in Figure 5.6, DAM allows control over the amount of details and length of descriptions with different prompts.

Zero-shot 3D Object Captioning. Our model can also describe objects in multi-view datasets such as Co3Dv2 [175], integrating information from multiple frames to provide coherent descriptions of 3D objects (Figure 5.5(b)).

Ablations

Visual Prompting. We analyze different prompting strategies and find that both localized inputs and contextual information are crucial for accurate descriptions. Using only the full image limits focus on specific regions (48.7%), while local crops improve detail but lose context (60.1%). Simply concatenating both performs poorly (42.4%). Adding cross-attention significantly improves performance (63.2%), and using focal crops further enhances results (65.4%). Our best approach, the **focal prompt**, integrates focal crops with cross-attention, achieving **67.3% accuracy** without increasing sequence length for the LLM.

Data Scaling. Expanding supervised datasets boosts performance, demonstrating the value of diverse regional data. Incorporating **semi-supervised learning (SSL)** with 10% of unannotated SA-1B images further improves accuracy to **67.3%**, showcasing our data pipeline’s scalability.

We also perform additional ablations, including training a model with the architecture from prior work on our data, comparing image-only vs image-video joint training, and ablating prompt augmentations.

5.7 Discussions and Conclusion

We introduced DAM, a model for detailed localized captioning in images and videos, balancing local detail and global context through a focal prompt and localized vision backbone. We developed DLC-SDP, an SSL data pipeline leveraging segmentation datasets and unlabeled

Method	SC	AD	TD	HD†	Avg.
<i>Zero-shot:</i>					
Qwen2-VL-7B [138]	3.30	2.54	2.22	2.12	2.55
InternVL2-26B [139]	4.08	3.35	3.08	2.28	3.20
GPT-4o-mini [100]	3.89	3.18	2.62	2.50	3.05
GPT-4o [100]	4.15	3.31	3.11	2.43	3.25
Osprey-7B [108]	3.30	2.66	2.10	1.58	2.41
Ferret-7B [107]	3.20	2.38	1.97	1.38	2.23
Elysium-7B [176]	2.35	0.30	0.02	3.59	1.57
Artemis-7B [150]	3.42	1.34	1.39	2.90	2.26
DAM-8B (Ours)	4.45	3.30	3.03	2.58	3.34
<i>In-domain*:</i>					
VideoRefer-7B [120]	4.44	3.27	3.10	3.04	3.46
DAM-8B (Ours)	4.69	3.61	3.34	3.09	3.68

Table 5.7: **Performance on detailed localized video description on VideoRefer-Bench-D [120].** †: We provide analysis on hallucination scores (HD). *: trained on in-domain VideoRefer-700k with regard to VideoRefer-Bench, both sourcing videos from Panda-70M [170].

Prompting	XAttn	#IT	Pos (%)	Neg (%)	Avg (%)
Full Image Only	No	196	32.1	65.4	48.7
Local Crop Only	No	196	43.5	76.6	60.1 (+11.4)
Full + Local Crop	No*	392	26.3	58.6	42.4 (-6.3)
Full + Local Crop	Yes	196	45.7	80.6	63.2 (+14.5)
Focal Crop Only	No	196	47.3	83.6	65.4 (+16.7)
Full + Focal Crop	Yes	196	52.3	82.2	67.3 (+18.6)

Table 5.8: **Ablation studies across different visual prompts, cross-attention settings, and the number of image tokens.** Local crop denotes cropping without surrounding context. XAttn: cross-attention. #IT: number of image tokens. * indicates the full image and the crop are concatenated on the sequence dimension. **Bold:** Our proposed **focal prompt**.

Data	# regions	Pos (%)	Neg (%)	Avg (%)
LVIS	373k	34.0	72.6	53.3
+ additional datasets	602k	47.5	80.0	63.8 (+10.5)
+ SSL on 10% of SA-1B	1.38M	52.3	82.2	67.3 (+14.0)

Table 5.9: **Our model benefits from diverse datasets generated by DLC-SDP.** Scaling the dataset in size and diversity significantly improves model performance, and SSL further enhances performance using widely available unannotated images.

web images for high-quality captions. We also proposed DLC-Bench, a benchmark using attribute-based evaluation to overcome limitations of reference-based scoring. DAM achieves SOTA in 7 benchmarks in multi-granular regional captioning.

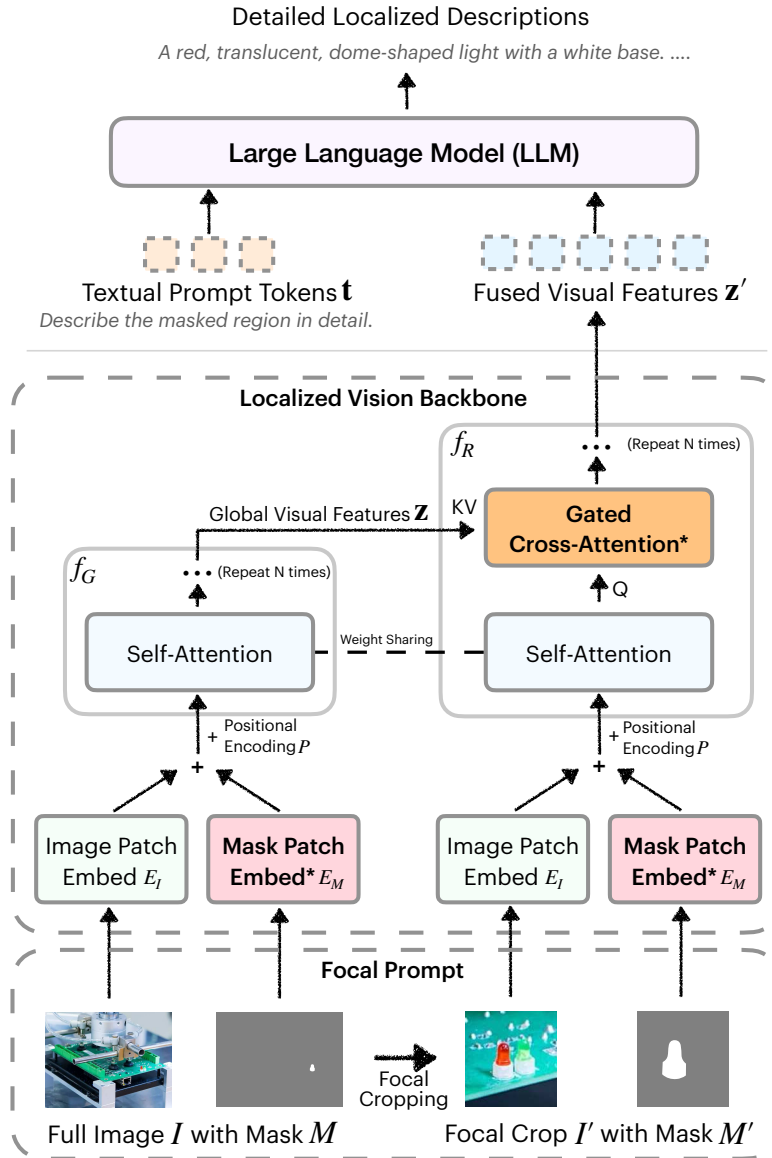


Figure 5.2: **Architecture of the Describe Anything Model (DAM)**. DAM employs a *focal prompt* to encode user-specified regions with high token density while preserving context for detailed understanding. Focal cropping is applied to the image and its corresponding mask, retaining surrounding areas for local context. Both the full image and the focal crop are the inputs into the *localized vision backbone*, where images and binary masks are embedded in a spatially aligned fashion. Global context from the full image is leveraged to help understand the focal crop through gated cross-attention. The resulting visual features and prompt tokens are fed into a large language model to generate detailed, context-aware descriptions. * indicates initialized to output zeros.

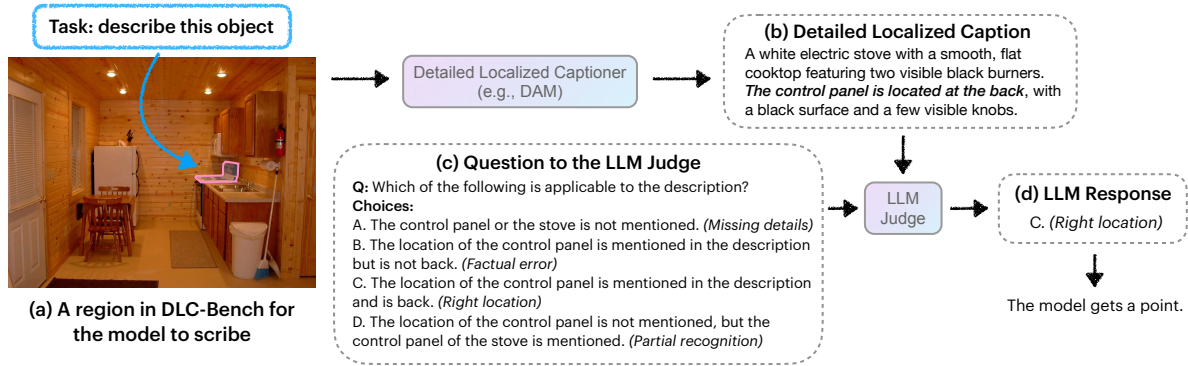


Figure 5.3: **We propose DLC-Bench, a benchmark tailored to detailed localized captioning.** In DLC-Bench, a captioning model is prompted to describe a specified image region (a). The generated description (b) is then evaluated by querying an LLM Judge (c). Points are assigned or deducted based on the LLM’s response (d). The question we show in (c) is an example of positive questions.

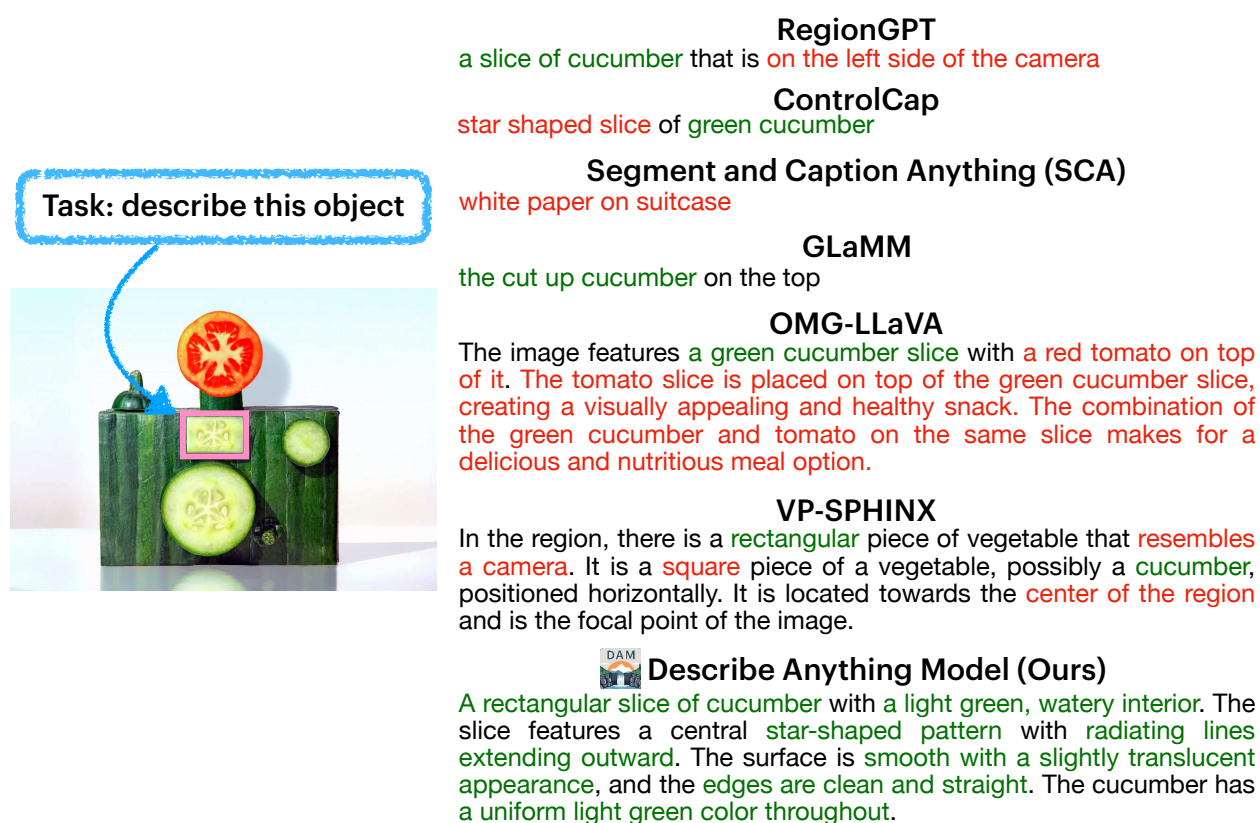
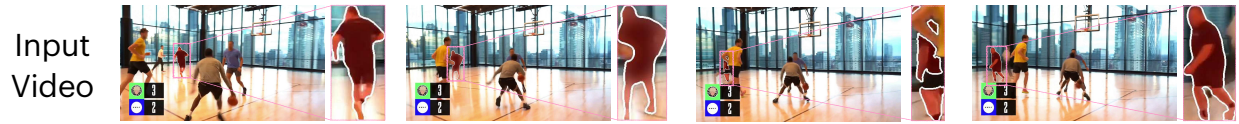


Figure 5.4: DAM generates detailed and localized descriptions, whereas prior works generate descriptions that are less precise. **Green**: correct description. **Red**: factual error or mislocalization.



A person wearing a **black shirt and dark shorts** is captured in a dynamic sequence of movement. Initially, the individual appears to be in a **running** motion, with their **body slightly leaning forward**, suggesting speed and agility. The **arms are bent at the elbows**, and the legs are positioned in a way that indicates a **swift, forward stride**. As the sequence progresses, the person maintains a consistent pace, with their legs alternating in a rhythmic pattern, indicative of a **sprinting** action. The posture remains upright, and the **head is slightly tilted forward**, emphasizing focus and determination. The overall movement conveys a sense of urgency and athleticism, as the person continues to move swiftly across the scene.

(a) DAM reliably describes user-specified objects in videos, even under strong camera and object motion and heavy occlusion.

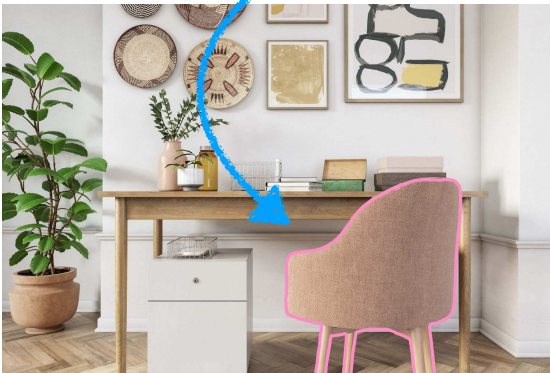


The car is a **blue SUV with a boxy design**, featuring a **spare tire** mounted on the rear door. It has a **four-door configuration** with **black side mirrors and tinted windows**. The vehicle is equipped with **silver alloy wheels** and black trim around the wheel arches. The front of the car includes a **rectangular grille** and **rectangular headlights**. The side of the car displays a **horizontal silver stripe** running along the length of the doors. The rear of the car has **vertical taillights** and a **visible license plate area**. The car's body is **slightly worn**, indicating use over time.

(b) DAM can describe objects in multi-view datasets, like this car from Co3Dv2 [175], by integrating information from multiple frames.

Figure 5.5: DAM accurately describes user-specified regions in *videos* and *multi-view scenes* under challenging conditions.

Task: describe this object



Prompt: Provide a **brief** description of the masked region.

A modern chair with a curved backrest and light wood legs.

Prompt: Provide a **highly detailed** description of the masked region.

A modern chair with a curved backrest and a textured, light brown fabric upholstery. The chair features a smooth, rounded top edge and a slightly tapered back. It is supported by four slender, light-colored wooden legs that are slightly splayed for stability.

Figure 5.6: DAM offers multi-granular localized descriptions.

Chapter 6

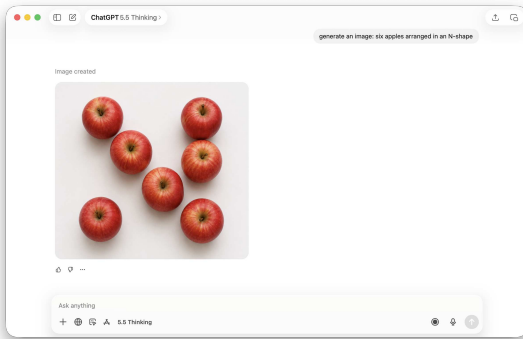
Relevance of Language-Guided Visual Decomposition

6.1 Why Explicit Visual Structure Still Matters

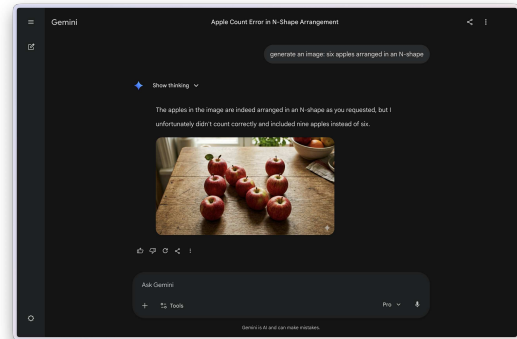
The visual generation methods in Chapters 2–4 were developed during a period when text-to-image and text-to-video diffusion models were rapidly improving but still struggled with compositional prompts. Since then, image and video generators have become substantially stronger. They can often produce realistic scenes, follow broad style instructions, and synthesize visually compelling motion without the explicit layout interfaces used in LLM-grounded Diffusion, LLM-grounded Video Diffusion, or Self-correcting LLM-controlled Diffusion.

This progress changes the role of these methods, but it does not make the underlying problem obsolete. The central issue studied in these chapters is not whether a model can generate a plausible image or video from text. It is whether the generation process can reliably satisfy structured constraints as instructed by the user: object counts, spatial relations, attribute binding, temporal ordering, and user-specified edits. Existing direct prompting interfaces still do not provide a reliable way to ensure that such constraints are represented, preserved, and checked throughout generation. A prompt such as “six apples arranged in an N-shape” can fail in ways that are visually subtle but semantically important: objects may be swapped, missing, duplicated, incorrectly colored, or placed in the wrong relation. Video generation adds another layer of difficulty because the same entities and relations must remain coherent across time.

Explicit decomposition remains useful because it separates what should be generated from how it should look. A layout, scene graph, bounding-box trajectory, mask, or correction plan gives the system an intermediate representation that can be inspected, edited, verified, and reused. Even when a modern generator can satisfy many prompts directly, explicit structure provides a control surface for cases where direct prompting is ambiguous or unreliable. This is especially important for applications that require correctness rather than only aesthetic



(a) GPT-image-2



(b) Nano Banana 2

Figure 6.1: **Current image generators are still not guaranteed to preserve user-specified spatial and counting constraints.** Given the prompt “generate an image: six apples arranged in an N-shape,” GPT-image-2 produces a plausible image but does not reliably satisfy the requested N-shaped arrangement, while Nano Banana 2 acknowledges that the result should contain six apples in an N-shape but generates the wrong number of apples. These examples illustrate that current systems can recognize the intended constraint while still failing to enforce it reliably in the visual output.

quality: instructional diagrams, robotics simulation, spatial reasoning data, visual editing, synthetic training data, and evaluation suites for multimodal models.

6.2 A Continuing Role for LMD, LVD, and SLD

LLM-grounded Diffusion, LLM-grounded Video Diffusion, and Self-correcting LLM-controlled Diffusion should be read as examples of a broader design pattern. They use language models not merely as prompt expanders, but as planners and controllers that expose latent structure in a visual task. LMD provides a concrete way to make image generation more reliable by turning a user instruction into an explicit spatial layout before synthesis. LVD extends the same idea to video by representing objects and their motion over time. SLD adds closed-loop verification and correction: after a model generates an image, the system checks whether the result matches the instruction and repairs the failure when possible. The particular base generators used in these chapters can be replaced by newer ones, but the decomposition remains relevant whenever a generator benefits from external planning, grounding, or verification.

This perspective is compatible with stronger end-to-end models. As generators improve, explicit planning may be needed less often for simple prompts, but it becomes more valuable for harder prompts, high-stakes use cases, and interactive workflows. A user may want to lock the position of an object, preserve an identity while changing an attribute, specify

motion over time, or repair only the part of an image that violates the instruction. These are system-level requirements, not just model-level capabilities. Decomposition offers a way to build generation pipelines whose behavior is more transparent and controllable than a single direct prompt-to-output call.

The same argument applies to evaluation. Modern generators can make failure cases harder to notice because the outputs often look polished even when they are semantically wrong. Explicit intermediate representations make it easier to ask whether the right entities exist, whether they are in the right places, and whether temporal relations are preserved. This makes the methods in Part I relevant not only as generation techniques, but also as tools for analyzing and stress-testing visual generators.

6.3 Why Describe Anything Remains Useful

Describe Anything addresses a complementary problem: how to produce detailed localized descriptions of images and videos. This task remains useful even as general vision-language models improve, because many applications require descriptions that are grounded in a specific region rather than in the whole image. A global caption may say that an image contains a street scene, but a user, dataset builder, or downstream model may need to know the fine-grained appearance, state, and context of one selected car, sign, person, tool, or action.

Localized description is valuable for data generation. Systems such as LMD and LVD need rich supervision about objects, attributes, spatial relations, and temporal behavior. Human-written captions are expensive and often underspecify the visual details needed to train or evaluate controllable generation. A model that can describe selected regions in detail can turn existing images and videos into structured synthetic data: object-level captions, region-level attributes, localized action descriptions, and examples of spatial relations grounded in real visual evidence.

The task is also useful beyond training data. Detailed localized descriptions can support accessibility for visually impaired people, interactive visual search, content moderation, robotics, medical or scientific annotation workflows, and visual editing tools. In these settings, the user often cares about one part of a scene and needs a description that preserves both local detail and global context. Describe Anything is designed around exactly this tension: it focuses on the selected region without discarding the surrounding scene that gives the region meaning.

For this reason, Describe Anything should not be viewed only as an auxiliary data generator for earlier diffusion systems. It is a standalone visual understanding task with continuing practical value. As image and video generation systems become more capable, localized captioning becomes more important, not less: stronger generators and editors need better localized supervision, better localized evaluation, and better interfaces for users to specify what part of a visual scene they care about.

6.4 Part I in Retrospect

The common thread across Part I is that visual intelligence benefits from explicit interfaces between language, vision, and generation. LMD, LVD, and SLD show how language models can plan, control, and correct visual synthesis. Describe Anything shows how vision-language models can produce the localized descriptions needed to supervise and evaluate such systems. The individual models and backbones will continue to change, but the underlying questions remain current: how should a system represent the user’s intent, how can that representation be grounded in visual structure, and how can the system verify that the final output satisfies the intended relations?

Those questions become more important as visual generators are deployed in real workflows. When the goal is only to produce an appealing image, direct prompting may be enough. When the goal is to produce a correct image, controllable video, editable scene, reliable dataset, or accessible visual description, decomposition remains a practical design principle.

Part II

Adaptive Parallel Reasoning

Chapter 7

Learning Adaptive Parallel Reasoning with Language Models

This chapter is based on the following publication:

Jiayi Pan*, Xiuyu Li*, Long Lian*, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. “Learning Adaptive Parallel Reasoning with Language Models.” *Conference on Language Modeling (COLM)*, 2025.
(*Equal contribution.)

7.1 Introduction

Recent progress in language model reasoning, like OpenAI o1 [126] and DeepSeek-R1 [177], has demonstrated the promise of exploiting test-time compute to perform search and of using reinforcement learning to optimize the search. However, these current approaches face fundamental limitations: serialized chain-of-thought methods [177] produce lengthy output sequences that increase latency and strain context window limits, while parallel methods like best-of-N or self-consistency [178] often lack coordination between inference paths and are not end-to-end optimized, leading to redundant computation and limiting improvement. Structured inference-time search methods like tree-of-thought [179] require hand-designed search structures, limiting their flexibility and scalability.

We propose Adaptive Parallel Reasoning (APR), a simple yet effective approach that enables language models to reason by adaptively distributing inference-time computation in a manner that exploits both serial and parallel operations. Our method generalizes existing approaches to reasoning with language models, including serialized chain-of-thought reasoning, parallelized inference with self-consistency, and structured search: rather than imposing fixed search structures through prompting or external orchestration, we train language models to *learn* when and how to parallelize their inference operations.

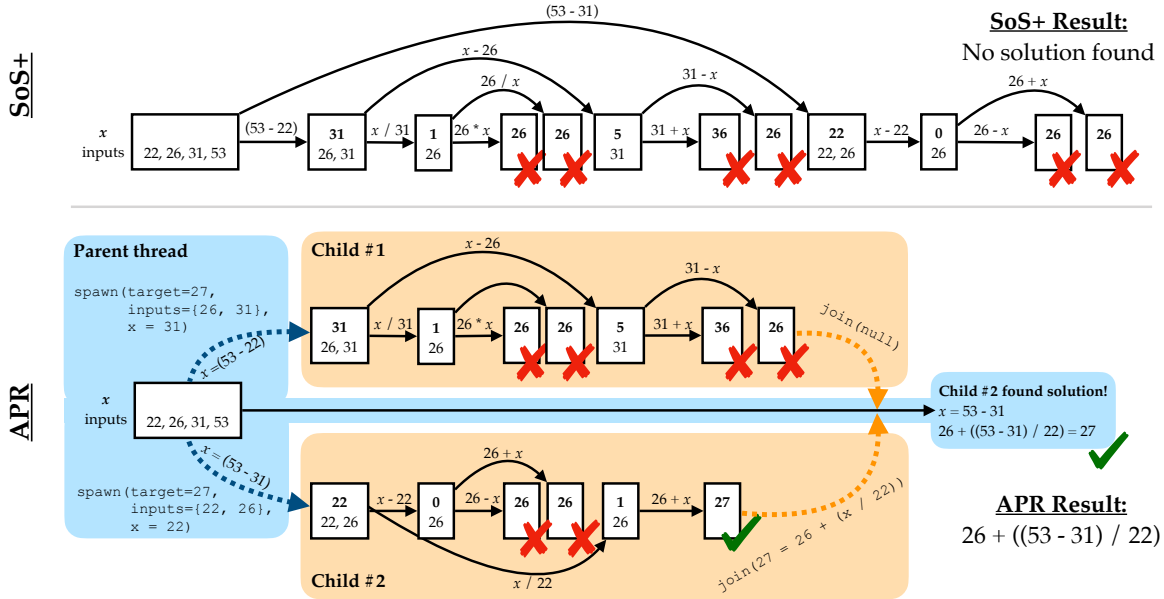


Figure 7.1: **Serialized search [4] (top) vs Adaptive Parallel Reasoning (bottom)** illustrated on an example of the Countdown task, with a target number of 27 and input numbers of {22, 26, 31, 53}. Each box represents a node in the search representing the value of the intermediate expression (x). Edges are annotated with explored arithmetic operations relative to the parent-node x using remaining input numbers. In serialized search, the context window of the single inference thread is exhausted before a solution is found. In Adaptive Parallel Reasoning, the parent thread (blue) spawns two child threads (orange), which are executed in parallel. Child threads have access only to a limited context passed to them by the parent thread and return a summary of their execution to the parent thread. The parent thread can then continue to decode with access to these summaries. This parallel distribution of computation prevents context window exhaustion while reducing latency.

Adaptive Parallel Reasoning employs two key innovations: First, we supply language models with a parent-child threading mechanism. Parent inference threads can, at any point during decoding, delegate subtasks to multiple child inference threads using a `spawn()` operation. Child threads independently explore distinct reasoning paths in parallel and return outcomes to the parent thread through a `join()` operation. The parent thread then continues decoding conditioned on the information returned by the children. We build on the model serving framework SGLang [180] to perform inference in child threads simultaneously through batching, which significantly reduces real-time latency. Our second key innovation is to fine-tune the language model that performs inference in both parent and child threads via end-to-end reinforcement learning, which optimizes overall task success and eliminates the need to predefine explicit reasoning structure.

Figure 7.1 illustrates how APR facilitates more efficient and effective reasoning in the

Countdown reasoning task [179, 4, 181] when compared to serialized methods. Our experiments demonstrate that we achieve three key benefits over prior approaches:

1. **Higher performance within same context window:** Our approach performs more effective search and reasoning within fixed context size constraints (83.4 vs 60.0% at 4k context) compared to sequential methods that quickly exhaust available context.

2. **Superior scaling behavior:** When scaling the total compute budget, APR exhibits better performance improvements (80.1 vs 66.6% with a budget of 20k total tokens) through wider parallelization in addition to increasing the length of individual reasoning chains.

3. **Improved performance at the same latency:** Adaptive Parallel Reasoning achieves significantly higher success rates compared to serialized search methods with same latency (75.2 vs 57.3% at around 5,000ms).

These results highlight the potential of training language models to adaptively allocate their own inference-time compute resources. By learning when to reason serially and when to branch out into parallel computation, models can more efficiently explore the solution space of complex reasoning problems.

7.2 Related Work

Inference-time scaling Prior work has shown that increasing test-time compute can improve language model performance on downstream tasks [182, 183, 184, 177, 126, 185, 4, 186, 187]. However, these methods typically result in significantly longer output sequences, introducing key limitations given the inherently sequential nature of autoregressive decoding: longer output sequences mean higher latency, and fitting an entire sequence into a single context window makes it hard for models to attend to relevant information when scaled with more output tokens. Compared to serialized inference, parallelizing reasoning traces reduces both latency and the pressure of context window limitations. Our experiments demonstrate that models trained end-to-end to adaptively distribute inference-time compute can outperform serialized approaches under the same computational budgets.

Parallelization in language model inference Parallelizing inference with multiple independent language model calls has emerged as another avenue for inference-time scaling towards improved reasoning performance [188, 178]. While these ensembling methods effectively enhance performance, their lack of coordination across parallel threads results in redundant computation and suboptimal resource utilization. Recent methods have aimed to address this limitation by proposing fixed parallelizable reasoning structures [179, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198] often without learning. However, these proposed search structures (and, in the case of multi-agent reasoning, the communication protocol) are fixed and hand-designed, limiting their flexibility and scalability. In contrast, our approach leverages reinforcement learning to train language models to adaptively structure search at inference time, dynamically allocating compute between parallel and serial operations. Concurrent to our work, PASTA [199] and Hogwild! Inference [200] also explore parallel

reasoning. However, PASTA decomposes tasks into parallel sub-tasks but ultimately merges the complete context from each sub-task back into the main inference trajectory, thus not effectively reducing context usage. Meanwhile, Hogwild! Inference employs parallel worker threads for collaborative reasoning, yet it relies exclusively on prompting without any end-to-end optimization. In contrast, our method uniquely integrates supervised training and reinforcement learning, enabling language models to adaptively and efficiently manage parallel explorations and selectively integrate successful outcomes, maximizing reasoning performance within constrained context windows.

Many parallel inference methods have shared prefixes across language model calls, which synergize with modern serving framework to improve serving efficiency via prefix caching and reuse [180]. APR builds on these optimizations, as parent-child threading mechanism creates natural opportunities for prefix sharing, which further reduces the computational overhead of APR parallelization.

Training language models to control their own outputs Our work is related to algorithms for training language models to directly control the inference process, for both efficiency and performance reasons. For example, PENCIL [201] trains models to adaptively discard parts of their context at inference time to reduce the context length, and thus the time complexity of producing a successful reasoning trace. [202] train language models to inject additional “pause” tokens into their output sequences, which are shown to improve end-task performance by increasing inference-time computation. Both approaches train using supervised demonstrations of token discarding (or injection). To the best of our knowledge, we are the first to use end-to-end reinforcement learning to optimize language model control over the allocation of inference compute.

7.3 Adaptive Parallel Reasoning

We now introduce Adaptive Parallel Reasoning (APR), which teaches language models to adaptively orchestrate parallel inference processes. By distributing compute across parent and parallel child threads, APR minimizes the end-to-end latency, achieves better performance within the same context limit constraint, and scales better with an increasing inference-time compute budget. We first describe our task setup, followed by our novel multi-thread setup for parallel search at inference time, and finally the corresponding optimization procedure that combines supervised training and RL fine-tuning.

Preliminaries

Stream of Search [4] propose Stream of Search (SoS), which serializes search traces for reasoning tasks as natural language strings; they train language models to generate SoS strings through supervised training and then further optimize the model through policy improvement algorithms, including STaR [183] and APA [184]. Experiments show this

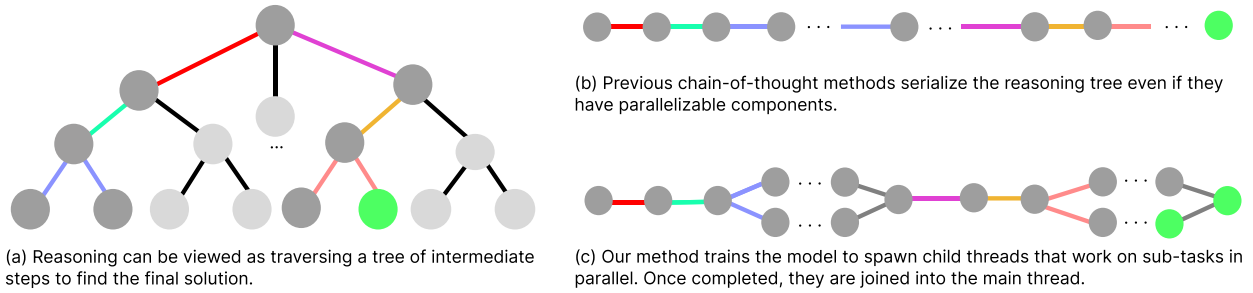


Figure 7.2: **Overview of Adaptive Parallel Reasoning (APR).** While previous chain-of-thought methods directly linearize the reasoning tree, Adaptive Parallel Reasoning alternates between the parent thread and parallel child threads to traverse the reasoning tree more efficiently.

approach allows models to search effectively and to self-improve by adapting and discovering new search and reasoning strategies. However, the serialized reasoning traces of SoS are naturally length-constrained by the context window, and can lead to high latency during inference due to requiring sequential, token-by-token generation. Our approach addresses both limitations by training language models to adaptively parallelize their search at inference time.

Task: Countdown Following existing work that prototypes language model reasoning algorithms [179, 4, 181], we perform experiments on the Countdown task, where a model must map from a set of four numbers to an arithmetic expression that uses each number exactly once and whose value exactly matches a given target (e.g., given $\{1, 4, 6, 8\}$ and target 10, one valid solution is $(8 - 6) \times (4 + 1) = 10$).

Multi-Threading at Inference Time

Inspired by the multi-threading functionality of operating systems, where a process distributes computation concurrently across multiple CPU cores, our framework outlines a generic approach to parallel inference using multi-threaded model execution. At inference time with Adaptive Parallel Reasoning (APR), reasoning proceeds via an adaptively orchestrated hierarchy of parent-child threads. An example trajectory is illustrated in Figure 7.3.

Spawning child threads with `spawn()` Existing reasoning methods rely on search structure provided by model developers either through prompting [e.g., 203] or through external orchestration and synthesis of inference calls [e.g., 178]. In contrast, in Adaptive Parallel Reasoning, search structure is determined at inference time entirely by the model itself. To support this, we provide the model access to a `spawn(msgs)` operation, which, when

selected during decoding, will spawn multiple child threads that are executed in parallel.¹ `spawn()`’s only argument, `msgs`, is a list of strings, each corresponding to a child node to spawn and the context it is passed. Coordination between child threads is achieved by having the parent thread pass each child a distinct context.

Child thread inference Child threads independently yet simultaneously execute inference using the same language model as the parent thread, with each thread’s context limited to the tokens passed to it by the parent thread in `msgs`. Similar to other approaches like self-consistency [178], parallelization in our approach affords diversification across reasoning traces. However, because child threads may be conditioned on different contexts provided by the parent thread, they can execute distinct reasoning subtasks and thus avoid the problem of lack of coordination as in independent inference methods like self-consistency.

When a child thread generates the `join(msg)` operation, it terminates its inference, specifying a sequence of tokens to return to the parent thread as `msg`. Critically, child threads have control over what they return to the parent thread. In the Countdown task, child threads discard intermediate traces and return only the successful solution path, enabling concise and targeted communication back to the parent. No solution paths are returned for child threads with failed searches.

Synthesis of child threads after `join()` Once all child threads have terminated and returned corresponding messages, the parent thread execution continues, conditioned only on its previous context before spawning the child threads and messages returned by the child threads. Keeping intermediate search traces only to the child threads reduces the parent thread’s token usage and thus the overall computational cost of inference.

Training Models to Adaptively Parallelize their Reasoning

Enabling language models to effectively use the `spawn()` and `join()` operations introduces two major challenges. First, pre-trained language models are simply not trained to call such operations at inference time.² Following [4], we use automatically-generated demonstrations to train a reasoning model from scratch with a supervised objective. In our case, we generate demonstrations of reasoning traces that use `spawn()` and `join()`. Second, even with supervised learning, the model is trained only to imitate demonstrations, without itself exploring the space of possible search structures which may include structures that are more computationally efficient or effective for the task. To address this, we fine-tune the pretrained model using end-to-end reinforcement learning.

¹We use the notation of a “thread” to indicate a consecutive decoding of tokens. The actual implementation depends on the serving framework.

²Following existing work that proposes new learning algorithms for reasoning [4], we focus on training reasoning models from scratch. We leave experiments on adapting language models pre-trained on general web corpus for future work.

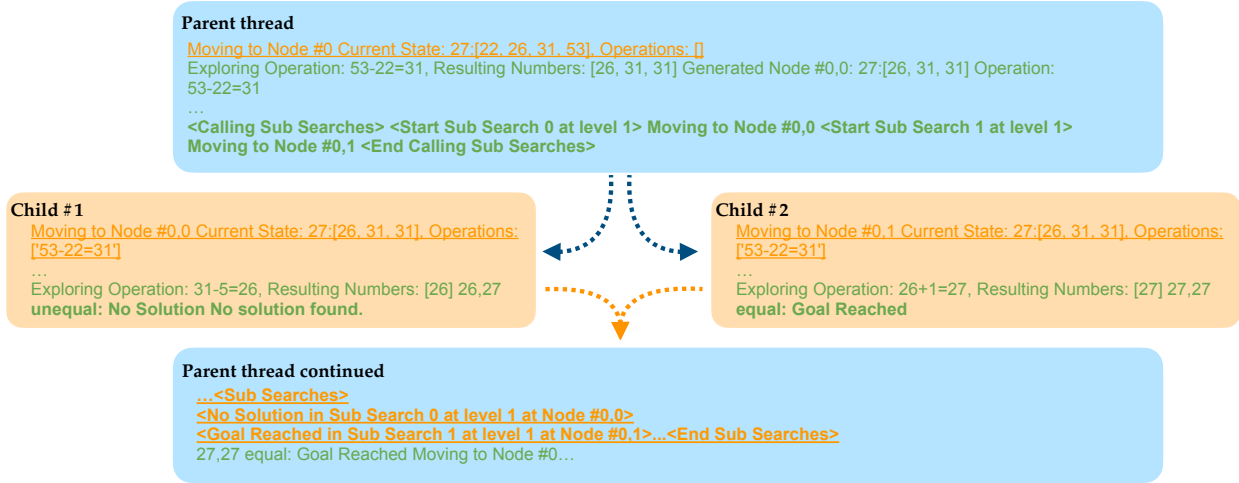


Figure 7.3: **An example trajectory of our APR method solving the Countdown task.** The model starts with the parent thread, solving the task through reasoning in language, and generates **spawn** commands to instantiate two child threads for parallel reasoning, which are **joined** back when finished, after which the main thread continues decoding. **Orange** denotes prefix input tokens, while **Green** denotes LM-generated tokens. Bold parts correspond to the **spawn** and **join** operations.

Supervised initialization of parallel reasoning We use supervised learning to train a language model from scratch to generate reasoning traces that use **spawn()** and **join()**. Following Stream of Search [4], we use a symbolic solver to generate reasoning traces. In SoS, each demonstration represents a single search strategy – either depth first search (DFS) or breadth first search (BFS). To diversify the types of search trajectories and allow for spontaneous parallelization behaviors, we instead develop *hybrid* search that includes examples of both BFS and DFS in the same search trace, which we also empirically find to have slightly better performance.

We implement two symbolic solvers: SoS+, which adapts the original SoS solver to produce serialized hybrid search paths without **spawn()** and **join()**; and APR, which creates hybrid demonstrations with parallelization.³ For the APR symbolic solver, during its execution, it selects certain nodes to explore in parallel. We delegate the search under these nodes to child LM threads, while the parent thread handles the rest.

Because APR decomposes search into multiple demonstrations, APR faces much less of a context window bottleneck than SoS+ both for training and inference, as each demonstration only includes tokens from part of the search, rather than the full search. Thus, APR can represent much more extensive searches.

³The original paper provides details and pseudocode for both symbolic solvers.

Reinforcement learning for end-to-end policy optimization While supervised training establishes a baseline understanding of parallel execution, it merely guides the model to imitate the provided demonstrations, and does not optimize computational efficiency or reasoning effectiveness. We further fine-tune the language model with fully end-to-end reinforcement learning (RL). During RL-based finetuning, we iteratively sample reasoning traces from our current policy, assign them a reward according to the correctness of the proposed solution, and optimize policy parameters with GRPO [204]. In this stage, the model learns to strategically determine when, how, and how broadly to invoke child threads, maximizing performance by balancing the trade-offs between parallel exploration and the context window constraint.

7.4 Experiments

We evaluate the effectiveness and efficiency of Adaptive Parallel Reasoning compared to serialized chain-of-thought reasoning. We first demonstrate that Adaptive Parallel Reasoning effectively trains models to distribute inference-time computation across multiple inference threads, none of which require the full search context, resulting in a more efficient use of a model’s limited context window, and thus supporting scaling to higher compute. We show that under the same latency and context window constraints, Adaptive Parallel Reasoning achieves superior performance. We also find that end-to-end reinforcement learning significantly enhances the effectiveness of Adaptive Parallel Reasoning, and that when tasked to optimize the accuracy end-to-end, RL results in wider, in addition to deeper searches, indicating the effectiveness of adding another dimension of scaling to the method.

Experiment setup Throughout our experiments, we use a standard decoder-only language model trained from scratch with the Llama2 architecture [205]. The model employs the Llama2 tokenizer, contains 228M non-embedding parameters, and supports a 4,096-token context window. All models are initialized via supervised learning on 500k trajectories generated using Countdown symbolic solvers. For our baseline, we use trajectories from the SoS+ solver, while for our method we use trajectories from the APR symbolic solver.

To directly measure the compute-accuracy trade-off, we adopt a budget constraint method similar to Chen et al. [206], training length-controlled models by conditioning on context-window sizes for SoS+ models. Specifically, we partition training sample lengths into bins of 512 tokens, using these bin sizes as conditioning signals during training. At inference time, specifying a bin size allows us to control the number of generated tokens. For APR models, however, conditioning on context window size is less suitable as child thread lengths can vary significantly. We condition these models on the number of child threads initiated per parent thread, which strongly correlates with the total number of tokens across all threads.

We use SGLang [180] for inference due to its high-performance and support for continuous batching and radix attention, which enables efficient running of APR.

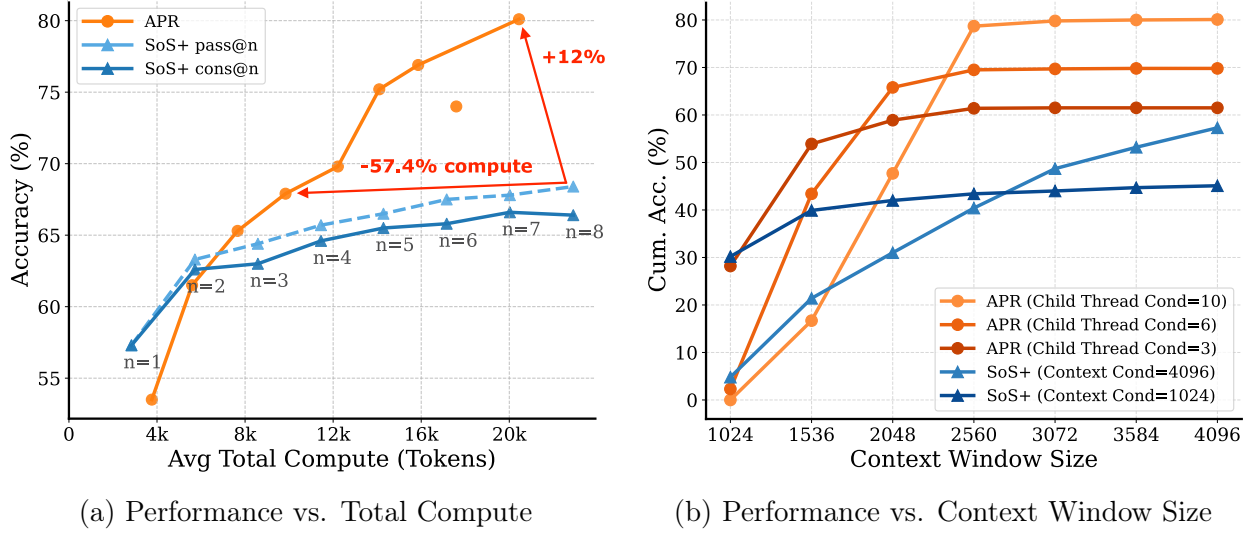


Figure 7.4: **Scaling performance of SoS+ vs APR.** (a) APR achieves higher accuracy with increasing compute budget compared to SoS+ with Best-of-N sampling. (b) APR more effectively utilizes fixed context windows across different thread configurations. The cumulative accuracy measures the total accuracy considering only the outputs that fall within the context window constraint—that is, only outputs no longer than the set constraint are counted as correct.

Baselines We compare our method against two baselines: search via long chain-of-thought reasoning (SoS+) and self-consistency selection (denoted **cons@n**), the standard parallel inference method for scaling test-time compute. The **cons@n** approach is implemented by independently sampling reasoning traces from SoS+, excluding outputs that fail to find a valid solution, then applying majority voting to the final search paths from the remaining outputs. The most frequently occurring solution is then selected as final answer. Additionally, we report **pass@n**, the rate at which at least one returned solution is correct, to illustrate the upper-bound performance achievable through repeated sampling with simple ensemble-based parallel inference.

Evaluation metrics We consider two categories of metrics to evaluate the model performance and efficiency. First, we measure **accuracy** as the percentage of Countdown problems in the test set successfully solved by the model. Second, we evaluate computational efficiency by measuring the **total tokens** generated during reasoning, which reflects the model’s compute usage at test time. Additionally, we evaluate **latency** using two complementary metrics: **sequential tokens**, representing the maximum number of causally dependent tokens that must be processed sequentially (i.e., parallel sub-thread generation considers only the longest token sequence among independently processable threads); and **real-world latency**, the wall-clock time required to solve a single task.

Bootstrapping Adaptive Parallel Reasoning from Supervised Training

Following Gandhi et al. [4], we pre-train our model by imitating sub-optimal search traces generated by the symbolic solver. We construct a training dataset of 500k Countdown problems with corresponding search traces using both SoS+ and APR symbolic solvers. Our results indicate that APR consistently outperforms SoS+ across multiple dimensions.

Scaling with higher compute We first analyze how allocating additional test-time compute improves performance. We measure compute by total tokens generated during reasoning; for SoS+, this is the length of the search trace, while for APR, it is the total token count accumulated across all parent and child calls. We scale test-time compute of APR by varying the number of conditioned child threads from 0 to 10. For SoS+, the context window size is fixed at the maximum number of 4,096 tokens. To scale compute for SoS+ beyond this window, we report $\text{cons}@n$ for sample sizes ranging from 1 to 8. We also report $\text{pass}@n$ as an upper bound. All scaling experiments use a sampling temperature of 1.0. As shown in Fig. 7.4a, we find that APR initially under-performs in low-compute regimes (below 4k tokens, or $\text{pass}@1$), which we attribute to “parallelism overhead” - some of the generated tokens are used to orchestrate threads rather than directly contribute to the search. However, as compute increases, APR significantly outperforms SoS+, achieving a 13.5% absolute improvement ($66.6\% \rightarrow 80.1\%$) over SoS+ $\text{cons}@7$ at 20k tokens. It also surpasses SoS+ $\text{pass}@8$ by 11.7% ($68.4\% \rightarrow 80.1\%$) at 24k tokens. Notably, it matches the $\text{pass}@8$ performance of SoS+ while consuming 57.4% less compute. This clearly indicates that APR scales more effectively with increased compute by enabling parallel exploration via independently executed child threads. Importantly, this parallel design maintains efficiency with minimal impact on latency, as further detailed in §7.4.

Scaling with context window size We also evaluate the performance of SoS+ and APR under varying context window constraints (1k to 4k tokens). For each setting we sample once and report cumulative accuracy: at every window size we count only those traces whose length is within that limit. Budgets below 1k are omitted because neither method can produce usable solutions. To investigate performance trade-offs, APR is evaluated with 3, 6, and 10 child threads, whereas SoS+ is trained with context conditioning at 1,024 and 4,096 tokens. The 1,024-conditioned SoS+ occasionally generates traces slightly longer than 1,024 tokens; these same outputs become valid when the limit is relaxed, so its curve remains meaningful beyond the 1,024 window. As illustrated in Fig. 7.4b, APR consistently exploits context more efficiently. With only 3 child threads, it already surpasses SoS+ at every window size, and with 6 or 10 threads, it achieves around 10% and 20% higher accuracy, respectively, at the 4k-token limit. The advantage comes from distributing reasoning across parallel threads, enabling more total tokens without packing the entire trace into one context window—a limitation inherent to serialized SoS+. Overall, APR more effectively exploits fixed context budgets, yielding significantly better performance.

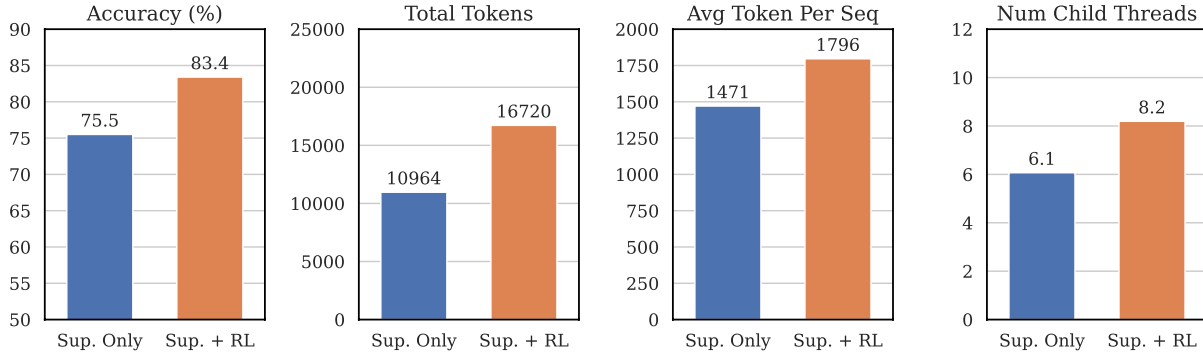


Figure 7.5: **Comparison of model performance and statistics before and after reinforcement learning (RL).** RL significantly improves performance by optimizing the APR policy beyond what can be learned from supervised imitation learning. Post-RL models strategically increase sequence length (total tokens and avg token per seq) and, more substantially, child thread count (the number of spawned child threads), demonstrating an advantage of broadening rather than deepening search in reasoning. This optimization results in substantially higher accuracy (from 75.5% to 83.4%).

End-to-end Policy Optimization through Reinforcement Learning

We employ reinforcement learning (RL) to optimize the APR policy end-to-end. As shown in Fig. 7.5, RL significantly improves model performance across different initial conditions, resulting in substantially higher accuracy (from 75.5% to 83.4%). Pre- and post-RL models exhibit markedly different behaviors. RL increases both the sequence length (from an average of 1,471 to 1,796 tokens; 22.1% relative increase) and number of child threads (from an average of 6.1 to 8.2 child threads; 34.4% relative increase). This implies that, for the Countdown task, a broader search is more optimal than a deeper one—as discovered by RL.

Efficiency of APR

We demonstrate that APR improves reasoning efficiency both theoretically and practically compared to SoS+ serialized chain-of-thought baselines (by measuring sequential token usage and real-world latency). We define sequential token usage as the average length of the longest non-parallelizable component across parent and child threads, serving as a lower bound on sequential computation. For SoS+, this directly equals the average output sequence length. Fig. 7.6 shows accuracy against sequential token usage for both methods. Results confirm that APR effectively leverages parallel exploration, significantly boosting accuracy with minimal additional sequential tokens beyond 2,048 tokens, rarely exceeding 2,500 tokens. In contrast, SoS+ achieves only marginal accuracy improvements despite quickly approaching

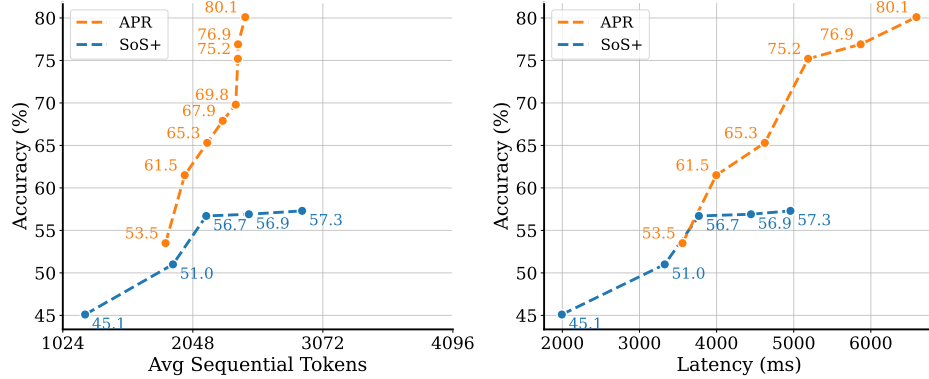


Figure 7.6: **Efficiency comparison between APR and SoS+.** Left: accuracy vs. sequential token usage. Right: accuracy vs. wall clock latency. APR consistently achieves higher accuracy with fewer tokens and lower latency, demonstrating superior efficiency.

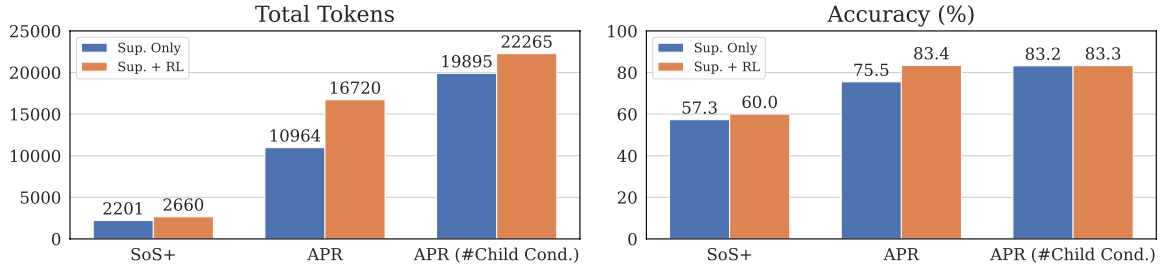


Figure 7.7: RL provides a larger performance gain to APR (7.9%) compared to SoS+ (2.7%), resulting in substantial advantages of APR over SoS+ after RL (83.4% vs 60.0%). Ablations with the number of child threads enforced show that such performance gain mainly comes from teaching the model how to allocate compute resources instead of making more optimal decisions with the same resources.

3,000 tokens, highlighting the scaling advantages of parallel exploration under constrained contexts.

Moreover, we evaluate the practical efficiency of APR by analyzing real-time latency. Inspired by the `spawn()` operation commonly used in computer systems, APR is specifically designed to leverage hardware parallelization effectively. We consider an API-based serving scenario where both SoS+ and the main call of APR initially consume the same fixed amount of compute, while subsequent child threads utilize additional computational resources concurrently. Specifically, we deploy the models on an 8-GPU NVIDIA RTX A6000 server, dedicating one GPU to handle the main inference thread, with the remaining GPUs allocated for executing child threads in parallel. We measure the latency per sample across the entire test set. Results in Fig. 7.6 demonstrate that APR achieves a substantially better accuracy-

latency trade-off compared to SoS+. Notably, at a latency of 5000ms per sample, APR reaches an accuracy of 75%, an 18% absolute improvement over SoS+’s 57%, implying the potential of APR for superior performance and efficiency in realistic deployment scenarios.

Ablation Study

RL on SoS+ vs APR As shown in Fig. 7.7, we observe that RL significantly improves the policy in APR, with a significant increase in the sequence length and the number of child threads the model uses. To evaluate RL’s impact on our baseline, we also applied GRPO to SoS+. As expected, RL increased the sequence length and improved SoS+ performance as well. However, the accuracy improvement for SoS+ (2.7%) was substantially smaller than for APR (7.9%). This difference in improvement rates correlates with changes in total token usage. While SoS+ only increased total tokens by 20.9%, APR showed a 52.5% increase. These results demonstrate APR’s effectiveness in allowing RL to scale token usage beyond context window limitations by distributing computation across multiple child threads.

Disentangling contributing factors in RL As shown in Fig. 7.7, RL significantly improves policy accuracy. However, this improvement may stem from two distinct factors: (1) test-time scaling, where RL encourages the model to use more tokens at test time and perform more extensive search (i.e., try more options); or (2) reasoning-quality improvements, where RL teaches the model to search more effectively (i.e., select options more cleverly).

To disentangle these factors, we conducted an experiment where we conditioned the model to use the maximum number of child threads (10 threads) both before and after RL training, so that it almost always uses the most test time compute possible even before RL. Our results reveal that (1) with maximum child threads enforced, we observe minimal changes in both thread count and sequence length after RL; and (2) accuracy remains nearly identical pre- and post-RL under these fixed compute conditions, from 83.2 to 83.3% accuracy. This indicates that in our experiments, the primary benefit of reinforcement learning comes from scaling test-time compute rather than improving decision quality within a fixed budget.

7.5 Conclusions, Limitations, and Future Work

We presented Adaptive Parallel Reasoning, which enables language models to adaptively distribute computation across serial and parallel reasoning paths using a parent-child threading mechanism. In this method, we employed supervised training and reinforcement learning to enable models to learn to develop parallelization strategies without manually designed structures. Supervised training establishes a baseline understanding of parallel execution, and to further optimize the effectiveness of reasoning, we fine-tune the language model with fully end-to-end reinforcement learning.

Experiments on the Countdown task demonstrate that APR achieves: (1) higher performance within the same context window (83.4% vs. 60.0% at a 4k context); and (2) superior

scaling behavior as compute budgets increase, resulting in more substantial performance improvements (80.1% vs. 66.6% using 20k tokens); (3) significantly higher success rates compared to serialized search methods at equivalent latency constraints (75.2% vs. 57.3% at around 5,000 ms). Overall, APR demonstrates the potential of reasoning systems that dynamically structure their inference processes to achieve improved scalability and efficiency.

We see many research directions that we are excited to explore in the future:

1. Extending to pre-trained language models and general tasks. Currently, our experiments are restricted to non-pretrained LMs on Countdown tasks. Moving forward, we plan to extend our methods to general reasoning tasks with pre-trained LMs, which would validate the approach’s broader applicability.

2. Reducing reliance on supervised training. Our current setup requires bootstrapping by mimicking the symbolic solver. We aim to use strong pre-trained LMs as initial checkpoints and, motivated by DeepSeek R1-Zero [177], explore whether we can bypass the supervised training stage and directly apply reinforcement learning.

3. Better orchestration and communication protocols. As a first step, this work uses `fork` and `join` operations for inter-thread communication and thread orchestration. We plan to explore other protocols, such as any-to-any messaging, all-to-all communication patterns, or subscription-based methods.

Acknowledgements The authors would like to thank Nicholas Tomlin, Boyi Li, Li Erran Li, Sanjay Subramanian, Ruiqi Zhong for their valuable discussions. This work was partially supported by compute resources provided through Google’s TPU Research Cloud (TRC) and Nvidia, and an Ai2 Young Investigator Award.

Chapter 8

ThreadWeaver: Adaptive Threading for Efficient Parallel Reasoning in Language Models

This chapter is based on the following publication:

Long Lian, Sida Wang, Felix Juefei-Xu, Tsu-Jui Fu, Xiuyu Li, Adam Yala, Trevor Darrell, Alane Suhr, Yuandong Tian, and Xi Victoria Lin. “ThreadWeaver: Adaptive Threading for Efficient Parallel Reasoning in Language Models.” *International Conference on Machine Learning (ICML)*, 2026.

8.1 Introduction

Large Language Models (LLMs) have advanced automated problem solving across domains demanding complex reasoning, including mathematics, programming and scientific discovery. This process is driven by scaling inference-time computation to produce longer and more detailed chain of thought (CoT) that iteratively navigates the problem space and revises earlier steps [182, 183, 179]. As task difficulty increases, the time required to reach a correct solution rises dramatically, in some cases reaching tens of hours for a single task [207]. On the other hand, the sequential nature of autoregressive LLMs imposes severe inference latency which cannot be easily reduced by adding more compute resources, as generating each token requires conditioning on all previous tokens.

Parallel reasoning has emerged as a popular technique for LLMs to address this limitation, offering an additional test-time scaling dimension beyond extending sequence length. Methods such as best-of-N and self-consistency [178] execute multiple independent reasoning threads and aggregate their outputs. Test-time search approaches such as tree-of-thought [179] guide exploration through hand-crafted structures tuned to the underlying problem. More recently, several works have begun to explore *adaptive parallel reasoning*, where LLMs are trained to decide when to branch reasoning into parallel threads and when to continue sequentially,

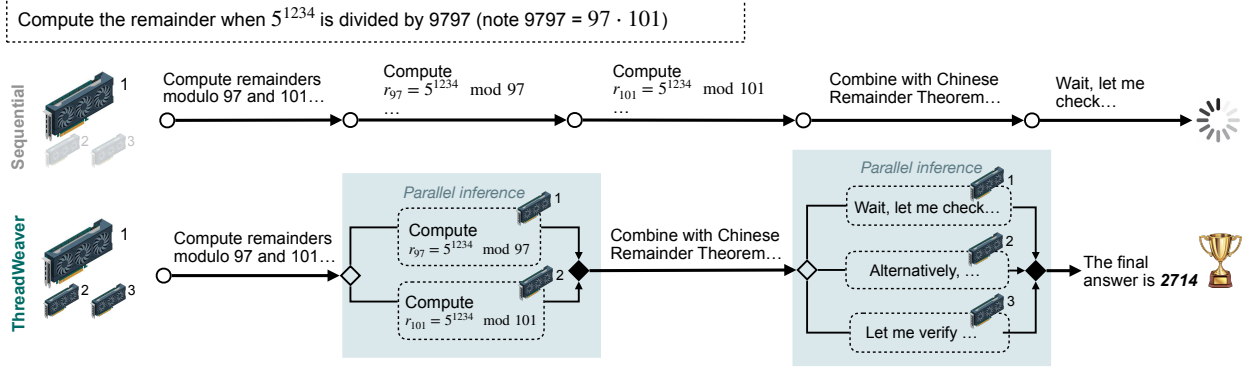


Figure 8.1: For complex reasoning problems, a sequential reasoner generates tokens one at a time, so latency scales with the trajectory length and is difficult to reduce with additional compute. ThreadWeaver instead adaptively **spawns** ($\diamond$) and **joins** ($\blacklozenge$) concurrent threads that work on different parts of the solution, using extra compute to shorten the critical reasoning path.

which mimics multi-threaded execution without pre-defining the branching structure [208, 8, 199, 9]. Such adaptive approaches hold the promise of allocating test-time computation more effectively across reasoning stages, enabling the model to self-determine when parallelization is beneficial for the task at hand.

However, despite its potential, *adaptive parallelization has yet to demonstrate competitive performance on complex real-world problems comparable to that of strong sequential reasoning models* [8, 9], due to several fundamental challenges. First, high-quality parallel reasoning trajectories for real-world problems are difficult to obtain for training purposes, leaving prior datasets limited in both scale and diversity. Second, most existing approaches require modifications to position embeddings, key-value caches, or attention masks at inference to support thread branching and joining [199, 9, 8], making deployment on standard inference engines difficult [209, 180]. Third, scaling reinforcement learning (RL) for parallel reasoning is further complicated by additional modeling and system challenges, such as advantage calculation across branches and maintaining consistency between train-time rollouts and test-time execution.

To address these challenges, we propose **ThreadWeaver**, a framework that brings RL-induced adaptive parallel reasoning to strong reasoning LLMs such as Qwen3 [210], without modifying the underlying inference engine. We target a setting in which abundant compute is available to reduce reasoning latency for problems requiring long chain-of-thoughts. For such examples, further latency reduction requires parallelism *within* a single reasoning trajectory. ThreadWeaver is designed specifically to operate effectively under this regime, with three core innovations: 1) *a two-stage parallel trajectory generator*, which generates high-quality structured reasoning traces with LLM-annotated parallelization control tokens, providing a scalable, low-cost cold start for downstream RL (§8.4); 2) *a trie-based training and inference*

co-design (§8.3), which enables parallel reasoning on standard autoregressive inference engines such as vLLM [209] and SGLang [180]; 3) *parallelization-aware GRPO (P-GRPO)*, which jointly optimizes accuracy and latency reduction through mathematically justified thread-wise advantage broadcast (§8.5) and a parallelization-aware reward design (§8.5) to induce effective adaptive parallelization while maintaining strong reasoning performance.

ThreadWeaver is the first adaptive parallel reasoning framework that achieves accuracy comparable to equally sized cutting-edge sequential reasoning models (79.9% accuracy for ThreadWeaver vs 78.3% for the thinking model Qwen3-8B [210] on AIME24, and 71.9% vs 72.2% on average across six widely used math benchmarks) while delivering substantial reductions in token latency: $1.14\times$ speedup on AIME24 [211], $1.16\times$ speedup on AMC23 [212], $1.23\times$ speedup on MATH500 [213], $1.21\times$ on OlympiadBench [214], and $1.53\times$ on Minerva Math [215]. These results establish a new Pareto frontier between accuracy and efficiency, opening a promising direction for enabling LLMs to tackle increasingly complex problems within feasible inference time budget.

8.2 Related work

Test-Time Scaling. Test-time scaling has proven effective for enhancing LLM performance across various downstream tasks [182, 183, 4, 186, 185, 216]. Prominent examples like OpenAI o1 [126] and DeepSeek R1 [177] scale the length of reasoning trajectories via reinforcement learning. However, the sequential nature of autoregressive decoding means that longer outputs incur significant latency, which is unsustainable for complex domains like theorem proving or scientific discovery. In contrast, parallelizing reasoning traces allows for adaptive allocation of computational budgets, achieving a superior Pareto frontier.

Parallel Reasoning with Self-Consistency. Parallel reasoning through self-consistency is a widely adopted technique to improve LLM inference robustness. By sampling multiple independent reasoning chains and aggregating the final answers, this approach provides a scaling dimension orthogonal to sequence length [188, 178, 217]. Related system results (e.g., Gemini Deep Think, Luong and Lockhart [218]) also emphasize inference-time parallel compute scaling, though the exact mechanisms can differ from standard self-consistency. Variants refine this paradigm with confidence-guided aggregation and adaptive consensus [219, 220]. Nevertheless, because these threads are uncoordinated and independently executed, they often result in redundant computation and limited efficiency.

Structured (Agentic) Parallel Reasoning. Structured parallel reasoning explores coordination via prompting-based agents or tree-of-thought frameworks [179, 189, 190, 191, 192, 193, 194, 195, 196, 197]. These methods decompose problems into subtasks using explicit structures like graphs or modular agents. While effective for symbolic or mathematical reasoning, they generally rely on hand-crafted heuristics and operate at inference time, lacking

the scalability of learned reasoning strategies. Recent agentic formulations automate this process but still lack gradient-based optimization across reasoning branches [198, 200, 221].

Adaptive Parallel Reasoning. Recent training-based methods enable models to self-determine when to branch via special tokens. PASTA [199] and Multiverse [8] utilize SFT and optionally preference optimization, but do not incorporate reinforcement learning (RL), which was s for test-time scaling. APR [208] introduces an SFT+RL pipeline but is limited to synthesized datasets. Parallel-R1 [9] incorporates RL, but its SFT data covers only low-complexity problems, with its final accuracy below that of official Qwen3 models [210]. Concurrent work NPR [222] also uses RL but is built upon instruct models rather than thinking models. Both Parallel-R1 and NPR models have shorter CoTs compared to ThreadWeaver. Compared to previous work, ThreadWeaver is based on thinking models (e.g., Qwen3-8B) and is capable of performing long chain-of-thought reasoning. It is the first adaptive parallel reasoning framework to achieve accuracy comparable to cutting-edge sequential thinking models of equivalent size. In addition, while Multiverse and NPR require significant modifications to the underlying inference engine, ThreadWeaver is compatible with standard sequential engines for easy deployment.

8.3 Model

Parallel Trajectory Format

We extend standard autoregressive (AR) generation with lightweight control tokens arranged in a fork-join pattern to realize adaptive parallel reasoning, following Multiverse [8]. Unlike Multiverse, our framework restricts trajectories to a single level of parallelization in which all branches rejoin the main thread.¹ Below we describe our parallel trajectory format, with an illustrative example shown in Figure 8.2:

- `<think>` and `</think>` mark the start and end of the reasoning trajectory, which may contain sequential segments and zero or more parallel blocks.
- `<Parallel> ... </Parallel>`: defines a parallel block.
- `<Outlines> ... </Outlines>`: consists of numbered `<Outline>` entries which declare independent sub-tasks.
- `<Thread> i`: execution trajectory of the i -th sub-task. *Threads are generated independently and must not reference other threads.* At inference, each thread is generated in parallel and stopped at `</Thread>`.

¹In practice, we found that single-level parallelization often offers significant speedup while preserving reasoning accuracy. It also reduces the engineering overhead and data-quality challenges introduced by nested parallelization branches. The techniques proposed in our work naturally generalize to nested branches by permitting parallel blocks within individual threads.

Prompt: Find the modulo 7 remainder of the sum $1 + 3 + 5 \dots + 199$. Think step by step and put your answer in `\boxed{}`.

Response:

<think>

We want the remainder when $S = 1 + 3 + 5 + \dots + 199$ is divided by 7.

<Parallel>

<Outlines>

<Outline>1: Use the arithmetic series formula to compute S exactly, then reduce S modulo 7. </Outline>

<Outline>2: Work directly modulo 7 by exploring the repeating pattern of odd numbers modulo 7. </Outline>

</Outlines>

<Thread>1: The sum S is an arithmetic sequence with first ... </Thread>

<Thread>2: The number of terms is $n = (199 - 1) / 2 + 1 \dots$ </Thread>

</Parallel>

Both methods agree that $S = 4 \pmod{7}$, so the remainder when S is divided by 7 is `\boxed{4}`.

</think>

The remainder when $1 + 3 + 5 + \dots + 199$ is divided by 7 is `\boxed{4}`.

Figure 8.2: **Format of our parallel reasoning trajectories.** <Parallel> encloses a fork-join block with <Outlines> and multiple <Thread>’s; content inside each <Thread> is generated *concurrently*, while the other spans are generated sequentially.

In data collection, this format is enforced by our data generator proposed in §8.4 for supervised fine-tuning. During RL rollout and at inference time, this structured trajectory is generated by an orchestrator calling an LLM inference engine, described in §8.3.

Parallel Inference

Our parallel structure introduces <Parallel> blocks that contain an <Outlines> section followed by multiple <Thread>’s. Importantly, *only the thread contents are generated in parallel*, while all remaining parts of the trajectory are generated via standard autoregressive (AR) decoding. As shown in Figure 8.3, this design allows parallel inference to be implemented

on top of any model exposed through a request-completion interface, such as an LLM inference server with API backed by an autoregressive inference engine like SGLang [180] or vLLM [209], without any modifications to their underlying system.

State-machine view. We implement inference as a minimal state machine operating on request-response pairs with a text-completion API. The orchestrator alternates between sequential phases (decoding until `</Outlines>`), parallel phases (launching independent completion requests for each thread until `</Thread>`), and join phases (concatenating results and continuing). This design requires no modifications to the LLM architecture or inference engine, inheriting existing optimizations like prefix caching and paged attention [180, 209].

Hybrid operation. This design naturally enables hybrid reasoning: operators may choose *parallel mode* (executing the state machine for low latency via parallel reasoning) or *autoregressive mode* (bypassing the state machine for high throughput). ThreadWeaver is the first adaptive framework maintaining sequential reasoning compatibility.

Efficient Training with Trie-based Sequence Merging

We fine-tune the LLM to emit these parallel structures by matching training targets to the inference state machine via trie construction and traversal.

Trie-based loss computation. As shown in Figure 8.4, using regex-based parsing, we first extract all the `<context, completion>` units that will be issued during inference. Specifically, each context corresponds to a fragment issued by the state machine to a text completion API, and each completion corresponds to the LLM-generated span between structural tokens (e.g., from `<think>` to `</Outlines>`). The pair of spans is concatenated into a token sequence. We then insert all units into a token-level prefix tree (root = shared prompt, branches = divergent continuations), then use depth-first traversal to linearize it into a single training sequence with an *ancestor-only* attention mask and position encoding to prevent cross-thread leakage.² Loss is applied only to tokens from the completion part of each unit. This ensures training matches the inference state machine’s isolation property.

Consistency and scalability. This co-design guarantees that all the tokens in a trajectory generated by the trie traversal will have the same context and position encodings as in each request-response pair that the orchestrator issues. Training with trajectories constructed in this manner also ensures that inferencing without our orchestrator recovers ordinary autoregressive reasoning behavior, since the overall trajectory obtained by autoregressive inferencing is always a valid subsequence of the trajectory obtained from the trie. We employ FlexAttention [223] for efficient attention masking in training.

²The change in attention masks and position encoding are only at training time and are not employed at inference time.

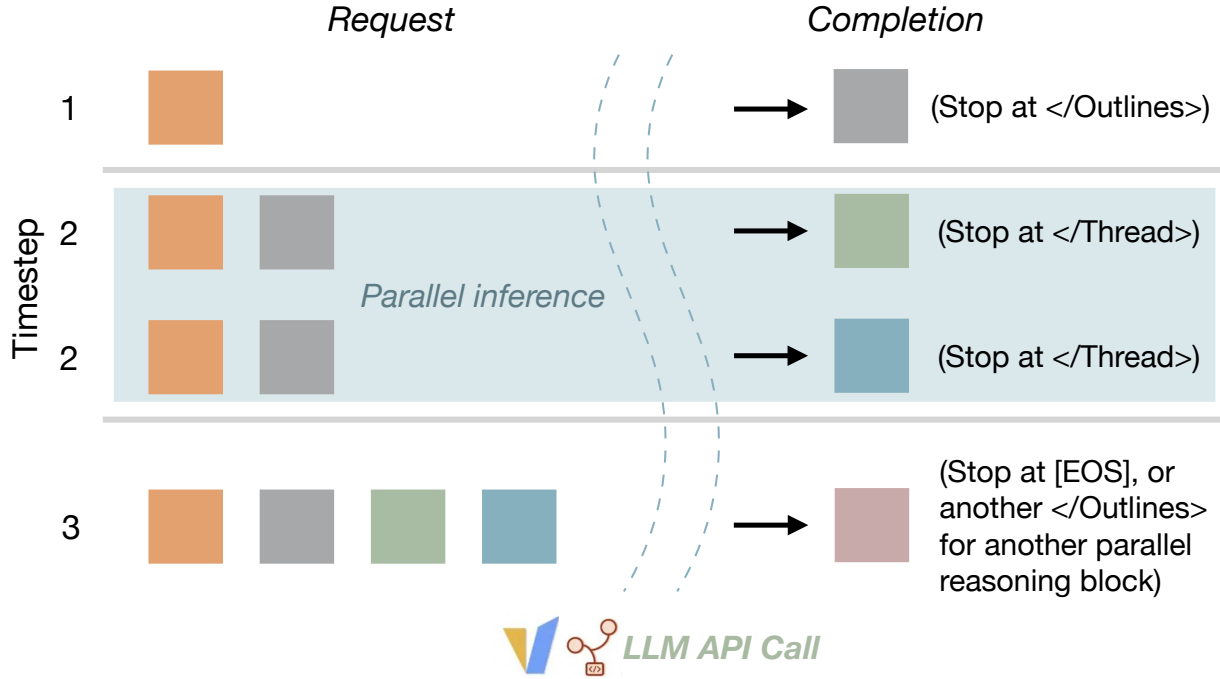


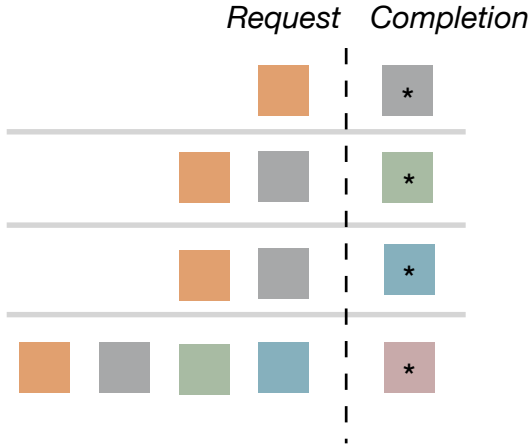
Figure 8.3: **Inference-time request sequence for a parallel reasoning trajectory.** For a given user prompt, Timestep 1 decodes the prefix and the <Outlines> block sequentially up to </Outlines>. Timestep 2 launches one completion request per outline in parallel, each seeded with its corresponding “<Thread> *i*” prefix and stopped at </Thread>. Timestep 3 resumes sequential decoding over the joined context until [EOS] or the next </Outlines>. All steps are executed on a standard LLM inference engine.

8.4 Parallel Trajectory Curation for Supervised Fine-tuning

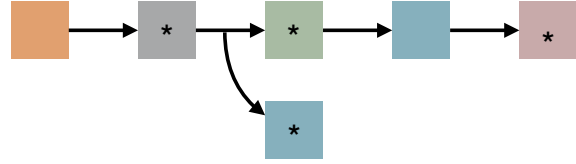
With our proposed inference pattern and training paradigm, a key challenge is to obtain suitable training data for supervised fine-tuning. Our training procedure requires parallel reasoning trajectories in the fork-join format described in Section 8.3. To construct these trajectories, we employ a two-stage pipeline: **1)** an LLM-driven annotation and a light-weight rewriting phase for SFT cold start, followed by **2)** self-training with reward-based filtering to scale data quality and improve performance.

LLM rewriting. We first extract sequential reasoning trajectories by running inference on Qwen3-8B [210] with a dataset of prompts. We then apply a five-step LLM-driven annotation pipeline using GPT-5 [224] to identify parallelizable spans, extract canonical thread boundaries, rewrite for clarity and independence, generate outlines, and perform

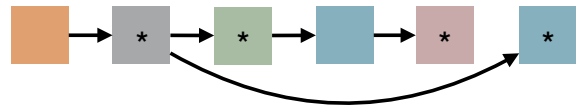
Step 1: Extracting all <request, completion> pairs from a training example



Step 2: Prefix-Tree Building



Step 3: Flattening to Single Sequence



*We apply masking and reset the position IDs s.t. * only sees orange * in its context.*

Figure 8.4: **Our trie-based training sequence construction has three steps: 1)** extract context and completion segments that the inference state machine will encounter in order to produce this trajectory; **2)** insert these segments into a token-level trie (prefix-tree); **3)** traverse the trie to produce a flat sequence with an ancestor-only attention mask and standard incremental positions counting from the root node. * denotes text spans with loss applied. Although we only show one branching in this example, ThreadWeaver supports branching and joining multiple times in a trajectory.

format validation. Crucially, we let the LLM recover structure from existing chain-of-thoughts by using line-based extraction, compared to regenerating solutions [8], preserving the model’s original reasoning semantics through minimal localized edits. Using this pipeline on 53k sequential Qwen3-8B trajectories from Polaris-53k [225], we obtain 959 high-quality parallelized trajectories for cold-start training.

Self-training with reward-based filtering. While effective for cold-start, the LLM-rewritten trajectories produced in the previous step reflect post-hoc decomposition rather than the model’s natural generative patterns. To bridge this gap, we fine-tune Qwen3-8B on the 959 cold-start items, then run parallel rollout on the full 53k prompts. We filter trajectories by format correctness and answer correctness, yielding a large corpus of format-faithful trajectories that align supervision with the model’s own generative patterns, which serves as high-quality SFT data to enable reliable parallel rollout in subsequent reinforcement learning.

8.5 Reinforcement Learning

We build on Group Relative Policy Optimization (GRPO) [204], a policy-gradient method that computes advantages via intra-prompt normalization. For a prompt p with k sampled trajectories having rewards $\{r_1, \dots, r_k\}$, let $\mu_p = \frac{1}{k} \sum_{j=1}^k r_j$ and $\sigma_p = \sqrt{\frac{1}{k} \sum_{j=1}^k (r_j - \mu_p)^2}$ denote the group mean and standard deviation. The group-normalized advantage is

$$A_{p,i}^{\text{GRPO}} = \frac{r_i - \mu_p}{\sigma_p + \varepsilon}, \quad (8.1)$$

with small $\varepsilon > 0$ for stability.

P-GRPO: GRPO with Parallel Rollout

State-machine based rollout. We perform rollout using the inference orchestrator described in Section 8.3. Let $\tau_p^{(i)}$ be composed of the ordered set of $\langle \text{context}, \text{completion} \rangle$ units emitted by the state machine, $\tau_p^{(i)} = \{u_p^{(i,1)}, u_p^{(i,2)}, \dots, u_p^{(i,M_i)}\}$, where $u_p^{(i,m)} = [\text{cont}_p^{(i,m)} \parallel \text{comp}_p^{(i,m)}]$ is the concatenation of the context $\text{cont}_p^{(i,m)}$ followed by its corresponding completion $\text{comp}_p^{(i,m)}$. Training with these units ensures the distribution seen during RL matches test-time execution so that there is no train–inference gap.

Advantage calculation. We compute a single scalar reward $r_{p,i}$ for the *entire* trajectory (see Section 8.5) and *broadcast* its group-normalized advantage $A_{p,i}$ to all units in that sample. Let $T_{p,i}$ be the number of tokens in trajectory i of prompt p and $Z = \sum_{p \in \mathcal{B}} \sum_{i=1}^k T_{p,i}$ be the total token count. The loss is

$$\mathcal{L}_{\text{P-GRPO}}(\theta) = -\frac{1}{Z} \sum_{p,i} A_{p,i} \sum_{m=1}^{M_i} \log \pi_{\theta}(c_p^{(i,m)} \mid x_p^{(i,m)}), \quad (8.2)$$

where $c_p^{(i,m)}$ and $x_p^{(i,m)}$ denote completion and context tokens. Following the practice in sequential reasoning and multi-turn RL, we do not apply loss on the context tokens of each thread. Specifically, Equation (8.2) is implemented with a token-level mask that zeros out the loss on context tokens, so the log-probability term only accumulates contributions from completion tokens. Therefore, each generated token receives loss at most once, even if it later reappears as part of the context for subsequent generation.

This broadcasting across the units spanning all the parallel threads is mathematically justified, keeps the implementation simple, and empirically stabilizes training without per-branch credit heuristics.

Parallelization-Aware Reward

Similar to sequential generation, we set the main reward to an indicator function of answer correctness. In addition, we softly encourage *acceleration* when parallelization reduces the

Algorithm 2 Parallelization-Aware GRPO (P-GRPO)

```

1: Input: Policy  $\pi_\theta$ , batch of prompts  $\mathcal{B}$ , rollouts per prompt  $k$ , reward function  $r(\cdot)$ 
2: Output: Updated policy  $\pi_\theta$ 
3: for each prompt  $p \in \mathcal{B}$  do
4:   for  $i = 1$  to  $k$  do
5:     Sample trajectory  $\tau_p^{(i)}$  using parallel inference state machine (§8.3)
6:     Extract  $\{u_p^{(i,1)}, \dots, u_p^{(i,M_i)}\}$  where  $u_p^{(i,m)} = [\text{cont}_p^{(i,m)} \parallel \text{comp}_p^{(i,m)}]$ 
7:     Compute reward  $r_{p,i} = \mathcal{R}_{\text{correct}}(\tau_p^{(i)}) + \mathcal{R}_{\text{accel}}(\tau_p^{(i)})$ 
8:   end for
9:   Compute group mean:  $\mu_p = \frac{1}{k} \sum_{j=1}^k r_{p,j}$ 
10:  for  $i = 1$  to  $k$  do
11:    Compute advantage:  $A_{p,i} = r_{p,i} - \mu_p$ 
12:  end for
13: end for
14: Compute total tokens:  $Z = \sum_{p \in \mathcal{B}} \sum_{i=1}^k T_{p,i}$ 
15: Compute loss:  $\mathcal{L} = -\frac{1}{Z} \sum_{p,i} A_{p,i} \sum_{m=1}^{M_i} \log \pi_\theta(c_p^{(i,m)} \mid x_p^{(i,m)})$ 
16: Update  $\theta$  by gradient descent on  $\mathcal{L}$ 

```

critical path. For sample τ , the total reward includes the two components added together:

$$r(\tau) = \mathcal{R}_{\text{correct}}(\tau) + \mathcal{R}_{\text{accel}}(\tau), \quad (8.3)$$

The correctness reward $\mathcal{R}_{\text{correct}}(\tau)$ evaluates the boxed final answer:

$$\mathcal{R}_{\text{correct}}(\tau) = \mathbf{1}\{\text{CORRECT}(\tau)\}. \quad (8.4)$$

The acceleration reward $\mathcal{R}_{\text{accel}}$ incentivizes the model to generate more efficient reasoning paths by leveraging parallelization:

$$\mathcal{R}_{\text{accel}}(\mathbf{s}) = \mathbf{1}\{\text{CORRECT}(\tau)\} \min(\rho \cdot \eta(\mathbf{s}), \rho_{\text{clip}}) \quad (8.5)$$

where $\mathbf{s}$ denotes the set of parallel threads in the trajectory, ρ is an acceleration reward scale, and ρ_{clip} is a clipping threshold.³ Acceleration ratio $\eta(\mathbf{s})$ is computed as:

$$\eta(\mathbf{s}) = 1 - \frac{L_{\text{longest}}}{L_{\text{total}}} \quad (8.6)$$

where L_{longest} is the sequence length for the longest thread (i.e., token latency), L_{total} is the total number of tokens in the entire trajectory. The acceleration ratio reward is only given when the answer is correct, which is indicated by the correctness term in Equation (8.5).

³In our experiments we set $\rho = 0.5$ and $\rho_{\text{clip}} = 0.2$. These values are chosen empirically to keep the acceleration bonus a small fraction of the correctness reward so that the model does not sacrifice accuracy for acceleration gains.

Removing std-dev normalization for stability. Empirically, standard-deviation normalization in GRPO leads to unstable training with the multi-term reward Equation (8.3): when all rollouts are correct, mean-centering removes the correctness signal and std-dev normalization cancels the acceleration scale, causing the model to optimize acceleration at the expense of correctness. We instead use simple mean-centering, preserving the intended reward balance:

$$A_{p,i}^{\text{P-GRPO}} = r_{p,i} - \mu_p. \quad (8.7)$$

Full ablation analysis in Table 8.4. This choice is aligned with the modifications proposed by Liu et al. [226], though our motivation originates from the multi-term reward structure proposed for incentivizing parallel reasoning.

Overall, since the rollout uses our parallel inference scheme specified in Section 8.3, P-GRPO learns parallel behavior that transfers directly to any off-the-shelf AR engine, along with correctness and acceleration improvements. We present the pseudo-code for our proposed P-GRPO in Algorithm 2.

8.6 Experiments

Setup

We evaluate ThreadWeaver trained on top of Qwen3-8B [210], a thinking model that demonstrates strong performance on mathematical problem solving with long CoT.

ThreadWeaver is trained through a three-stage pipeline: **1)** Fine-tuning Qwen3-8B on 959 LLM-rewritten parallel trajectories for 8 epochs, **2)** Self-training with 17,491 model-generated correct and structurally-valid trajectories, and **3)** RL using P-GRPO for 350 steps with batch size 128 and group size 8. Training prompts from all stages come from An et al. [225]. Sequential RL baseline uses the same training setup.

Evaluation. We evaluate performance on six math reasoning benchmarks: AIME24 [211], AIME25 [227], Minerva Math [215], AMC23 [212], OlympiadBench [214], and MATH500 [213]. Parallel reasoning in inference is performed via our proposed orchestrator.

Main Results

We report the performance of ThreadWeaver in Table 8.1, comparing it against a sequential GRPO baseline on Qwen3-8B using the same RL prompts across six math benchmarks. Both models are trained with identical RL configurations and evaluated with multiple rollouts per problem.

We report accuracy and token latency to assess reasoning correctness and efficiency. To quantify efficiency gains, we define three speedup metrics based on the ratio of token latencies between the sequential GRPO baseline and ThreadWeaver. *Speedup* is the average of these per-problem ratios across a dataset. *Speedup (correct only)* restricts this calculation to rollouts

Table 8.1: Comparison between the sequential GRPO baseline and ThreadWeaver on Qwen3-8B across six math benchmarks. ThreadWeaver matches the baseline’s accuracy (71.9% vs. 72.2%) while **reducing mean critical-path length from 15.1k to 13.2k tokens** and achieving **up to 1.53 $\times$ average speedup** and **up to 3.56 $\times$ max speedup**. Speedups are computed as ratios of the token latencies of the sequential reasoning baseline and ThreadWeaver.

Model	AIME24 (Avg@32)	AIME25 (Avg@32)	AMC23 (Avg@8)	MATH500 (Avg@1)	Minerva Math (Avg@4)	OlympiadBench (Avg@8)	Average
Accuracy (%)							
Qwen3-8B + Seq. RL	78.3	61.6	92.6	91.8	43.9	65.0	72.2
Qwen3-8B + ThreadWeaver	79.9	60.5	92.3	91.4	43.7	63.5	71.9
Token Latency (tokens)							
Qwen3-8B + Seq. RL	19.4k	24.6k	13.8k	7.2k	10.6k	15.2k	15.1k
Qwen3-8B + ThreadWeaver	16.9k	24.0k	12.0k	6.4k	7.3k	12.8k	13.2k
Speedup	1.14 $\times$	1.03 $\times$	1.16 $\times$	1.23 $\times$	1.53 $\times$	1.21 $\times$	1.22 $\times$
Speedup (correct only)	1.14 $\times$	1.05 $\times$	1.17 $\times$	1.23 $\times$	1.47 $\times$	1.20 $\times$	1.21 $\times$
Max Speedup (correct only)	1.47 $\times$	1.21 $\times$	1.67 $\times$	3.05 $\times$	3.56 $\times$	1.92 $\times$	—

yielding correct final answers for both models. Finally, *Max Speedup (correct only)* identifies the maximum per-problem speedup achieved among correct rollouts, serving as an empirical upper bound for our method’s acceleration. For each problem, we average token latency over all rollouts.

On average, ThreadWeaver matches the baseline’s accuracy (71.9% vs. 72.2%) while reducing mean token latency from 15.1k to 13.2k and yielding up to 1.53 $\times$ speedup.

The highest latency reductions occur on benchmarks with long, decomposable reasoning traces (e.g., AIME24, OlympiadBench), where independent subtasks can be processed concurrently.

Comparisons

We compare ThreadWeaver with recently proposed adaptive parallel reasoning methods on AIME24, focusing on Multiverse [8] and Parallel-R1 [9]. As summarized in Table 8.2, ThreadWeaver attains substantially higher accuracy (79.9%) compared to Multiverse (up to 53.8%) and Parallel-R1 variants (up to 19.4%). Furthermore, ThreadWeaver achieves higher *self-parallelism speedup* and *activation ratio*. The former is the ratio between the total number of tokens and the token latency in reasoning trajectory, and the latter is defined as the fraction of trajectories with parallelization. **ThreadWeaver achieves both higher accuracy and higher efficiency, establishing a stronger Pareto frontier than prior adaptive parallel reasoning approaches.**

We also observed that ThreadWeaver built on Qwen3-8B outperforms Multiverse, which is built on Qwen2.5-32B [228], both in accuracy and in measured self-parallelism speedup. This

Table 8.2: Comparison with other adaptive parallel reasoning approaches on AIME24. Self-parallelism speedup is the ratio between the total number of tokens and the critical path length. Activation ratio is defined as the percentage of samples with parallelization in the inference trajectories. ThreadWeaver achieves higher accuracy and stronger self-parallelism than Multiverse and Parallel-R1. Results from Yang et al. [8] and Zheng et al. [9].

Model	Size	Self-Par. Speedup	Act. Ratio	Acc.
Multiverse-zero	32B	1.04×	-	52.1%
Multiverse	32B	1.18×	-	53.8%
Parallel-R1-Seen	4B	-	27.3	19.4%
Parallel-R1-Unseen (S1)	4B	-	13.6	18.3%
Parallel-R1-Unseen (S2)	4B	-	63.0	16.3%
ThreadWeaver (ours)	8B	1.25×	85.2	79.9%

indicates that our training recipe effectively induces adaptive parallelization behavior of the model at a much smaller model size. Compared to Parallel-R1, which is also based on a model from the Qwen3 family (Qwen3-4B-Base), ThreadWeaver operates in a long chain-of-thought regime that closely matches the sequence lengths of strong sequential reasoning models in the Qwen3 family. In contrast, Parallel-R1 is trained to produce significantly shorter chains of thought, which limits its test-time scaling ability and accuracy.

Acceleration Distribution Analysis

To analyze how acceleration varies across problems, Figure 8.5 shows per-problem speedup distributions on four math benchmarks. Speedup is defined as the ratio between the token latency of the sequential baseline and that of ThreadWeaver. For AIME24, AMC23, and OlympiadBench, we average ratios over 32, 8, and 8 rollouts per prompt to reduce sampling noise, while MATH500 uses a single rollout per problem.

Across all datasets, most mass lies to the right of the 1.0×

 line, indicating consistent acceleration, with a wide spread reflecting strong question dependence. Some problems admit large gains (up to 1.47× on AIME24 and 3.05× on MATH500), while others show little benefit or mild slowdowns. Overall, ThreadWeaver yields substantial critical-path reductions on decomposable problems and naturally approaches parity on intrinsically sequential ones.

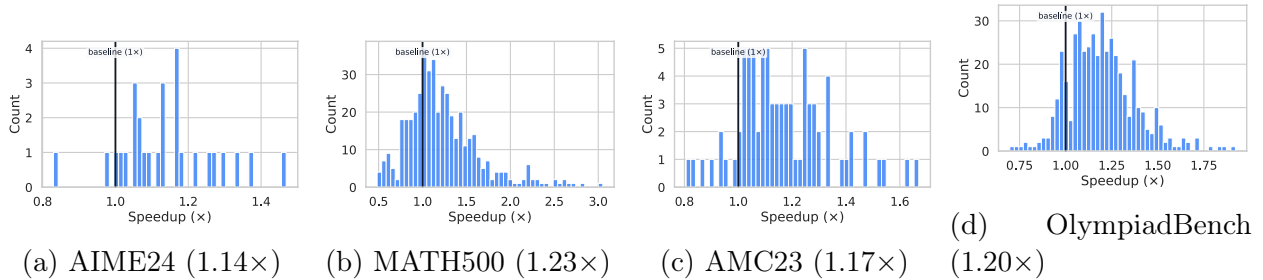


Figure 8.5: **Per-problem speedup distributions on math benchmarks.** The speedup is the ratio between the token latency of the sequential reasoning baseline and the token latency of ThreadWeaver. A vertical reference line at $1.0\times$ marks parity with the sequential baseline. Parentheses show mean speedup per dataset.

Table 8.3: Wall-clock latency on 50 MATH problems (averaged over 8 runs per problem).

Parallelization	Avg Latency (s)
Parallel	142.21
Sequential	162.34
Speedup	1.14×

Ablations

Adding compute makes ThreadWeaver faster. To verify that critical-path reductions yield real-world gains, we measured wall-clock latency on 50 problems from MATH500 (8-run average). We compared standard autoregressive decoding on a single GPU against the orchestrator distributing threads across four GPUs at batch size 1.

As shown in Table 8.3, parallelization achieves a $1.14\times$ speedup. While lower than token-latency gains due to system overhead, this confirms that ThreadWeaver effectively converts additional compute into reduced inference latency.

Removing standard-deviation normalization in GRPO improves stability. We compared GRPO variance normalization against simple mean-centering. As shown in Table 8.4, mean-centering preserves the intended balance between correctness and acceleration, resulting in higher accuracy and shorter critical paths.

Parallel rollout and self-training improve performance. We ablate two training components: *parallel rollout* during RL and *self-training* post-SFT. As shown in Table 8.5, both significantly enhance accuracy and efficiency. Without self-training, accuracy drops to

Table 8.4: Effect of std-dev normalization on AIME 24 (Avg@32).

Setting	Acc.	Seq. Len.	Longest
With Std. Norm.	74.8%	30.1k	18.7k
Mean-Centered (Ours)	79.9%	21.1k	16.9k

Table 8.5: Ablation on parallel rollout and self-training (AIME 24).

Par. Rollout	Self-Train	Acc.	Latency
✓	✗	77.9%	17.6k
✗	✓	78.4%	17.3k
✓	✓	79.9%	16.9k

77.9% and latency increases to 17.6k, confirming that the 17k high-quality samples are vital for aligning teacher-curated data with Qwen3-8B’s reasoning capabilities (Section 8.4).

Meanwhile, parallel rollout allows the policy to optimize execution efficiency directly, eliminating the mismatch between training rollouts and parallel inference. Note that self-training can be considered as a form of filtered behavior cloning or STaR [229], and since our self-training data is always generated with parallel rollout in the ablations, extensive self-training can be considered as taking part of the role in reinforcement learning. When combined, these techniques enable ThreadWeaver to achieve 79.9% accuracy and a 16.9k token latency.

Additional analysis. We present a comprehensive analysis of ThreadWeaver’s behavior in the appendix, including per-stage ablations (Table F.2) and efficiency metrics (Table F.5). We also discuss why reward hacking was not observed due to strong SFT priors (Section F.7) and provide a qualitative study in Section F.5 that illustrates ThreadWeaver spontaneously forking into distinct parallel strategies (e.g., algebraic vs. numerical) to solve complex problems, demonstrating adaptive task decomposition.

8.7 Conclusion

We introduce ThreadWeaver, an adaptive parallel reasoning framework that addresses key obstacles limiting prior work: 1) a two-stage parallel trajectory generator for scalable data creation, 2) a trie-based training-inference co-design enabling deployment on standard autoregressive engines, and 3) P-GRPO, an RL framework that optimizes for both accuracy and acceleration. ThreadWeaver achieves up to $1.53\times$ speedup in token latency compared

to sequential baselines while maintaining comparable accuracy with cutting-edge thinking models, establishing a new Pareto frontier.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This dissertation studied task decomposition with multi-agent systems across visual generation, visual data generation, and language-model reasoning. Across these settings, the central result is that complex tasks can often be made more controllable and efficient when a single opaque generation process is replaced by explicit intermediate structure and coordinated sub-computations. The same design pattern appears throughout the thesis: language models act as planners, parsers, controllers, annotators, and reasoning agents, while visual models, captioning systems, and inference orchestrators ground and execute the decomposed computation.

In visual generation, Chapters 2–4 demonstrate that language models can make spatial and temporal structure explicit before and during generation. LLM-grounded Diffusion translates prompts into object layouts and uses them to guide diffusion, substantially improving generation accuracy across negation, counting, attribute binding, and spatial reasoning benchmarks over Stable Diffusion. LLM-grounded Video Diffusion extends this idea to dynamic scene layouts, producing spatiotemporal plans and videos that align much more reliably with complex prompts than the baseline video generator. Self-correcting LLM-controlled Diffusion shows that decomposition can also be iterative: generation, perception, diagnosis, and repair form a feedback loop that improves correctness for both open-source and proprietary image generators.

In visual data generation, Describe Anything shows that decomposition is also valuable for interpreting and annotating visual content. By combining local focal crops with global context, it produces detailed localized image and video descriptions rather than only global captions. This design achieves state-of-the-art results across in-domain and zero-shot benchmarks, with consistent improvements on keyword-level, phrase-level, detailed image captioning, and detailed video captioning evaluations. These results support a broader role for localized description as both a standalone capability and a source of structured supervision for controllable generation systems.

The takeaway of Part I is that stronger image and video generators do not remove the need for decomposition. As models improve, direct prompting will solve more ordinary cases, but explicit structure remains useful when users need controllability, inspection, repair, spatial or temporal guarantees, or synthetic data with known local semantics. In this sense, the visual chapters are examples of a system design principle for making visual generation more accountable to user-specified structure.

In reasoning, Chapters 7–8 apply the same principle to language-model inference. Adaptive Parallel Reasoning introduces explicit `spawn` and `join` operations so a model can distribute search across parallel threads rather than serialize all exploration into one chain of thought. On Countdown, this improves performance under fixed context and latency constraints while scaling more effectively with additional test-time compute. ThreadWeaver then shows that the idea transfers to strong reasoning models, matching the accuracy of a sequential reinforcement-learning baseline while reducing critical-path token latency across mathematical reasoning benchmarks.

Taken together, these contributions support the thesis that task decomposition is a way to expose the structure that foundation models need in order to plan, verify, generate data, allocate compute, and execute work in parallel. The specific methods differ across images, videos, localized captioning, and reasoning, but their shared lesson is that capable AI systems should not only predict outputs. Instead, they should organize their own computation around the structure of the task.

9.2 Future Directions

The work in this thesis points to several future directions. First, language-guided visual generation systems can be made more tightly integrated. The methods in this dissertation use LLMs to provide layouts, corrections, or control signals, but future systems could jointly reason over layout, appearance, physical constraints, and user feedback in a more unified loop. This would be especially valuable for long-horizon video generation, interactive editing, and creative tools where users refine a result over multiple turns.

Second, localized description systems can become stronger data engines for visual generation. Describe Anything suggests that region-level captions can bridge perception and synthesis by turning images and videos into structured supervision. Future systems could use such descriptions to train or evaluate layout-grounded image and video generators, bootstrap rare object and attribute combinations, and support human-facing applications where detailed visual descriptions are directly useful.

Third, adaptive parallel reasoning remains an open problem. The results in this thesis show that language models can learn to use parallel structure and that such structure can reduce latency, but future work should scale these ideas to harder domains, deeper nested decompositions, and heterogeneous compute environments. A mature parallel reasoning system should decide not only what to think about, but also when to branch, how much

compute to allocate to each branch, when to stop unproductive paths, and how to merge partial answers reliably.

Finally, task decomposition raises broader questions about how AI systems should be designed. As models become more capable, the challenge is not simply to make larger monolithic predictors, but to build systems that can organize their own computation. The methods in this dissertation are steps toward AI systems that plan, specialize, verify, and parallelize their work, making complex visual and reasoning tasks more controllable and more efficient.

Bibliography

- [1] *SdXL: improving latent diffusion models for high-resolution image synthesis*. arXiv preprint arXiv:2307.01952, 2023. 1952.
- [2] *Visual chatgpt: Talking, drawing and editing with visual foundation models*. arXiv preprint arXiv:2303.04671, 2023. 2023.
- [3] *Generating images with multimodal language models*. arXiv preprint arXiv:2305.17216, 2023. 2023.
- [4] Kanishk Gandhi, Denise HJ Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah Goodman. “Stream of Search (SoS): Learning to search in language”. In: *COLM*. 2024.
- [5] *Gligen: Open-set grounded text-to-image generation*. arXiv preprint arXiv:2301.07093, 2023. 2023.
- [6] *T2i-compbench: A comprehensive benchmark for open-world compositional text-to-image generation*. arXiv preprint arXiv:2307.06350, 2023. 2023.
- [7] *Modelscope text-to-video technical report*. arXiv preprint arXiv:2308.06571, 2023. 2023.
- [8] Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. “Multiverse: Your Language Models Secretly Decide How to Parallelize and Merge Generation”. In: *arXiv preprint arXiv:2506.09991* (2025).
- [9] Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Xinyu Yang, Rumpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, et al. “Parallel-r1: Towards parallel thinking via reinforcement learning”. In: *arXiv preprint arXiv:2509.07980* (2025).
- [10] *Stable beluga models, 2023*. URL. 2023.
- [11] *Mixtral of experts*. arXiv preprint arXiv:2401.04088, 2024. 2024.
- [12] *Llama 2: Open foundation and fine-tuned chat models, 2023*. 2023.
- [13] *High-resolution image synthesis with latent diffusion models*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10684–10695, 2022. 2022.
- [14] *Multidiffusion: Fusing diffusion paths for controlled image generation*. arXiv preprint arXiv:2302.08113, 2, 2023. 2023.

- [15] *Training-free layout control with cross-attention guidance*. arXiv preprint arXiv:2304.03373, 2023. 2023.
- [16] *Boxdiff: Text-to-image synthesis with training-free box-constrained diffusion*. arXiv preprint arXiv:2307.10816, 2023. 2023.
- [17] *Hierarchical text-conditional image generation with clip latents*. arXiv preprint arXiv:2204.06125, 2022. 2022.
- [18] *Photorealistic text-to-image diffusion models with deep language understanding*. Advances in Neural Information Processing Systems, 35: 36479–36494, 2022. 2022.
- [19] *Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models*. arXiv preprint arXiv:2301.12597, 2023. 2023.
- [20] *Flamingo: a visual language model for few-shot learning*. Advances in Neural Information Processing Systems, 35: 23716–23736, 2022. 2022.
- [21] *Layouttransformer: Layout generation and completion with self-attention*. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1004–1014, 2021. 2021.
- [22] *Attention is all you need*. Advances in neural information processing systems, 30, 2017. 2017.
- [23] *Layoutgpt: Compositional visual planning and generation with large language models*. arXiv preprint arXiv:2305.15393, 2023. 2023.
- [24] *Semantic image synthesis with spatially-adaptive normalization*. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 2337–2346, 2019. 2019.
- [25] *Blobgan: Spatially disentangled scene representations*. In Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XV, pp. 616–635. Springer, 2022. 2022.
- [26] *Image generation from layout*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8584–8593, 2019. 2019.
- [27] *Scene graph generation by iterative message passing*. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5410–5419, 2017. 2017.
- [28] *Image generation from scene graphs*. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1219–1228, 2018. 2018.
- [29] *Learning canonical representations for scene graph to image generation*. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI 16, pp. 210–227. Springer, 2020. 2020.
- [30] *Adding conditional control to text-to-image diffusion models*. arXiv preprint arXiv:2302.05543, 2023. 2023.

- [31] *Spatext: Spatio-textual representation for controllable image generation*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 18370–18380, 2023. 2023.
- [32] *Layoutdiffuse: Adapting foundational diffusion models for layout-to-image generation*. arXiv preprint arXiv:2302.08908, 2023. 2023.
- [33] *Layoutdiffusion: Controllable diffusion model for layout-to-image generation*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 22490–22499, 2023. 2023.
- [34] *Reco: Region-controlled text-to-image generation*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14246–14255, 2023. 2023.
- [35] *Microsoft coco: Common objects in context*. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13, pp. 740–755. Springer, 2014. 2014.
- [36] *Lora: Low-rank adaptation of large language models*. arXiv preprint arXiv:2106.09685, 2021. 2021.
- [37] *Instructpix2pix: Learning to follow image editing instructions*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 18392–18402, 2023. 2023.
- [38] *Visual programming: Compositional visual reasoning without training*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14953–14962, 2023. 2023.
- [39] *Segment anything*. arXiv:2304.02643, 2023. 2023.
- [40] *Gpt-4 technical report*. arXiv preprint arXiv:2303.08774, 2023. 2023.
- [41] *Language models are few-shot learners*. Advances in neural information processing systems, 33: 1877–1901, 2020. 1901.
- [42] *Simple open-vocabulary object detection with vision transformers*. arXiv preprint arXiv:2205.06230, 2022. 2022.
- [43] *Orca: Progressive learning from complex explanation traces of gpt-4, 2023*. 2023.
- [44] *Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation*. In International Conference on Machine Learning, pp. 12888–12900. PMLR, 2022. 2022.
- [45] *Simple multi-dataset detection*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7571–7580, 2022. 2022.
- [46] *Stable bias: Analyzing societal representations in diffusion models*. arXiv preprint arXiv:2303.11408, 2023. 2023.

- [47] *Denoising diffusion probabilistic models*. Advances in Neural Information Processing Systems, 33: 6840–6851, 2020. 2020.
- [48] *Diffusion models beat gans on image synthesis*. Advances in neural information processing systems, 34: 8780–8794, 2021. 2021.
- [49] *Cascaded diffusion models for high fidelity image generation*. The Journal of Machine Learning Research, 23 (1): 2249–2281, 2022. 2022.
- [50] *Glide: Towards photorealistic image generation and editing with text-guided diffusion models*. arXiv preprint arXiv:2112.10741, 2021. 2021.
- [51] *Improved denoising diffusion probabilistic models*. In International Conference on Machine Learning, pp. 8162–8171. PMLR, 2021. 2021.
- [52] *Make-a-video: Text-to-video generation without text-video data*. arXiv preprint arXiv:2209.14792, 2022. 2022.
- [53] *Align your latents: High-resolution video synthesis with latent diffusion models*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 22563–22575, 2023. 2023.
- [54] *Text2video-zero: Text-to-image diffusion models are zero-shot video generators*. arXiv preprint arXiv:2303.13439, 2023. 2023.
- [55] *Llm-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models*. arXiv preprint arXiv:2305.13655, 2023. 2023.
- [56] *Grounded text-to-image synthesis with attention refocusing*. arXiv preprint arXiv:2306.05427, 2023. 2023.
- [57] *Emergent abilities of large language models*. arXiv preprint arXiv:2206.07682, 2022. 2022.
- [58] *Language models like gpt-3.* , 2020. 2020.
- [59] *A dataset of 101 human action classes from videos in the wild*. Center for Research in Computer Vision, 2 (11), 2012. 2012.
- [60] *Msr-vtt: A large video description dataset for bridging video and language*. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5288–5296, 2016. 2016.
- [61] *Generative modeling by estimating gradients of the data distribution*. Advances in neural information processing systems, 32, 2019. 2019.
- [62] *Compositional visual generation with composable diffusion models*. In European Conference on Computer Vision, pp. 423–439. Springer, 2022. 2022.
- [63] *Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 22500–22510, 2023. 2023.

- [64] *Diffusion models in vision: A survey*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023. 2023.
- [65] *Diffusion models: A comprehensive survey of methods and applications*. arXiv preprint arXiv:2209.00796, 2022. 2022.
- [66] *Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation*. arXiv preprint arXiv:2212.11565, 2022. 2022.
- [67] *Shape-guided diffusion with inside-outside attention*. arXiv e-prints, pp. arXiv–2212, 2022. 2022.
- [68] *Control-a-video: Controllable text-to-video generation with diffusion models, 2023*. 2023.
- [69] *Generating long videos of dynamic scenes*. Advances in Neural Information Processing Systems, 35: 31769–31781, 2022. 2022.
- [70] *Improved conditional vrns for video prediction*. In Proceedings of the IEEE/CVF international conference on computer vision, pp. 7608–7617, 2019. 2019.
- [71] *Stochastic video generation with a learned prior*. In International conference on machine learning, pp. 1174–1183. PMLR, 2018. 2018.
- [72] *Long video generation with time-agnostic vqgan and time-sensitive transformer*. In European Conference on Computer Vision, pp. 102–118. Springer, 2022. 2022.
- [73] *Cogvideo: Large-scale pretraining for text-to-video generation via transformers*. arXiv preprint arXiv:2205.15868, 2022. 2022.
- [74] *A good image generator is what you need for high-resolution video synthesis*. arXiv preprint arXiv:2104.15069, 2021. 2021.
- [75] *Godiva: Generating open-domain videos from natural descriptions*. arXiv preprint arXiv:2104.14806, 2021. 2021.
- [76] *U-net: Convolutional networks for biomedical image segmentation*. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, pp. 234–241. Springer, 2015. 2015.
- [77] *Imagen video: High definition video generation with diffusion models*. arXiv preprint arXiv:2210.02303, 2022. 2022.
- [78] *Videodirectorgpt: Consistent multi-scene video generation via llm-guided planning*. arXiv preprint arXiv:2309.15091, 2023. 2023.
- [79] *Diffusion self-guidance for controllable image generation*. arXiv preprint arXiv:2306.00986, 2023. 2023.
- [80] *Frozen in time: A joint video and image encoder for end-to-end retrieval*. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1728–1738, 2021. 2021.

- [81] *Grounded language-image pre-training*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10965–10975, 2022. 2022.
- [82] *Simple online and realtime tracking*. In 2016 IEEE International Conference on Image Processing (ICIP), pp. 3464–3468, 2016. 2016.
- [83] *Magicvideo: Efficient video generation with latent diffusion models*. arXiv preprint arXiv:2211.11018, 2022. 2022.
- [84] *Dall-e 3 system card, 2023*. 2023.
- [85] *Llm-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models*. arXiv preprint arXiv:2305.13655, 2023. 2023.
- [86] *Paint by example: Exemplar-based image editing with diffusion models*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 18381–18391, 2023. 2023.
- [87] *Localizing object-level shape variations with text-to-image diffusion models*. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2023. 2023.
- [88] *Controllable text-to-image generation with gpt-4*. arXiv preprint arXiv:2305.18583, 2023. 2023.
- [89] *Prompt-to-prompt image editing with cross attention control*. arXiv preprint arXiv:2208.01626, 2022. 2022.
- [90] *SDEdit: Guided image synthesis and editing with stochastic differential equations*. In International Conference on Learning Representations, 2022. 2022.
- [91] *Diffedit: Diffusion-based semantic image editing with mask guidance*. In The Eleventh International Conference on Learning Representations, 2023. 2023.
- [92] *Plug-and-play diffusion features for text-driven image-to-image translation*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 1921–1930, 2023. 1921.
- [93] *Diffusion self-guidance for controllable image generation*. arXiv preprint arXiv:2306.00986, 2023. 2023.
- [94] *Segment anything*. arXiv preprint arXiv:2304.02643, 2023. 2023.
- [95] *Scaling open-vocabulary object detection*. NeurIPS, 2023. 2023.
- [96] *Microsoft coco captions: Data collection and evaluation server*. arXiv:1504.00325, 2015. 2015.
- [97] *Regionclip: Region-based language-image pretraining*. In CVPR, pages 16793–16803, 2022. 2022.
- [98] *Grounding dino: Marrying dino with grounded pre-training for open-set object detection*. In European Conference on Computer Vision, pages 38–55. Springer, 2024. 2024.

- [99] *Grounded language-image pre-training. 2022 ieee*. In CVPR, pages 10955–10965, 2021. 2022.
- [100] *Gpt-4o system card, 2024*. 2024.
- [101] *Caption anything: Interactive image description with diverse multimodal controls*. arXiv:2305.02677, 2023. 2023.
- [102] *Grit: A generative region-to-text transformer for object understanding*. arXiv:2212.00280, 2022. 2022.
- [103] *Controlcap: Controllable region-level captioning*. pages 21–38, 2024. 2024.
- [104] *Segment and caption anything*. In CVPR, 2024. 2024.
- [105] *Shikra: Unleashing multimodal llm’s referential dialogue magic*. arXiv:2306.15195, 2023. 2023.
- [106] *Gpt4roi: Instruction tuning large language model on region-of-interest*. arXiv:2307.03601, 2023. 2023.
- [107] *Ferret: Refer and ground anything anywhere at any granularity*. In ICLR, 2024. 2024.
- [108] *Osprey: Pixel understanding with visual instruction tuning*. In CVPR, 2024. 2024.
- [109] *Omg-llava: Bridging image-level, object-level, pixel-level reasoning and understanding*. arXiv:2406.19389, 2024. 1938.
- [110] *Draw-and-understand: Leveraging visual prompts to enable MLLMs to comprehend what you want*. In ICLR, 2025. 2025.
- [111] *Referitgame: Referring to objects in photographs of natural scenes*. In EMNLP, 2014. 2014.
- [112] *Generation and comprehension of unambiguous object descriptions*. In CVPR, 2016. 2016.
- [113] *Visual genome: Connecting language and vision using crowdsourced dense image annotations*. IJCV, 2017. 2017.
- [114] *Regiongpt: Towards region understanding vision language model*. In CVPR, 2024. 2024.
- [115] *Cider: Consensus-based image description evaluation*. In CVPR, 2015. 2015.
- [116] *Meteor: An automatic metric for mt evaluation with improved correlation with human judgments*. In ACL Workshops, 2005. 2005.
- [117] *Bleu: a method for automatic evaluation of machine translation*. In ACL, pages 311–318, 2002. 2002.
- [118] *Rouge: A package for automatic evaluation of summaries*. In Text summarization branches out, pages 74–81, 2004. 2004.
- [119] *Spice: Semantic propositional image caption evaluation*. In ECCV, pages 382–398. Springer, 2016. 2016.

- [120] *Vidorefer suite: Advancing spatial-temporal object understanding with video llm.* arXiv:2501.00599, 2024. 2024.
- [121] *Mixmatch: A holistic approach to semi-supervised learning.* NeurIPS, 2019. 2019.
- [122] *Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring.* arXiv:1911.09785, 2019. 1911.
- [123] *Fixmatch: Simplifying semi-supervised learning with consistency and confidence.* NeurIPS, 2020. 2020.
- [124] *Unsupervised data augmentation for consistency training.* NeurIPS, 2020. 2020.
- [125] *Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.* In Workshop on challenges in representation learning, ICML, page 896. Atlanta, 2013. 2013.
- [126] OpenAI. *Learning to reason with LLMs.* 2024. URL: <https://openai.com/index/learning-to-reason-with-llms/>.
- [127] *Openflamingo: An open-source framework for training large autoregressive vision-language models.* arXiv:2308.01390, 2023. 2023.
- [128] *Instructblip: Towards general-purpose vision-language models with instruction tuning. arxiv 2023.* arXiv:2305.06500, 2023. 2023.
- [129] *Obelics: An open web-scale filtered dataset of interleaved image-text documents.* NeurIPS, 2024. 2024.
- [130] *The llama 3 herd of models.* arXiv:2407.21783, 2024. 2024.
- [131] *Visual instruction tuning.* NeurIPS, 2024. 2024.
- [132] *Improved baselines with visual instruction tuning, 2023.* 2023.
- [133] *Llava-next: Improved reasoning, ocr, and world knowledge, 2024.* 2024.
- [134] *On scaling up a multilingual vision and language model.* In CVPR, 2024. 2024.
- [135] *Fuyu-8b: A multimodal architecture for ai agents, 2023.* 2023.
- [136] *Minigpt-4: Enhancing vision-language understanding with advanced large language models.* arXiv:2304.10592, 2023. 2023.
- [137] *Qwen technical report.* arXiv:2309.16609, 2023. 2023.
- [138] *Qwen2-vl: Enhancing vision-language model’s perception of the world at any resolution.* arXiv:2409.12191, 2024. 2024.
- [139] *Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks.* arXiv:2312.14238, 2023. 2023.
- [140] *Vila: On pre-training for visual language models.* In CVPR, 2024. 2024.
- [141] *Paligemma: A versatile 3b vlm for transfer.* arXiv:2407.07726, 2024. 2024.

- [142] *Vila-u: a unified foundation model integrating visual understanding and generation.* arXiv:2409.04429, 2024. 2024.
- [143] *The dawn of lmms: Preliminary explorations with gpt-4v (ision).* arXiv:2309.17421, 2023. 2023.
- [144] *Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v.* arXiv:2310.11441, 2023. 2023.
- [145] *Kosmos-2: Grounding multimodal large language models to the world.* arXiv:2306.14824, 2023. 2023.
- [146] *The all-seeing project: Towards panoptic visual recognition and understanding of the open world.* arXiv:2308.01907, 2023. 1907.
- [147] *Glamm: Pixel grounding large multimodal model.* In CVPR, 2024. 2024.
- [148] *Alpha-clip: A clip model focusing on wherever you want.* In CVPR, 2024. 2024.
- [149] *Merlin: Empowering multimodal llms with foresight minds.* In ECCV, pages 425–443. Springer, 2024. 2024.
- [150] *Artemis: Towards referential understanding in complex videos.* NeurIPS, 37: 114321–114347, 2024. 2024.
- [151] *Ferret-v2: An improved baseline for referring and grounding with large language models.* In COLM, 2024. 2024.
- [152] *Sentence-bert: Sentence embeddings using siamese bert-networks.* arXiv preprint arXiv:1908.10084, 2019. 1908.
- [153] *Focal self-attention for local-global interactions in vision transformers.* arXiv:2107.00641, 2021. 2021.
- [154] *Tosa: Token selective attention for efficient vision transformers.* arXiv:2406.08816, 2024. 2024.
- [155] *Token merging: Your vit but faster.* arXiv:2210.09461, 2022. 2022.
- [156] *Beyond fixation: Dynamic window visual transformer.* In CVPR, 2022. 2022.
- [157] *Vision transformers with mixed-resolution tokenization.* In CVPR, 2023. 2023.
- [158] *V*: Guided visual search as a core mechanism in multimodal llms.* arXiv:2312.14135, 2023. 2023.
- [159] *Sam 2: Segment anything in images and videos.* arXiv:2408.00714, 2024. 2024.
- [160] *An image is worth 16x16 words: Transformers for image recognition at scale.* arXiv:2010.11929, 2020. 2010.
- [161] *Lvis: A dataset for large vocabulary instance segmentation.* In CVPR, 2019. 2019.
- [162] *Paco: Parts and attributes of common objects.* In CVPR, pages 7141–7151, 2023. 2023.

- [163] *Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models*. In ICCV, pages 2641–2649, 2015. 2015.
- [164] *Revisiting referring expression comprehension evaluation in the era of large multimodal models*. arXiv:2406.16866, 2024. 2024.
- [165] *Self-training with noisy student improves imagenet classification*. In CVPR, pages 10687–10698, 2020. 2020.
- [166] *Clair: Evaluating image captions with large language models*. arXiv:2310.12971, 2023. 2023.
- [167] *Claude 3.7 sonnet and claude code, 2025*. 2025.
- [168] *Gemini 2.5: Our most intelligent ai model*. 2025. 2025.
- [169] *Human-centric spatio-temporal video grounding with visual transformers*. IEEE Transactions on Circuits and Systems for Video Technology, 32 (12): 8238–8249, 2021. 2021.
- [170] *Panda-70m: Captioning 70m videos with multiple cross-modality teachers*. In CVPR, pages 13320–13331, 2024. 2024.
- [171] *Gemini: a family of highly capable multimodal models*. arXiv:2312.11805, 2023. 2023.
- [172] *Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling*. arXiv:2412.05271, 2024. 2024.
- [173] *Enhancing the reasoning ability of multimodal large language models via mixed preference optimization*. arXiv:2411.10442, 2024. 2024.
- [174] *Qwen2.5-vl, 2025*. 2025.
- [175] *Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction*. In ICCV, 2021. 2021.
- [176] *Elysium: Exploring object-level perception in videos via mllm*. In ECCV, pages 166–185. Springer, 2024. 2024.
- [177] DeepSeek-AI. “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning”. In: *ArXiv* abs/2501.12948 (2025).
- [178] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. “Self-consistency improves chain of thought reasoning in language models”. In: *ICLR* (2023).
- [179] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. “Tree of thoughts: Deliberate problem solving with large language models”. In: *NeurIPS* (2023).
- [180] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. “Sglang: Efficient execution of structured language model programs”. In: *Advances in neural information processing systems* 37 (2024), pp. 62557–62583.

- [181] Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. *TinyZero*. 2025. URL: <https://github.com/Jiayi-Pan/TinyZero>.
- [182] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. “Chain-of-thought prompting elicits reasoning in large language models”. In: *NeurIPS*. 2022.
- [183] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. “STaR: Bootstrapping Reasoning With Reasoning”. In: *NeurIPS*. 2022.
- [184] Banghua Zhu, Hiteshi Sharma, Felipe Vieira Frueger, Shi Dong, Chenguang Zhu, Michael Jordan, and Jiantao Jiao. *Fine-Tuning Language Models with Advantage-Induced Policy Alignment*. 2024. URL: <https://openreview.net/forum?id=RtOTTdWbZd>.
- [185] Kimi Team. “Kimi k1. 5: Scaling reinforcement learning with llms”. In: *arXiv* 2501.12599 (2025).
- [186] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. “Scaling LLM test-time compute optimally can be more effective than scaling model parameters”. In: *ICLR* (2025).
- [187] Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E Gonzalez, and Ion Stoica. “S*: Test time scaling for code generation”. In: *arXiv preprint arXiv:2502.14382* (2025).
- [188] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. “Training Verifiers to Solve Math Word Problems”. In: *ArXiv* 2110.14168 (2021).
- [189] Yilun Du, Shuang Li, A. Torralba, J. Tenenbaum, and Igor Mordatch. “Improving Factuality and Reasoning in Language Models through Multiagent Debate”. In: *ICML* (2023).
- [190] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. “An LLM compiler for parallel function calling”. In: *ICML*. 2024.
- [191] Gabriel Grand, Joshua B Tenenbaum, Vikash K Mansinghka, Alexander K Lew, and Jacob Andreas. “Self-Steering Language Models”. In: *arXiv preprint arXiv:2504.07081* (2025).
- [192] Philip Schroeder, Nathaniel Morgan, Hongyin Luo, and James Glass. “THREAD: Thinking Deeper with Recursive Spawning”. In: *arXiv preprint arXiv:2405.17402* (2024).
- [193] Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. “Skeleton-of-Thought: Prompting LLMs for Efficient Parallel Generation”. In: *ICLR*. 2024.

- [194] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. “Chain of agents: Large language models collaborating on long-context tasks”. In: *NeurIPS* (2024).
- [195] Wenyue Hua, Mengting Wan, Shashank Vadrevu, Ryan Nadel, Yongfeng Zhang, and Chi Wang. “Interactive speculative planning: Enhance agent efficiency through co-design of system and user interface”. In: *arXiv preprint arXiv:2410.00079* (2024).
- [196] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. “Language Agents as Optimizable Graphs”. In: *ICML* (2024).
- [197] Fengwei Teng, Zhaoyang Yu, Quan Shi, Jiayi Zhang, Chenglin Wu, and Yuyu Luo. “Atom of Thoughts for Markov LLM Test-Time Scaling”. In: *arXiv* 2502.12018 (2025).
- [198] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. “Mixture-of-Agents Enhances Large Language Model Capabilities”. In: *ICLR*. 2025.
- [199] Tian Jin, Ellie Y Cheng, Zack Ankner, Nikunj Saunshi, Blake M Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. “Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding”. In: *arXiv preprint arXiv:2502.11517* (2025).
- [200] Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Vage Egiazarian, Anton Sinitin, Denis Kuznedelev, and Dan Alistarh. “Hogwild! Inference: Parallel LLM Generation via Concurrent Attention”. In: *arXiv* 2504.06261 (2025).
- [201] *PENCIL: Long thoughts with short memory*. In First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models, 2025. 2025.
- [202] *Think before you speak: Training language models with pause tokens*. In *ICLR*, 2024. 2024.
- [203] *Measuring and narrowing the compositionality gap in language models*. In Findings of EMNLP, 2023. 2023.
- [204] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. “DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models”. In: *ArXiv* 2402.03300 (2024).
- [205] Llama Team. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: *arXiv* 2307.09288 (2023).
- [206] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. “Decision transformer: Reinforcement learning via sequence modeling”. In: *NeurIPS* (2021).
- [207] Anthropic. *Introducing Claude Sonnet 4.5*. <https://www.anthropic.com/news/claude-sonnet-4-5>. Sept. 2025.

- [208] Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. “Learning adaptive parallel reasoning with language models”. In: *arXiv preprint arXiv:2504.15466* (2025).
- [209] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. “Efficient Memory Management for Large Language Model Serving with PagedAttention”. In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. 2023.
- [210] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. “Qwen3 technical report”. In: *arXiv preprint arXiv:2505.09388* (2025).
- [211] Mathematical Association of America. *American Invitational Mathematics Examination 2024*. https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination. 2024.
- [212] Mathematical Association of America. *American Mathematics Competitions 2023*. <https://huggingface.co/datasets/AI-MO/aimo-validation-amc>. 2023.
- [213] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. “Measuring mathematical problem solving with the math dataset”. In: *arXiv preprint arXiv:2103.03874* (2021).
- [214] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. “Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems”. In: *arXiv preprint arXiv:2402.14008* (2024).
- [215] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. “Solving quantitative reasoning problems with language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 3843–3857.
- [216] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hamaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. “s1: Simple test-time scaling”. In: *arXiv preprint arXiv:2501.19393* (2025).
- [217] Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. *Universal Self-Consistency for Large Language Model Generation*. 2023. arXiv: 2311.17311 [cs.CL]. URL: <https://arxiv.org/abs/2311.17311>.

- [218] Thang Luong and Edward Lockhart. *Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad*. <https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/>. July 2025.
- [219] Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. *Deep Think with Confidence*. 2025. arXiv: 2508.15260 [cs.LG]. URL: <https://arxiv.org/abs/2508.15260>.
- [220] Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. “Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute”. In: *arXiv preprint arXiv:2509.04475* (2025).
- [221] Yongchao Chen, Jiefeng Chen, Rui Meng, Ji Yin, Na Li, Chuchu Fan, Chi Wang, Tomas Pfister, and Jinsung Yoon. *TUMIX: Multi-Agent Test-Time Scaling with Tool-Use Mixture*. 2025. arXiv: 2510.01279 [cs.CL]. URL: <https://arxiv.org/abs/2510.01279>.
- [222] Tong Wu, Yang Liu, Jun Bai, Zixia Jia, Shuyi Zhang, Ziyong Lin, Yanting Wang, Song-Chun Zhu, and Zilong Zheng. “Native Parallel Reasoner: Reasoning in Parallelism via Self-Distilled Reinforcement Learning”. In: *arXiv preprint arXiv:2512.07461* (2025).
- [223] Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. “Flex attention: A programming model for generating optimized attention kernels”. In: *arXiv preprint arXiv:2412.05496* (2024).
- [224] OpenAI. *Introducing GPT-5*. <https://openai.com/index/introducing-gpt-5/>. Aug. 2025.
- [225] Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. *POLARIS: A Post-Training Recipe for Scaling Reinforcement Learning on Advanced Reasoning Models*. 2025. URL: <https://hkunlp.github.io/blog/2025/Polaris>.
- [226] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. “Understanding rl-zero-like training: A critical perspective”. In: *arXiv preprint arXiv:2503.20783* (2025).
- [227] Mathematical Association of America. *American Invitational Mathematics Examination 2025*. https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination. 2025.

- [228] Qwen, : An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. *Qwen2.5 Technical Report*. 2025. arXiv: 2412.15115 [cs.CL]. URL: <https://arxiv.org/abs/2412.15115>.
- [229] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. “Star: Bootstrapping reasoning with reasoning”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 15476–15488.
- [230] *Denoising diffusion implicit models*. arXiv preprint arXiv:2010.02502, 2020. 2010.
- [231] *Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models*. arXiv preprint arXiv:2211.01095, 2022. 2022.
- [232] *Classifier-free diffusion guidance*. arXiv preprint arXiv:2207.12598, 2022. 2022.
- [233] *Learning transferable visual models from natural language supervision*. In International conference on machine learning, pp. 8748–8763. PMLR, 2021. 2021.
- [234] *Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models*. arXiv preprint arXiv:2211.01095, 2022. 2022.
- [235] *Gpt-4v(ision) system card, 2023*. 2023.
- [236] *Visual instruction tuning*. In NeurIPS, 2023. 2023.
- [237] *A prompt pattern catalog to enhance prompt engineering with chatgpt*. arXiv preprint arXiv:2302.11382, 2023. 2023.
- [238] *Chain-of-thought prompting elicits reasoning in large language models*. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022. 2022.
- [239] *Objects365: A large-scale, high-quality dataset for object detection*. In ICCV, 2019. 2019.
- [240] *Sigmoid loss for language image pre-training*. In ICCV, 2023. 2023.
- [241] *Dataseg: Taming a universal multi-dataset multi-task segmentation model*. NeurIPS, 2024. 2024.
- [242] *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. arXiv:2403.05530, 2024. 2024.
- [243] *Claude 3.5 sonnet, 2024*. 2024.
- [244] *Mose: A new dataset for video object segmentation in complex scenes*. In CVPR, 2023. 2023.
- [245] *The mapillary vistas dataset for semantic understanding of street scenes*. In ICCV, 2017. 2017.

- [246] *Coco-stuff: Thing and stuff classes in context.* *arxiv*. arXiv:1612.03716, 2016. 2016.
- [247] *Openimages: A public dataset for large-scale multi-label and multi-class image classification.* Dataset available from <https://storage.googleapis.com/openimages/web/index.html>, 2017. 2017.
- [248] *The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale.* IJCV, 2020. 2020.
- [249] *Scaling open-vocabulary object detection.* NeurIPS, 2024. 2024.
- [250] *Exploring plain vision transformer backbones for object detection.* In ECCV, 2022. 2022.
- [251] *Cascade r-cnn: High quality object detection and instance segmentation.* TPAMI, 2019. 2019.
- [252] *Eva-02: A visual representation for neon genesis.* Image and Vision Computing, 2024. 2024.
- [253] *Sharegpt4v: Improving large multi-modal models with better captions.* arXiv:2311.12793, 2023. 2023.
- [254] *Graph of thoughts: Solving elaborate problems with large language models.* In AACL, 2024. 2024.
- [255] *Algorithm of thoughts: Enhancing exploration of ideas in large language models.* ICML, 2024. 2024.
- [256] Coleman Hooper, Sehoon Kim, Suhong Moon, Kerem Dilmen, Monishwaran Maheswaran, Nicholas Lee, Michael W. Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. “ETS: Efficient tree search for inference-time scaling”. In: *arXiv* 2502.13575 (2025).
- [257] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: <https://arxiv.org/abs/1707.06347>.
- [258] Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. “Dapo: An open-source llm reinforcement learning system at scale”. In: *arXiv preprint arXiv:2503.14476* (2025).
- [259] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. “Pytorch fsdp: experiences on scaling fully sharded data parallel”. In: *arXiv preprint arXiv:2304.11277* (2023).
- [260] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. “Training Verifiers to Solve Math Word Problems”. In: *arXiv preprint arXiv:2110.14168* (2021).

Appendix A

LLM-grounded Diffusion

A.1 Preliminary introduction to latent diffusion models

The layout-to-image stage (i.e., the image generation stage) of our method LMD builds on off-the-shelf text-to-image Stable Diffusion models, which is based on the latent diffusion framework [13]. We present a preliminary introduction to the latent diffusion framework in this section and define the key terms used in our work. We encourage the readers to check [13] for a detailed explanation of the latent diffusion framework.

Latent diffusion models [13] are powerful generative models that learn the data distribution of complex, high-resolution image datasets. Before training a latent diffusion model, [13] first trains an image encoder that converts an image $\mathbf{x}$ into a vector $\mathbf{z}$ in the high-dimensional *latent* space and a decoder that converts $\mathbf{z}$ back to a vector in the image space that is similar to $\mathbf{x}$ in appearance. By training and sampling a diffusion model in the latent space, latent diffusion lowers the cost of training and sampling from high-resolution diffusion models and is widely used in text-to-image generation, with Stable Diffusion as a popular model based on the latent diffusion framework. Our method improves the prompt understanding of Stable Diffusion without adapting the weights.

During training, the latent diffusion framework first maps each training image, denoted as $\mathbf{x}_0$, into latent $\mathbf{z}_0$ with the image encoder that is frozen during the diffusion training stage:

$$\mathbf{z}_0 = \text{Encode}(\mathbf{x}_0) \quad (\text{A.1})$$

A timestep t is sampled uniformly from $\{1, \dots, T\}$, where T is a hyperparameter.

Noise ϵ is then sampled from a Gaussian distribution parameterized by timestep t and added to the latent $\mathbf{z}_0$ to obtain noisy latent $\mathbf{z}_t$. A neural network with parameter θ learns to predict the added noise ϵ for the forward process by minimizing the training objective:

$$\mathcal{L} = \|\epsilon - \epsilon_\theta(\mathbf{z}_t, t)\|^2 \quad (\text{A.2})$$

The neural network described above often uses a variant of U-Net [76] architecture that has attention layers [22], and thus is also referred to as the diffusion U-Net.

At inference time, there are many sampling methods that allow the synthesis of samples from a diffusion model trained in the fashion described above. The general intuition is to go through a *reverse* process (also called *denoising* process) in which the diffusion model ϵ_θ iteratively predicts a noise vector $\epsilon_\theta(\mathbf{z}_t, t)$ from $\mathbf{z}_t$ and subtracts it to transform $\mathbf{z}_t$ into a sample $\mathbf{z}_{t-1}$ that has less noise and is closer to the distribution of the training set, with t initialized as T and $\mathbf{z}_T \sim \mathcal{N}(0, \mathbf{I})$. The denoised sample $\mathbf{z}_0$ resembles the clean data in the latent space.

One can use DDPM [47] to perform sampling from a noise prediction model ϵ_θ . DDPM predicts the noise ϵ for each of the T denoising steps and then obtains $\mathbf{z}_{t-1}$ from $\mathbf{z}_t$ using this formula:

$$\mathbf{z}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{z}_t - \frac{1 - \alpha_t}{\sqrt{1 - \prod_{i=1}^t \alpha_i}} \epsilon_\theta(\mathbf{z}_t, t) \right) + \sigma_t \epsilon_t \quad (\text{A.3})$$

where $\epsilon_t \sim \mathcal{N}(0, \mathbf{I})$, α_t and σ_t are parameterized by a variance schedule $\{\beta_t \in (0, 1)\}_{t=1}^T$ that controls the size of the denoising step.

Denoising diffusion implicit models (DDIM, *Denoising diffusion implicit models* [230]) are a generalization to DDPM which allows sampling with fewer iterations. DDIM applies the following update rule:

$$\mathbf{z}_{t-1} = \sqrt{\alpha_{t-1}} \left(\frac{\mathbf{z}_t - \sqrt{1 - \alpha_t} \epsilon_\theta(\mathbf{z}_t, t)}{\sqrt{\alpha_t}} \right) + \sigma_t \epsilon_t \quad (\text{A.4})$$

Note that DDIM shares the same training procedure with DDPM, which means we can choose to perform DDIM or DDPM for a trained diffusion model. When σ_t is set to 0, which is the case for our setting, the denoising becomes deterministic given $\mathbf{z}_T$. The results shown in our work are obtained with DDIM with $\sigma_t = 0$, with other faster sampling methods such as *Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models* [231] also applicable to our method.

Since there are many sampling methods given a trained diffusion model that are applicable in the latent diffusion framework, we denote the denoising process, such as the one in Equation (A.3) and Equation (A.4), as

$$\mathbf{z}_{t-1} \leftarrow \text{Denoise}(\mathbf{z}_t) \quad (\text{A.5})$$

After getting the denoised sample $\mathbf{z}_0$, we then decode the image with an image decoder:

$$\mathbf{x}_0 = \text{Decode}(\mathbf{z}_0) \quad (\text{A.6})$$

Text-conditional generation through cross-attention. The above formulation describes the unconditional generation process of latent diffusion models. Models such as Stable Diffusion take text as input and perform conditional generation. The difference between conditional and unconditional generation process involves processing the input text into

text features, passing the feature tokens to diffusion U-Net, and performing classifier-free guidance [232], which is described as follows.

Rather than only taking the noisy input $\mathbf{x}_t$ and timestep t , the conditional diffusion U-Net $\epsilon_\theta(\mathbf{z}_t, t, \tau_\theta(\mathbf{y}))$ takes in an additional text condition $\mathbf{y}$ processed by a text encoder $\tau_\theta(\cdot)$. The text encoder is a CLIP [233] text encoder in Stable Diffusion. After $\mathbf{y}$ is tokenized by the tokenizer into discrete tokens, it is processed by a Transformer [22] to text features $\tau_\theta(\mathbf{y}) \in \mathbb{R}^{l \times d_{\text{text}}}$, where l is the number of text tokens in $\mathbf{y}$ after tokenization and d_{text} is the dimension of features.

The text features $\tau_\theta(\mathbf{y})$ are then processed by the cross-attention layers in the diffusion U-Net so that the output of the U-Net can also change depending on the text. For simplicity, we only consider one cross-attention head in this preliminary introduction and refer the readers to *High-resolution image synthesis with latent diffusion models* [13] and *Attention is all you need* [22] for details with the multi-head cross-attention used in the U-Net in the latent diffusion framework.

Specifically, each cross-attention layer linearly maps the text features $\tau_\theta(\mathbf{y})$ into key and value vectors $\mathbf{k}, \mathbf{v} \in \mathbb{R}^{l \times d_{\text{attn}}}$, where d_{attn} is the attention dimension. Each cross-attention layer also takes in the flattened 2D feature from the previous layer in the U-Net and linearly maps the feature into a query vector $\mathbf{q} \in \mathbb{R}^{m \times d_{\text{attn}}}$ where m is the dimension of the previous flattened 2D image feature.

Then, a cross-attention map $\mathbf{A}$ is computed from the query $\mathbf{q}$, key $\mathbf{k}$, and value $\mathbf{v}$ vectors, which describes the affinity from the image feature to the text token feature:

$$\mathbf{A}_{uv} = \text{Softmax}(\mathbf{q}_u^T \mathbf{k}_v) \quad (\text{A.7})$$

where $\mathbf{q}_u$ and $\mathbf{k}_v$ are linearly transformed image feature at spatial location u and text feature at token index v in the prompt, respectively.

The attention map is then used for computing a weighted combination of the values $\mathbf{v}$:

$$\mathbf{o}_u = \sum_v \mathbf{A}_{uv} \mathbf{v}_v \quad (\text{A.8})$$

$\mathbf{o} \in \mathbb{R}^{m \times d_{\text{attn}}}$ is then linearly transformed to become the output of the cross-attention layer. The residual connections and layer norms are omitted in this introduction for simplicity.

Samples are generated by classifier-free guidance to ensure alignment with text prompt $\mathbf{y}$. At training time, with a small probability, the input condition $\tau_\theta(\mathbf{y})$ is randomly replaced with a learnable null token $\tau_\emptyset$. At inference time, classifier-free guidance uses the following term $\tilde{\epsilon}_\theta(\mathbf{x}_t, t, \tau_\theta(\mathbf{y}))$ in place of the predicted noise $\epsilon_\theta(\mathbf{x}_t, t)$ in the update rule for unconditional generation:

$$\tilde{\epsilon}_\theta(\mathbf{x}_t, t, \tau_\theta(\mathbf{y})) = w \epsilon_\theta(\mathbf{x}_t, t, \tau_\theta(\mathbf{y})) + (1 - w) \epsilon_\theta(\mathbf{x}_t, t, \tau_\emptyset) \quad (\text{A.9})$$

where w is the strength of classifier-free guidance, set to 7.5 by default in Stable Diffusion.

A.2 Pseudo-code for layout-grounded image generation

We present the pseudo-code for our layout-grounding stage (stage 2) in Algorithm 3. We explain the functionality of the functions used in the pseudo-code:

1. **SampleGaussian** samples i.i.d standard Gaussian as the initial noise for the latent tensor.
2. **PromptForBox** simply sets “[*background prompt*] with [*box caption*]” (e.g., “a realistic image of an indoor scene with a gray cat”) as the denoising prompt.
3. **AttnControl** performs backward guidance to minimize the energy function Equation (2.2) described in §2.3 to encourage the attention to the area within the box and discourage the attention on area outside the box. The cross-attention maps $A_t^{(i)}$ are also returned in order to allow obtaining a mask for each box.
4. **Denoise** denotes one denoising step by the diffusion model.
5. **TemporalAverage** averages the cross-attention map across the timestep dimension.
6. **SAMRefine** refines the attention map by internally decoding the latent and refining with SAM. If SAM is not enabled, we perform an attention thresholding instead.
7. **ComposedPrompt** composes the prompt for overall generation. We offer two options for the overall prompt: using the original input prompt or composing the prompt as “[*background prompt*] with [*box caption 1*], [*box caption 2*], ...”. The former one allows capturing the object as well as foreground-background interactions that are not captured in the layout. The latter allows captions in languages unsupported by the diffusion model and stays robust when the caption is misleading (e.g., “neither of the apples is red”). We use the latter by default but also allow the former for fine-grained adjustments.
8. **LatentCompose** spatially composes each of the latents $\mathbf{z}^{(i)}$ with respect to the corresponding mask $\mathbf{m}^{(i)}$, replacing the content of the destination latent on the masked locations. As for the order of composition, we compose the masked latents with the largest area after masking first.
9. **AttnTransfer** performs backward guidance to minimize the energy function Equation (2.7) in §2.3 to encourage the attention in overall generation within the box to be similar to the attention in per-box generation in addition to attention control.

A.3 Additional features and use cases from instruction-based scene specification

As shown in §2.3, LMD, equipped with instruction-based scene specification, allows the user to apply follow-up instruction requests in addition to the initial prompt.

Furthermore, we demonstrate two additional use cases supported by instruction-based scene specification in Fig. A.1 without additional training.

In Fig. A.1(a), instruction-based scene specification allows the users to try out different adjustments on the same generation while preserving the overall image style and layout, facilitating fine-grained content creation.

The LLM equipped in LMD can also respond to open-ended requests and present suggestions for improving the scene. Moreover, different from instruction-based image editing methods that only take one instruction without context, our instruction-based scene specification parses the instruction in its context, allowing for more natural dialog with users. For example, in Fig. A.1(b), our method can respond to instructions with phrases such as “What are some objects that you can add to make it lively?”, “undo the last edit”, and “adding a small pond instead”.

A.4 Additional ablation studies

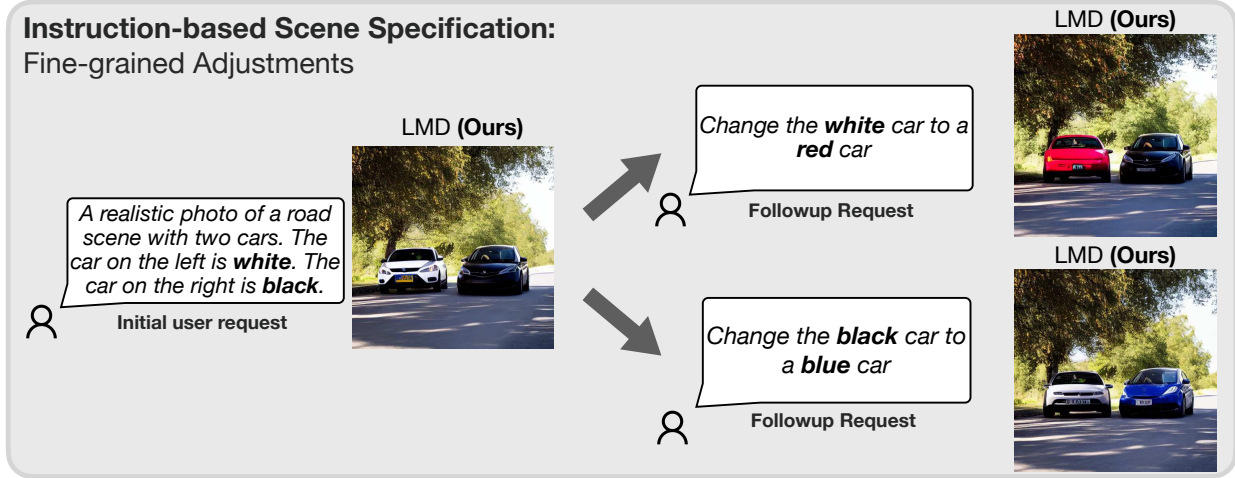
Text-to-layout stage

Varying the LLM types. All LLMs in Tab. A.1 generate layouts that almost perfectly follow the requirements in the prompts, indicating the bottleneck to be the layout-to-image stage. `gpt-4` shows improved results in layout and the subsequent image generation, compared to `gpt-3.5-turbo`. The capability to generate high-quality layouts are *not limited to proprietary LLMs*, with Llama2-based `StableBeluga2` [10, 12, 43] and `Mixtral-8x7B-Instruct-v0.1` [11] also able to perform text-to-layout generation in the stage 1. We believe that fine-tuning these models will lead to even better performance in terms of text-to-layout generation.

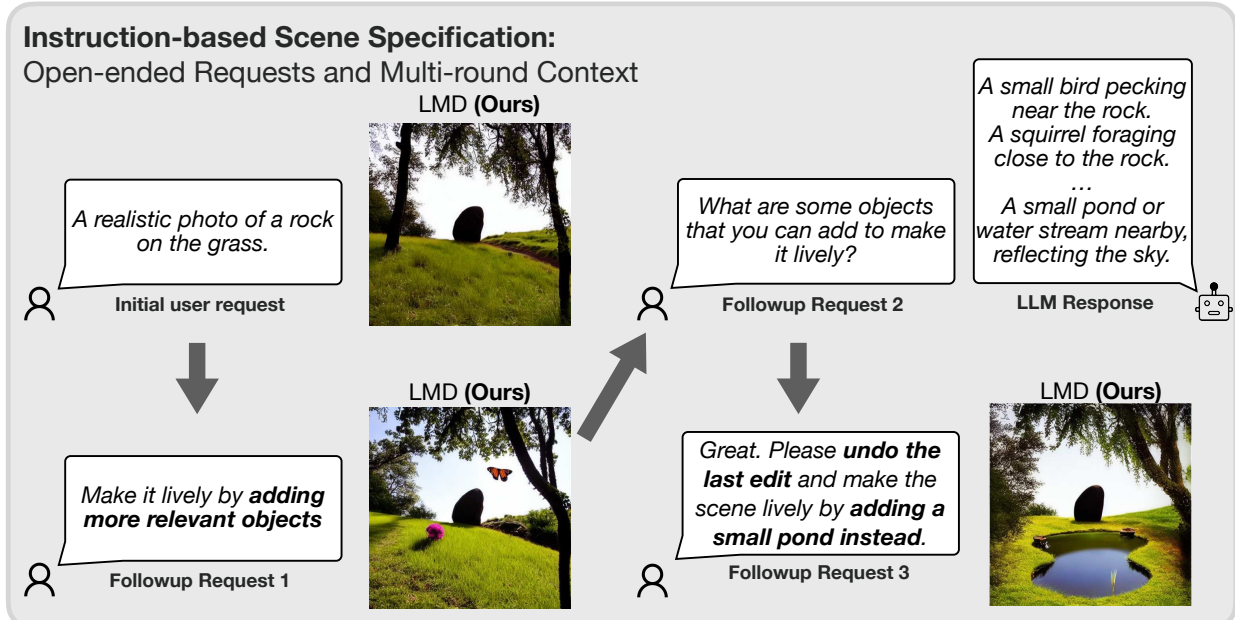
Varying the LLM sizes. We also tested the ability of layout generation on LLMs of different model sizes. As shown in Tab. A.2, larger LLMs offer much better layout generation capabilities.

Layout-to-image stage

Varying ω . ω is the weight for balancing the loss term on the foreground and the term on the background (Equation (2.2)). While we set $\omega = 4$ by default, we ablate this design choice. As shown by the experimental results in Tab. A.3a, our method is relatively stable in terms of hyperparameter selection. Moreover, even though we did not perform hyperparameter search prior to determining our default hyperparameter value, our default hyperparameter



(a) LMD allows the users to try out different detailed adjustments while preserving the overall image style and layout, enabling fine-grained content creation.



(b) The LLM used by LMD can perform open-ended scene adjustments, give suggestions, and understand user requests based on the contexts over multiple rounds of user dialog.

Figure A.1: Additional features and use cases enabled by instruction-based scene specification.

$\omega = 4$ already leads to the optimal performance among the hyperparameter values that we searched in this ablation for both LMD and LMD+.

Varying λ . λ is the weight for the attention transfer term in Equation (2.7). As shown in Tab. A.3b, we found that setting $\lambda = 3$ leads to better performance compared to our default

Stage 1 Model	Layout (Image) Accuracy
	Average of 4 tasks
StableBeluga2	96.5% (67.0%)
Mixtral-8x7B-Instruct-v0.1	98.3% (77.5%)
gpt-3.5-turbo	99.0% (80.5%)
gpt-4	100.0% (86.3%)

Table A.1: **Ablations on different LLMs in stage 1.** Although proprietary models such as GPT-3.5 and GPT-4 perform the best, the ability to generate high-quality layouts is also present in open-source models *Stable beluga models, 2023* [10], *Mixtral of experts* [11], and *Llama 2: Open foundation and fine-tuned chat models, 2023* [12]. The image accuracy is benchmarked using LMD+ as stage 2.

Stage 1 Model	Layout Accuracy
	Average of 4 tasks
StableBeluga-7B	59.3%
StableBeluga-13B	84.0%
StableBeluga2 (70B)	96.5%

Table A.2: **Ablations on the LLM model size on StableBeluga Models [10] based on Llama-2 [12] for layout generation (stage 1 only).** Larger LLMs offer more accurate layout generation compared to smaller LLMs.

hyperparameter setting with $\lambda = 2$, which indicates that the performance of our method can be further improved through hyperparameter tuning.

Ablation results on T2I-CompBench [6]. In addition to comparing our method with the baseline method Stable Diffusion in Tab. 2.6, we further combine our text-to-layout stage (stage 1) with other layout-to-image methods as stage 2 in this ablation, similar to Tab. 2.2. The results are in Tab. A.4, with the results for the SD baseline from [6]. Our method surpasses not only the base diffusion model SD but also several variants of our method that combine our stage 1 with previous layout-to-image methods as stage 2, which shows the effectiveness of our layout-grounded controller.

	Image Accuracy (Average of 4 tasks)			
Method	$\omega = 1$	$\omega = 2$	$\omega = 4$	$\omega = 8$
LMD	72.3%	75.8%	<u>76.5%</u>	72.5%
LMD+	79.8%	80.0%	<u>80.5%</u>	78.3%

(a) Ablations on hyperparameter ω .

	Image Accuracy (Average of 4 tasks)				
Method	$\lambda = 0$	$\lambda = 1$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
LMD	70.8%	75.0%	<u>76.5%</u>	77.3%	75.0%
LMD+	79.3%	79.5%	<u>80.5%</u>	81.8%	78.8%

(b) Ablations on hyperparameter λ .

Table A.3: **Ablations on hyperparameter ω and λ .** Our method is relatively stable in terms of hyperparameter values ω and λ . While we did not perform hyperparameter search, our default hyperparameter $\omega = 4$ allows optimal performance for both LMD and LMD+. For the hyperparameter λ , we found that setting $\lambda = 3$ leads to better performance compared to our default hyperparameter setting with $\lambda = 2$, which indicates that the performance of our method can be further improved through hyperparameter tuning. Underlined numbers indicate performance with our default hyperparameter selection ($\omega = 4$, $\lambda = 2$). **Bold numbers** indicate the best performance among all the hyperparameters ablated.

A.5 Are the generated layouts distributed similarly to the in-context examples?

Since our LLM takes a few in-context examples in our text-to-layout stage, it is possible that the LLM prefers to generate samples that are similar to the in-context examples in terms of spatial distribution. To test whether this is the case, we present the LLM with only one in-context example and query it with a prompt that is similar to the example. The results are shown in Fig. A.2. Even though each of the query prompts shares a similar form to the corresponding in-context example, the LLM still generates layouts that are tailored to the objects in the query prompt (e.g., the apple and the bears) rather than copying or mimicking the layout boxes from the in-context examples. This qualitative analysis shows that even with the in-context examples as references, the LLM often generates natural layouts according to the prompts, relieving the users from heavy prompt engineering to prevent overly similar layouts between the generation and the examples.

Stage 1/Stage 2	Color	Shape	Texture	Spatial
SD	0.3765	0.3576	0.4156	0.1246
LMD/MultiDiffusion	0.4631	0.4497	0.4007	0.1604
LMD/Backward Guidance	0.4877	0.5069	0.4643	0.2361
LMD/Boxdiff	0.4579	0.4967	0.4720	0.1965
LMD/LMD (Ours)	0.5495	0.5462	0.5241	0.2570

Table A.4: Our method surpasses the base diffusion model SD as well as several variants of our method that combines our stage 1 with previous layout-to-image methods as stage 2 on T2I-CompBench [6].

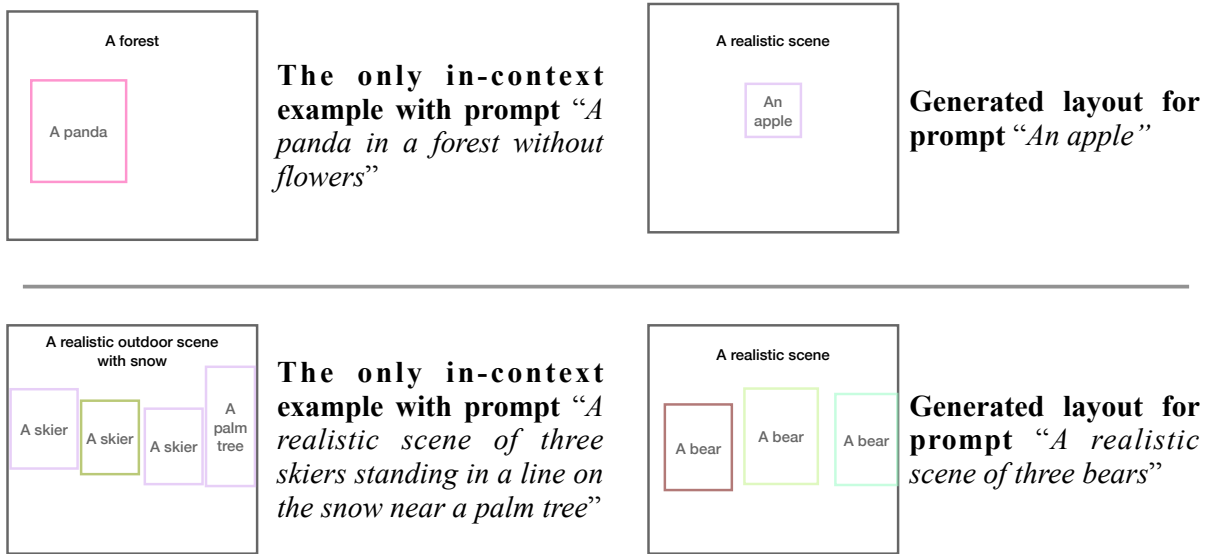


Figure A.2: **The generated layouts are not necessarily similar to the in-context examples in terms of the spatial distribution of boxes.** We present the LLM with *only one in-context example* and query it with a prompt that is similar to the example. **Top:** While the query and the example shares a similar structure (only one object), the LLM generates a box for “an apple” that is very different from “a panda” in terms of the size and position. **Bottom:** The LLM does not simply copy boxes for the three skiers in the in-context example to generate the boxes for three bears.

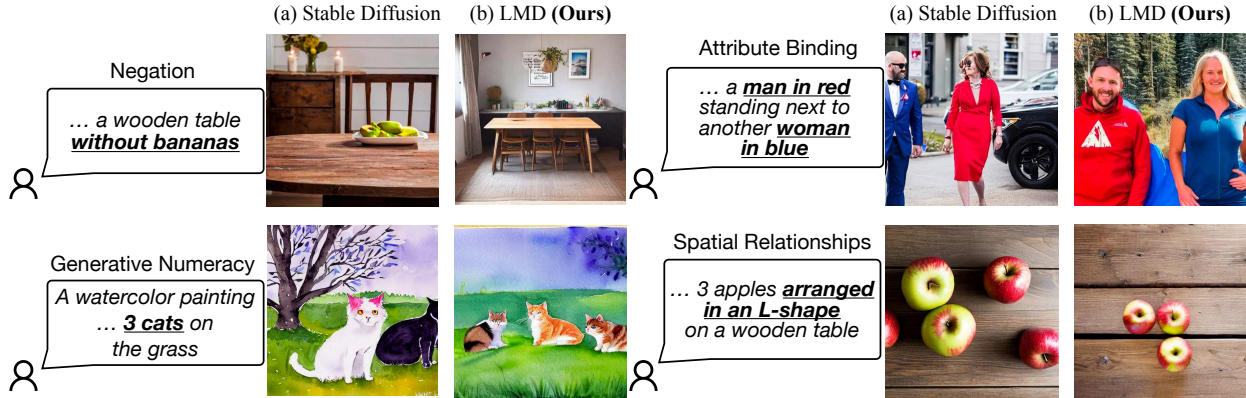


Figure A.3: We also generate images with the same text prompts as Fig. 2.1 with SDv1.5 and LMD on SDv1.5. We observe similar results which show that while Stable Diffusion [13] (a) often struggles to accurately follow several types of complex prompts, our method LMD (b) achieves enhanced prompt understanding capabilities and accurately follows these types of prompts.

A.6 Additional visualizations

We also present Fig. A.3, which includes a qualitative comparison with Stable Diffusion v1.5 (abbreviated as SDv1) and shares the prompts with Fig. 2.1.

A.7 Benchmarking VisualChatGPT and GILL for multi-round instruction-based scene specification

VisualChatGPT [2] and GILL [3] involve LLM in their image generation pipelines and thus could potentially take instructions from multiple rounds of dialog for image generation. Therefore, in addition to the qualitative benchmark in Fig. 2.8, we also benchmark both methods for multi-round scene specification. As shown in Fig. A.4, the generated images quickly degrade starting from the second iteration, showing that neither method is able to take instructions from multiple rounds of dialog for image generation. In contrast, our method is able to handle several rounds of sequential requests on image generation without generation degradation, shown in Fig. 2.6.

A.8 Details for SDXL integration

Thanks to the training-free nature of our work, our method is applicable to various diffusion models without additional training. Therefore, we also apply our method on SDXL

1.0 [1], the latest stable diffusion model which has a $3\times$ larger U-Net module compared to previous stable diffusion models [13].

It is straightforward to apply the LMD pipeline directly to SDXL UNet, which has a very similar procedure to applying the LMD pipeline to SD v1/v2. This approach only requires marginal modifications of the LMD pipeline: different from SD v1/v2 that use only one text encoder for encoding the prompts, SDXL involves two text encoders for text feature generation, and the attention control proposed in LMD needs to be applied to the cross-attention with both text encoders taken into account. The rest follows from the standard LMD pipeline.

Inspired by methods such as *Hierarchical text-conditional image generation with clip latents* [17] that generate low resolution images and then upsample the generation to the target resolution, an alternative approach is to perform denoising with the standard LMD with a standard SDv1/v2 resolution (i.e., 512×512) and then perform upsampling with SDXL refiner for a few steps to the intended resolution (e.g., 1024×1024). Since most of the generation still happens in the standard resolution latent space, with the SDXL only involved a limited number of steps for high-resolution latents, this approach is more efficient compared to the former approach. We compare the generation for the same scene with SDXL baseline and both approaches in Fig. A.5. Both approaches present much better prompt following ability compared to SDXL baseline. We observe similar generation quality on both approaches. Therefore, we use the latter approach by default.

For Fig. 2.1 and Fig. 2.7, we use SDXL 1.0 as the base model of LMD and compare against SDXL as a strong baseline. For all other settings, including the qualitative evaluation setting, we use Stable Diffusion v1.5 (denoted as SDv1) unless stated otherwise. For fair comparison with [2] and [3] that only use Stable Diffusion v1.5, we also generate images for the same set of prompts of Fig. 2.1 with Stable Diffusion v1.5 in Fig. A.3.

A.9 Generating images from languages not supported by the underlying diffusion model

As shown in Fig. A.6, by asking the LLM to always output layouts in English even if the prompt is non-English (e.g., Korean or Chinese as in Fig. A.6) and providing an in-context example of non-English input and English layout, LMD is able to generate images from prompts in languages not supported by the underlying diffusion model. We simply translate the prompt input of the last in-context example to non-English, while keeping the output in this example in English. No adaptation is needed on the diffusion model since the underlying diffusion model still takes in an English layout as input.

A.10 Details for text-to-image benchmarks

We pick 10 common object types from the COCO dataset [35] for generation¹.

For negation and generative numeracy task, each prompt requires the model to generate a layout of a scene with some number of a certain object or without a certain object. Then we count the number of objects and consider the layout to be correct if the number of the object of that particular type matches the one in the prompt, with the number ranging from 1 to 5.

The objective for each prompt in the attribute binding task is to generate an object of a color and another object of another color, for which the evaluation is similar to other tasks.

For the spatial relationship task, we generate an object at a certain location and another object at an opposite location (left/right and top/bottom). We then check the spatial coordinates of the boxes to ensure the layout exactly matches the prompt. In each task, we generate 100 text prompts, with 400 text prompts in total.

Prompts. For the negation benchmark, we use the prompt *A realistic photo of a scene without [object name]*.

For generative numeracy, we use the prompt *A realistic photo of a scene with [number] [object name]*.

For attribute assignment, we use the prompt *A realistic photo of a scene with [modifier 1] [object name 1] and [modifier 2] [object name 2]*, where the two modifiers are randomly chosen from a list of colors (red, orange, yellow, green, blue, purple, pink, brown, black, white, and gray).

For the spatial relationship benchmark, we use the prompt *A realistic photo of a scene with [object name 1] on the [location] and [modifier 2] [object name 2] on the [opposite location]*, where the location is chosen from left, right, top, and bottom.

Implementation details. For LMD, we use Stable Diffusion v1.5 by default. For LMD+, we use GLIGEN [5] model without additional training or adaptation. We selected the GLIGEN [5] model trained based on Stable Diffusion v1.4, which is the latest at the time of writing. We use $\eta = 5$, $\lambda = 2.0$, $r = 0.4$, guidance scale 7.5. The energy minimization is repeated 5 times for each denoising timestep and linearly decreases for every five denoising steps until the repetition is reduced to 1, and we do not perform guidance after 30 steps. k in the $\text{Topk}(\cdot)$ in Equation (2.2) is set to 20% of the area of the mask for each mask. The background part (second term) of Equation (2.2) is weighted by $\omega = 4.0$. We run the denoising process with 50 steps by default. We only perform latent compose in the first half of the denoising process (first 25 steps). The qualitative visualizations/quantitative comparisons are generated by LMD+/LMD, respectively, by default unless stated otherwise.

A.11 Our LLM prompt

Our LLM prompt is listed in Tab. A.5. Our in-context examples are listed in Tab. A.6.

¹Backpack, book, bottle, bowl, car, cat, chair, cup, dog, and laptop.

Algorithm 3 Layout-grounded image generation.

Input: A set of captioned bounding boxes $\{(\mathbf{b}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$. Background caption $\mathbf{y}^{(\text{bg})}$.

Output: Image $\mathbf{x}_0$.

```

1:  $\mathbf{z}_T \leftarrow \text{SampleGaussian}(\mathbf{0}, \mathbf{I})$ 
2: Per-box masked latent generation:
3: for each captioned box  $(\mathbf{b}^{(i)}, \mathbf{y}^{(i)})$  do
4:    $\mathbf{z}_T^{(i)} \leftarrow \mathbf{z}_T$ 
5:    $\mathbf{y}^{(i)} \leftarrow \text{PromptForBox}(\mathbf{y}^{(i)}, \mathbf{y}^{(\text{bg})})$ 
6:   for  $t \leftarrow T$  to 1 do
7:      $\mathbf{z}_t^{(i)}, A_t^{(i)} \leftarrow \text{AttnControl}(\mathbf{z}_t^{(i)}, \mathbf{y}^{(i)}, \mathbf{b}^{(i)})$ 
8:      $\mathbf{z}_{t-1}^{(i)} \leftarrow \text{Denoise}(\mathbf{z}_t^{(i)}, \mathbf{y}^{(i)})$ 
9:   end for
10:   $A^{(i)} \leftarrow \text{TemporalAverage}(A_t^{(i)})$ 
11:   $\mathbf{m}^{(i)} \leftarrow \text{SAMRefine}(A^{(i)}, \mathbf{z}_0^{(i)})$  (Optional: This could be replaced with an attention
    thresholding instead.)
12:   $\hat{\mathbf{z}}_t^{(i)} \leftarrow \mathbf{z}_t^{(i)} \otimes \mathbf{m}^{(i)}$ 
13: end for
14: Composed image generation:
15:  $\mathbf{z}_T^{(\text{comp})} \leftarrow \mathbf{z}_T$ 
16:  $\mathbf{y} \leftarrow \text{ComposedPrompt}((\mathbf{y}^{(i)})_{i=1}^N, \mathbf{y}^{(\text{bg})})$ 
17: for  $t \leftarrow T$  to 1 do
18:   if  $t \geq rT$  then
19:      $\mathbf{z}_t^{(\text{comp})} \leftarrow \text{LatentCompose}(\mathbf{z}_t^{(\text{comp})}, \hat{\mathbf{z}}_t^{(i)}, \mathbf{m}^{(i)}) \quad \forall i$ 
20:      $\mathbf{z}_t^{(\text{comp})} \leftarrow \text{AttnTransfer}(\mathbf{z}_t^{(\text{comp})}, \mathbf{y}^{(\text{comp})}, (A_t^{(i)})_{i=1}^N)$ 
21:   end if
22:    $\mathbf{z}_{t-1}^{(\text{comp})} \leftarrow \text{Denoise}(\mathbf{z}_t^{(\text{comp})}, \mathbf{y}^{(\text{comp})})$ 
23: end for
24:  $\mathbf{x}_0 \leftarrow \text{Decode}(\mathbf{z}_0^{(\text{comp})})$ 

```

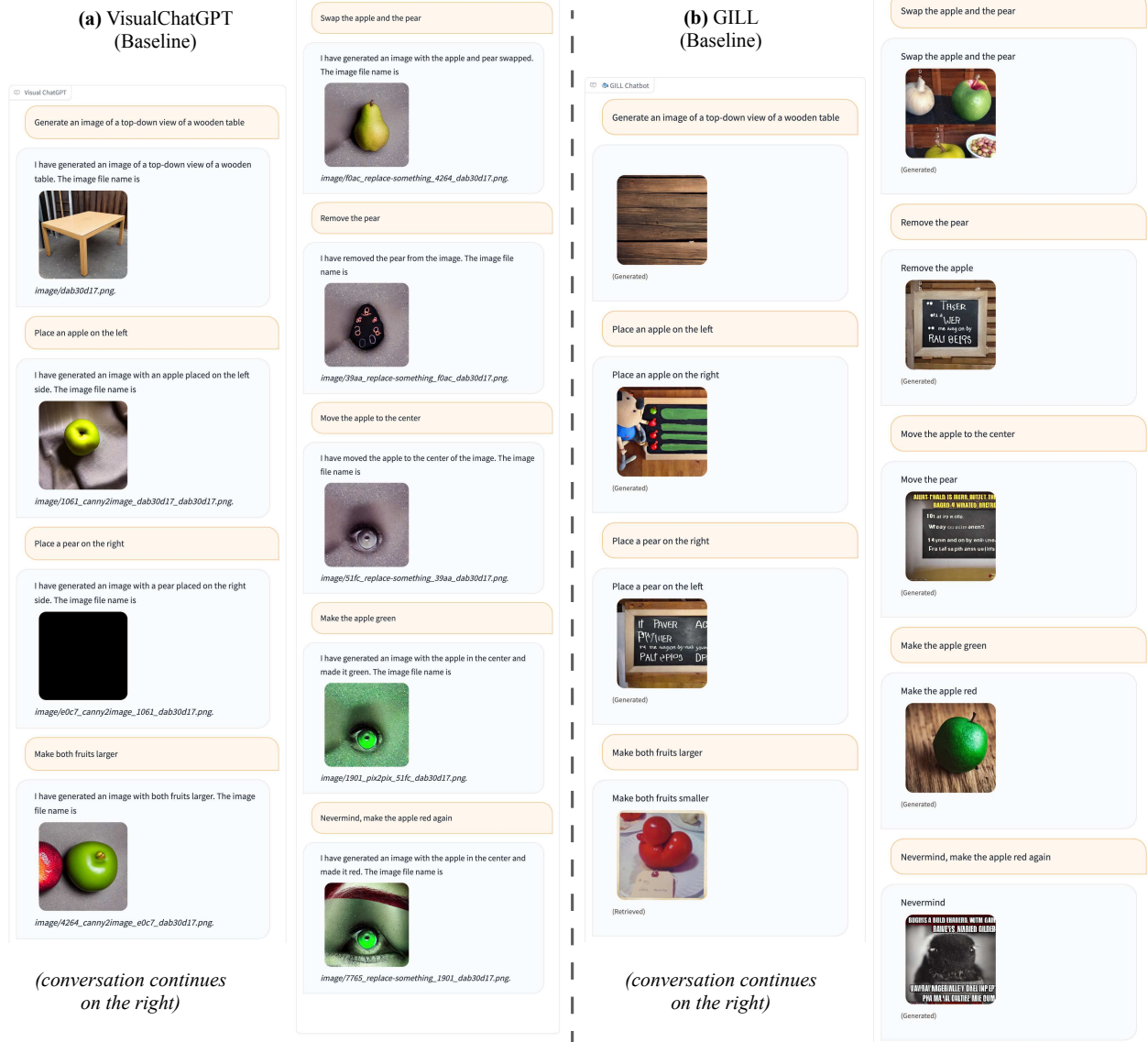


Figure A.4: VisualChatGPT [2] and GILL [3] generally cannot handle more than one round of image generation requests, with the generated image degraded starting from the second request. In contrast, our method is able to handle several rounds of sequential requests on image generation without generation degradation, shown in Fig. 2.6.



Figure A.5: **LMD can be easily applied on the latest stable diffusion model SDXL [1].** We compare generated images from text prompt “A realistic photo of a **gray cat** and an **orange dog** on the grass”. (a) directly generates the image from the text prompt. SDXL does not accurately generate the image from the prompt, showing that *simply scaling the diffusion model does not necessarily lead to improved prompt following ability*. (b) Thanks to our method being training-free, our method can be directly applied on SDXL without additional training. (c) An alternative way to integrate our method with SDXL is to use our method to generate low-resolution images with SD and then refine the image in high-resolution in SDXL. Since most denoising is completed in low-resolution latents, this approach is more efficient.

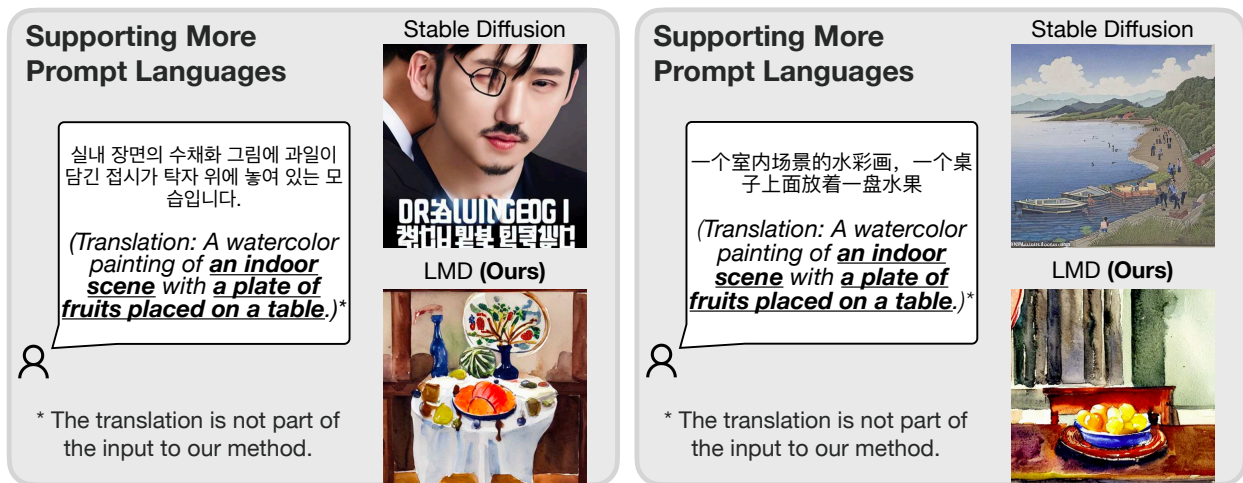


Figure A.6: By asking the LLM to always output layouts in English, **LMD is naturally able to generate images from prompts in languages not supported by the underlying diffusion model.**

1 You are an intelligent bounding box generator. I will provide you with a caption for a photo, image, or painting. Your task is to generate the bounding boxes for the objects mentioned in the caption, along with a background prompt describing the scene. The images are of size 512x512. The top-left corner has coordinate [0, 0]. The bottom-right corner has coordinnate [512, 512]. The bounding boxes should not overlap or go beyond the image boundaries. Each bounding box should be in the format of (object name, [top-left x coordinate, top-left y coordinate, box width, box height]) and should not include more than one object. Do not put objects that are already provided in the bounding boxes into the background prompt. Do not include non-existing or excluded objects in the background prompt. Use "A realistic scene" as the background prompt if no background is given in the prompt. If needed, you can make reasonable guesses. Please refer to the example below for the desired format.

2

3 [In-context Examples]

4

5 Caption: [User Prompt]

6 Objects:

Table A.5: Our full prompt to the LLM for layout generation. LLM starts completion from "Objects:".

```

1 Caption: A realistic image of landscape scene depicting a green car parking on the
  left of a blue truck, with a red air balloon and a bird in the sky
2 Objects: [('a green car', [21, 281, 211, 159]), ('a blue truck', [269, 283, 209,
  160]), ('a red air balloon', [66, 8, 145, 135]), ('a bird', [296, 42, 143,
  100])]
3 Background prompt: A realistic landscape scene
4 Negative prompt:
5
6 Caption: A realistic top-down view of a wooden table with two apples on it
7 Objects: [('a wooden table', [20, 148, 472, 216]), ('an apple', [150, 226, 100,
  100]), ('an apple', [280, 226, 100, 100])]
8 Background prompt: A realistic top-down view
9 Negative prompt:
10
11 Caption: A realistic scene of three skiers standing in a line on the snow near a
  palm tree
12 Objects: [('a skier', [5, 152, 139, 168]), ('a skier', [278, 192, 121, 158]), ('a
  skier', [148, 173, 124, 155]), ('a palm tree', [404, 105, 103, 251])]
13 Background prompt: A realistic outdoor scene with snow
14 Negative prompt:
15
16 Caption: An oil painting of a pink dolphin jumping on the left of a steam boat on
  the sea
17 Objects: [('a steam boat', [232, 225, 257, 149]), ('a jumping pink dolphin', [21,
  249, 189, 123])]
18 Background prompt: An oil painting of the sea
19 Negative prompt:
20
21 Caption: A cute cat and an angry dog without birds
22 Objects: [('a cute cat', [51, 67, 271, 324]), ('an angry dog', [302, 119, 211,
  228])]
23 Background prompt: A realistic scene
24 Negative prompt: birds
25
26 Caption: Two pandas in a forest without flowers
27 Objects: [('a panda', [30, 171, 212, 226]), ('a panda', [264, 173, 222, 221])]
28 Background prompt: A forest
29 Negative prompt: flowers
30
31 Caption: An oil painting of a living room scene without chairs with a painting
  mounted on the wall, a cabinet below the painting, and two flower vases on the
  cabinet
32 Objects: [('a painting', [88, 85, 335, 203]), ('a cabinet', [57, 308, 404, 201]),
  ('a flower vase', [166, 222, 92, 108]), ('a flower vase', [328, 222, 92, 108])]
33 Background prompt: An oil painting of a living room scene
34 Negative prompt: chairs

```

Table A.6: Our in-context examples. We use fixed in-context examples for layout generation.

Appendix B

LLM-grounded Video Diffusion

B.1 Reasoning Statements from GPT-4

As described in §3.3, we ask the model to output a reasoning statement for interpretability. We present examples of the reasoning statements for DSL generation in Tab. B.1. The corresponding DSLs and generated videos are presented in Fig. 3.4, Fig. 3.1, and Fig. 3.5. The first three statements are from ChatGPT web interface. The last four statements are from OpenAI API. All statements are from GPT-4.

These statements provide useful insights about why GPT-4 generates these DSLs in specific ways. For example, the first reasoning statement indicates that GPT-4 takes the flying pattern of birds into account. The second statement shows that GPT-4 applies the buoyancy for the wooden barrel. GPT-4 also considers the reaction of the cat in the third statement. For the fourth statement, GPT-4 demonstrates its understanding that the rock, unlike the ball provided in the in-context example, does not bounce (note that our y-coordinate is top-to-bottom as introduced in the text instructions, so it is correct that the y-coordinate increases).

Interestingly, we observe that GPT-4 with ChatGPT web interface often outputs longer reasoning statements compared to GPT-4 called with APIs. We hypothesize that GPT-4 used in ChatGPT interface has prompts encouraging more explanations rather than following the patterns in the examples.

B.2 Additional Qualitative Visualizations

We present additional qualitative visualizations of LVD in Fig. B.1.

Handling ambiguous prompts. As shown in Fig. B.1(a), our LVD can successfully generate videos for “a walking dog”, which is a prompt that has ambiguity, as the walking direction of the dog is not given. Despite the missing information, the LLM is still able to make a reasonable guess for the walking direction in DSL generation, leading to reasonable video generation. Furthermore, this assumption is included in the reasoning statement:

Table B.1: Reasoning statements from GPT-4.

Query Prompt	LLM Reasoning statements
1 A bird is flying from the left to the right	A bird flying from left to right will have its x-coordinate steadily increasing. Birds tend to fly in a somewhat leveled manner, but they can have slight vertical movements (up and down) due to flapping or gliding. Given the short timeframe of six frames, we'll assume the bird has a slight upward movement initially and then a glide downward, reflecting a flap-glide pattern typical for many birds.
2 A wooden barrel is drifting on a river	A wooden barrel drifting on a river will follow the flow direction, typically moving from left to right or right to left. The barrel, being buoyant, will remain on the surface and will be subjected to the river's currents. As such, the x-coordinate of the barrel will change, indicating its drift, but its y-coordinate will remain largely consistent since it is floating on the surface. For simplicity, let's assume the river flows from left to right.
3 A raccoon is walking towards a cat	The raccoon starts from one side and moves closer to the cat. As both the raccoon and the cat are made larger, their bounding boxes will be more extensive. The raccoon's x-coordinate should increase and it should get closer to the x-coordinate of the cat. The cat might be stationary at first, but as the raccoon approaches, it might move backward a bit due to its natural instinct to maintain a distance. Both the raccoon and the cat will have a constant y-coordinate because they are on the same ground level.
4 A rock is thrown out from the right	A rock is thrown from the right, so its x-coordinate should decrease. Due to gravity, its y-coordinate should increase, and the speed should be faster in later frames until it hits the ground. Unlike a ball, a rock does not bounce back when it hits the ground.
5 A car viewed from the side is driving forward	As the car is driving forward, its x-coordinate should increase over time. The y-coordinate should remain constant as the car is moving on a flat surface. The size of the car should remain constant as it is viewed from the side.
6 A car viewed from the top is driving forward	The car is moving forward, so its y-coordinate should decrease. The x-coordinate should remain the same as there is no lateral movement. The car's size remains the same as the camera is stationary.
7 A car viewed from its back is driving forward	As the car is driving forward and the camera is viewing from the back, the car will appear to get smaller as it moves away from the camera. The x-coordinate and y-coordinate will remain the same, but the width and height of the bounding box will decrease to represent the car moving away.

Reasoning: A dog walking would have a smooth, continuous motion across the frame. Assuming it is walking from left to right, its x-coordinate should gradually increase. The dog’s y-coordinate should remain relatively consistent, with minor fluctuations to represent the natural motion of walking. The size of the bounding box should remain constant since the dog’s size doesn’t change.

This explicitly indicates that the LLM can detect the missing information and make reasonable assumptions for generation.

Handling overlapping boxes. Our DSL-to-video stage can process highly overlapping boxes. In addition to video frames, we also show the dynamic scene layouts generated by the LLM in dashed boxes (white for the raccoon and blue for the barrel) in Fig. B.1(b). Although the box of the raccoon overlaps about half of the barrel, the generation of the raccoon on the barrel is still realistic and the motion still makes sense.

Handling dynamic background. ModelScope often prefers to generate a background with little motion. Although this preference is inherited by our training-free method, our model still works when asked to generate a background that changes significantly. For example, in Fig. B.1(c) we query the model to generate a brown bear taking a shower in a lake when the lightning strikes, and our method generates both successfully, with significant variations on the background. Note that the model generates the effect of rain when only lightning is mentioned in the prompt, which is a reasonable assumption inherited from our base model.

B.3 Our Prompts and In-context Examples for DSL Generation

Tab. B.5 presents the prompts that we use for DSL generation in the quantitative evaluation. To generate DSLs, we replace the “User Text Prompt for DSL Generation” in the template with the text query that we would like to generate DSLs from and pass the prompt to an LLM. We use the chat completion API¹ for both GPT-3.5 and GPT-4. For API calls, line 1-3 are assigned with role “system”. Each caption line in the in-context examples and user query are assigned with role “user”. The example outputs are assigned with role “assistant”. The final “Reasoning” line is omitted. For querying through ChatGPT web interface², we do not distinguish the roles of different lines. Instead, we merge all the lines into one message. The benchmarks are conducted by querying the LLMs through the API calls. The visualizations in Fig. 3.1 are conducted by querying the GPT-4 through the ChatGPT web interface (with horizontal videos generated by Zeroscope³, a model fine-tuned from ModelScope optimized for horizontal video generation).

¹<https://platform.openai.com/docs/guides/gpt/chat-completions-api>

²<https://chat.openai.com/>

³https://huggingface.co/cerspense/zeroscope_v2_576w

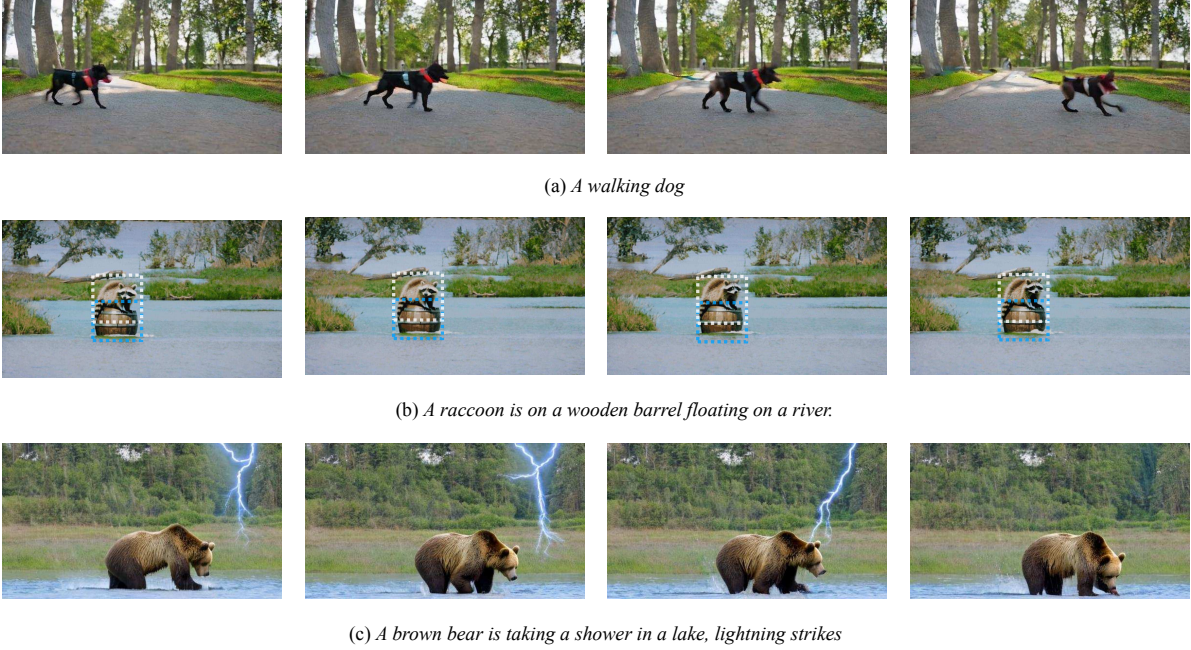


Figure B.1: Additional visualizations of LVD. **(a)** Given a prompt that has ambiguity (the walking direction of the dog is not given), LLM is still able to make a reasonable guess and often includes the assumption in the reasoning statement, leading to reasonable generation. **(b)** Our DSL-to-video stage is able to process highly overlapping boxes (**outlined in the dashed boxes**): the box of the raccoon overlaps about half of the barrel, yet the generation of the raccoon on the barrel is still realistic and not abrupt. **(c)** Although our base video diffusion model ModelScope often generates static background, which is inherited by our training-free method, our model still works when asked to generate background that changes significantly such as lightning and raining. Note that the model generates raining when only lightning is mentioned in the prompt, which is a reasonable assumption.

B.4 Details about Our Setup and Benchmarks

Setup. Since our method is training-free, we introduce our inference setup in this section. For DSL-grounded video generation, we use DPMSolverMultiStep scheduler [231, 234] to denoise 40 steps for each generation. We use the same hyperparams as the baselines, except that we employ DSL guidance. For ModelScope [7], we generate videos of 16 frames. For qualitative assessment in our benchmarks, we generate videos with resolution of both 256×256 and 512×512 for comprehensive comparisons. For FVD evaluation, we generate videos with resolution 256×256 , following our baseline. For evaluator-based assessment, we generate videos with resolution 512×512 with only E_{topk} . For Zeroscope examples in visualizations, we generate 576×320 videos of 24 frames. For DSL guidance, we scale our energy function by a

factor of 5. We perform DSL guidance 5 times per step only in the first 10 steps to allow the model to freely adjust the details generated in the later steps. We apply a background weight of 4.0 and a foreground weight of 1.0 to each of the terms in the energy function, respectively. The k in **Topk** was selected by counting 75% of the positions in the foreground/background in the corresponding term, inspired by previous work for image generation [16]. E_{CoM} is weighted by 0.03 and added to E_{topk} to form the final energy function E . The energy terms for each object, frame, and cross-attention layers are averaged. The “learning rate” for the gradient descent follows $\sqrt{1 - \hat{\alpha}_t}$ for each denoising step t , where the notations are introduced in [48].

Proposed benchmark. To evaluate the alignment of the generated DSLs and videos with text prompts, we propose a benchmark comprising five tasks: generative numeracy, attribute binding, visibility, spatial dynamics, and sequential actions. Each task contains 100 programmatically generated prompts that can be verified given DSL generations or detected bounding boxes using a rule-based metric, with 500 text prompts in total. We then generate 2 videos for each text prompt, resulting in 1000 videos generated. For LLM-grounded experiments, we generate one layout per prompt and generate two videos per layout.

The generative numeracy prompts are generated from template *A realistic lively video of a scene with [number] [object type]*, where object types are sampled from ['car', 'cat', 'bird', 'ball', 'dog'].

The attribute binding prompts are generated from template *A realistic lively video of a scene with a [color1] [object type1] and a [color2] [object type2]* where colors are sampled from ['red', 'orange', 'yellow', 'green', 'blue', 'purple', 'pink', 'brown', 'black', 'white', 'gray'].

The visibility prompts are generated from template *A realistic lively video of a scene in which a [object type] appears only in the [first/second] half of the video.*

The spatial dynamics prompts are generated from template *A realistic lively video of a scene with a [object type] moving from the [left/right] to the [right/left]* and template *A realistic lively video of a scene with a [object type1] moving from the [left/right] of a [object type2] to its [right/left]*.

The sequential actions prompts are generated from template *A realistic lively video of a scene in which a [object type1] initially on the [location1] of the scene. It first moves to the [location2] of the scene and then moves to the [location3] of the scene.* The three locations are sampled from [('lower left', 'lower right', 'upper right'), ('lower left', 'upper left', 'upper right'), ('lower right', 'lower left', 'upper left'), ('lower right', 'upper right', 'upper left')].

UCF-101 and MSR-VTT benchmarks. For UCF-101, we generate 10k videos, each of 16 frames and 256×256 resolution, with either the baseline or our method and using captions randomly sampled from UCF-101 data, and evaluate FVD between the 10k generated videos and videos in UCF-101 following [53, 52]. We first generate a template prompt for each class, use LLM to generate multiple sets of layouts for each prompt so that there are in total 10k video layouts from all the prompts following the class distribution of the training set. We

then calculate the FVD score with the training set of UCF-101. The video resolution is 256×256 and each video contains 16 frames. For evaluation on MSR-VTT, we use LLM to generate layout for each caption of each video in the test set, which contains 2990 videos. We then randomly select 2048 layouts, generate videos for each layout, and calculate the FVD score with the test set videos.

B.5 Failure Cases

As shown in Fig. B.3 (left), the base model [7] does not generate high-quality cartoons for “an apple moving from the left to the right”, with the apples in weird shapes as well as random textures in the background. Although our model is able guide the motion of the apple generation with DSL-guidance, shown in Fig. B.3 (right), our model still inherits the other problems that are present in the baseline generation. Techniques that can be used for enhancing the ability for domain-specific generation such as performing LoRA fine-tuning [36] on the diffusion models may alleviate such issues. Since the scope of this work does not include fine-tuning diffusion models for higher quality generation, we leave the investigation to future work.

Another failure case is shown in Fig. B.4, in which the DSL grounding process actually minimizes the energy function in an unexpected way. Instead of moving the barrel to the right in order to render the drifting effect, the model attempts to hide the barrel on the left with leaves and pop a barrel up from the water to reach the desired energy minimization effect in the DSL grounding process. Reducing the ambiguity caused by cross-attention-based control is needed to resolve this problem. We leave its investigation as a future direction.

We also show a failure case of LLM generating DSLs. From Table 3.1 we show that DSLs generated by GPT4 have 100% accuracy in all categories except for sequential movements. This means LLM can still fail when generating layouts for long-horizon dynamics. Here we show a failure case in Figure B.2. The prompt is *A ball first moves from lower left to upper left of the scene, then moves to upper right of the scene*. We can see that instead of moving to top left, the generated box only moves to slightly above the middle of the frame box, which is inconsistent with the prompt. Since LLM queries are independent, we found that this accounts for most of the fluctuations in the performance in our layout generation stage.

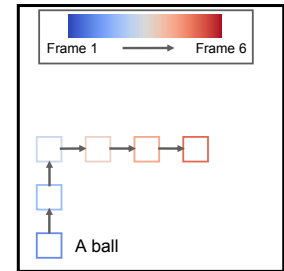


Figure B.2: Failure case of LLM-generated DSL.

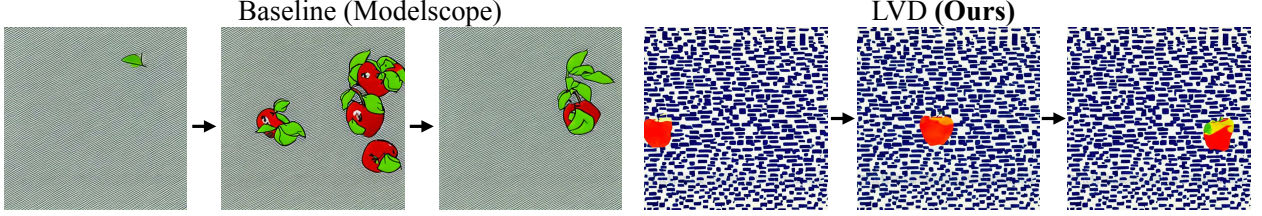


Figure B.3: **One failure case:** *A cartoon of an apple moving from the left to the right.* Since LVD is training-free, it does not improve the quality on domains/styles that the base model is not good at. Although LVD correctly generates a video that is aligned with the prompt, neither the baseline or LVD generates background that fits the apple in the foreground.

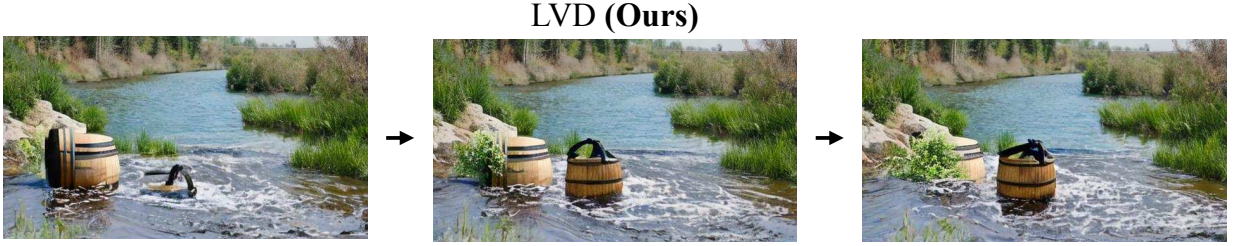


Figure B.4: **Another failure case:** *A wooden barrel is drifting on a river.* Since LVD controls the cross-attention map between the image and the text, the DSL guidance sometimes leads to undesirable generations during the energy minimization process. In this failure case, the model attempts to hide the barrel on the left with leaves and pop a barrel up from the water to reach the desired energy minimization effect, while the DSL means to move the barrel to the right.

B.6 Additional qualitative results with higher video resolution

We also present additional results on our proposed benchmark in Tab. B.2 with resolution 512×512 . We found that as the resolution increases, the performance gaps between our method and baselines are more significant, which is likely because it is easier to control the motion with high-resolution latents and the detection boxes are more accurate and have lower levels of noise in evaluation. Our full method, with E_{CoM} , surpasses the baseline without guidance by a large margin. We also ablated other choices of energy functions, and show that E_{ratio} , proposed in [15], leads to degradation in performance. Adding E_{corner} proposed in [16] leads to similar performance in terms of text-video alignment. Since different types of energy terms are involved, we also experimented with giving more weights to E_{corner} . However, the performance fluctuates between 59.2% and 60.9%, as we give $1\times$ to $10\times$ more weights to E_{corner} . We hypothesize that the spatial normalization term

Table B.2: **Additional results and ablations on videos of higher resolution.** Generated videos are of 512×512 . $\dagger$ indicates unnormed corner energy function.

	E_{topk}	E_{ratio}	E_{corner}	E_{CoM}	Numeracy	Attribution	Visibility	Dynamics	Sequential	Average
No guidance	✗	✗	✗	✗	8%	66%	2%	19%	0%	18.6%
With only E_{ratio}	✗	✓	✗	✗	29%	75%	13%	49%	23%	37.5%
With E_{corner}	✓	✗	✓	✗	64%	94%	47%	66%	36%	60.9%
With $E_{\text{corner}}^{\dagger}$	✓	✗	✓	✗	17%	86%	10%	36%	5%	30.6%
With only E_{topk}	✓	✗	✗	✗	62%	92%	48%	65%	32%	59.6%
With E_{CoM}	✓	✗	✗	✓	56%	92%	62%	75%	52%	67.2%

in E_{corner} leads to its insignificant change in performance, and thus we also implemented a version of the E_{corner} that removed the spatial normalization. However, we observe that this significantly degrades the text-video alignment compared to using only E_{topk} .

B.7 Additional ablation experiments

Ablations on the LLM-grounded DSL generation stage

In-context examples. We test prompts with 1, 3, and 5 in-context examples. We first test the performance of the LLM-grounded DSL generator with each of the 3 in-context examples that we composed. As shown in Tab. B.3, we find that GPT-4 could still generate high-quality DSLs with only 1 in-context example, despite slightly degraded performance and increased performance fluctuation. However, for GPT-3.5, we observe instabilities that two of three runs could not finish the whole benchmark within three attempts for each prompt, mainly because they are not able to generate DSLs that can be parsed (e.g., some DSLs do not have boxes for all the frames). Therefore, we still recommend at least three examples. Furthermore, we composed two additional in-context examples and evaluated the quality of the DSLs with **1)** 3 randomly chosen DSLs from the 5 DSLs, and **2)** all 5 DSLs as in-context examples. We find that our method is stable with different sets of in-context examples when 3 examples are given. In addition, we did not find additional benefits from adding more DSLs. We therefore keep 3 DSLs as our default choice.

Ablations on the DSL-grounded video generation stage

DSL guidance steps. We test 1, 3, 5, and 7 steps of DSL guidance between denoising steps, respectively. The results are shown in Tab. B.4. We can see that the average accuracy improves when adding more DSL guidance steps from 1 to 5 steps, and the performance plateaus when we use more than 5 steps. This means 5 steps is enough and probably the best choice considering the trade-off between performance and efficiency. We therefore use 5 DSL guidance steps.

Table B.3: **Ablations on the number of in-context examples.** We found that 3 in-context examples are sufficient for stable performance. †: We observe instabilities when we only use one example with GPT-3.5, where two of the three runs could not finish due to invalid DSLs generated.

Number of in-context examples	1	3 (default)	5
DSL evaluation accuracy (GPT-4)	96.2%±1.9% [†]	97.3%±0.6%	96.6%

Table B.4: **Ablations on the number of DSL guidance steps between two denoising steps in LVD.** We use 5 DSL guidance steps as the performance saturates with more DSL guidance steps.

DSL guidance steps	1	3	5 (default)	7
Detection-based evaluation accuracy	35.3%	46.9%	49.4%	49.4%

Table B.5: Our prompt for DSL generation. When using API calls, line 1-3 presented to GPT-4 with role “system”. Each caption line in the in-context examples and user query are assigned with role “user”. The example outputs are assigned with role “assistant”. These lines are merged into one message when querying GPT-4 through the ChatGPT web interface.

```

1 You are an intelligent bounding box generator for videos. You don't need to
  generate the videos themselves but need to generate the bounding boxes. I will
  provide you with a caption for a video with six frames, with two frames per
  second. Your task is to generate a list of realistic bounding boxes for the
  objects mentioned in the caption for each frame as well as a background keyword
  . The video frames are of size 512x512. The top-left corner has coordinates [0,
  0]. The bottom-right corner has coordinates [512, 512]. The bounding boxes
  should not overlap or go beyond the frame boundaries.
2
3 Each frame should be represented as `[{'id': unique object identifier incrementing
  from 0, 'name': object name, 'box': [box top-left x-coordinate, box top-left y
  -coordinate, box width, box height]}, ...]`. Each box should not include more
  than one object. Your generated frames must encapsulate the whole scenario
  depicted by the caption. Assume objects move and interact based on real-world
  physics, considering aspects such as gravity and elasticity. Assume the camera
  follows perspective geometry. Boxes for an object should have the same id
  across the frames, even if the object may disappear and reappear. If needed,
  you can make reasonable guesses. Provide a concise reasoning statement that is
  not longer than a few sentences before each generation. Refer to the examples
  below for the desired format. Never use markdown or other formats not in the
  examples. Do not start each frame with `.`. Do not include any comments in your
  response.
4
5 [In-context examples]
6
7 Caption: {User Text Prompt for DSL Generation}
8 Reasoning:

```

Table B.6: Our in-context examples for DSL generation. We use these three in-context examples that we use throughout our work unless stated otherwise.

```

1 Caption: A woman walking from the left to the right and a man jumping on the right
  in a room
2 Reasoning: A woman is walking from the left to the right so her x-coordinate
  should increase with her y-coordinate fixed. A man is jumping on the right so
  his x-coordinate should be large, and his y-coordinate should first decrease (
  upward movement) and then increase (downward movement due to gravity).
3 Frame 1: [{ 'id': 0, 'name': 'walking woman', 'box': [0, 270, 120, 200]}, { 'id': 1,
  'name': 'jumping man', 'box': [380, 290, 120, 180]}]
4 Frame 2: [{ 'id': 0, 'name': 'walking woman', 'box': [50, 270, 120, 200]}, { 'id':
  1, 'name': 'jumping man', 'box': [380, 205, 120, 200]}]
5 Frame 3: [{ 'id': 0, 'name': 'walking woman', 'box': [100, 270, 120, 200]}, { 'id':
  1, 'name': 'jumping man', 'box': [380, 175, 120, 200]}]
6 Frame 4: [{ 'id': 0, 'name': 'walking woman', 'box': [150, 270, 120, 200]}, { 'id':
  1, 'name': 'jumping man', 'box': [380, 175, 120, 200]}]
7 Frame 5: [{ 'id': 0, 'name': 'walking woman', 'box': [200, 270, 120, 200]}, { 'id':
  1, 'name': 'jumping man', 'box': [380, 205, 120, 200]}]
8 Frame 6: [{ 'id': 0, 'name': 'walking woman', 'box': [250, 270, 120, 200]}, { 'id':
  1, 'name': 'jumping man', 'box': [380, 290, 120, 180]}]
9 Background keyword: room
10
11 Caption: A red ball is thrown from the left to the right in a garden
12 Reasoning: A ball is thrown from the left to the right, so its x-coordinate should
  increase. Due to gravity, its y-coordinate should increase, and the speed
  should be faster in later frames until it hits the ground. Due to its
  elasticity, the ball bounces back when it hits the ground.
13 Frame 1: [{ 'id': 0, 'name': 'red ball', 'box': [0, 206, 50, 50]}]
14 Frame 2: [{ 'id': 0, 'name': 'red ball', 'box': [80, 246, 50, 50]}]
15 Frame 3: [{ 'id': 0, 'name': 'red ball', 'box': [160, 326, 50, 50]}]
16 Frame 4: [{ 'id': 0, 'name': 'red ball', 'box': [240, 446, 50, 50]}]
17 Frame 5: [{ 'id': 0, 'name': 'red ball', 'box': [320, 366, 50, 50]}]
18 Frame 6: [{ 'id': 0, 'name': 'red ball', 'box': [400, 446, 50, 50]}]
19 Background keyword: garden
20
21 Caption: The camera is moving away from a painting
22 Reasoning: Due to perspective geometry, the painting will be smaller in later
  timesteps as the distance between the camera and the object is larger.
23 Frame 1: [{ 'id': 0, 'name': 'painting', 'box': [156, 181, 200, 150]}]
24 Frame 2: [{ 'id': 0, 'name': 'painting', 'box': [166, 189, 180, 135]}]
25 Frame 3: [{ 'id': 0, 'name': 'painting', 'box': [176, 196, 160, 120]}]
26 Frame 4: [{ 'id': 0, 'name': 'painting', 'box': [186, 204, 140, 105]}]
27 Frame 5: [{ 'id': 0, 'name': 'painting', 'box': [196, 211, 120, 90]}]
28 Frame 6: [{ 'id': 0, 'name': 'painting', 'box': [206, 219, 100, 75]}]
29 Background keyword: room

```

Appendix C

Self-correcting LLM-controlled Diffusion

Method	Accuracy					
	Negation	Numeracy	Attribute	Spatial	Average	
DALL-E 3 [84]	25%	38%	74%	71%	52.0%	
+ 1-round SLD (OWL-ViT v1)	50%	51%	71%	82%	63.5%	(+11.5)
+ 1-round SLD (OWL-ViT v2)	90%	61%	80%	83%	78.5%	(+26.5)
LMD+ [85]	100%	82%	49%	86%	79.3%	
+ 1-round SLD (OWL-ViT v1)	100%	85%	59%	89%	83.3%	(+ 4.0)
+ 1-round SLD (OWL-ViT v2)	100%	98%	63%	92%	88.3%	(+ 9.0)

Table C.1: Our method can be applied on various image generation methods and improves the generation accuracy by a large margin.

C.1 More Comparisons with Image Editing Methods

In Fig. 4.5 of the main paper, we showcased an example of object replacement contrasting our SLD method with previous approaches. In this section, we provide more visual examples in Fig. C.1 that highlight the differences between SLD and prior diffusion-based image editing methods. While InstructPix2Pix [37] is confined to pixel-aligned changes (e.g., style changes), DiffEdit [91] often fails with precise object-level edits. Our SLD framework excels in these detailed, fine-grained editing tasks.

C.2 Comparison between LMMs and Detector-LLM Combinations

In our main paper, we introduce a self-correction framework that utilizes both LLMs and object detectors for image assessment, followed by the provision of correction suggestions. With the rapid evolution of Large Multimodal Models (LMMs) such as GPT-4V [235] and LLaVA [236, 132], we have investigated the feasibility of using an LMM to conduct image assessment. In this setup, the LMM evaluates images generated by the open-loop generator alongside user prompts, aiming to provide precise, object-level editing recommendations.

However, GPT-4V’s analysis of the “princess and dwarfs” image from DALL-E 3 (refer to Fig. 4.1 in our paper) reveals inaccuracies, as the model miscounts the characters, identifying seven dwarfs rather than the actual five, and struggles to define precise bounding box coordinates. In contrast, as highlighted in Fig. C.2, the latest generation open-vocabulary detector, OWL-ViT v2, demonstrates remarkable proficiency in detecting minute objects (seagulls in the sky), which is vital for the correction process. These limitation underscores our current approach, which combines detectors with LLMs for more accurate assessment and editing suggestions. Nevertheless, the potential of integrating advanced LMMs for streamlined

image generation and editing remains a compelling and promising direction for future research and development.

C.3 Comprehensive Image Generation Results

In the main paper, we demonstrate the significant performance gain achieved by integrating the OWL-ViT v2 open-vocabulary detector into our SLD pipeline. Since this detector is also used in the benchmark, we also explore using other detectors in our setting, which makes our self-correction detector distinct from the detector used in the evaluation benchmark.

Tab. C.1 presents a comparison between the results obtained using OWL-ViT v1 [42] and OWL-ViT v2 [95] detectors within our framework. The results indicate that SLD consistently enhances overall accuracy compared to the baseline text-to-image generators, irrespective of the detector used. The marginal reduction in performance gains when substituting the v2 detector can be attributed to two main factors: **1)** the relatively inferior detection capabilities of the alternate detector employed in the SLD, and **2)** the variations in object recognition between the two detectors. This situation parallels real-world experiences, where individual perceptions and recognition of objects or attributes can vary significantly. For instance, a bowl perceived as distinctly blue by one might be seen as less blue by another. These perceptual variances contribute to the marginally lower attribute binding scores of SLD compared to original DALL-E 3 results. Despite these discrepancies, the overall accuracy of our method, especially in areas such as numeracy and spatial reasoning, confirms the effectiveness of SLD, *regardless of the specific detector used in our self-correction pipeline.*

C.4 Our Prompts and Instructions to the LLM

As outlined in the main paper, we leverage two LLMs to steer the self-correction process: we employ one LLM parser to identify key objects from user prompts and another LLM controller to propose bounding box adjustments. The specific prompts for both the parser and the controller are detailed in Tab. C.2 and Tab. C.3, respectively. We also provide in-context examples for the LLM controller, tailored for self-correcting generation and image editing, in Tab. C.4 and Tab. C.5, respectively.

In crafting our LLM prompts, we emphasized clarity in defining the roles and guidelines for the LLM, drawing inspiration from prior work [237]. Notably, we discovered that GPT-4 possesses the capability to manipulate bounding box coordinates, a task that involves mathematical reasoning. We achieved improved results by guiding the model to employ chain-of-thought reasoning [238], where the model explicates its reasoning process during generation. This approach yielded more accurate suggestions compared to instances where the model’s reasoning was not explicitly stated.

```

1 # Your Role: Excellent Parser
2
3 ## Objective: Analyze scene descriptions to identify objects and their attributes.
4
5 ## Process Steps
6 1. Read the user prompt (scene description).
7 2. Identify all objects mentioned with quantities.
8 3. Extract attributes of each object (color, size, material, etc.).
9 4. If the description mentions objects that shouldn't be in the image, take note at the negation part.
10 5. Explain your understanding (reasoning) and then format your result (answer / negation) as shown in the
    examples.
11 6. Importance of Extracting Attributes: Attributes provide specific details about the objects. This helps
    differentiate between similar objects and gives a clearer understanding of the scene.
12
13 ## Examples
14
15 - Example 1
16   User prompt: A brown horse is beneath a black dog. Another orange cat is beneath a brown horse.
17   Reasoning: The description talks about three objects: a brown horse, a black dog, and an orange cat. We
    report the color attribute thoroughly. No specified negation terms.
18   Objects: [('horse', ['brown']), ('dog', ['black']), ('cat', ['orange'])]
19   Negation:
20
21 - Example 2
22   User prompt: There's a white car and a yellow airplane in a garage. They're in front of two dogs and
    behind a cat. The car is small. Another yellow car is outside the garage.
23   Reasoning: The scene has two cars, one airplane, two dogs, and a cat. The car and airplane have colors.
    The first car also has a size. No specified negation terms.
24   Objects: [('car', ['white and small', 'yellow']), ('airplane', ['yellow']), ('dog', [None, None]),
    ('cat', [None])]
25   Negation:
26
27 - Example 3
28   User prompt: A car and a dog are on top of an airplane and below a red chair. There's another dog sitting
    on the mentioned chair.
29   Reasoning: Four objects are described: one car, airplane, two dog, and a chair. The chair is red color.
    No specified negation terms.
30   Objects: [('car', [None]), ('airplane', [None]), ('dog', [None, None]), ('chair', ['red'])]
31   Negation:
32
33 - Example 4
34   User prompt: An oil painting at the beach of a blue bicycle to the left of a bench and to the right of a
    palm tree with five seagulls in the sky.
35   Reasoning: Here, there are five seagulls, one blue bicycle, one palm tree, and one bench. No specified
    negation terms.
36   Objects: [('bicycle', ['blue']), ('palm tree', [None]), ('seagull', [None, None, None, None, None]),
    ('bench', [None])]
37   Negation:
38
39 - Example 5
40   User prompt: A realistic photo of a scene without backpacks.
41   Reasoning: The description clearly states no backpacks, so this must be acknowledged. The user provides
    the negative prompt of backpacks.
42   Objects: [('backpacks', [None])]
43   Negation: backpacks
44
45 Your Current Task: Follow the steps closely and accurately identify objects based on the given prompt. Ensure
    adherence to the above output format.
46
47 User prompt: {the input user prompt}
48 Reasoning:

```

Table C.2: Our full prompt for the LLM parser.

```

1 # Your Role: Expert Bounding Box Adjuster
2
3 ## Objective: Manipulate bounding boxes in square images according to the user prompt while maintaining
   visual accuracy.
4
5 ### Bounding Box Specifications and Manipulations
6 1. Image Coordinates: Define square images with top-left at [0, 0] and bottom-right at [1, 1].
7 2. Box Format: [Top-left x, Top-left y, Width, Height]
8 3. Operations: Include addition, deletion, repositioning, and attribute modification.
9
10 ## Key Guidelines
11 1. Alignment: Follow the user's prompt, keeping the specified object count and attributes.
12 2. Boundary Adherence: Keep bounding box coordinates within [0, 1].
13 3. Minimal Modifications: Change bounding boxes only if they don't match the user's prompt.
14 4. Overlap Reduction: Minimize intersections in new boxes and remove the smallest, least overlapping objects.
15
16 ## Process Steps
17 1. Interpret prompts: Read and understand the user's prompt.
18 2. Implement Changes: Review and adjust current bounding boxes to meet user specifications.
19 3. Explain Adjustments: Justify the reasons behind each alteration and ensure every adjustment abides by the
   key guidelines.
20 4. Output the Result: Present the reasoning first, followed by the updated objects section, which should
   include a list of bounding boxes in Python format.
21
22 ## Examples
23
24 {In-context examples for self-correcting image generation or image editing}
25
26 Your Task: Carefully follow the provided guidelines and steps to adjust bounding boxes in accordance with the
   user's prompt. Ensure adherence to the above output format.
27
28 User prompt: {the input user prompt}
29 Current Objects: {a list of detected key objects}
30 Reasoning:

```

Table C.3: Our full prompt for the LLM controller.

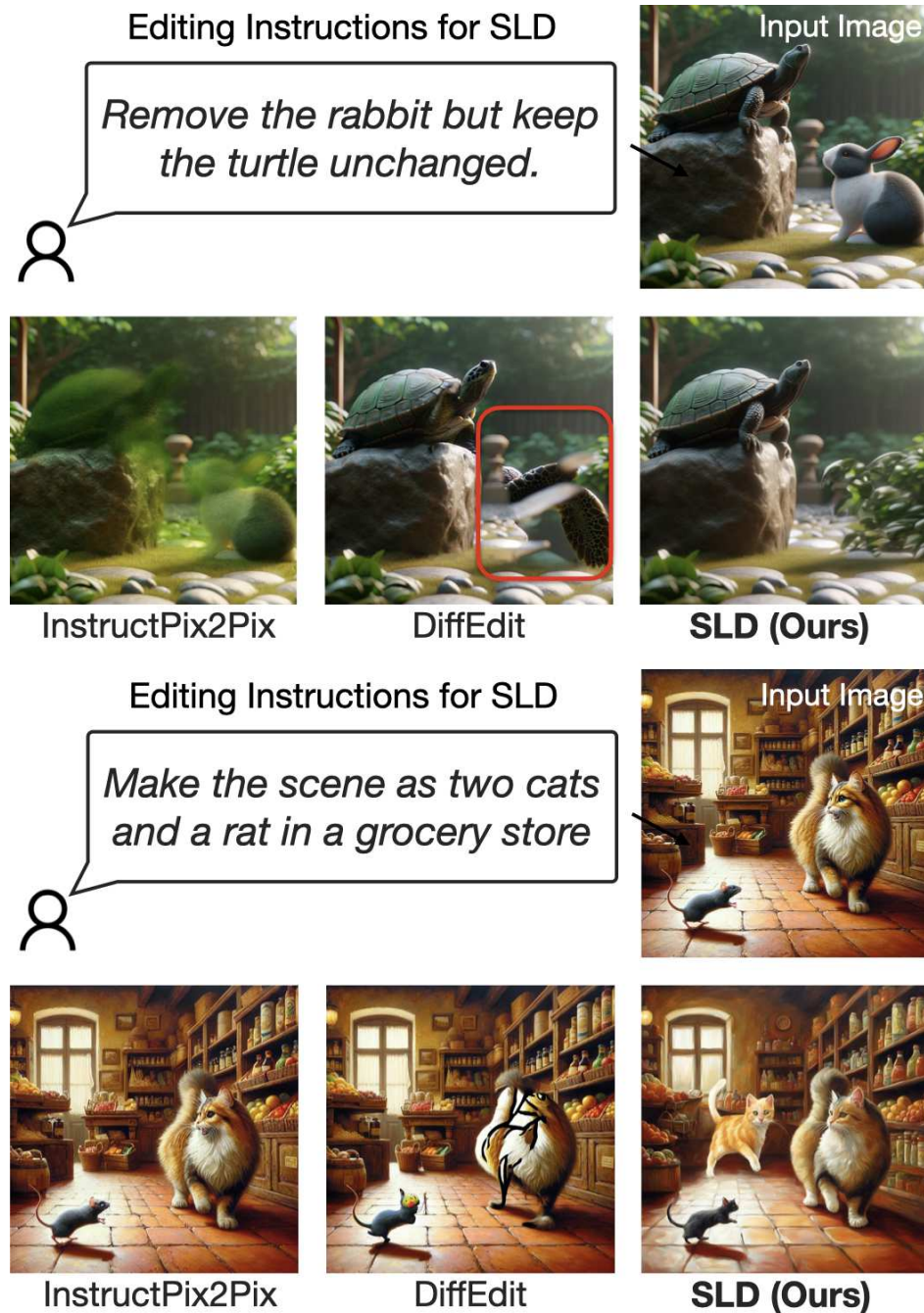
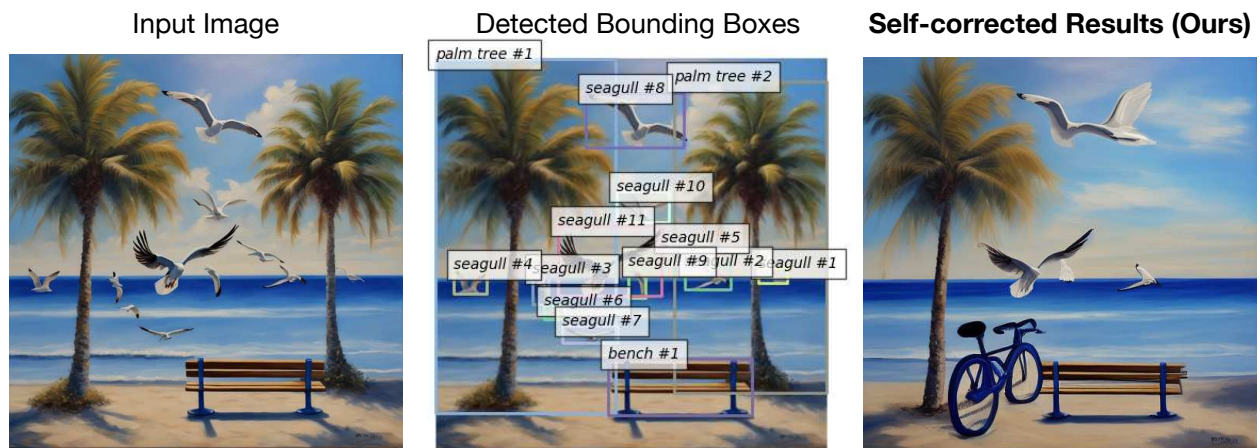


Figure C.1: We demonstrate that current diffusion-based image editing methods, such as InstructPix2Pix and DiffEdit, face challenges in fundamental operations like object deletion (top example) and addition (bottom example). In contrast, our proposed SLD pipeline can easily handle these tasks. As highlighted in the red box, despite DiffEdit’s ability to identify the object for removal, it falls short in generating a plausible background.



*An oil painting at the beach of a **blue bicycle** to the left of a bench and to the right of a palm tree with **three seagulls** in the sky*

Figure C.2: Leveraging the advanced localization abilities of OWL-ViT v2 open-vocabulary detectors, we accurately identify all seagulls in the image, enabling selective removal to align with the user’s prompt.

1	# Examples
2	
3	- Example 1
4	User prompt: A realistic image of landscape scene depicting a green car parking on the left of a blue truck, with a red air balloon and a bird in the sky
5	Current Objects: [('green car #1', [0.027, 0.365, 0.275, 0.207]), ('blue truck #1', [0.350, 0.368, 0.272, 0.208]), ('red air balloon #1', [0.086, 0.010, 0.189, 0.176])]
6	Reasoning: To add a bird in the sky as per the prompt, ensuring all coordinates and dimensions remain within [0, 1].
7	Updated Objects: [('green car #1', [0.027, 0.365, 0.275, 0.207]), ('blue truck #1', [0.350, 0.369, 0.272, 0.208]), ('red air balloon #1', [0.086, 0.010, 0.189, 0.176]), ('bird #1', [0.385, 0.054, 0.186, 0.130])]
8	
9	- Example 2
10	User prompt: A realistic image of landscape scene depicting a green car parking on the right of a blue truck, with a red air balloon and a bird in the sky
11	Current Output Objects: [('green car #1', [0.027, 0.365, 0.275, 0.207]), ('blue truck #1', [0.350, 0.369, 0.272, 0.208]), ('red air balloon #1', [0.086, 0.010, 0.189, 0.176])]
12	Reasoning: The relative positions of the green car and blue truck do not match the prompt. Swap positions of the green car and blue truck to match the prompt, while keeping all coordinates and dimensions within [0, 1].
13	Updated Objects: [('green car #1', [0.350, 0.369, 0.275, 0.207]), ('blue truck #1', [0.027, 0.365, 0.272, 0.208]), ('red air balloon #1', [0.086, 0.010, 0.189, 0.176]), ('bird #1', [0.485, 0.054, 0.186, 0.130])]
14	
15	- Example 3
16	User prompt: An oil painting of a pink dolphin jumping on the left of a steam boat on the sea
17	Current Objects: [('steam boat #1', [0.302, 0.293, 0.335, 0.194]), ('pink dolphin #1', [0.027, 0.324, 0.246, 0.160]), ('blue dolphin #1', [0.158, 0.454, 0.376, 0.290])]
18	Reasoning: The prompt mentions only one dolphin, but two are present. Thus, remove one dolphin to match the prompt, ensuring all coordinates and dimensions stay within [0, 1].
19	Updated Objects: [('steam boat #1', [0.302, 0.293, 0.335, 0.194]), ('pink dolphin #1', [0.027, 0.324, 0.246, 0.160])]
20	
21	- Example 4
22	User prompt: An oil painting of a pink dolphin jumping on the left of a steam boat on the sea
23	Current Objects: [('steam boat #1', [0.302, 0.293, 0.335, 0.194]), ('dolphin #1', [0.027, 0.324, 0.246, 0.160])]
24	Reasoning: The prompt specifies a pink dolphin, but there's only a generic one. The attribute needs to be changed.
25	Updated Objects: [('steam boat #1', [0.302, 0.293, 0.335, 0.194]), ('pink dolphin #1', [0.027, 0.324, 0.246, 0.160])]
26	
27	- Example 5
28	User prompt: A realistic photo of a scene with a brown bowl on the right and a gray dog on the left
29	Current Objects: [('gray dog #1', [0.186, 0.592, 0.449, 0.408]), ('brown bowl #1', [0.376, 0.194, 0.624, 0.502])]
30	Reasoning: The leftmost coordinate (0.186) of the gray dog's bounding box is positioned to the left of the leftmost coordinate (0.376) of the brown bowl, while the rightmost coordinate (0.186 + 0.449) of the bounding box has not extended beyond the rightmost coordinate of the bowl. Thus, the image aligns with the user's prompt, requiring no further modifications.
31	Updated Objects: [('gray dog #1', [0.186, 0.592, 0.449, 0.408]), ('brown bowl #1', [0.376, 0.194, 0.624, 0.502])]

Table C.4: Self-correction in-context examples for the LLM controller.

```

1 ## Examples
2
3 - Example 1
4   User prompt: Move the green car to the right and make the blue truck larger in the image.
5   Current Objects: [('green car #1', [0.027, 0.365, 0.275, 0.207]), ('blue truck #1', [0.350, 0.368, 0.272,
6   0.208])]
7   Reasoning: To move the green car rightward, its x-coordinate needs to be increased from 0.027. The
8   dimensions (height and width) of the blue truck must be enlarged. While adjusting bounding boxes, ensure
9   they do not overlap excessively. All other elements remain unchanged.
10  Updated Objects: [('green car #1', [0.327, 0.365, 0.275, 0.207]), ('blue truck #1', [0.350, 0.369, 0.472,
11  0.408])]
12
13 - Example 2
14   User prompt: Swap the positions of a green car and a blue truck in this landscape scene with an air
15   balloon.
16   Current Output Objects: [('green car #1', [0.350, 0.369, 0.275, 0.207]), ('blue truck #1', [0.027, 0.365,
17   0.272, 0.208]), ('red air balloon #1', [0.086, 0.010, 0.189, 0.176])]
18   Reasoning: Exchange locations of the car and truck to align the bottom right part; other objects remain
19   unchanged.
20   Updated Objects: [('green car #1', [0.027, 0.365, 0.275, 0.207]), ('blue truck #1', [0.350, 0.364,
21   0.272, 0.208]), ('red air balloon #1', [0.086, 0.010, 0.189, 0.176])]
22
23 - Example 3
24   User prompt: Change the color of the dolphin from blue to pink in this oil painting of a dolphin and a
25   steamboat.
26   Current Objects: [('steam boat #1', [0.302, 0.293, 0.335, 0.194]), ('blue dolphin #1', [0.027, 0.324,
27   0.246, 0.160])]
28   Reasoning: Alter only the dolphin's color from blue to pink, without modifying other elements.
29   Updated Objects: [('steam boat #1', [0.302, 0.293, 0.335, 0.194]), ('pink dolphin #1', [0.027, 0.324,
30   0.246, 0.160])]
31
32 - Example 4
33   User prompt: Remove the leftmost bowl in this photo with two bowls and a dog.
34   Current Objects: [('dog #1', [0.186, 0.592, 0.449, 0.408]), ('bowl #1', [0.376, 0.194, 0.324, 0.324]),
35   ('bowl #2', [0.676, 0.494, 0.324, 0.324])]
36   Reasoning: There are two bowls in the image and bowl #1 is identified as the leftmost one because its x
37   coordinates (0.376) is smaller than that of bowl #2 (0.676). Thus, eliminate bowl #1 without modifying
38   any remaining instances.
39   Updated Objects: [('dog #1', [0.186, 0.592, 0.449, 0.408]), ('bowl #2', [0.676, 0.494, 0.324, 0.324])]
40
41 - Example 5
42   User prompt: Add a pink bowl between two existing bowls in this photo.
43   Current Objects: [('bowl #1', [0.076, 0.494, 0.324, 0.324]), ('bowl #2', [0.676, 0.494, 0.324, 0.324])]
44   Reasoning: There are two bowls in the existing image. To add a pink bowl between the two, the x
45   coordinates should be placed between 0.076 and 0.676 and the y coordinates should be between 0.494 and
46   0.494. When adding the object, be sure to prevent overlapping between existing objects and make sure the
47   [top-left x-coordinate, top-left y-coordinate, top-left x-coordinate+box width, top-left
48   y-coordinate+box height] lie between 0 and 1.
49   Updated Objects: [('bowl #1', [0.076, 0.494, 0.324, 0.324]), ('bowl #2', [0.676, 0.494, 0.324, 0.324]),
50   ('bowl #3', [0.376, 0.494, 0.324, 0.324])]

```

Table C.5: Image editing in-context examples for the LLM controller.

Appendix D

Describe Anything

D.1 Challenges in Generating Detailed Localized Descriptions with Off-the-Shelf VLMs

Although cutting-edge Vision-Language Models (VLMs), such as GPT-4o [100] and LLaVA [131, 133, 132], excel at generating global-level image descriptions, producing detailed *localized* image captions remains an open problem. Specifically, these VLMs only take in RGB images along with text prompts and do not allow users to accurately specify regions of interest.

While users could employ text to localize the object to be described, this approach is often cumbersome and inefficient, requiring precise referring phrases that may still be difficult for the VLM to interpret. This can lead to mislocalization of the intended object, as illustrated in Fig. D.1(a).

The required effort for both the user and the model can be significantly reduced if the user is allowed to specify the region directly using a representation in 2D coordinates that the model can understand. With this idea in mind, we focus on *generating detailed localized descriptions* by enabling users to specify a region in an image for the model to describe in detail. Since spatial representations such as points and boxes can be converted into masks using SAM [94] and SAM 2 [159], we concentrate on regions specified by mask inputs.

A first attempt to address this problem with existing VLMs is to reduce the task to global image captioning by presenting only the region to the VLM through masking or cropping, as shown in Fig. D.1(b). While this forces the VLM to focus solely on the specified region, freeing users from the burden of expressing localizations as phrases, the lack of contextual information makes the task much more challenging and often confuses the VLM. This confusion can prevent the model from correctly identifying the object, let alone providing detailed descriptions of its parts. In more extreme cases, the model may even refuse to caption the region due to insufficient information in the cropped or masked image. Therefore, generating detailed localized captions requires more than just the local region.

An alternative approach to prompt existing off-the-shelf VLMs for localized descriptions

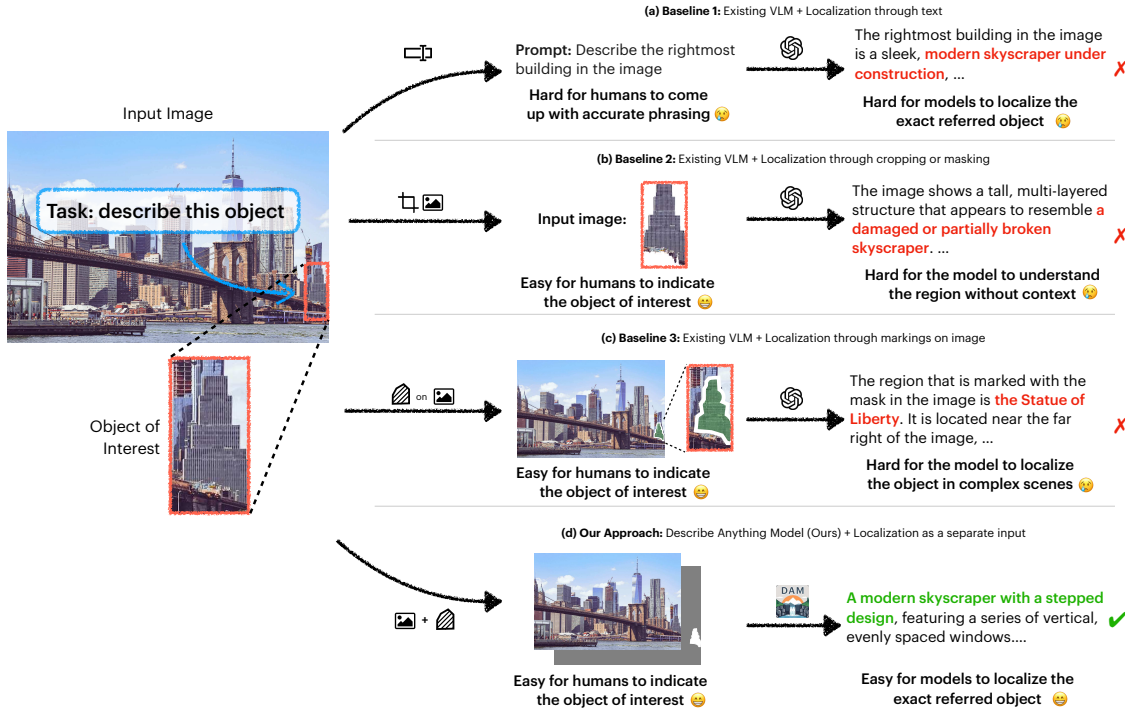


Figure D.1: **Existing Vision-Language Models (VLMs) do not perform well in generating localized descriptions.** (a) to (c) demonstrate several ways to prompt existing VLMs, but none achieves satisfactory performance, leading to the need for a new method that is capable of providing detailed and localized descriptions. In (d), we propose a model that accepts the condition in a separate form of input, making it easy for users to specify the object of interest and for the models to accurately localize the referred object. Note that our focal prompt, proposed in §5.3, is considered part of the Describe Anything Model and is not shown in the figure for simplicity.

is to overlay markings such as points, scribbles, contours, and alpha masks on the image [143, 144], as shown in Fig. D.1(c). However, these markings may blend into the object or the background in highly complex scenes, making them unrecognizable to the VLMs. This issue is especially common for small objects that are not the main focus of the scene. Furthermore, the markings may render the image out-of-distribution, confusing the VLMs and disrupting the quality of output that they were originally capable of generating.

The exploration above highlights a conflict between the precision of localization and the availability of context. On one hand, we want the model to accurately focus on a specific region without mentioning other regions, such as other objects or the background. On the other hand, the model needs to leverage contextual information to correctly identify the object in the region of interest. This conflict makes it very difficult for current VLMs to

produce high-quality localized descriptions.

Our proposed model overcomes this challenge by taking the localization as a *separate* input in 2D space. This approach has the advantage of making the localization more explicit for the VLMs to parse while keeping the image within its original distribution, thus preventing the model from being distracted by the markings. This technique leads to accurate localization even in complex scenes, as illustrated in Fig. D.1(d). Note that since Fig. D.1(d) mainly focuses on explaining the design choices of inputting mask inputs to the model, focal prompting is included as a part of the model and is omitted in this figure for simplicity. We refer readers to Fig. 5.2 for illustrations on focal prompting.

D.2 Evaluation Benchmarks

Our DAM is designed to perform well at *localized image and video captioning* across *multiple granularities*, including keyword, phrase, and detailed captions. Therefore, we evaluate and achieve SOTA in 7 in-domain and zero-shot benchmarks:

1. The LVIS open-class keyword-level benchmark in Tab. 5.2.
2. PACO open-class keyword-level benchmark (including object and parts as regions) in Tab. 5.2.
3. Flickr30k Entities phrase-level benchmark in Tab. 5.3.
4. Ref-L4 detailed captioning benchmark in Tab. 5.4.
5. Our proposed DLC-Bench detailed localized captioning benchmark in Tab. 5.5.
6. HC-STVG detailed video captioning benchmark in Tab. 5.6.
7. VideoRefer detailed video captioning benchmark in Tab. 5.7.

We offer an explanation for each setup.

Keyword-level Localized Captioning Benchmarks

Open-class keyword-level localized captioning benchmarks, proposed in [108], require the model to output keywords containing the object and part entities to describe the region. In contrast to closed-class keyword-level localized captioning, which constraints the model output to several choices provided, open-class keyword-level localized captioning takes free-form text outputs from the model. The evaluation results are in Tab. 5.2.

1. For LVIS [161], this involves predicting the class name as a keyword, given the segmentation mask of an object. A typical class name ranges from one word to four words.

2. For PACO [162], this involves predicting the class name of an object in the mask if the mask contains a full object, or the object name and the part name if the mask contains an object part. This is especially challenging because it would require the model to understand nuances between full objects and object parts.

Phrase-level Localized Captioning Benchmarks

Phrase-level localized captioning task requires the model to output a phrase containing a brief description for each the region that includes object identification and attributes typically within a few words. The metrics typically used in phrase-level benchmarks are CIDER, METEOR, BLEU, ROUGE_L, and SPICE [115, 116, 117, 118, 119]. We refer these metrics as short captioning metrics, as opposed to metrics from LLM-based evaluations that support evaluating detailed captions.

We perform zero-shot evaluation on the grounded phrases in Flickr30k Entities [163], where our model is not trained on the entities annotated in the training split of Flickr30k Entities. Results are in Tab. 5.3.

Detailed Localized Captioning Benchmarks

Detailed localized captioning task requires the model to output a detailed description for each the region with the length spanning from a long sentence to multiple sentences.

1. We perform zero-shot evaluation on detailed captions in the Objects365 [239] split of Ref-L4 [164] since we do not train on Objects365 dataset. We evaluate the prediction quality by computing short captioning metrics and CLAIR [166] score against the reference captions in the dataset. We use CLAIR to evaluate raw detailed outputs, while we summarize both the prediction and ground truth with GPT-4o-mini [100] before evaluation with short captioning metrics. No ground truth or reference captions are provided to GPT-4o-mini, with the LLM setting exactly the same for all models for fairness. Results are in Tab. 5.4.
2. We evaluate our model with DLC-Bench, our proposed benchmark for fine-grained region-based captioning. This evaluation is also zero-shot. We present details about our benchmark in §D.4. Results are in Tab. 5.5.

Detailed Localized Video Captioning Benchmarks

1. We conduct evaluation on HC-STVG [169], a spatial-temporal video grounding dataset with detailed captions used in prior and concurrent work [150, 120]. Following prior work [150], we evaluate the quality of localized captions with CIDER, METEOR, BLEU, ROUGE_L, and SPICE [115, 116, 117, 118, 119]. Results are in Tab. 5.6.

2. We also perform evaluation on the detailed localized video description benchmark in VideoRefer-Bench proposed by concurrent work [120]. GPT-4o is used to provide four dimensions of scores on a scale of 1 to 5. The four dimensions are Subject Correspondence (SC), Appearance Description (AD), Temporal Description (TD), and Hallucination Detection (HD). Zero-shot setting indicates that our model is not trained on Panda-70M [170], the dataset that VideoRefer-Bench sources the videos from. In-domain setting indicates mixing the detailed caption subset of VideoRefer-700k, which is also curated from Panda-70M [170], into our training data. Results are in Tab. 5.7.

D.3 Discussions

The Caveats of Using Referring Boxes in Data Pipeline

Caveats exist when boxes are used to refer to regions in the data pipeline. As shown in Fig. D.2, boxes can be ambiguous in terms of what they are referring to, causing uncertainty for the VLM that we use in our data pipeline. In contrast, masks are much more specific in terms of the region that it is referring to. This motivates us to use manually annotated masks in existing segmentation datasets rather than bounding boxes in order to curate high-quality data for DLC with little referring ambiguity. We additionally take in manually annotated keywords (e.g., class names, part names, entities) in the datasets for regions we are annotating in our data pipeline to further reduce the ambiguity and potential confusion for our VLM in the data pipeline.

The Pitfall of Using Reference Captions in Benchmarks

As discussed in Sections 5.5 and 5.6, caveats exist for using a “ground truth” reference caption for benchmarking localized descriptions. Specifically, since such a reference caption is hardly comprehensive and may not contain all the details about the region of interest, the metrics from the benchmark will treat the correct details in the caption prediction about the region of interest that are not mentioned in the ground truth reference caption as hallucinations. This discourages the model from generating detailed captions.

We analyzed the performance of our method in HD (hallucination detection) sub-task in VideoRefer-Bench [120] and found that our model often predicts correct details that are not present in the reference caption. Specifically, the example in Fig. D.3 shows this phenomenon. While our model’s prediction includes appearance and motion details about the change of the person’s gesture and expression, such details are not mentioned in the reference caption in the dataset. Since the GPT evaluator does not see the video and uses the ground truth caption as the only source of information, it incorrectly believes that the gestures and expressions are hallucinations and gives our caption a very low score for the hallucination detection dimension. However, the evaluation is not valid, as our model is correct in the descriptions about the gestures and expressions.



I don't have enough information to know what the box is referring to! 😞

Possibility 1: the dish

Possibility 2: the plate



in the data pipeline



It's the dish! 😊

a dish that appears to be a seafood meal, specifically including shrimp ...



in the data pipeline



It's the plate! 😊

a white, oval-shaped dish with a slightly raised rim ...



in the data pipeline

Figure D.2: **Caveats for using boxes to indicate region of interests.** Top: Using a box to indicate the region of interest leads to ambiguity. Middle and Bottom: Switching to a mask representation leads to more specific referring and correct descriptions.

Input Video	
DAM Output	A man with short, dark hair and a slight stubble is wearing a denim shirt with rolled-up sleeves. He begins by standing with his hands clasped together in front of him, exuding a calm and composed demeanor. As the sequence progresses, he raises his hands slightly, maintaining a gentle and expressive gesture. His facial expression shifts subtly, suggesting he is engaged in conversation or explanation. The man's movements are smooth and deliberate, as he occasionally opens his hands and gestures outward, indicating emphasis or clarification. Throughout the sequence, his posture remains upright and attentive, conveying a sense of engagement and focus.
Reference Caption	A man with short black hair is standing on the left, wearing a black jacket, as if reporting news
GPT Evaluation on Hallucination Detection (HD) Dimension	Hallucination Detection: 1 Explanation: The predicted answer includes several imaginative elements, such as gestures and expressions , that are not mentioned in the correct answer, indicating hallucinations in the description.

Figure D.3: **The pitfall of using reference captions for caption evaluation.** Evaluation benchmarks based on reference captions may incorrectly treat correct details in the predicted caption as hallucination. Since the GPT evaluator relies solely on the ground truth caption without viewing the video, it mistakenly flags gestures and expressions as hallucinations, resulting in a low score. However, the evaluation is invalid since the predicted details are correct.

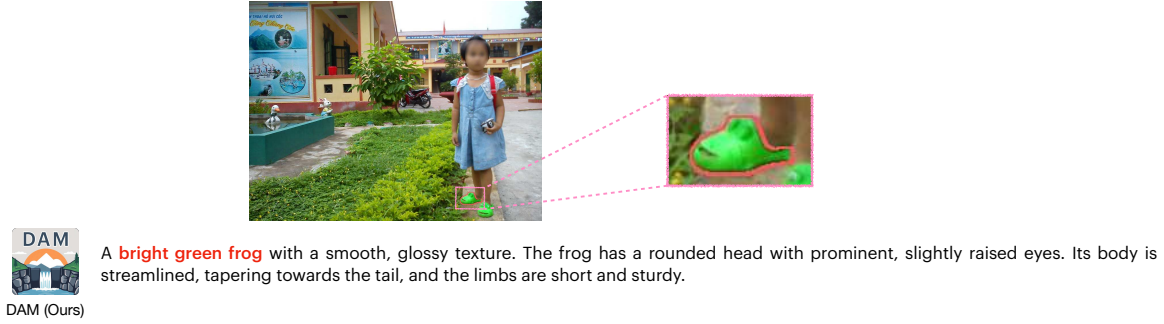
This indicates that the lower score on this sub-task is *not due to the hallucination of our model*, but rather due to the missing details in the reference caption and the fact that our model, evaluated in a zero-shot setting, does not have awareness for what types of details are preferred by or included in the reference caption.

Failure Cases

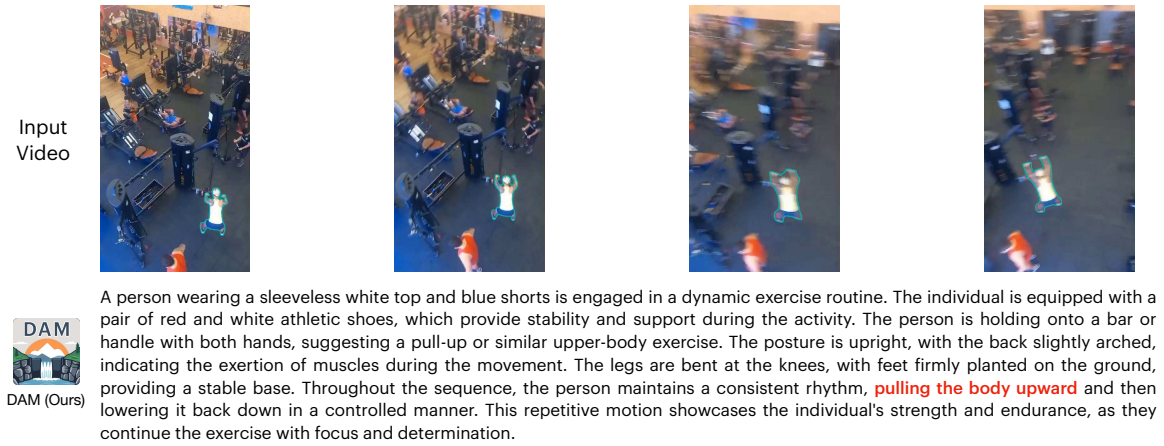
We show two failure cases of DAM in Fig. D.4. In Fig. D.4(a), DAM misrecognizes the frog-shaped slipper to be a frog. In Fig. D.4(b), DAM describes the person as pulling the body upward. We expect these errors to be mitigated by broader data coverage.

Potential Limitations

DAM is only trained for multi-granular localized captioning, especially for detailed localized captioning (DLC) and is not specifically optimized for other general vision-language tasks. However, DAM is designed for in-depth analysis for the task of multi-granular image and video localized descriptions rather than for breadth for general vision-language understanding, which justifies the design choice.



(a) DAM might still misrecognize a region and output an incorrect description. For example, it misrecognizes the frog-shaped slipper to be a frog.



(b) DAM might also be confused by the combination of the object motion and the camera motion. In this example, it makes the mistake of describing the person as pulling the body upward.

Figure D.4: Failure cases for our proposed DAM.

Computational Efficiency

DAM incorporates our proposed localized vision encoder. It differs from the SigLIP [240] vision encoder used in [140] by adding two key components: *patch embedding layers* for encoding the mask and *cross-attention blocks*. Importantly, these components do not alter the dimensions or sequence length of the vision features passed to the large language model, ensuring that the parameter count and computational efficiency of the large language model are unaffected. Since the vision encoder represents only a small fraction of the total parameters and computational operations, the overall increase in FLOPs and parameter count remains marginal, maintaining the model's efficiency.

To be more specific, unlike prior works that derive regional features from *image features* for each region, the regional feature used in our approach comes directly from *a global and a*

focal view of the input image, with cross-attention enhancing the focal representation. This design is justified as the vision encoder is much smaller than the LLM (400M vs. 3B/8B parameters), with minimal latency impact (*0.06s compared to 1.49s for 3B LLM* as measured in our pipeline). This overhead is outweighed by the benefits of preserving fine details that global image features miss as indicated in Tab. 5.8), especially for small regions. Finally, DAM 3B outperforms much larger models in challenging (Tab. 5.5), showing our efficiency.

Training Data

In addition to the details in data annotation presented in §D.8, we discuss the training data of our work in this section and present a comparison with recent works. Compared with recent work Ferret [107] which used 1.1M *unreleased* samples and RegionGPT [114] which used 1.5M *unreleased* samples, we train our model on a comparable amount of data (1.5M samples). However, we obtain much better performance (Tab. 5.5), which shows the effectiveness of DAM.

Performances of Baseline Models on DLC-Bench

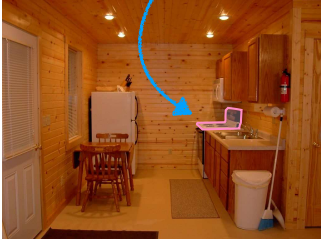
Interestingly, region-specific VLMs often perform on par or worse than generic VLMs. This is likely because many are trained on datasets with short regional captions, leading them to produce brief, phrase-level descriptions. Even when prompted for longer descriptions [114, 109], these models tend to include irrelevant details about the background, speculations, or hallucinations, due to insufficient regional information. Providing crops instead of full images leads to mixed results for different region-specific VLMs since these models are not designed to describe regions in crops.

D.4 Details for DLC-Bench

Image and Instance Selection. We leveraged a subset of the Objects365 v2 [239] validation set, which was manually annotated with segmentation masks in [241], for image and instance selection. We collected a set of 892 challenging questions from this subset, each containing one object of interest. Each question is manually inspected, and questions with ambiguous or unclear answers are filtered out. To maintain the integrity of our benchmark, we conducted de-duplication to ensure that no images used in the benchmark were present in our training dataset for detailed localized captioning.

Positive Question Generation. For each masked region, we prompted an off-the-shelf Visual Language Model (VLM) to generate a list of parts. Subsequently, for the whole object and each part, we asked the VLM to generate a list of properties covering aspects such as color, shape, texture, materials, and size. Each property is stored in the form ([object name], [part name], [property name], [property value]). For example, if the masked region

Task: describe this region



(a) An example region in the benchmark for the model to describe

A white electric stove with a smooth, flat cooktop featuring two visible black burners. The control panel is located at the back, with a black surface and a few visible knobs. The stove has a slightly raised backguard.

(b) An example description to be evaluated

Q: Which of the following is applicable to the description?

Choices:

- A. The control panel or the stove is not mentioned. (score: 0)
- B. The location of the control panel is mentioned in the description but is not back. (score: -1)
- C. The location of the control panel is mentioned in the description and is back. (score: 1)**
- D. The location of the control panel is not mentioned, but the control panel of the stove is mentioned. (score: 0.5)

(c) An example positive question in DLC-Bench

Q: Which of the following is applicable to the description?

Choices:

- A. The induction cooktop surface of the stove is mentioned in the description. (score: -1)
- B. The induction cooktop surface of the stove is not mentioned in the description. (score: 1)**
- C. The stove is not mentioned in the description. (score: 0)

(d) An example negative question in DLC-Bench

Figure D.5: **An example from DLC-Bench for detailed localized captioning.** (a) The process begins by prompting a model to describe a specified region within the image. The resulting description is then evaluated using a text-only LLM as a judge that rates each response by answering positive and negative questions. (b) shows an example description to be evaluated. (c) Positive questions are designed to test whether the model correctly identifies specific details within the described region. The model receives points for accurate details and is penalized for factual errors. The bold option (option C) indicates that the LLM judge believes that option C is applicable, allowing the model to get a point for this example positive question. (d) Negative questions ensure the model refrains from mentioning irrelevant or nonexistent details. Mislocalization or hallucination results in penalties to prevent false positives. The bold option (option B) indicates that the LLM judge believes that option B is applicable, allowing the model to get a point for this negative question.

is a corgi, the VLM could describe the brown fur of the corgi as (corgi, fur, color, brown).

We used this list of properties as a starting point for manual curation. We then manually added significant properties that the VLM missed, revised inaccurate properties, and removed hallucinated or ambiguous properties from the VLM outputs. Finally, we turned these properties into questions that test whether a description accurately covers the property.

Negative Question Generation. We targeted mislocalization and hallucination, which are two types of negatives (i.e., cases in which a property or an object should not be included in the descriptions). Specifically, for mislocalization errors, we prompted the VLMs to generate a list of objects in the image that are not in the masked region. We also prompted the VLMs to generate a list of parts that are commonly associated with the object type of the masked region but are not present or visible in the masked object in the image (e.g., the head of a corgi if it is occluded and thus not included in the masked region).

To avoid biasing towards one specific off-the-shelf VLM, we leveraged multiple VLMs for different instances to generate initial positives and negatives. Specifically, we annotated 34 regions using GPT-4o [100], 35 using Gemini 1.5 Pro [171, 242], and 31 using Anthropic Claude 3.5 Sonnet [243] for the initial property generation. We used the same image prompting method for all VLMs as we did when prompting the VLMs in the first stage of our data pipeline.

Note that the choices for each question are mutually exclusive, which ensures one option is always valid and leaves no room for two options to be true at the same time.

Scoring Mechanism. Our evaluation methodology involves scoring the models based on their ability to include correct details and exclude incorrect or irrelevant information.

To evaluate a model like DAM for its ability to output detailed localized captions, we first prompt the model to generate descriptions for each of the masked instances. Then, instead of directly asking our model to provide answers to these questions, we prompt a text-only LLM, Llama 3.1 8B [130], to serve as a judge to rate the localized descriptions according to the positive and negative questions.

For each model-generated description, we apply the following scoring rules:

- **Positive Scoring:** For each positive question, if the description correctly includes the specified detail, the model receives a point. To prevent models from artificially inflating their scores by generating excessively long descriptions and guessing details, we penalize incorrect details and discourage models from including uncertain or erroneous content. If the detail is mentioned but incorrectly (e.g., wrong color), a penalty of one point is applied. No point is awarded if the description does not mention the detail. Partial points (0.5 points) are awarded for answers that are partially correct but insufficiently detailed. Note that the model gets positive points only when the object recognition is correct, as the correctness of the details depends on the correctness of the overall region recognition. We present a positive example in Fig. D.5(c).
- **Negative Scoring:** For each negative question, if the description appropriately excludes the incorrect or irrelevant detail, the model gets a point. If the description includes the detail, indicating mislocalization or hallucination, a penalty is applied. The model gets zero or negative points when the object recognition is incorrect, since otherwise a caption that is random and completely off could get high scores on the negative questions. We present a negative example in Fig. D.5(d).

The positive (negative) score for a model is the sum of points for positive (negative) questions, normalized by the maximum possible score to yield a percentage for comparison. We also average the positive and negative scores to obtain an overall score, which represents the model’s overall capability in detailed localized captioning.

We present an example from DLC-Bench in Fig. D.5. The example region in Fig. D.5(a) features a stove with coil burners. An example description of the region is presented in Fig. D.5(b). For the example positive question in Fig. D.5(c), the LLM judge selects option

	VP-SPHINX Arch	Our Arch
Avg (%)	50.2	63.8

Table D.1: **Ablations on architecture design compared to our strongest baseline VP-SPHINX [110].** We trained a model with VP-SPHINX [110] architecture on our curated DLC data from various segmentation datasets. The results on DLC-Bench indicate the advantages of our model architecture that allows detailed localized features to be presented to the LLM for DLC.

Prompt Augmentation	Pos (%)	Neg (%)	Avg (%)
No	52.3	82.2	67.3
Yes	51.3	82.2	66.7

Table D.2: **Comparison of performance of DAM with and without prompt augmentation.** Prompt augmentation has minimal effect on DAM’s performance on DLC-Bench. While descriptions generated by the model may occasionally be less detailed, leading to a slight decrease in the performance on positive questions, we observed that prompt augmentation enhances instruction following when prompts include specific guidelines, such as length constraints. We use the model without prompt augmentation with our benchmark, including ablations, by default.

Setting	Pos (%)	Neg (%)	Avg (%)
Image-only Training	52.3	82.2	67.3
Image+Video Joint Training	52.4	85.4	68.9

Table D.3: **Comparison of performance of our image-only DAM and DAM trained with both localized image description task and localized video description task.** Joint training benefits generating high-quality localized image descriptions compared to image-only training.

C, as the caption correctly mentions that the control panel is at the back, allowing the model to get a point for this positive question. For the negative question in Fig. D.5(d), the LLM judge selects option B, as the caption correctly indicates that it is not an induction cooktop, allowing the model to get a point for this negative question.

Evaluation Setting. For our models, we follow our inference setting described in §D.8.

D.5 Additional Ablation Studies

Model Architecture with the Same Training Data. A model’s performance is largely due to two factors: model architecture design and training data. Since both factors differ for different models, it is hard to compare the effectiveness of different model architectures head-to-head.

To this end, we compare our model architecture against VP-SPHINX [110], the strongest prior baseline in most benchmarks that we tested on. By continuously training a VP-SPHINX model [110] on our data after pre-training on the originally proposed datasets. This is a fair comparison since our method is also fine-tuned from a pretrained VLM, VILA-1.5 [140], with two stages of training prior to training on our region-specific dataset.

As shown in Tab. D.1, our model architecture achieves much better performance on detailed localized captioning benchmark DLC-Bench with trained on the same dataset from our proposed data pipeline. This justifies that our proposed focal prompt and localized visual backbone are able to provide more detailed features compared to just the global image features extracted by a vision encoder on the full image with a regional referring feature, as employed in [110].

Prompt Augmentation. We compared variants of our model with and without prompt augmentation. As shown in Tab. D.2, incorporating prompt augmentation slightly degrades our model’s performance on the positive questions in our benchmark. We hypothesize that despite introducing variations in the prompts and enhancing the model’s instruction-following capabilities, prompt augmentation creates a mismatch between the prompts used during training and those used during evaluation (as we always use the same prompt for evaluation, which is detailed in §D.8). Since the prompt used during evaluation might not be the same as the prompt used in training, the model may also occasionally reference other tasks from our mixing dataset ShareGPT-4V for the length of outputs. This may cause the model to produce outputs that are not as detailed as when it is trained exclusively with the original prompt. Importantly, the model’s performance on the negative questions remains unchanged, indicating that prompt augmentation does not lead to hallucinations or mislocalization.

Despite the slight degradation of the performance in the benchmark (0.6% in the overall accuracy), we observed that prompt augmentation improves instruction-following capabilities when prompts include additional instructions, particularly those specifying requirements on the length of the outputs. Therefore, we default to using the model without prompt augmentation in our benchmark evaluations, including ablations. In contrast, we employ the model with prompt augmentation in the qualitative evaluations.

Image-only Training vs Image+Video Joint Training. We also compared our image-only DAM with DAM with image + video joint training in Tab. D.3. We show that our model with image-video joint training slightly outperforms our model with image-only training on detailed localized image captioning. Note that for this ablation study, we keep the model size the same and use the 3B model for both image-only training and image-video joint training. We use image-only training as the default option for results in our benchmark for simplicity.

Method	#Params	Pos (%)	Neg (%)	Avg (%)
<i>API-only General VLMs:</i>				
GPT-4o (SOM) [100]	-	5.0	29.2	17.1
o1 (SOM) [126] [†]	-	0.8	28.0	14.4
Claude 3.7 Sonnet (SOM) [167] [†]	-	0.5	40.2	20.4
Gemini 2.5 Pro (SOM) [171, 168] [†]	-	13.2	65.0	39.1
<i>Open-source General VLMs:</i>				
Llama-3.2 Vision (SOM) [130]	11B	16.8	40.4	28.6
Llama-3 VILA1.5 (SOM) [140]	8B	0.6	0.6	0.6
InternVL2.5 (SOM) [139, 172, 173]	8B	8.6	28.6	18.6
LLaVA v1.6 (SOM) [131, 132, 133]	7B	2.2	3.8	3.0
Qwen2.5-VL (SOM) [138, 174]	7B	8.5	27.2	17.8
VILA1.5 (SOM) [140]	3B	-0.4	15.4	7.5
DAM (Ours)	3B	52.3	82.2	67.3

Table D.4: **Additional results with existing general VLMs using Set-of-Mark (SoM) prompting [144].** The results are accuracies on detailed localized captioning in DLC-Bench. Compared with results in Tab. 5.5 which are obtained with the same prompt engineering as we use in the stage 1 of our data pipeline, SoM leads to degraded quality. In this comparison, the advantages of our method, compared with prior baselines, are much larger. Negative numbers are due to penalties from factual errors. Note that region-specific VLMs, including our proposed DAM, have predefined ways of inputting regions, and thus SoM prompting is not applicable to these models. †: models with thinking mode.

D.6 Additional Quantitative Results

Set-of-Marks Prompting. We present a comparison with baseline VLMs that use Set-of-Marks (SoM) prompting [144] in Tab. D.4. SoM leads to degraded results compared to the prompt engineering method used in stage one of our data annotation pipeline. This is mostly because the marks proposed by SoM blend in with the object or the background in complex scenes. They might also mask out some part of the object, which interferes with the model’s understanding capabilities. Therefore, for fair comparisons, we use the same prompt engineering method as we use in stage one of our data annotation pipeline in our main result in Tab. 5.5. Importantly, region-specific VLMs, including DAM, have predefined ways of encoding regional inputs, making SoM inapplicable to these models.

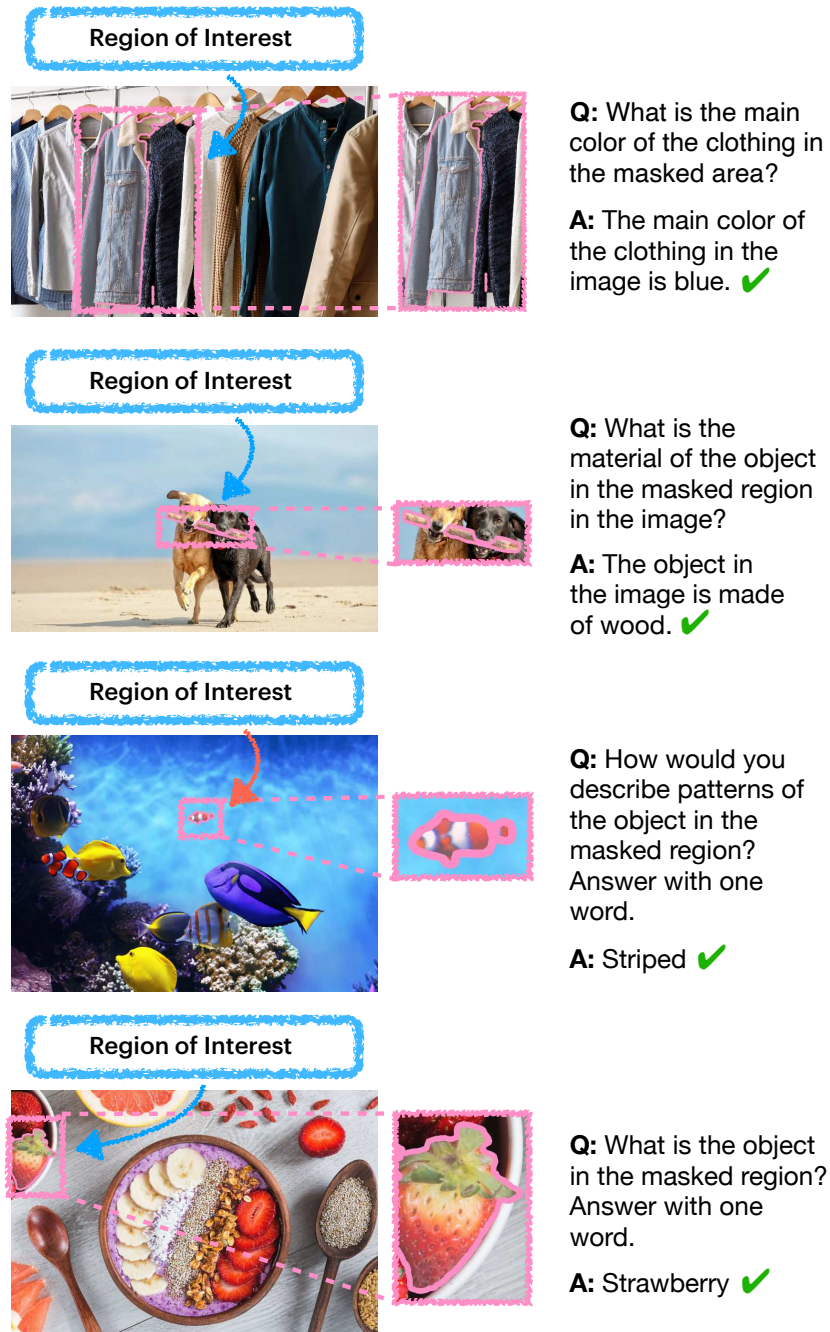


Figure D.6: **Emerging zero-shot QA capabilities.** DAM could answer questions about regions in an image, showcasing capabilities such as object recognition and property identification.

D.7 Additional Qualitative Results

Detailed Localized Image Captioning

In Fig. D.7, we present additional examples from LVIS [161] to show our model’s strong performance on detailed localized image captioning.

Our model demonstrates robust localization and region understanding capabilities. In the first example, it accurately describes the sofa cushion without mentioning the dog that is outside the masked region. In the second example, it correctly identifies the roller blind, which would be challenging to recognize based solely on a local crop without context. In the third example, the model provides a detailed description of the giraffe without referencing the birds perched on it, as they fall outside the masked region. These examples highlight our model’s precise localization abilities and its effectiveness in perceiving regional details with contextual understanding.

Zero-shot QA Capabilities

Although not trained on any regional QA datasets, DAM surprisingly exhibits emerging zero-shot capabilities on regional QA.

In Fig. D.6, we show examples of our model performing zero-shot QA. DAM is able to identify properties of objects in the masked regions. For example, it is able to identify the color of the clothing, the material of the stick, and the textural pattern of the fish in the first three examples. DAM is also capable of performing object recognition for a region in the image, identifying the strawberry in the last image.

Detailed Localized Video Captioning

We present more examples for detailed localized video captioning in Fig. D.8 and Fig. D.9. Our model can describe objects in videos with large object motion and camera motion. DAM can also identify stationary objects by indicating that they are stationary in the description.

Qualitative Comparisons with Strong Baselines

Detailed Localized Image Captioning. We also present qualitative comparisons with GPT-4o [100] and our strongest open-weight baseline VP-SPHINX [110] in detailed localized image captioning in Fig. D.10.

In both examples, GPT-4o could not correctly recognize the objects in the masked regions, providing only vague descriptions. VP-SPHINX, while better than GPT-4o, still struggles with accurate object recognition and detailed descriptions. In the left image, VP-SPHINX incorrectly describes a group of seals when the masked region contains only one seal. In the right image, VP-SPHINX identifies the towel but provides minimal detail, missing key attributes like its color and texture.

Dataset	# Images	# Regions
<i>Stage 1:</i>		
LVIS [161]	90,613	373,551
Mapillary Vistas v2.0 [245]	17,762	100,538
COCO Stuff [246]	28,365	32,474
OpenImages v7 [247, 248]	64,874	96,006
PACO [162]	24,599	81,325
<i>Stage 2:</i>		
SA-1B (10%)	592,822	774,309
Total	819,035	1,458,203

Table D.5: **Dataset statistics across stages with total images and regions for training detailed localized image captioning.** In stage 1, we annotated 684k regions across 226k images from existing instance and semantic segmentation datasets. In stage 2, we perform SSL on 10% of SA-1B images without using the masks provided by the datasets, resulting in 774k regions across 593k images. In total, we annotated 1.46M regions across 819k images with detailed localized descriptions. This diverse and high-quality dataset is the key to our model’s performance. Note that due to filtering the number of instances and images are lower than the number of instances and images in the original dataset.

Our model outputs detailed and high-quality descriptions of the seal and the towel. This improvement stems from our model’s design which enables the fusion of object-specific information with broader contextual understanding.

Detailed Localized Video Captioning. We present comparisons with three strong video understanding models, GPT-4o [100], Qwen2.5-VL [174], and recent work VideoRefer- [120], in detailed localized video captioning in Fig. D.11. In the top example, we observed that GPT-4o struggles to interpret the cow’s movements. Similarly, Qwen2.5-VL-7B incorrectly perceives the cow as stationary. VideoRefer-7B provides minimal motion and appearance details. In contrast, our 8B model accurately identifies the motion of the cow, providing more detailed information about it.

In the bottom example, GPT-4o misidentifies the object, mistakenly assuming the animal is transforming into a wolf or a pig. Meanwhile, Qwen2.5-VL-7B believes only the sheep’s head is moving. VideoRefer-7B recognizes that the sheep is moving but provides little detail about the appearance of the sheep. In contrast, our model correctly identifies the animal in the masked region as a sheep throughout the video and accurately recognizes its full movement, providing details about its motion and appearance.

Dataset	# Videos	# Regions
SA-V [159]	36,922	93,969

Table D.6: **Dataset statistics across stages with total videos and regions for training detailed localized *video* captioning.** We label 94k regions across 37k videos from SA-V dataset [159] for detailed localized video captioning. Note that each region indicates an instance across multiple frames in the video.

D.8 Implementation Details

Data Annotation Pipeline

Stage 1. We annotate four existing instance and semantic segmentation datasets for detailed localized descriptions. We use off-the-shelf VLMs for region annotations, with 603k regions across 202k images with detailed localized descriptions in total in stage 1. For the model variant used in PACO [162] open-class dataset evaluation, we additionally merged in 81k annotated instances from PACO [162] to improve its part description capabilities, leading to 684k annotated regions, as detailed in Tab. D.5. To prompt a VLM to output detailed localized descriptions, we input a cropped image and a masked image. While the cropped image allows coarse localization and provides high token density per pixel for clear descriptions, the masked image helps localize the object of interest when there are multiple instances with the same category. The category name is also provided in the text prompt, relieving the model from having to identify the object without the context from the image. We present the prompt for data annotation in Tab. D.7.

Stage 2. We annotate 10% of SA-1B through self-labeling, resulting in 774k annotations across 593k images, as detailed in Tab. D.5. Due to filtering, the final number of instances and images is lower than the original 10% subset of SA-1B. We do not use the masks provided with SA-1B, as they contain a large number of masks for parts. Instead, we employ the open-vocabulary detector OWL-ViT v2 [42, 249] to detect objects in the images, and then use SAM [159] to generate masks for the detected instances. Finally, we use SigLIP [240] to evaluate the image-text similarity, taking the region as an image.

To ensure data quality, we apply extensive filtering (i.e., rejection sampling) based on confidence scores from OWL-ViT v2, SAM, and SigLIP image-text similarity. We also ensure we have at most two instances per image, and for images with two instances, these two instances have to be from different classes. The object category names produced by OWL-ViT v2 are then put into a variant of our Describe Anything model, which is trained on data from stage 1 and optimized for self-labeling. This variant generates descriptions with a 50% probability of incorporating class names during training, as during self-labeling we have a class name as a part of each input. The object category proposals used by OWL-ViT v2 are generated by VILA 1.5 [140].

Detailed localized video captioning. We annotated 94k regions across 37k videos from SA-V dataset [159] for detailed localized video captioning, as detailed in Tab. D.6. Note that each region, also called masklet, indicates an instance across multiple frames in the video. In contrast to the use of SA-1B, where we did not use the masks that come with the dataset, we use the high-quality masklets that come with the videos. We found that many masklets cover parts of an instance, which is not necessarily helpful in describing the whole object as a common use case of our model. Therefore, we performed instance segmentation on the videos with ViTDet [250] + Cascade Mask R-CNN [251] trained by EVA-02 [252] and used voting to match the segmentation masks with the masklets. In this way, we filter out most of the masklets that are parts, since they likely do not correspond to instance masks. The matched masklets carry the class name from the matched instance segmentation mask, which is used in the annotation process to obtain a detailed localized caption for each masklet.

-
- 1 You are responsible to write a very descriptive caption to describe the {{category}} in the provided SEGMENTED image. You may leverage the surrounding context of the SEGMENTED image provided in the CROPPED image.
 - 2 You must not mention any background in the caption and only describe the {{category}} in the SEGMENTED image! The caption must ONLY contain sufficient details to reconstruct the same {{category}} in the SEGMENTED image but nothing else!
 - 3 Here are some additional rules you need to follow when describing the {{category}} in the SEGMENTED image:
 - 4 1. If there are multiple {{category}} in the CROPPED image, focus on the {{category}} in the SEGMENTED image.
 - 5 2. If the {{category}} in the SEGMENTED image is occluded by other objects, only describe the visible part. DO NOT mention anything that is not directly related to the visible part of {{category}}, such as "A segment of", which part is invisible, etc. For objects with text written on it, describe the object instead of just outputting the text written on it.
 - 6 Here is the SEGMENTED image that needs caption:
-

Table D.7: Our prompt for data annotation in stage 1.

Model Training

We start from off-the-shelf VILA 1.5 [140] models that are publicly available on HuggingFace. For image-only training, we fine-tune VILA 1.5 3B model. For joint image-video training, we use VILA 1.5 8B model. We use SigLIP [240] vision encoder, following VILA 1.5. To prevent catastrophic forgetting and to maintain instruction following capabilities, we mix in ShareGPT-4V [253] with our localized image/video captioning dataset collected with our proposed data pipeline. Following the VILA 1.5 training and inference recipe, we treat videos as 8 images concatenated in the sequence.

We closely follow VILA 1.5’s recipe of the supervised fine-tuning stage and train all modules, including the vision backbone, the projector, and the LLM. We fine-tune the model for 1 epoch. For the 3B model, we use a batch size of 2048 with a learning rate of 1e-4 on 8

Nvidia A100 GPUs. For the 8B model, we use a batch size of 2048 with a learning rate of $1e-5$ on 32 Nvidia A100 GPUs. Both models take less than a day to train. We use a cosine scheduler with a warmup ratio of 0.03. No weight decay is used. For training our model that takes in a class name for self-labeling, we randomly put the class name in the prompt with 50% probability. For models without prompt augmentation, which is detailed below, we simply use the prompt “Describe the masked region in detail.” Following VILA, we always put image tokens in front of the textual tokens. As for the setting for the focal crop, we extend the crop by $1\times$ the width towards left and right, and $1\times$ the height towards top and bottom, unless we hit the boundaries of the image, in which case we take the boundaries, i.e. $\alpha = 3$ and the total area of the crop is enlarged up to $9\times$. If either the height or width is less than 48 pixels, we take 48 pixels for that direction to encode more context for very small regions, since the small regions themselves do not have much useful information.

Prompt Augmentation. We trained a variant of our model with prompt augmentation to enhance generalization capabilities beyond detailed localized captioning, as analyzed in §D.7. For these models, during training, we randomly select one of 15 prompts from a predefined set. These prompts may or may not include a `{prompt_suffix}`. The default prompt suffix is *in detail*. However, we introduce variability by conditioning the prompt on the number of words or sentences in the target caption.

Specifically, with a 20% probability, we condition the prompt on the number of sentences, using suffixes like *in one sentence* or *in [number of sentences] sentences* (e.g., *in 2 sentences*). If the caption contains only one sentence, we use phrases like *in a sentence* or *in one sentence*.

With another 20% probability, we condition the prompt on the number of words in the target caption. For captions with a small word count, we use exact numbers (e.g., *in 3 words*). For longer captions (up to 200 words), we may round the word count to the nearest ten and use phrases like *in about 50 words* or *in around 50 words*. If the caption exceeds 200 words, we use the suffix *in more than 200 words*.

The list of prompts that include a `{prompt_suffix}` is as follows:

1. Describe the masked region `{prompt_suffix}`.
2. Describe the masked area `{prompt_suffix}`.
3. What can you describe about the masked region `{prompt_suffix}`?
4. Can you describe the masked region `{prompt_suffix}`?
5. Provide an explanation of the masked region `{prompt_suffix}`.
6. Depict the masked area `{prompt_suffix}`.
7. Portray the masked area `{prompt_suffix}`.
8. Describe what the masked region looks like `{prompt_suffix}`.
9. Illustrate the masked region `{prompt_suffix}`.

10. How would you explain the masked area {prompt_suffix}?
11. What details can you provide about the masked region {prompt_suffix}?
12. What does the masked region entail {prompt_suffix}?
13. How would you illustrate the masked region {prompt_suffix}?
14. How would you depict the masked area {prompt_suffix}?
15. How would you portray the masked area {prompt_suffix}?

Additionally, we have prompts that inherently request detailed descriptions without requiring a suffix:

1. Give a detailed description of the masked region.
2. Provide a thorough description of the masked region.
3. Can you explain the details of the masked area?
4. Give a detailed account of the masked region.
5. Describe the masked area comprehensively.
6. Provide an in-depth description of the masked region.
7. Explain the specifics of the masked area.
8. Can you provide a thorough explanation of the masked region?
9. What are the details of the masked area?
10. Provide a comprehensive description of the masked area.
11. What specific details can you provide about the masked region?
12. Can you give an in-depth account of the masked section?
13. What are the main characteristics of the masked region?
14. Give a thorough description of the masked area's details.
15. Provide detailed information about the masked area.

For prompts without a suffix, we do not condition the generation on the number of words or sentences.

During training, we select prompts based on the `prompt_suffix`:

- If the `prompt_suffix` is *in detail* (the default option), we may choose from either set of prompts.
- If the `prompt_suffix` specifies word or sentence counts, we select only from prompts that include `{prompt_suffix}`.

This approach introduces variability in the prompts, encouraging the model to generate responses with controls from the prompts in mind, thereby enhancing its generalization and instruction-following capabilities.

Inference Setting

Unless otherwise mentioned, our prompt for obtaining detailed localized image descriptions at inference time is the following:

`Describe the masked region in detail.`

Our prompt for obtaining detailed localized video descriptions at inference time is the following:

`Given the video in the form of a sequence of frames above, describe the object in the masked region in the video in detail. Focus on appearance, motion, and actions. If the motion involves multiple stages or steps, break down each stage and describe the movements or changes sequentially. Ensure each phase of motion is described clearly, highlighting transitions between actions.`

For Co3Dv2 [175] sequences that we treat as videos, we use the following prompt:

`Describe the masked region in the video in detail. The video consists of multiple views of a stationary object. Focus on the appearance of the object without mentioning any motion or actions.`

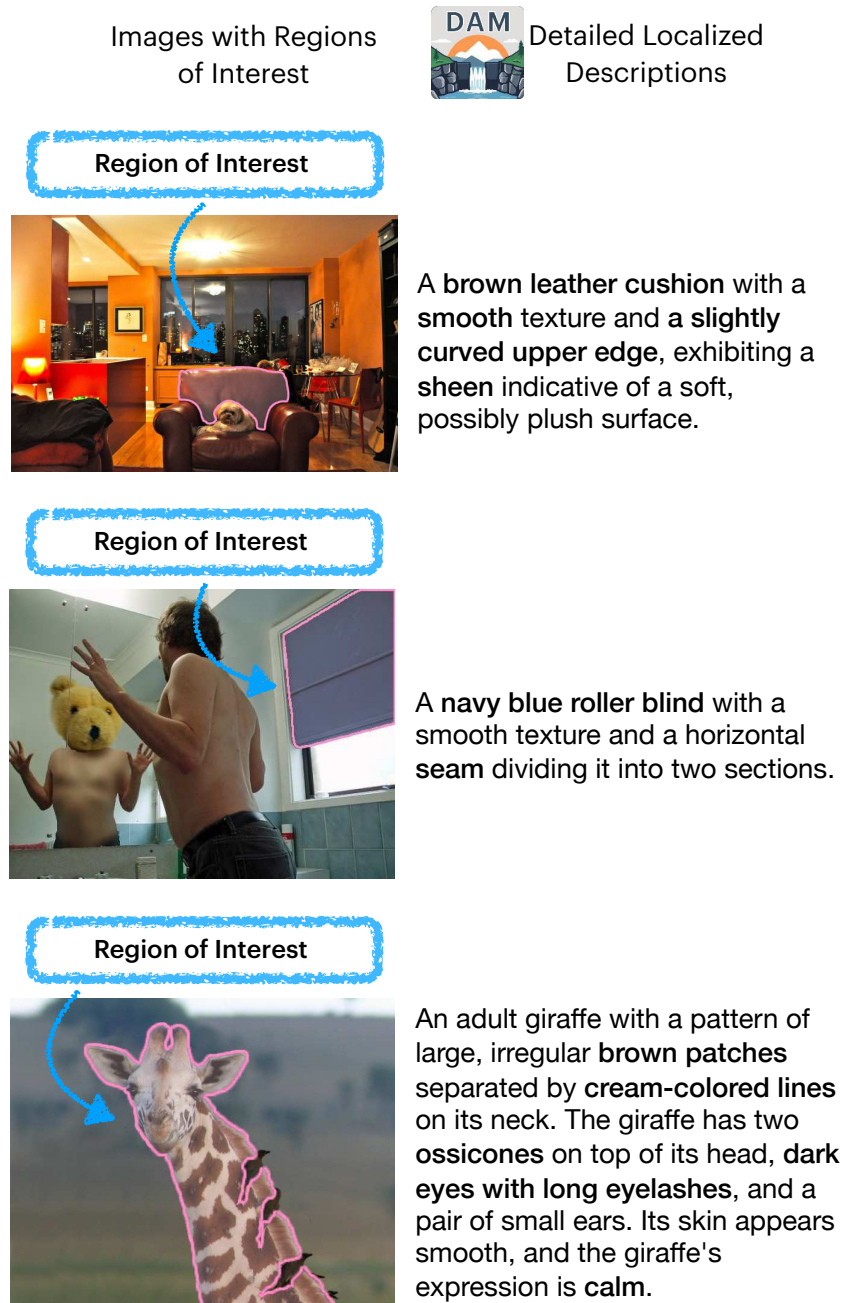


Figure D.7: Additional results from LVIS [161] demonstrating DAM’s detailed localized image captioning capabilities. Our model exhibits robust region understanding and localization across diverse scenarios. It produces precise descriptions of objects within masked regions while successfully identifying challenging details like the roller blind in the second example through effective use of contextual cues.



Figure D.8: **Additional results from DAM on detailed localized video captioning (Part 1)**. Our model is able to accurately describe small objects in complex scenes that involve large object motion and camera motion. Our model also correctly identifies stationary objects captured by a non-stationary camera by saying they are stationary. Videos visualized in this figure are from SA-V [159] dataset.

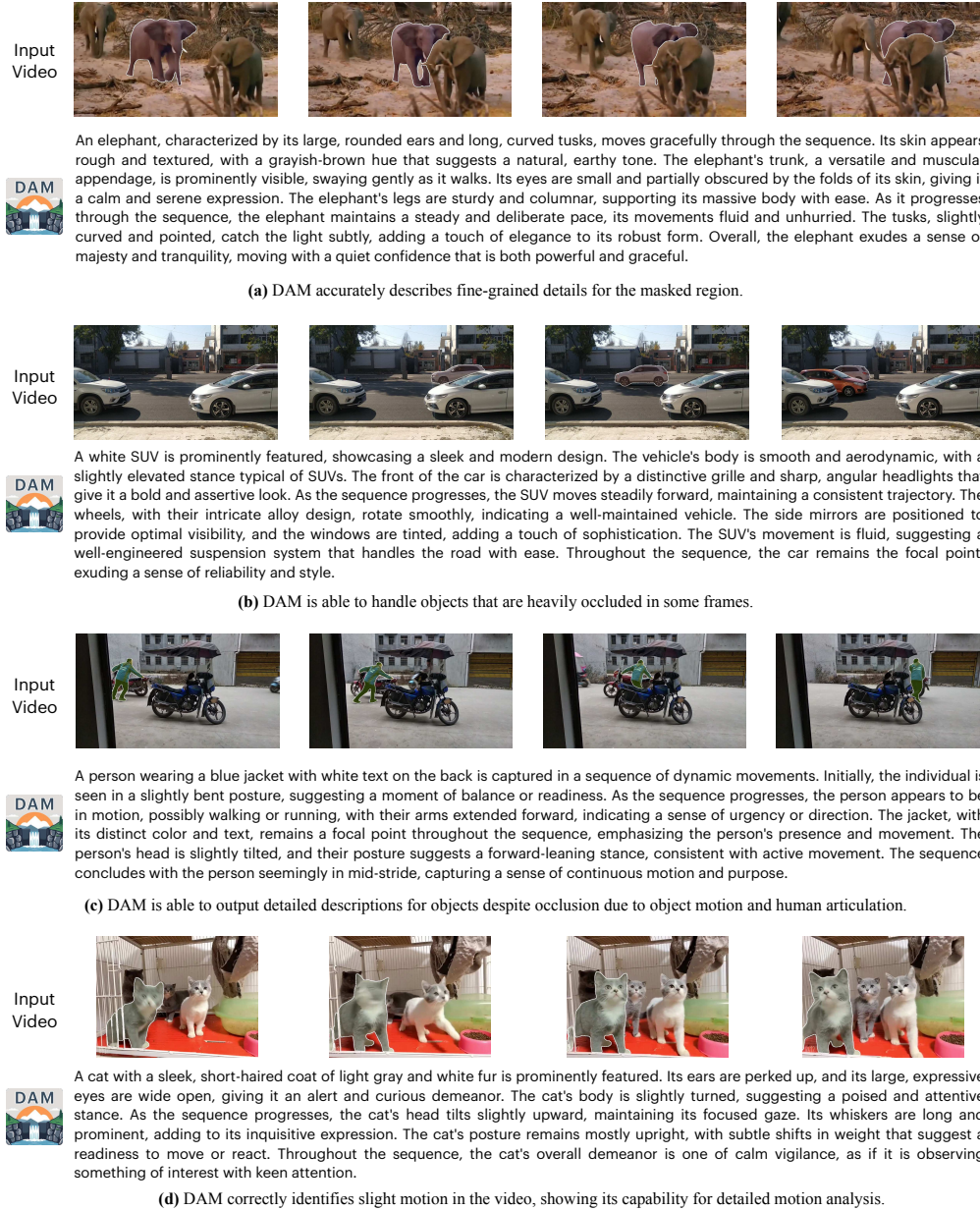


Figure D.9: **Additional results from DAM on detailed localized video captioning (Part 2).** Our model is able to accurately describe objects that are partially occluded and is able to perceive and describe slight motion. Videos visualized in this figure are from MOSE [244] dataset.

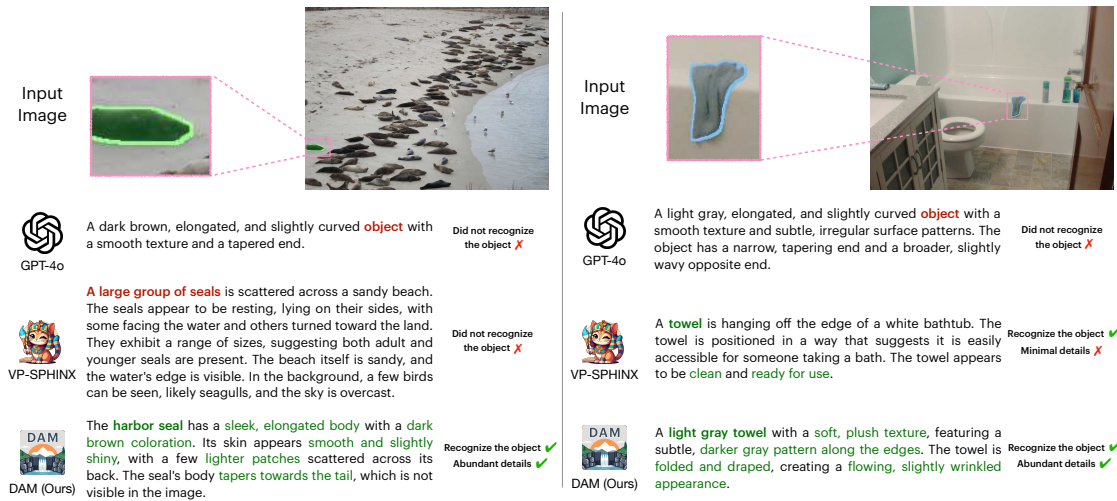


Figure D.10: **Qualitative comparisons demonstrate the superior localized image understanding capabilities of our model compared to GPT-4o [100] and VP-SPHINX [110], our strongest open-weight baseline.** GPT-4o struggles to recognize objects in masked regions accurately, offering only vague descriptions. In the left image, VP-SPHINX incorrectly describes a group of seals when the masked region contains only one seal. In the right image, VP-SPHINX identifies the towel but provides minimal detail, missing key attributes like its color. In contrast, our model delivers precise, detailed descriptions and captures the seal's sleek elongated body, dark brown coloration with lighter patches, and the towel's light gray color, wrinkled texture, and darker edge pattern. This superior performance stems from our model's architecture that effectively fuses object-specific details with broader contextual understanding.

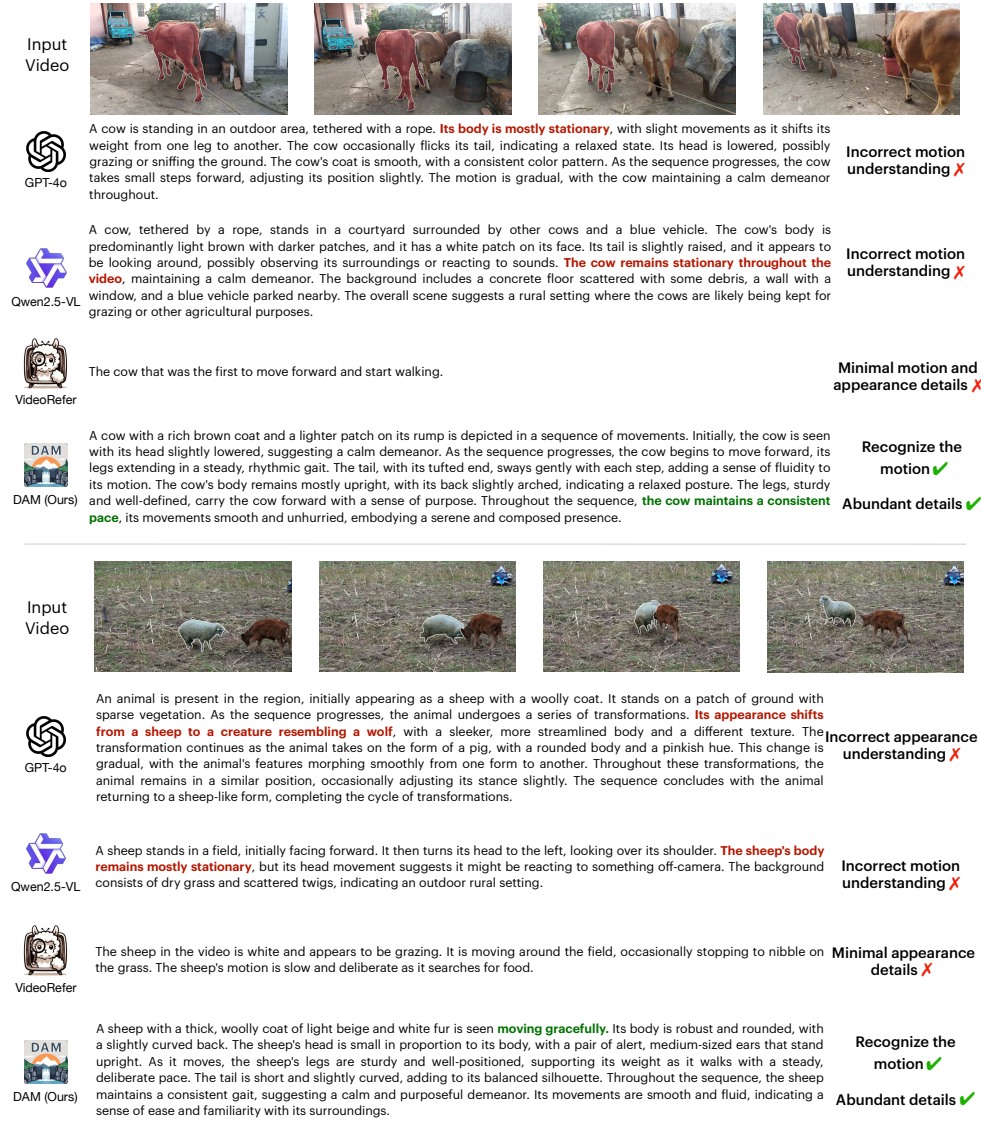


Figure D.11: Our proposed DAM demonstrates superior localized video understanding compared to GPT-4o [100], QwenVL-2.5 [174], and VideoRefer [120]. **Top figure:** DAM accurately captures the cow's forward movement with comprehensive details, whereas GPT-4o and QwenVL-2.5 mistakenly perceive the cow as stationary. Compared to VideoRefer, DAM provides richer descriptions of both motion and appearance. **Bottom figure:** DAM correctly recognizes the animal as a sheep and accurately describes its graceful movement, while GPT-4o erroneously identifies it as transforming into other animals, and QwenVL-2.5 incorrectly perceives that only the sheep's head is moving. VideoRefer provides limited appearance details, while DAM offers extensive, accurate descriptions. These cases highlight DAM's precise understanding of motion and appearance throughout video sequences.

Appendix E

Adaptive Parallel Reasoning

E.1 Additional Related Works

Language model reasoning algorithms Chain-of-thought (CoT) prompting [182] was the first approach to show how, with the right prompt, additional test-time compute can be exploited to improve language model performance on an end task. Building on the promise of CoT, models were evaluated on increasingly complex reasoning tasks, and the output sequences produced by CoT and its variants became increasingly out-of-distribution from model training data. This motivated the development of approaches that fine-tune language models to perform reasoning more effectively by making their reasoning traces in-domain, and by searching for optimal reasoning strategies at training time [183, 184]. In these approaches, during learning, reasoning traces are sampled from a language model on training task instances, and the sampled traces are used during model optimization, for example by fine-tuning only on traces that lead to task success, or by reinforcement learning. Such exploration-based learning is the key element of success for popular models like DeepSeek-R1 [177], OpenAI’s o1 [126], Kimi 1.5 [185], and Stream-of-Search [SoS; 4]. However, these approaches all result in models that produce significantly longer output sequences, which imposes several limitations due to the fact that autoregressive generation is inherently serial: longer output sequences means higher latency, and context window constraints on serving system cap the feasible output length and complexity of reasoning. We also experiment with exploration-based learning for reasoning, but with a crucial difference from these prior approaches: we provide a special action to the language model, allowing it to distribute their inference-time computation into serial *and* parallel operations. In comparison to serialized inference, parallelizing reasoning traces both reduces latency and the stress of context window limitations. Not only do our experimental results confirm this, but we also show that models trained end-to-end to distribute their inference-time compute significantly outperform these existing approaches at the same computational budgets.

Parallelizing inference with multiple independent language model calls has emerged as another avenue for scaling reasoning tasks. For example, in self-consistency [178], multiple

candidate solutions to a reasoning problem are generated independently, and the most common solution among them is chosen. While ensembling methods like these effectively improve task performance, their core limitation is the independence between inference threads; the lack of coordination among threads leads to redundant computation and thus suboptimal resource utilization.

Methods such as Tree of Thought [179], Graph of Thought [254], Skeleton of Thought [193], Atom of Thought [197], and Self-Ask [203] build on these simple ensembling methods by structuring the exploration of reasoning paths into multiple parallelizable calls to run inference on a language model. Several recent approaches also propose to perform reasoning tasks by simulating multi-agent interaction [189, 255, 190, 194, 196, 256, 198]. However, the proposed search structures (and, in the case of multi-agent reasoning, the communication protocol) of these methods are fixed and hand-designed, which limits their flexibility and scalability. Furthermore, these structured inference methods are predominantly prompting-based,¹ without end-to-end optimization, which imposes a wide distributional gap between sequences processed at training and inference-time. Instead, we train language models to structure their own search at inference time, distributing computation between parallelized and serialized operations. In theory, our framework could result in language models that implement the same search structures as existing approaches, such as Tree-of-Thought [179], without explicit prompting or hand-designed orchestration of language model calls.

Concurrent to our work, PASTA [199] and Hogwild! Inference [200], released in February and April 2025 respectively, also explore parallel reasoning. PASTA decomposes a sequential task into parallel sub-tasks that eventually merge back into a single main thread. Our method, however, spawns multiple exploratory threads in parallel and selectively incorporates only successful outcomes, effectively scaling to better performance as compute increases. Hogwild! Inference employs parallel workers to collaboratively solve problems; yet, it remains a prompting method, while APR optimizes collaborative reasoning in an end-to-end manner through supervised training and reinforcement learning, providing a more integrated solution to efficient inference parallelization.

Our work is related to existing algorithms for training language models to directly control the inference process, for both efficiency and performance reasons. For example, PENCIL [201] trains models to adaptively discard parts of their context at inference time to reduce the context length, and thus the time complexity of producing a successful reasoning trace. [202] train language models to inject *additional* “pause” tokens into their output sequences, which are shown to improve end-task performance by increasing inference-time computation. Both approaches train using supervised demonstrations of token discarding (or injection). To the best of our knowledge, we are the first to use end-to-end reinforcement learning to optimize language model control over inference.

¹While [196] employ reinforcement learning for optimizing prompts and connectivity between agents, the underlying language models are still fixed.

Language model serving systems Language model inference is primarily memory-bounded, with modern serving frameworks optimizing performance through batched processing of multiple requests in parallel. Systems like vLLM [209] and SGLang [180] have significantly improved token throughput per request and total throughput across all requests. While these systems can scale horizontally by adding more GPUs or increase total throughput by expanding batch size, they face a fundamental constraint: the token generation rate per individual thread remains limited (typically 50-100 tokens per second for models like GPT-4o). This limitation creates significant latency issues for sequential chain-of-thought methods, as generating reasoning traces for complex problems can take 10-60+ seconds, degrading user experience. Additionally, long reasoning traces quickly consume the available context window. Adaptive Parallel Reasoning addresses these bottlenecks by distributing computation across parallel sub-threads, reducing end-to-end latency while enabling more complex reasoning within practical time constraints and maintaining compatibility with existing serving systems.

E.2 Implementation Details

Model architecture We use a model following Llama2 [205] with 228M non-embedding parameters (293M total parameters) with the following key specifications: 18 hidden layers, 1024 hidden dimension, 16 attention heads, and a 4096 token context window.

Supervised training We conduct supervised training using 128 TPUv3 cores. Across all experiments, we use a batch size of 256 and a learning rate of $5e-5$. The model is trained for 19,000 steps—approximately 10 epochs—following the setup in [4].

Reinforcement learning We run reinforcement learning with 2 Nvidia GPUs with GRPO algorithm [204]. The training batch size is 64, with each sample rolled out 5 times (i.e., the group size is 5). The model is rolled out with SGLang. The temperature for rollout in training is set to 1.0. We use greedy sampling for evaluation unless otherwise stated, since it achieves the best performance for both SoS+ and APR compared to other evaluation temperatures.

Training is conducted using a learning rate of 1×10^{-5} and a PPO clip ratio of 0.2. We run training for 150 total steps, each consisting of 2 inner PPO optimization steps. We apply gradient clipping with a maximum norm of 1.0 and validate performance every 25 steps. For the KL divergence factor used in the GRPO framework, we set the factor to 0.01 for SoS+ baseline and 0.001 for APR for training stability.

E.3 Additional Results with Larger Models

We evaluated both APR and the SoS+ baseline using a 600M-parameter Llama2 model to study scaling with model size. Figure E.1 reports accuracy over average total compute (in

thousands of tokens), following §7.4. Both methods improve with increased model capacity, and APR maintains a substantial lead over SoS+ at every compute level.

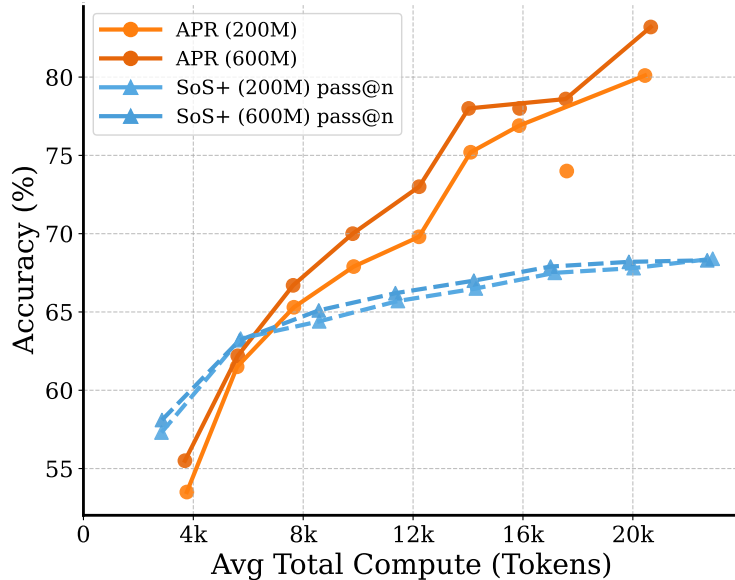


Figure E.1: Accuracy vs. average total compute (in thousands of tokens) for 200M and 600M parameter models. APR consistently outperforms SoS+ and scales more strongly with model size.

E.4 Additional Results with Pretrained Models

To verify that APR’s gains extend beyond models trained from scratch, we fine-tuned a pretrained Qwen2.5 1.5 B model on the same SoS+ and APR demonstration data (approximately 4 k supervised steps). As Table E.1 shows, APR significantly outperforms the SoS+ baseline on Qwen, mirroring the trend observed with Llama2. This demonstrates that our parallel reasoning framework is agnostic to model family, size, and pretraining status.

	Llama2 200M	Qwen2.5 1.5B (pretrained)
SoS+	57.4 %	57.5 %
APR	83.2 %	80.2 %

Table E.1: Accuracy of SoS+ and APR when fine-tuning on a pretrained Qwen2.5 1.5 B model versus a Llama 2 200 M model.

E.5 Extended Context Window Experiments

We evaluated APR and SoS+ on a five-number Countdown variant—whose search space is $40\times$ larger—using context budgets up to 8k tokens. Figure E.2 shows that APR continues to improve up to about 6k tokens and outperforms SoS+ beyond 3.5k tokens, achieving gains of 7% and 11% at an 8k-token budget.

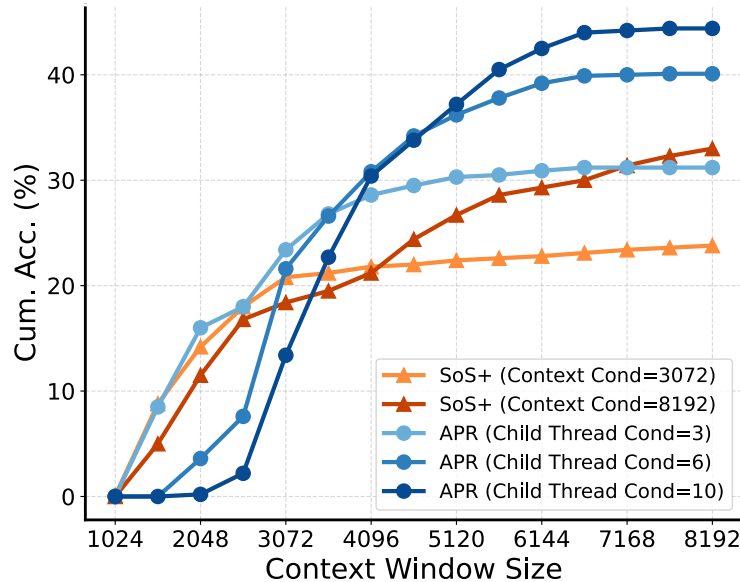


Figure E.2: Accuracy vs. context window size on the five-number Countdown task for SoS+ and APR.

E.6 Symbolic Search Algorithm

We build upon the BFS and DFS algorithm in Stream-of-Search [4] and implement our improved version SoS+ as shown in Algorithm 4.

For the parallel search version, we mostly follow the same algorithm, but when the model decides to do DFS on a certain node, it will spawn a list of sub-searches and explore the nodes in parallel. The algorithm is illustrated in Algorithm 5. We recommend referring to our official implementation available at github.com/Parallel-Reasoning/APR.

SE is a state expansion function that decides what are the next step to explore, while `is_promising` a heuristic function to judge if the current node is promising or not.

Concretely, the state expansion function follows [4], where we select the top K operations ranked by the multiple heuristic. The multiply heuristic considers the target number T and the input numbers and the factors of the target. If T has factors $\{f_1, f_2, \dots, f_m\}$, the multiply

Algorithm 4 SoS+ Symbolic Solver

Input: start_state, goal_state

```

1: state_deque  $\leftarrow$  SE(start_state)
2: while state_deque  $\neq \emptyset$  do
3:   current_state  $\leftarrow$  state_deque.popleft()
4:   if current_state = goal_state then
5:     return "Goal reached"
6:   end if
7:   if IS_PROMISING(current_state) then
8:     result  $\leftarrow$  SOS+(current_state, goal_state)
9:     if result = "Goal reached" then
10:      return "Goal reached"
11:    end if
12:   else
13:     states_to_explore  $\leftarrow$  SE(current_state)
14:     state_deque.extend(states_to_explore)
15:   end if
16: end while
17: return "No result"

```

heuristic can be written as $h_{multiply} = \min(\{|f_j - \sum_{i=1}^{|I|} n_i| \forall j \in [1, m]\})$. Due to the difficulty in implementing an accurate `is_promising` function, we simply let it return True for 10% of the time, which we leave reinforcement learning to further optimize the promising node selection strategies.

E.7 SoS vs. SoS+

Method	Temp = 0.0	Temp = 0.1	Temp = 0.5	Temp = 1.0
SoS	49.5%	49.6%	47.1%	37.9%
SoS+	57.3%	57.1%	52.0%	48.1%

Table E.2: Accuracy comparison between SoS and SoS+ across different temperature settings. SoS+ consistently achieves higher performance, especially at lower temperatures.

Here, we compare the performance of models trained with SoS and SoS+ demonstrations without conditions under different sampling temperatures. As shown in Tab. E.2, our SoS+ approach significantly outperforms the original SoS method.

Algorithm 5 APR Symbolic Solver

```

1: function APR(start_state, goal_state, main = True)
2:   state_deque  $\leftarrow$  SE(start_state)
3:   while state_deque  $\neq \emptyset$  do
4:     current_state  $\leftarrow$  state_deque.popleft()
5:     if current_state = goal_state then
6:       return "Goal reached"
7:     end if
8:     states_to_explore  $\leftarrow$  SE(current_state)
9:     if main  $\wedge$  IS_PROMISING(current_state) then            $\triangleright$  Parallel exploration in
promising states
10:       parallel_results  $\leftarrow$  [APR(s, goal_state, False) | s  $\in$ 
states_to_explore]
11:       for r  $\in$  parallel_results do
12:         if r = "Goal reached" then
13:           return "Goal reached"
14:         end if
15:       end for
16:     else
17:       state_deque.extend(states_to_explore)
18:     end if
19:   end while
20:   return "No result"
21: end function

```

E.8 Effect of SoS+ Supervised Training Data

We study whether improving the quality of supervised demonstrations can narrow the performance gap between SoS+ and APR. Specifically, we evaluate two enhanced training strategies for SoS+: (1) increasing the maximum beam size from 5 to 15 during demonstration generation to mirror the setup used in APR, and (2) applying rejection sampling to curate 500K high-quality training examples that are both correct and fit within the context window. We report results at two decoding temperatures: 0.0 and 1.0.

As shown in Tab. E.3, SoS+ performance is fundamentally limited by the context window size, as it cannot learn from long search paths without parallel exploration. Increasing the beam size improves symbolic solver accuracy but leads to longer trajectories that exceed the model’s context length, ultimately reducing performance. Rejection sampling improves training quality by filtering for context-bounded correct examples, yielding gains at temperature 1.0, but the overall impact remains modest and still falls short of closing the gap to APR.

	Temp = 0.0	Temp = 1.0
SoS+ (baseline)	57.3%	48.1%
SoS+ (beam size = 15)	50.9%	42.5%
SoS+ (rejection sampling)	56.5%	54.5%

Table E.3: Accuracy of SoS+ trained with different supervision strategies.

E.9 Effect of Temperature

We ablate the effect of temperature on how performance scales with compute, extending Fig. 7.4a. As shown in Fig. E.3, APR consistently outperforms SoS+ regardless of temperature, exhibiting greater robustness and efficiency under varied sampling conditions. Again, Table E.4 shows that our results—and their relative advantages—remain consistent across temperatures, both before and after RL.

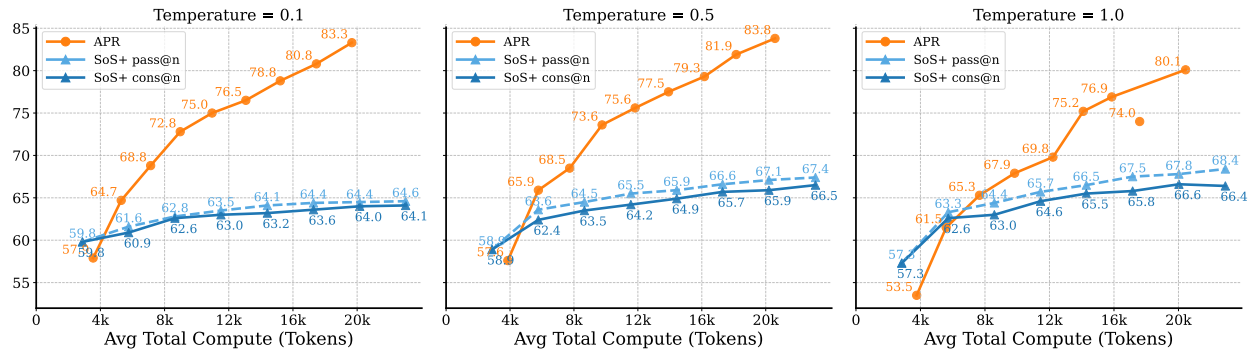


Figure E.3: **Ablation of APR and SoS+ vs. total compute under different temperature settings.** Across all settings, APR demonstrates more reliable and efficient scaling behavior.

E.10 Differences Between Sequential Tokens and Wall-Clock Time

While the number of sequential tokens correlates well with wall-clock time, in practice, there is a slight mismatch due to hardware constraints. Specifically, we utilize an 8-GPU server; however, in scenarios with higher compute demands—such as those involving up to 10 child threads—some GPUs may end up handling multiple child threads, leading to uneven workloads and increased computational load on certain devices. This issue can be mitigated by allocating additional GPUs and improving load balancing during model serving.

Method	Temp = 0.0	Temp = 0.1	Temp = 0.5	Temp = 1.0
<i>SoS+</i>	57.3%	57.1%	52.0%	48.1%
+ RL	60.0%	59.3%	59.1%	58.1%
<i>APR</i>	75.5%	75.9%	74.9%	67.8%
+ RL	83.4%	82.9%	81.3%	76.4%
<i>APR (Num Child Thread Enforced)</i>	83.2%	83.3%	83.8%	80.1%
+ RL	83.3%	83.3%	83.8%	81.9%

Table E.4: Accuracy of different search strategies across sampling temperatures. Each base method is followed by its variant trained with RL.

Appendix F

ThreadWeaver

F.1 Preliminaries

Adaptive Parallel Reasoning

Many reasoning problems naturally contain parallelizable structure, such as independent sub-cases, alternative solution paths, or self-reflection on partial results that can be handled along with exploration. Adaptive parallel reasoning (APR) [208] is a reasoning paradigm in which a language model dynamically decides when to exploit such structure within a single inference trajectory.

Formally, let π_θ be a policy over an extended action space that includes text token emissions and two control actions, **spawn** and **join**. Executing **spawn** at time t creates $m \geq 1$ child threads $\{\tau^{(j)}\}_{j=1}^m$ that evolve independently with no cross-thread reads under π_θ until a **join**. At **join**, a designated reducer aggregates the child outputs (e.g., via concatenation or a learned summarization pattern) and returns the resulting information back to the parent thread, which then continues decoding sequentially.

APR [208, 8, 9] differs from *inter-trajectory* parallelism such as self-consistency and majority vote [178], where multiple complete chains are sampled independently and combined only at the end. In APR, branching and joining occur *inside* a single trajectory with explicit coordination across threads. This allows the model to parallelize decomposable subproblems while maintaining a single coherent solution path. Our framework, ThreadWeaver, follows this paradigm.

Test-time Speed-Accuracy Pareto Frontier

Test-time scaling methods trade additional inference compute for higher accuracy. Prior evaluations typically emphasize accuracy alone, e.g., by generating longer chains of thought or sampling more trajectories for majority vote, while overlooking the accompanying latency costs.

We instead adopt a **speed-accuracy Pareto view**: given a difficult problem and additional compute, we ask whether a method can *reduce the end-to-end time from receiving a user prompt to producing the final answer*, rather than merely increasing accuracy at the cost of slower inference. In our setting, latency is proxied by **token latency**, defined as the number of tokens on the longest thread, or critical path, of a parallel inference trajectory (Section 8.6). For sequential models, this reduces to the total number of generated tokens. This metric correlates well with wall-clock time when parallelism is utilized at inference. Unlike Multiverse [8] and Parallel-R1 [9], which focus on measuring latency per token and the degree of parallelism within a parallel trajectory, we emphasize the speedup to answer generation by comparing the token latency of a parallel model with a sequential model with comparable reasoning capability under the same prompts.

Viewing existing approaches under this measurement reveals sharp differences. Self-consistency [178] and other ensemble approaches improve robustness by sampling multiple full solutions in parallel, yet the critical-path latency remains that of an entire trajectory. Moreover, because trajectories are sampled independently without coordination, they often replicate large portions of the same reasoning process, leading to substantial redundant computation. In contrast, APR introduces structured *intra-run* branching: the model decomposes a problem into multiple threads that execute concurrently and then rejoin. When this decomposition aligns with the underlying problem structure, the longest thread can be significantly shorter than a monolithic long chain-of-thought, yielding strictly lower token latency.

GRPO Preliminary

Group Relative Policy Optimization (GRPO) is a policy-gradient method that computes advantages with intra-prompt normalization [204]. For a prompt p , let $\mathcal{G}_p = \{r_1, \dots, r_k\}$ be the scalar rewards for k trajectories sampled from p . The group-normalized advantage for trajectory i is given in Equation (8.1), which yields advantages that are comparable across prompts, mitigating reward-scale and difficulty variance.

While GRPO can be paired with a PPO-style clipping surrogate [257], in our setting we use fully *on-policy* updates (single rollout, single optimization step), so the importance ratio is effectively 1 and clipping is inactive. We thus optimize the score-function objective

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{p, i, \tau_p^{(i)} \sim \pi_\theta} \left[A_{p, i}^{\text{GRPO}} \sum_{t=1}^{T_{p, i}} \log \pi_\theta(a_t \mid s_t) \right], \quad (\text{F.1})$$

whose gradient is obtained from the standard REINFORCE estimator. In practice, we minimize the pseudo-loss

$$\mathcal{L}_{\text{GRPO}}(\theta) = - \frac{1}{\sum_{p \in \mathcal{B}} \sum_{i=1}^k T_{p, i}} \sum_{p \in \mathcal{B}} \sum_{i=1}^k A_{p, i}^{\text{GRPO}} \sum_{t=1}^{T_{p, i}} \log \pi_\theta(a_t \mid s_t), \quad (\text{F.2})$$

where $T_{p,i}$ is the number of completion tokens for the trajectory $\tau_p^{(i)}$. This loss has gradient $-\nabla_{\theta} \mathcal{J}_{\text{GRPO}}$. Note that we follow DAPO [258] and normalize over tokens across sequences in the batch $(\sum_{p \in \mathcal{B}} \sum_{i=1}^k T_{p,i})$.

F.2 Derivations for P-GRPO

From clipped GRPO to the score-function objective

For completeness, we first establish that P-GRPO with strictly on-policy sampling reduces to a score-function (REINFORCE) estimator with a baseline. Define the PPO-style clipped GRPO surrogate over a batch $\mathcal{B}$ with group size k :

$$\mathcal{J}_{\text{clip}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{p \in \mathcal{B}} \frac{1}{k} \sum_{i=1}^k \mathbb{E}_{\tau_p^{(i)} \sim \pi_{\theta_{\text{old}}}} \left[\min(r_{\theta}(\tau_p^{(i)}) A_{p,i}, \text{clip}(r_{\theta}(\tau_p^{(i)}), 1 - \epsilon, 1 + \epsilon) A_{p,i}) \right], \quad (\text{F.3})$$

where $A_{p,i}$ is the advantage computed via GRPO and the importance ratio is:

$$r_{\theta}(\tau) = \frac{\pi_{\theta}(\tau)}{\pi_{\theta_{\text{old}}}(\tau)}. \quad (\text{F.4})$$

The gradient of the unclipped objective with respect to θ is:

$$\nabla_{\theta} \mathbb{E}[r_{\theta}(\tau) A] = \mathbb{E}[A \nabla_{\theta} r_{\theta}(\tau)] = \mathbb{E}[A r_{\theta}(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau)]. \quad (\text{F.5})$$

In our setting, we perform a single gradient update immediately after data collection (fully on-policy), implying $\theta_{\text{old}} = \theta$. Consequently, $r_{\theta}(\tau) = 1$ identically, rendering the clipping operation inactive. Substituting $r_{\theta}(\tau) = 1$ into Equation (F.5) yields the standard score-function gradient:

$$\nabla_{\theta} \mathcal{J}_{\text{P-GRPO}}(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [A(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau)]. \quad (\text{F.6})$$

Factorization of Parallel Reasoning Trajectories

To implement Equation (F.6) efficiently for tree-structured parallel reasoning, we factorize the probability of the trajectory. Let a parallel trajectory $\tau_p^{(i)}$ be represented as a sequence of M_i disjoint generation segments (or units) produced by the state machine:

$$\tau_p^{(i)} = ((\text{cont}^{(i,1)}, \text{comp}^{(i,1)}), \dots, (\text{cont}^{(i,M_i)}, \text{comp}^{(i,M_i)}))$$

Here, each unit m consists of a context $\text{cont}^{(i,m)}$ (which may include the prompt and previously generated segments) and a newly generated completion $\text{comp}^{(i,m)}$. Without loss of generality, in this derivation, we ignore tokens that we do not apply loss on (e.g., the user prompt and the tokens inserted by the state machine orchestrator). Under this assumption, the set of all

completion segments $\{\text{comp}^{(i,m)}\}_{m=1}^{M_i}$ forms a partition of all tokens generated by the model in the trajectory $\tau_p^{(i)}$.

By the chain rule of probability, the log-probability of the entire trajectory is the sum of the log-probabilities of its generated tokens. Since each completion segment $\text{comp}^{(i,m)}$ is generated conditioned on its specific history $\text{cont}^{(i,m)}$, we have:

$$\log \pi_\theta(\tau_p^{(i)}) = \sum_{m=1}^{M_i} \log \pi_\theta(\text{comp}^{(i,m)} \mid \text{cont}^{(i,m)}). \quad (\text{F.7})$$

This factorization holds regardless of the parallel structure. For example, a shared `<Outline>` segment is simply the completion of the first unit (conditioned on the prompt), while subsequent `<Thread>` segments are completions of later units (conditioned on the prompt and the Outline). The independence of parallel threads is naturally captured by the fact that their contexts branch from the same parent node in the trie.

Broadcasted Credit Assignment

We now show that broadcasting the trajectory-level advantage to all tokens in all threads is mathematically justified by relating it to the gradient of the total trajectory loss. Substituting the factorization from Equation (F.7) into the gradient objective in Equation (F.6):

$$\begin{aligned} \nabla_\theta \mathcal{J}_{\text{P-GRPO}}(\theta) &= \mathbb{E}_{\tau_p^{(i)} \sim \pi_\theta} [A_{p,i}^{\text{P-GRPO}} \nabla_\theta \log \pi_\theta(\tau_p^{(i)})] \\ &= \mathbb{E}_{\tau_p^{(i)} \sim \pi_\theta} \left[A_{p,i}^{\text{P-GRPO}} \nabla_\theta \sum_{m=1}^{M_i} \log \pi_\theta(c_p^{(i,m)} \mid x_p^{(i,m)}) \right] \\ &= \mathbb{E}_{\tau_p^{(i)} \sim \pi_\theta} \left[\sum_{m=1}^{M_i} A_{p,i}^{\text{P-GRPO}} \nabla_\theta \log \pi_\theta(c_p^{(i,m)} \mid x_p^{(i,m)}) \right], \end{aligned} \quad (\text{F.8})$$

where $c_p^{(i,m)} = \text{comp}^{(i,m)}$ and $x_p^{(i,m)} = \text{cont}^{(i,m)}$ for brevity. Equation (F.8) demonstrates that the gradient of the full trajectory objective is simply the sum of the gradients for each completion segment, weighted by the *same* scalar trajectory advantage $A_{p,i}^{\text{P-GRPO}}$.

In our implementation, we approximate the expectation using a Monte Carlo estimate over the batch $\mathcal{B}$. The broadcasting operation described in Section 8.5 assigns the scalar $A_{p,i}^{\text{P-GRPO}}$ to every token in every completion $\text{comp}^{(i,m)}$. When we compute the loss on these tokens weighted by $A_{p,i}^{\text{P-GRPO}}$, we are computing exactly the inner term of the summation in Equation (F.8). Thus, P-GRPO’s advantage broadcasting is not a heuristic approximation. It is the policy gradient estimator for the parallel trajectory reward structure.

F.3 Implementation Details

We provide implementation details for ThreadWeaver training pipeline.

Data Preparation. Training prompts are sourced from the Polaris 53k dataset [225], which contains 53,291 high-quality math reasoning problems. Each prompt contains a question and a reference answer. We perform initial inference with Qwen3-8B to obtain a single reasoning trajectory per prompt. In all generated trajectories, 36,296 (68.1%) contain both a correct final answer (validated by symbolic evaluation) and a valid reasoning structure. These form the base candidate trajectory pool for SFT.

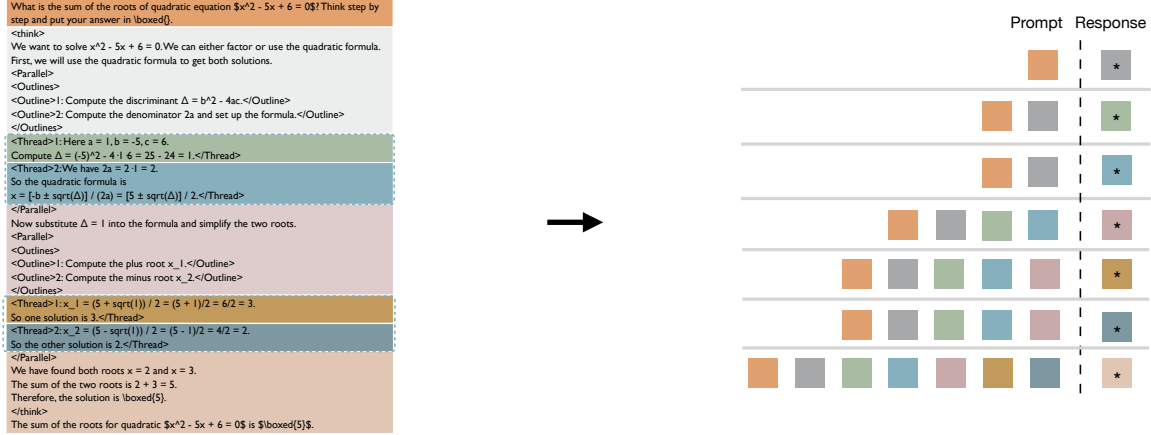
Trie-based Sequence Merging In addition to the single branching case in the example provided by Figure 8.4, our trie-based sequence merging also works on trajectories that contain multiple `<Parallel>` blocks interleaved with sequential reasoning segments. We explain how trie-based sequence merging works with multiple `<Parallel>` blocks in detail.

As illustrated in Figure F.1, we follow the same three steps as in the single-block case: extract all `<context, completion>` units that the inference state machine would issue, insert their token sequences into a shared prefix tree, and traverse the trie to obtain a single packed sequence with an ancestor-only attention mask. Loss is still applied only to completion tokens, so the total number of supervised tokens matches training on each unit independently. Because the model also supports hybrid reasoning (i.e., it supports pure autoregressive decoding by skipping the state machine), we additionally treat the full trajectory as a single `<context, completion>` unit, with the user prompt as the context and the full model completion as the completion. For simplicity, we omit this full trajectory unit in Figure 8.4 and Figure F.1, as including the unit is only to allow the model to perform sequential reasoning with standard autoregressive decoding.

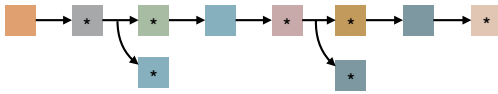
Detailed State-Machine Specification. The inference orchestrator implements a state machine with the following detailed phases, starting with a sequential phase with the user prompt wrapped in a chat template defined by the model as the initial context:

1. **Sequential phase.** Decode sequentially until `</Outlines>` is generated, which is registered as a stop token for the sequential phase.
2. **Parse outlines.** Extract the numbered outline entries from the preceding outline block.
3. **Parallel phase.** For each `<Outline>` entry i , issue a completion request with `<Thread>` i : appended to the previous context that includes all tokens that the LLM generated previously, including the `<Outlines>` block. Generate completions independently until `</Thread>`, which is registered as a stop token for independent generation in the parallel phase. Threads do not attend to each other as they are treated as different requests by the inference engine.

Step 1: Extracting all <context, completion> pairs from a training example



Step 2: Prefix-Tree Building



Step 3: Tree Flattening to Single Sequence

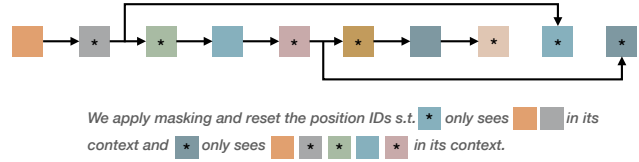


Figure F.1: Trie-based sequence merging with more than one parallel block. We still build a prefix-tree (trie) based on the <context, completion> pairs from the training example and then flatten the tree into one packed training sequence with an ancestor-only attention mask.

4. **Join.** Concatenate the context (prompt, outlines, and all thread texts) and append `</Parallel>` to form the context for the next round of text completion. While we expose all the reasoning in subthreads to the main thread, further optimizations are possible to only return necessary information for subsequent computation, reducing the context usage for reasoning.
5. **Continue.** Proceed sequentially until the next `</Outlines>` token or the `[EOS]` token.

Because we introduce no changes to the LLM architecture or context representation, our method inherits existing serving optimizations (e.g., paged attention, prefix caching) and runs out of the box on common LLM inference stacks that expose a text-completion API. Note that common inference engines such as vLLM and SGLang support prefix caching, which prevents regenerating the KV cache (i.e., rerunning prefilling) for context tokens if they share prefix with a prior request. When the host API supports taking and returning token IDs, all steps above can operate directly on token IDs without text decoding for additional efficiency. The main potential sources of overhead are repeated API calls, which can increase queueing

and scheduling costs especially in trajectories with many short branches, and needing to prefill for threads beyond the first one, which can be expensive when later branches are much longer. In our experiments, these factors did not become bottlenecks, as reflected in the wall-clock speedups reported in Section 8.6, in part because we use an efficient vLLM-based implementation with low scheduling overhead and because prefilling is significantly cheaper than decoding.

Supervised Fine-tuning (SFT). From the 36,296 valid trajectories, we sample 1,000 for rewriting using GPT-5 [224]. Each sample is rewritten with a structured instruction prompt to improve readability and ensure canonical parallelization, while preserving the exact step order and final boxed answer. After automatic filtering for format and semantic alignment, we use the remaining 959 high-quality rewritten trajectories as the training data for SFT.

SFT is performed for 8 epochs with the following hyperparameters:

- **Optimizer:** AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay 0.01.
- **Learning rate:** initial 1×10^{-5} with 5% warmup and cosine decay.
- **Batch size:** 16.
- **Sequence length:** maximum 40,960 tokens.
- **Loss:** standard cross-entropy on the tokens with loss mask specified in Section 8.3.

Self-training. We apply the SFT-trained model to the full 53k Polaris prompt set to generate new reasoning trajectories. Each trajectory is automatically scored by a dual-criteria filter:

1. **Answer correctness:** whether the final boxed expression matches the ground truth, using symbolic equivalence checks implemented via SymPy.
2. **Structural validity:** whether all `<Parallel>` and `</Parallel>` markers form a valid structure, ensuring no partial threads or misuse of special tokens.

The trajectories generated with this process are used for the second-stage fine-tuning.

The self-training SFT stage uses identical hyperparameters to the initial SFT, except we only run for 1 epoch.

Reinforcement Learning. The final stage employs the proposed **Parallel-GRPO (P-GRPO)** algorithm. Parallel rollouts are performed so that each of the prompts is sampled 8 times in each batch, and sequence-level advantages are normalized by group mean within each prompt. Additional hyperparameters are listed as follows:

- **Optimizer:** Adam with learning rate 5×10^{-6} , no weight decay.

Table F.1: Statistics for the three training trajectory datasets. “Sequential” are raw Qwen3–8B trajectories subsampled from the 36k trajectories with the correct answers; “First-stage” are GPT–5–rewritten parallel traces; “Self-train” are trajectories generated by the model trained with data in the first stage and filtered as in §8.4. Parallel and acceleration ratios are computed per trajectory then averaged.

	Sequential Trajectories (sampled)	First-stage	Self-train
Size (format-correct)	1000	959	17,491
Avg. parallel ratio	0.00	0.77	0.68
Avg. longest seq. length	10,377	8,278	9,153
Avg. total tokens	10,377	11,143	11,176
Self-parallelism speedup	1.00	1.35	1.22
Avg. # parallel blocks	0.0	6.26	4.00
Avg. parallel path length	–	541	969
Avg. tokens per parallel block	–	1,542	2,458
Avg. threads per parallel block	–	2.58	2.46

- **Batch size:** 128 (8 rollouts $\times$ 16 prompts).
- **KL penalty:** No KL penalty is applied.
- **Temperature:** 1.0 during rollout and at evaluation.
- **Precision:** bf16.

We run the training for 350 optimization steps. We use vLLM [209] as our parallel inference backend. We leverage fully-sharded data parallel (FSDP, Zhao et al. [259]) in distributed training.

Evaluation Setup.

Evaluation is conducted across six datasets: AIME24 [211], AIME25 [227], AMC23 [212], MATH500 [213], Minerva Math [215], and OlympiadBench [214]. Inference uses a 40k total-token budget across the parallel branches.

F.4 Dataset Statistics

We report core statistics of the three trajectory datasets used in ThreadWeaver training: **1)** the original sequential Qwen3–8B trajectories on Polaris–53K, **2)** the dataset with parallel annotations processed by GPT–5 [224] and used in the first stage of training (Section 8.4), and **3)** the larger self–training dataset generated by the model itself (Section 8.4).

The original trajectories are long and purely sequential, with no acceleration from parallelization. Rewriting with GPT-5 introduces substantial parallel structure: about 77% of tokens lie in parallel blocks, and the critical path is on average about 26% shorter than the total token count, despite a slight increase in overall length. This corresponds to an idealized self-parallelism speedup of roughly $1.35\times$ (computed by dividing the token latency with total number of tokens averaged across the dataset) and provides strong supervision for the fork-join format. We also observe rich structural annotations, with about 2.6 threads per parallel block on average.

As discussed in Section 8.4, however, the rewritten traces are produced post hoc from complete solutions and do not always match how the model naturally decomposes problems at inference time: the strong teacher can rely on global hindsight and sometimes parallelizes by skipping or compressing intermediate steps that the student needs. To close this distribution gap, we perform self-training with the trajectories extracted by the model obtained from the first stage under our proposed parallel inference state machine, then filter trajectories to keep only structurally valid and answer-correct samples, with length and acceleration constraints. The resulting self-training corpus is much larger (17,491 trajectories). Since self-training only takes parallel trajectories that the smaller student model is able to generate ad hoc during reasoning, where it does not have full view of the overall trajectories except for the tokens it has generated, self-training reduces the self-parallelism speedup to $1.22\times$ while remaining strongly parallel, and better reflects what the model itself can reliably parallelize.

Together, our dataset from the two-stage data pipeline provides robust training signal for both supervised fine-tuning and prepares the model for the subsequent reinforcement learning.

F.5 Additional Results

Performance Across Training Stages

We evaluate performance on AIME 24 across successive training stages, measuring both *accuracy* (Avg@32) and *format correctness*, which reflects whether all opened `Parallel` blocks are properly closed. Format correctness serves as a proxy for structural stability, though it does not directly affect answer accuracy since the final result depends only on the last `\boxed{}` expression.

As shown in Table F.2, accuracy drops after the first SFT on 959 examples (74.5%), with limited format correctness (56.4%). This indicates that small-scale supervised data is insufficient for mastering the parallel reasoning. Expanding to 17k examples via self-training substantially improves format correctness (77.0%) while maintaining similar accuracy, showing that model-generated data helps the model internalize structural consistency. Reinforcement learning (RL) further improves accuracy in both cases (77.9% and 79.9%), but format correctness decreases slightly when initialized from the smaller dataset. This is expected since RL directly optimizes outcome and efficiency rather than explicit structural closure.

Table F.2: Stagewise performance on AIME 24. *Format correctness* checks proper closure of all opened `Parallel` blocks and well-formed thread markers (a proxy for structural stability). Common reasons for format error are truncation under token budgets and early stopping without closing the parallel block. Correctness of the final answer depends only on the last `\boxed{}`.

Model	Format Correctness	Accuracy (Avg@32)
Qwen3-8B (Baseline without SFT)	–	76.0%
Qwen3-8B + 1st SFT (959 SFT examples)	56.4%	74.5%
Qwen3-8B + 1st SFT + RL	41.9%	77.9%
Qwen3-8B + self-training (17k SFT examples)	77.0%	74.0%
Qwen3-8B + self-training + RL	72.4%	79.9%

Table F.3: Ablation of sequential–parallel cold-start mixing strategies during SFT on AIME 24. “No sequential mixture cold start” uses only parallel data in SFT. “Multiverse-style mixing recipe” follows a non-gradual schedule that switches between mostly sequential, mixed, and mostly parallel phases. “Linear mixing recipe” uses an eight-stage curriculum that linearly increases the proportion of parallel data across epochs. Neither strategy improves over the parallel-only SFT setting, suggesting that additional sequential data does not help Qwen3-8B transition to parallel reasoning once it already has strong sequential capabilities.

Model	Accuracy (Avg@32)
Qwen3-8B (No sequential mixture cold start)	74.5%
Qwen3-8B (Cold start with Multiverse-style mixing recipe)	73.4%
Qwen3-8B (Cold start with linear mixing recipe)	73.1%

Overall, self-training yields a more stable structural foundation, enabling RL with parallel rollout to achieve higher accuracy and lower token latency.

Cold-start with Sequential Data Mixture

Multiverse [8] proposed to cold-start with a mixture of sequential and parallel data in order to reduce the degradation that can occur when directly fine-tuning on parallel reasoning trajectories. We also ablated such cold-start strategies in our SFT stage, using a dynamic mixture between two datasets: a purely sequential dataset $\mathcal{D}_{\text{seq}}$ built from autoregressive Qwen3-8B trajectories, and a parallel dataset $\mathcal{D}_{\text{par}}$ built from our parallelized trajectories rewritten from $\mathcal{D}_{\text{seq}}$. We then constructed curriculum schedules over SFT epochs that control how often each dataset is sampled. Concretely, we considered two families of curricula: a *staged* schedule that switches in a few coarse phases, following [8], and a gradual schedule that increases the share of parallel data linearly over eight epochs. Both variants are trained

for 8 epochs, following the training schedule of ThreadWeaver, for fair comparison. Only SFT is conducted for this comparison to isolate the effect of each cold-start strategy.

Staged mixture (Multiverse-style mixing recipe). For the staged schedule, we follow [8] and organize SFT into three phases:

- **Phase 1: fully sequential phase.** The model is trained for 2 epochs using only $\mathcal{D}_{\text{seq}}$.
- **Phase 2: mixed phase.** The model is trained for 4 epochs on mixtures of sequential and parallel data, implemented as four one-epoch subphases with different fixed mixing ratios.
- **Phase 3: fully parallel phase.** The model is trained for 2 epochs using only $\mathcal{D}_{\text{par}}$.

Conceptually, the curriculum can be summarized as

$$\underbrace{\mathcal{D}_{\text{seq}}}_{\text{fully sequential phase}} \rightarrow \underbrace{\alpha \mathcal{D}_{\text{seq}} + (1 - \alpha) \mathcal{D}_{\text{par}}}_{\text{mixed phase}} \rightarrow \underbrace{\mathcal{D}_{\text{par}}}_{\text{fully parallel phase}},$$

where $\alpha \in (0, 1)$ controls the fraction of sequential data. In Multiverse’s implementation, α is not a single constant but is implicitly determined by how the two datasets are sliced and concatenated. The four mixed subphases correspond to:

- A first mixed epoch that uses 90% $\mathcal{D}_{\text{seq}}$ and 10% $\mathcal{D}_{\text{par}}$ (effectively $\alpha = 0.9$)
- Two middle mixed epochs that each combine 50% $\mathcal{D}_{\text{seq}}$ and 50% $\mathcal{D}_{\text{par}}$ (effectively $\alpha = 0.5$)
- A final mixed epoch that uses 10% $\mathcal{D}_{\text{seq}}$ and 90% $\mathcal{D}_{\text{par}}$ (effectively $\alpha = 0.1$)

Intuitively, Phase 2 therefore sweeps from mostly sequential, through approximately balanced mixtures, to mostly parallel before transitioning to the fully parallel Phase 3.

Gradual mixture (linear mixing recipe). In the second variant, we construct a smooth curriculum over eight stages, each corresponding to roughly one epoch. We again treat $\mathcal{D}_{\text{seq}}$ as the compatible supervised signal and $\mathcal{D}_{\text{par}}$ as the target format. At epoch $k \in \{1, \dots, 8\}$ we set a target fraction for parallel data

$$\lambda_k = 0.125 \times k,$$

and sample a batch that contains a fraction λ_k from $\mathcal{D}_{\text{par}}$ and $(1 - \lambda_k)$ from $\mathcal{D}_{\text{seq}}$. This produces the progression

$$12.5\% \text{ parallel} \rightarrow 25\% \rightarrow 37.5\% \rightarrow 50\% \rightarrow 62.5\% \rightarrow 75\% \rightarrow 87.5\% \rightarrow 100\% \text{ parallel},$$

Table F.4: Comparison of base model performance on AIME 24 (Avg@32). AR SFT uses sequential Qwen3-8B trajectories; Our SFT uses GPT-5-rewritten parallel trajectories.

Model	Accuracy (Avg@32)
Qwen3-4B-Base + AR SFT (1k SFT examples)	37.4%
Qwen3-4B-Base + Our first-stage SFT (959 SFT examples)	32.4%
Qwen3-8B-Base + AR SFT (1k SFT examples)	55.1%
Qwen3-8B-Base + Our first-stage SFT (959 SFT examples)	45.5%

so the model starts in a regime that is almost fully sequential and ends in a purely parallel regime, with the proportion of parallel examples changing smoothly at each stage. Within each stage we shuffle both datasets and sample without additional bias, so the only variable that changes across epochs is the global ratio between sequential and parallel trajectories.

In both settings, the goal is to cushion the transition from sequential to parallel reasoning, either by a small number of coarse phases (staged mixture) or by a fine-grained eight-epoch ramp (gradual mixture). However, as shown in Table F.3, neither curriculum yielded clear benefits over directly training on parallel data without any sequential mixture. We hypothesize that this is because Qwen3-8B is already a strong reasoning model with robust sequential behavior. When sequential data is continually mixed in, the model receives a strong training signal that reinforces its existing autoregressive reasoning style, which slows down the adaptation to the new parallel format. In other words, the mixed curricula make it harder for the model to fully commit to parallel structure, so the additional sequential examples do not translate into better performance in our setting, despite their success as a cold-start recipe in Multiverse.

ThreadWeaver Reuses Existing Reasoning Capabilities for Parallel Reasoning

To verify that ThreadWeaver can reuse strong reasoning capabilities from the base model without degradation, we compare Qwen3-4B-Base and Qwen3-8B-Base trained on our SFT data from the first stage (959 GPT-5 rewritten trajectories) against the same base models trained on standard autoregressive (AR) SFT data (randomly sampled 1k correct sequential Qwen3-8B trajectories). Both variants are trained for 8 epochs. As shown in Table F.4, the gap between AR SFT and our SFT is roughly 5% for both 4B and 8B models. This demonstrates that ThreadWeaver preserves most of the base model’s reasoning ability despite structural changes and remains compatible with standard reasoning models without extensive re-training. Performance gains from the stronger 8B base carry over to the parallel setting, confirming that our approach scales with model capacity.

Table F.5: **Two notions of efficiency.** Speedup vs. sequential baseline reflects switching from sequential to parallel reasoning. Self-parallelism speedup reflects internal parallel breadth. Speedups are computed per-problem and averaged.

Dataset	ThreadWeaver Longest	Seq. Baseline Longest	Speedup vs. Seq. Baseline	ThreadWeaver Total Tokens	Self-Parallelism Speedup
AIME24	16.9k	19.4k	1.14×	21.1k	1.25×
AIME25	24.0k	24.6k	1.03×	27.3k	1.18×
AMC23	12.0k	13.8k	1.16×	15.0k	1.25×
MATH500	6.4k	7.2k	1.23×	7.7k	1.22×
Minerva Math	7.3k	10.6k	1.53×	8.4k	1.14×
OlympiadBench	12.8k	15.2k	1.21×	15.5k	1.22×

Detailed Experimental Analysis

Critical-Path Speedup Against Sequential Baseline as Robust Efficiency Measurement

We find that metrics that only quantify the amount of parallelization within a parallel reasoning trajectory do not accurately reflect true inference-time acceleration obtained when switching from sequential to parallel reasoning. To rigorously assess efficiency gains, we recommend making comparisons directly against *sequential baselines* trained and evaluated under comparable settings.

As summarized in Table F.5, we report two quantities: 1) *speedup vs. sequential baseline* ($L_{\text{baseline longest}}/L_{\text{ours longest}}$), and 2) *self-parallelism speedup* ($L_{\text{ours total}}/L_{\text{ours longest}}$). Note that for the sequential baseline, the token latency is the same as the total number of generated tokens.

In three of six benchmarks, the self-parallelism speedup significantly exceeds the speedup measured against sequential baselines. This contrast is informative: AIME 25 shows a clear self-parallelism gain (1.18×) but only a modest 1.03× improvement over the baseline.

Thus, self-parallelism speedup is not a complete efficiency metric. For deployment, the decisive measure is the *critical-path speedup vs. the sequential baseline*. As in other forms of parallelization, APR introduces overhead, including the need to generate outline tokens and additional redundancy because threads operate without visibility into each other. This motivates benchmarking ThreadWeaver against sequential RL on Qwen3-8B.

High-Quality SFT Data is Critical for Model Performance

Both our SFT dataset and the Multiverse dataset are constructed from LLM-generated trajectories, but differ in data origin and model alignment. While Multiverse [8] uses trajectories from s1 [216] generated by DeepSeek-R1 [177], our dataset is generated directly from Qwen3-8B outputs and then lightly rewritten by GPT-5 [224]. To compare these sources fairly, we fine-tune Qwen3-8B on each dataset under identical settings (about 1k samples,

Table F.6: SFT data comparison on AIME 24. Our data yields stronger performance. *Results from [9] (Avg@16); others use Avg@32.

Model	Acc.
Qwen3-8B + Multiverse SFT (1k samples)	62.2%
Qwen3-8B + Our SFT (959 samples)	74.5%
Qwen3-4B-Base + Parallel-GSM8K (7.5k)	10.6%*
Qwen3-4B-Base + Parallel-GSM8K + RL	19.4%*
Qwen3-4B-Base + Our SFT (959 samples)	32.4%

8 epochs). In this ablation, we expand the internal nested parallelization structures in the Multiverse dataset to match the behavior of our model, following how Yang et al. [8] curates data for autoregressive baselines. This is not expected to impact accuracy significantly since the accuracy only fluctuates slightly even when all parallelization structures are expanded into sequential structures, as reported in Yang et al. [8]. To further ablate the impact of data quality, we also fine-tune Qwen3-4B-Base, the backbone used in Parallel-R1 [9], on our dataset. Parallel-R1 uses its proposed Parallel-GSM8K, curated with DeepSeek-R1-0528-Qwen3-8B [177] based on GSM8K [260].

As shown in Table F.6, training on the Multiverse dataset yields lower performance despite its stronger teacher model. This indicates that dataset-model compatibility is more important than teacher strength in our setting. We hypothesize that DeepSeek-R1 trajectories differ stylistically from Qwen3-8B’s native reasoning patterns, introducing distributional mismatch and limiting generalization. In contrast, our dataset preserves Qwen3-8B’s reasoning semantics while primarily adjusting structural organization to expose parallel reasoning patterns. This allows the model to acquire parallel structure effectively without disrupting its underlying reasoning behavior.

Our SFT model also substantially outperforms Parallel-R1. This is partially due to the fact that Parallel-GSM8k is curated from grade school math dataset GSM8k [260], which is likely too simple compared to competition-level math benchmarks. Although Parallel-R1 applies RL, the gains are modest and performance remains much lower. This suggests that *RL cannot fully compensate for lower-quality SFT data*. Both comparisons indicate that high-quality, model-aligned SFT data is a key factor in downstream performance and parallelization ability, whereas additional RL provides only limited gains when the SFT distribution is poorly matched to the target data distribution.

Qualitative Study

To illustrate how ThreadWeaver works in practice, we analyze a trigonometric evaluation example from MATH500 below, in which the model spontaneously forks into multiple reasoning branches to explore distinct solution strategies and joins the reasoning threads to

continue the next steps of reasoning. Due to the excessive length of the reasoning trajectories, we use brackets with summarized content in place of bulk reasoning text.

In the first parallelization block, the model explores two distinct approaches to the problem. While the second one reaches a numerical solution, the first one ends up switching to another direction that is more promising. Then in the second parallelization block, the first branch continues to explore the direction that the first branch of the first parallelization block is exploring to get to an exact answer, while the second branch performs self-reflection to double check the numerical answer. These branches parallelize well because they operate on disjoint subgoals that are independent of each other given the shared context. Finally, the main thread integrates the information from all the previous reasoning results and output solutions to the user. This example highlights how ThreadWeaver enables adaptively parallelized exploration and self-reflection within a single inference trajectory.

ThreadWeaver Example: Trigonometric Expression Evaluation

Prompt:

Evaluate

$$\sin(\arcsin 0.4 + \arcsin 0.5) \cdot \sin(\arcsin 0.5 - \arcsin 0.4).$$

Let's think step by step and output the final answer within `\boxed{}`.

Response:

<think>

[Identifies the problem as evaluating $\sin(\arcsin 0.4 + \arcsin 0.5) \cdot \sin(\arcsin 0.5 - \arcsin 0.4)$. Defines $A = \arcsin(0.4)$ and $B = \arcsin(0.5)$, rewriting the expression as $\sin(A + B) \sin(B - A)$. Reviews basic sine addition/subtraction formulas and considers expanding directly before looking for a better simplification strategy.]

<Parallel>

<Goal>

<Outline>1: Use $\sin(A + B) = \sin A \cos B + \cos A \sin B$ and $\sin(B - A) = \sin B \cos A - \cos B \sin A$, then expand the product to get a difference of squares of the form $(\sin B \cos A)^2 - (\cos B \sin A)^2$.</Outline>

<Outline>2: Compute $\cos A$ from $\sin A = 0.4$ via $\cos A = \sqrt{1 - \sin^2 A}$ and compute $\cos B$ from $\sin B = 0.5$ via $\cos B = \sqrt{1 - \sin^2 B}$, then substitute these values into the simplified-product expression and carry out the arithmetic.</Outline>

</Goal>

<Path>

1: [Attempts expansion, realizes it's cumbersome, and switches to using the product-to-sum identity. Applies $\sin \alpha \sin \beta = (\cos(\alpha - \beta) - \cos(\alpha + \beta))/2$ with $\alpha = A + B$ and $\beta = A - B$, simplifying the product to $(\cos(2A) - \cos(2B)) / 2$.]

</Path>

<Path>

2: [Uses numerical approximations for $\cos A$ and $\cos B$, then evaluates $\sin(A + B)$ and $\sin(B - A)$ numerically, confirming the product is about 0.09.]

</Path>

</Parallel>


```

<Parallel>
<Goal>
<Outline>1: Use the product-to-sum identity  $\sin(A+B) \cdot \sin(B-A) = [\cos(2B) - \cos(2A)]/2$ , compute  $\cos(2A)$  and  $\cos(2B)$  from  $\sin A$  and  $\sin B$  via  $\cos(2\theta) = 1 - 2\sin^2\theta$ , then substitute to get the exact value and check it matches the earlier numerical estimate.</Outline>
<Outline>2: Compute  $A = \arcsin(0.4)$  and  $B = \arcsin(0.5)$  numerically, evaluate  $\sin(A+B)$  and  $\sin(B-A)$  directly, multiply to obtain a numerical approximation, and compare with the symbolic result.</Outline>
</Goal>
<Path>
1: [Computes  $\cos(2A)=0.68$  and  $\cos(2B)=0.5$  using the double-angle identity, plugs into  $(\cos(2A)-\cos(2B))/2$ , and obtains exactly 0.09.]
</Path>
<Path>
2: [Approximates  $A$  and  $B$  numerically, computes  $\sin(A+B)$  and  $\sin(B-A)$ , and confirms the product closely matches 0.09.]
</Path>
</Parallel>

```

[Concludes that both algebraic and numerical methods agree: the value of the expression is 0.09.]

```

**Final Answer**
\boxed{0.09}
</think>

```

To evaluate the expression:

$$\sin(\arcsin 0.4 + \arcsin 0.5) \cdot \sin(\arcsin 0.5 - \arcsin 0.4),$$

we begin by introducing substitutions for simplicity:

Let:

- $A = \arcsin(0.4)$, so $\sin A = 0.4$,
- $B = \arcsin(0.5)$, so $\sin B = 0.5$.

Then the expression becomes:

$$\sin(A + B) \cdot \sin(B - A).$$

Step 1: Use Trigonometric Identity

We apply the identity:

$$\sin(A + B) \cdot \sin(B - A) = \frac{\cos(2A) - \cos(2B)}{2}.$$

This identity is derived from the product-to-sum formulas and is particularly useful here since we can compute $\cos(2A)$ and $\cos(2B)$ using the double-angle identity:

$$\cos(2\theta) = 1 - 2\sin^2(\theta).$$

Step 2: Compute $\cos(2A)$ and $\cos(2B)$

$$\cos(2A) = 1 - 2\sin^2(A) = 1 - 2(0.4)^2 = 1 - 0.32 = 0.68,$$

$$\cos(2B) = 1 - 2\sin^2(B) = 1 - 2(0.5)^2 = 1 - 0.5 = 0.5.$$

Step 3: Apply the Identity

$$\sin(A + B) \cdot \sin(B - A) = \frac{\cos(2A) - \cos(2B)}{2} = \frac{0.68 - 0.5}{2} = \frac{0.18}{2} = 0.09.$$

Step 4: Final Answer

Thus, the exact value of the original expression is:

$$\boxed{0.09}$$

Per-problem Speedup Without Filtering

We also study per-problem speedup without filtering for correctness, as shown in Figure F.2. In this setting every rollout is included, so the distributions reflect the speedup that a user would observe when running each model end to end, irrespective of whether the final answer is correct. While the maximum values in the upper tail are harder to interpret, since they could capture a potential case that our model terminates earlier incorrectly while the autoregressive baseline continues reasoning and succeeds, the overall distributions remain very similar to those in Figure 8.5. In particular, the bulk of the mass still lies to the right of the $1.0\times$ reference line and the shape of the histograms changes only slightly. Empirically, this indicates that the measured acceleration is not primarily driven by sacrifices in accuracy, but instead reflects real reductions in token latency relative to the sequential baseline.

Error Analysis: Repetition in Branches

We show a failure case below. Although the answer is correct, this is a failure case because the purportedly parallel threads do not provide genuinely complementary reasoning and instead largely duplicate the same computation: both compute the exponent of 5 in $42!$ via the standard floor-sum formula and both verify the method on smaller factorials such as $10!$ and $25!$, so the parallel structure adds little informational value.

Moreover, the alignment between outlines and threads is not respected: rather than having one path focus on the main calculation for $42!$ and the other confined to checking the method on smaller examples, each path blends these tasks and performs both the primary computation and the auxiliary validations. This redundancy and mixing of roles undermines the intended division of labor across paths and illustrates why the example does not realize the benefits of structured parallel reasoning, indicating the room for improvements in the instruction following capabilities.

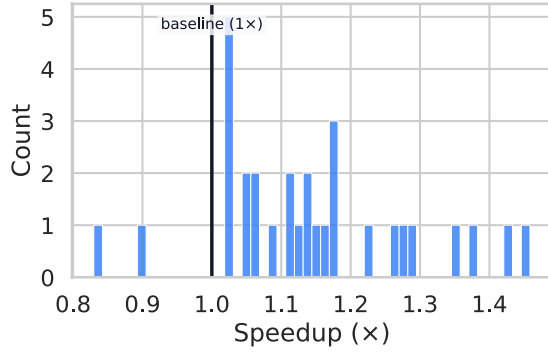
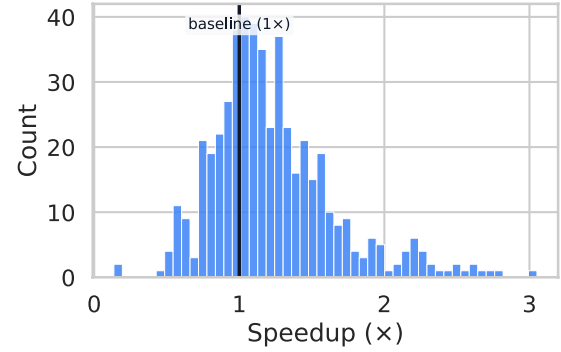
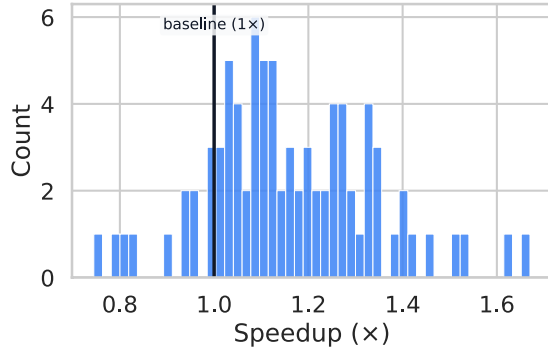
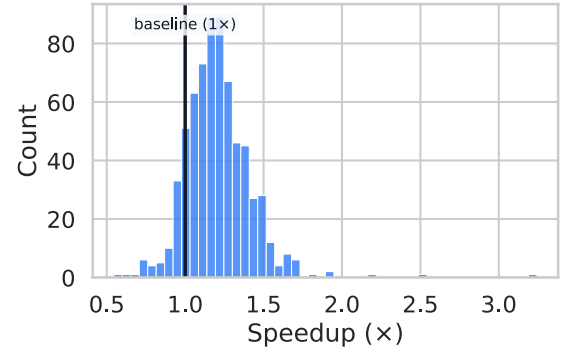
(a) AIME24 (Mean speedup: 1.14 $\times$)(b) MATH500 (Mean speedup: 1.23 $\times$)(c) AMC23 (Mean speedup: 1.17 $\times$)(d) OlympiadBench (Mean speedup: 1.20 $\times$)

Figure F.2: **Per-problem speedup distributions on math benchmarks without filtering for correctness.** The speedup is the ratio between the token latency of the autoregressive baseline and the token latency of ThreadWeaver. A vertical reference line at 1.0 $\times$ marks parity with the sequential baseline. In contrast to Fig. 8.5, these histograms include all rollouts rather than restricting to samples with correct final answers. However, taking both sets of histograms into account, we show that ThreadWeaver achieves substantial speedup on most problems no matter whether correctness filtering is performed.

ThreadWeaver Example (Failure Case): Number of trailing zeros in 42!

Prompt:

How many zeroes are at the end of 42! (42 factorial)? (Reminder: The number $n!$ is the product of the integers from 1 to n . For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$.) Let's think step by step and output the final answer within `\boxed{}`.

Response:

<think>

[Recognizes that trailing zeros come from factors of $10 = 2 \cdot 5$ in the prime factorization, and that in factorials there are many more 2s than 5s, so the number of trailing zeros is the exponent of 5 in $42!$. Plans to use the standard floor-sum formula.]

<Parallel>

<Outlines>

<Outline>1: Use the formula for the exponent of 5 in $42!$: $\text{sum floor}(42/5^k)$ over $k = 1, 2, \dots$ until the terms are zero.</Outline>

<Outline>2: Validate the method on smaller factorials such as $10!$ and $25!$ where the number of trailing zeros is known.</Outline>

</Outlines>

<Thread>

1: [Computes $\text{floor}(42/5) = 8$, $\text{floor}(42/25) = 1$, $\text{floor}(42/125) = 0$, then adds $8 + 1 + 0 = 9$ and concludes that $42!$ has 9 trailing zeros, since 5s are the limiting factor compared to 2s.]

</Thread>

<Thread>

2: [Applies the same floor-sum to $10!$ and $25!$, obtaining 2 and 6 trailing zeros respectively, which match the known values and confirm the correctness of the counting method.]

</Thread>

</Parallel>

<Parallel>

<Outlines>

<Outline>1: Reinterpret the floor-sum as directly counting multiples of 5 and extra factors from higher powers such as 25 among the integers from 1 to 42.</Outline>

<Outline>2: Optionally count powers of 2 in $42!$ to confirm that there are more than enough 2s so that 5s indeed determine the number of trailing zeros.</Outline>

</Outlines>

<Thread>

1: [Lists multiples of 5 up to 42: 5, 10, 15, 20, 25, 30, 35, 40 (8 numbers), each contributing at least one factor 5. Notes that $25 = 5^2$ contributes one additional factor 5, giving $8 + 1 = 9$ factors of 5 in total, consistent with the floor-sum calculation.]

</Thread>

<Thread>

2: [Computes the exponent of 2 in $42!$ as $\text{floor}(42/2) + \text{floor}(42/4) + \text{floor}(42/8) + \text{floor}(42/16) + \text{floor}(42/32) = 21 + 10 + 5 + 2 + 1 = 39$, which is much larger than 9, so the count of trailing zeros is indeed determined by the 9 factors of 5.]

</Thread>

</Parallel>

[Concludes that, since there are 9 factors of 5 and more than enough factors of 2, $42!$ has 9 trailing zeros.]

****Final Answer****

\boxed{9}

</think>

To determine how many trailing zeros are in $42!$, we look at how many times 10 divides $42!$. Since $10 = 2 \cdot 5$ and factorials contain many more factors of 2 than 5, the number of trailing zeros is the exponent of 5 in the prime factorization of $42!$.

We use the standard formula for the exponent of a prime p in $n!$:

$$v_p(n!) = \sum_{k=1}^{\infty} \left\lfloor \frac{n}{p^k} \right\rfloor.$$

Here $p = 5$ and $n = 42$:

$$\left\lfloor \frac{42}{5} \right\rfloor = 8, \quad \left\lfloor \frac{42}{25} \right\rfloor = 1, \quad \left\lfloor \frac{42}{125} \right\rfloor = 0.$$

So the total number of factors of 5 in $42!$ is

$$8 + 1 + 0 = 9.$$

Since there are far more factors of 2 than 5 in $42!$, each of these 9 factors of 5 pairs with a factor of 2 to form a factor of 10, which contributes one trailing zero.

Therefore, the number of trailing zeros in $42!$ is

$$\boxed{9}.$$

F.6 Reasoning Trajectory Annotation Pipeline for ThreadWeaver

This section describes the multi-step pipeline that converts reasoning trajectories extracted from Qwen3-8B into a structured, parallelizable supervision signal for ThreadWeaver. Our pipeline consists of five steps:

1. Stage 1 uses GPT-5 to identify potential parallel structures within sequential reasoning traces.
2. Stage 2 rewrites these parallel structures into canonical <Thread> format.
3. Stage 3 performs revision against allowed and forbidden contexts to make the transition between context and each thread smooth.
4. Stage 4 regenerates an outline for every branch.
5. Stage 5 runs filtering to remove samples with failed rewriting or format errors to produce the final training corpus.

We describe the implementations of each step in detail.

Step 1 In Stage 1, raw trajectories are line-numbered and assigned stable identifiers (Lxx). This step ingests the resulting question–answer trajectories and has the language model reconstruct the entire chain of thought into a rigorously indexed outline that reuses these line identifiers. Each reasoning trace is presented in full, the major conceptual transitions are enumerated as top-level steps with precise line spans, and aborted explorations are retained to preserve the empirical distribution of strategies. The model is further instructed to decompose these top-level steps into contiguous substeps whenever they contain multiple thought units, producing a two-level hierarchy.

After the hierarchical labeling, the model analyzes dependencies to determine which steps or blocks can run concurrently, emitting sequential and parallel structures together with short explanations. Finally, the model serializes everything (step descriptions, substeps, and parallel threads) into a normalized XML-style summary. The line-range annotations introduced at the input are preserved in this summary and are propagated verbatim through all subsequent steps.

All the tasks in this step are conducted in a multi-round conversation, and the inputs and outputs from previous tasks are visible to later tasks.

Task 1: Main-step extraction and global indexing. Task 1 ingests the fully line-numbered reasoning transcript, where every sentence is prefixed with immutable tags such as L1, L2, and so on. Leveraging this dense context, the model enumerates all top-level steps sequentially, captures dead ends, and records contiguous line ranges Lxx-Lyy for each step, thereby establishing the canonical coordinate system that downstream tasks will reuse verbatim.

Task 1 Prompt for Step 1 Processing in ThreadWeaver

```

**Task 1: Main-Step Extraction**
Analyze the given long reasoning chain for a math problem and extract every *main*
step of it. Ignore substeps-only list the top-level thinking processes or actions.

**Use the following output format:**
...
S1: [Description of step 1] (Lxx-Lyy)
S2: [Description of step 2] (Lxx-Lyy)
...
SX: [Description of step X] (Lxx-Lyy)
...

**Guidelines:**
- Label each top-level step consecutively (`S1`, `S2`, ...).
  - Do *not* include any sub-numbering (no `S2.1`, etc.).
- Please capture the entire thought process presented in the reasoning chain, and
do not skip any step that includes, but not limited to, the following:
  - Initial problem understanding and analysis
  - All exploration paths (both successful and unsuccessful)

```

- Case studies, checks, or tests performed
- Any “aha” or correction (re-evaluation or re-thinking) moments
- The final reasoning that yields the solution
- Split multiple cases or scenarios into different steps. Each case should be allocated an independent step.
- If a reasoning path is started but later abandoned because it cannot reach a conclusion, still capture it as a distinct step.
- Please indicate the line numbers corresponding to where each main step starts and ends in the original reasoning chain (`Lxx-Lyy`).
- Use the format `Lxx-Lyy` to specify the range of lines for each step.
- S1 starts at the first line and ends at the line you list in the parentheses (`S1: [Description of step 1] (L1-Lyy)`).
- S(k+1) starts on the next line after Sk ends. The delta between the starting line number of S(k+1) and the ending line number of S(k) is always 1.
- Ensure there is no gap between the end of one step and the start of the next.
- The last step must end at the final line of the original reasoning chain.
- Provide a brief summary of the content in each step that is one or two sentences long (e.g. `SX: [Description of step X]`).
- Keep the description for each step concise yet descriptive.
- A reasoning chain may contain any number of main steps. There is no limit to the total number of steps you can identify.

Task 2: Substep extraction within fixed ranges. Task 2 reuses both the raw transcript and the Task 1 step list as shared context, slicing each Sx span into adjacent substeps (Sx.1, Sx.2, ...) without gaps or overlaps. Because every line inside the parent range is still labelled with the original line numbers, the subdivision maintains identical Lxx-Lyy coverage while adding finer-grained anchors for later tasks.

Task 2 Prompt for Step 1 Processing in ThreadWeaver

```

**Task 2: Substep Extraction**
Given the previous output containing all main steps of a reasoning chain, break
down each main step into substeps **only if** its content from the original
reasoning chain can be meaningfully divided into smaller thought units. You can use
the line ranges to reference the original reasoning chain and better capture the
content of each main step.

**Output format:**
...
S1: [Description of step 1] (Lxx-Lyy)
  S1.1 [Description of substep 1.1] (Lxx-Lyy)
  S1.2 [Description of substep 1.2] (Lxx-Lyy)
  ...
  S1.X [Description of substep 1.X] (Lxx-Lyy)
S2: [Description of step 2] (Lxx-Lyy)
S3: [Description of step 3] (Lxx-Lyy)
S4: [Description of step 4] (Lxx-Lyy)
  S4.1 [Description of substep 4.1] (Lxx-Lyy)

```

```

...
...
SX: [Description of step X] (Lxx-Lyy)
...
...

**Guidelines:**
- A substep always uses the same parent index (`X`) as its main step (e.g. if
  breaking down `S2`, label `S2.1`, `S2.2`, ...).
- For each main step, capture the entire thought process presented in it, and do
  not skip any substep that includes, but not limited to, the following:
  - Initial problem understanding and analysis
  - All exploration paths (both successful and unsuccessful)
  - Case studies, checks, or tests performed
  - Any “aha” or correction (re-evaluation or re-thinking) moments
  - The final reasoning that yields the solution
- Split multiple cases or scenarios into different substeps. Each case should be
  allocated an independent substep.
- Provide a brief summary of the content in each substep (e.g., `S1.2 [Description
  of step 1.2]`) that is one or two sentences long.
- Keep each substep description concise and focused.
- Provide the line range for each substep (`Lxx-Lyy`).
  - The format Lxx-Lyy indicates the range of lines where the substep appears.
  - SX.1 starts at the first line of SX.
  - SX.(k+1) starts on the next line after SX.k ends.
  - Ensure there is no gap between the end of one substep and the start of the
    next.
  - The last substep must end at the final line of the parent main step.
- Keep the main step description and line ranges (`Lxx-Lyy`) as marked in Task 1.
- A main step may contain any number of substeps. There is no limit to the total
  number of substeps you can identify for a main step.
- If a main step does not contain substeps, copy over its step number, description
  and line range exactly as they appear in Task 1.
- Do not introduce nesting deeper than 2 levels. For example, `S2.1.1` is not
  allowed.

```

Task 3: Parallelizing main steps. Task 3 consumes the Task 1 summary and its line ranges, analyzes dependency relationships between adjacent steps, and expresses valid parallel groupings only when two steps do not rely on the same numbered lines. The reuse of the original annotations ensures that the resulting sequential and parallel blocks are traceable back to the same context without recomputing boundaries.

Task 3 Prompt for Step 1 Processing in ThreadWeaver

****Task 3: Parallelizing Main Steps****

Using only the **main steps** (`S1`, `S2`, ...) extracted in Task 1, first identify all steps or contiguous step groups that can be executed in parallel without violating logical dependencies; then rewrite them as a structured outline for parallel execution.

Output format:

...

Parallel groups:

P1: Parallel(S2, S3)

P2: Parallel(Sequential(S5, S6), S7)

...

PX: Parallel(Si, Sj, Sk...)

Outline for parallel execution:

S1: [Description of step 1] (Lxx-Lyy)

P1: Parallel[Parallel reason: ...](

 S2: [Description of step 2] (Lxx-Lyy),

 S3: [Description of step 3] (Lxx-Lyy)

)

S4: [Description of step 4] (Lxx-Lyy)

P2: Parallel[Parallel reason: ...](

 Sequential(

 S5: [Description of step 5] (Lxx-Lyy),

 S6: [Description of step 6] (Lxx-Lyy)

),

 S7: [Description of step 7] (Lxx-Lyy)

)

...

PX: Parallel[Parallel reason: ...](

 Si: [Description of step i] (Lxx-Lyy),

 ...

)

...

Guidelines:

1. **Identify Parallel Groups**

- Find groups of adjacent main steps with **no dependencies** among them.
- Label the groups as `P1`, `P2`, ... and represent the structure of steps within the group using the syntax pattern as shown above.
- Group steps Si, Sj, ... in `Parallel(Si, Sj, ...)` **if and only if** Sj does NOT depend on any part of Si.
- In a parallel block, you can combine adjacent steps that depend on each other into a `Sequential(Sx, Sy)` block. A sequential block can be executed in parallel to other main steps and sequential blocks.
- Do not create `Sequential` blocks that are not nested in any parallel block. Simply list the steps in order in those cases.
- Groups must cover contiguous step ranges in the original order.
- Interpretation of the problem should not be parallelized with the process of problem-solving.

- Different ways of self-reflection are often independent of each other and thus parallelizable.
 - Self-reflection on an intermediate step is often parallelizable with the subsequent steps.
 - In conditional logic, treat the ****if**** branch and ****else**** branch as independent tasks and parallelize them even though their outputs cannot both occur at runtime.
 - Only include steps that can be executed in parallel with others in a parallel group; the parallel groups do not need to cover every step.
 - A parallel group should not consist of only one single step or one ``Sequential(...)`` block. If this is the case, do not include those steps into any parallel groups.
2. ****Write the Structured Outline for Parallel Execution****
- Annotate the main steps with the parallel grouping structures.
 - Keep each main step's description and line range the same as those annotated in Task 1 (preserve as much as you can).
 - Include ****every**** step exactly once, either alone, inside a sequential block, or inside a parallel group.
 - If a main step does not belong to any sequential block or parallel group, copy over its step number, description and line range exactly as they appear in Task 1. Include it in the outline in its original position.
 - For a sequential block, list each step in order.
 - If a parallel group contains multiple units, briefly state why its units are independent in the ``parallel reason`` bracket (e.g. ``PX: Parallel[Parallel reason: ...]``).
- ****Contiguous Blocks:**** You may combine multiple adjacent steps (e.g. ``Sequential(S1, S2, ...)``) and treat them as a single unit in the parallel group alongside another step or block (e.g. ``S3``), but do ****not**** combine non-adjacent steps into a block.
 - ****Contiguous grouping:**** Each parallel group must cover a continuous sequence of steps or blocks. In other words, you may only parallelize adjacent steps. The occurrence of steps in parallel groups must follow their original order. For example, `P1: Parallel(S5, S8)` is not allowed.
 - ****Strict Parallelism Only:**** Build an explicit dependency graph in your analysis: draw an edge from step A to step B if B uses A's output or insight. A group ``Pi`` may include steps (or contiguous blocks) ****only**** if there are ****no**** dependency edges between any two units.

Task 4: Parallelizing substeps. Task 4 layers on top of the substep catalog from Task 2 and the parallel structure from Task 3, examining each main step in isolation to see which adjacent substeps can proceed concurrently. Because the inputs still reference the exact Lxx-Lyy intervals, the model can safely bundle parallel-friendly fragments while citing the same line numbers, preserving context reuse down to the second hierarchy level.

Task 4 Prompt for Step 1 Processing in ThreadWeaver

****Task 4: Parallelizing Substeps****

Examining the outline you extracted in Task 3, for any main steps that are not in a `Parallel` block, refer to its ****substeps**** you extracted in Task 2 if there is any (e.g. `S1.1`, `S1.2`, ...). Among these substeps, identify all substeps or contiguous substep groups that can be executed in parallel without violating logical dependencies. Then rewrite the structured parallel execution outline to include both the parallel structures of the main steps and those of the substeps you just identified.

****Output format:****

...

Parallel groups of substeps:

```
* Parallel(Sequential(S1.1, S1.2), S1.3)
* Parallel(S4.2, S4.3)
* ...
* Parallel(...)
```

Outline for parallel execution:

```
S1: [Description of step 1] (Lxx-Lyy)
  P-sub: Parallel[Parallel reason: ...](
    Sequential(
      S1.1: [Description of substep 1.1] (Lxx-Lyy),
      S1.2: [Description of substep 1.2] (Lxx-Lyy)
    ),
    S1.3: [Description of substep 1.3] (Lxx-Lyy)
  )
P1: Parallel[Parallel reason: ...](
  S2: [Description of step 2] (Lxx-Lyy),
  S3: [Description of step 3] (Lxx-Lyy)
)
S4: [Description of step 4] (Lxx-Lyy)
  S4.1: [Description of substep 4.1] (Lxx-Lyy)
  P-sub: Parallel[Parallel reason: ...](
    S4.2: [Description of substep 4.2] (Lxx-Lyy),
    S4.3: [Description of substep 4.3] (Lxx-Lyy)
  )
P2: Parallel[Parallel reason: ...](
  Sequential(
    S5: [Description of step 5] (Lxx-Lyy),
    S6: [Description of step 6] (Lxx-Lyy)
  ),
  S7: [Description of step 7] (Lxx-Lyy)
)
...
PX: Parallel[Parallel reason: ...](
  Si: [Description of step i] (Lxx-Lyy),
  ...
)
```

```

...

**Guidelines:**
1. **Identify Parallel Groups**
  - Only parallelize the substeps whose main steps are not in a parallel block.
  - Find sets of adjacent substeps with no dependencies among them.
  - Label the groups of substeps `P-sub`, and list their substep members (e.g.
    `Parallel(Sequential(S1.1, S1.2), S1.3)`, `Parallel(S4.2, S4.3)`).

2. **Write the Structured Outline for Parallel Execution**
  - Annotate the main steps and substeps with the parallel group structures.
  - Keep the parallel group structure, numbering, description and line ranges of
    the main steps the same as those annotated in Task 3 (preserve as much as you
    can).
  - Keep each substep's numbering, description and line range the same as those
    annotated in Task 2 (preserve as much as you can).
  - For a sequential block, list the description of each substep in order.
  - If a parallel group contains multiple units, briefly state why its units are
    independent in the `parallel reason` bracket (e.g. `PX: Parallel[parallel reason:
    ...]` )
  - If a substep does not belong to any parallel or sequential group, copy over its
    step number, description and line range exactly as they appear in Task 2. Include
    it in the outline in its original position.

- **Coverage:** For each main step you choose to annotate, include every
  substep exactly once, either alone or inside a parallel group.
- **Contiguous Blocks:** In a parallel block, you may combine adjacent substeps
  that depend on each other into a contiguous block (e.g. `Sequential(S2.1, S2.2)`)
  and treat them as a single unit in the parallel group alongside another substep or
  block (e.g. `S2.3`). Do not combine non-adjacent substeps into a block.
- Do not create `Sequential` blocks that are not nested in any parallel block.
  Simply list the substeps in order in those cases.
- Only group substeps corresponding to the same main step in a single-level
  parallel or sequential block (e.g. `Parallel(S2.2, S3.1)` or `Sequential(S2.1, S2.2,
  S3.1)` is not allowed).
- Strict Parallelism Only: Build an explicit dependency graph in your analysis:
  draw an edge from substep A to substep B if B uses A's output or insight. A group
  `Pi` may include steps (or contiguous blocks) only if there are no
  dependency edges between any two units.
- In conditional logic, treat the if branch and else branch as independent
  tasks and parallelize them even though their outputs cannot both occur at runtime.
- Contiguous grouping only: Each parallel group must cover a continuous sequence of
  steps or blocks. In other words, you may only parallelize adjacent steps or
  substeps. The occurrence of steps or substeps in parallel groups must follow their
  original order. For example, `P1: Parallel(S2.2, S3.1)` is not allowed.

```

Task 5: XML-style serialization with ranges. Task 5 repackages everything derived so far—main steps, substeps, and parallel groupings—into the prescribed XML-like outline. Each `<Thread>` and `<Parallel>` element explicitly repeats the inherited Lxx-Lyy spans, so

the serialized structure is a faithful rendering of the earlier outputs that downstream tooling can align with either the raw transcript or the intermediate tables.

Task 5 Prompt for Step 1 Processing in ThreadWeaver

****Task 5: Annotate the Overall Structured Outline with Line Range for Each Parallel Block****

According to the conversation above, output the overall parallel execution structure of the long reasoning chain with the main steps, substeps and the parallel structures clearly defined in a simple XML format.

In addition, include the line ranges for each parallel block and its threads.

****Output format:****

...

```
S1: [Description of step 1]
  <Parallel>[Parallel reason: ...]
    <Thread>
      S1.1: [Description of substep 1.1]
      S1.2: [Description of substep 1.2]
    </Thread> (range: Lxx-Lyy)
    <Thread>
      S1.3: [Description of substep 1.3]
    </Thread> (range: Lxx-Lyy)
  </Parallel> (range: Lxx-Lyy)
</Parallel>[Parallel reason: ...]
<Thread>
S2: [Description of step 2]
</Thread> (range: Lxx-Lyy)
<Thread>
S3: [Description of step 3]
</Thread> (range: Lxx-Lyy)
</Parallel> (range: Lxx-Lyy)
S4: [Description of step 4]
  S4.1: [Description of substep S4.1]
  <Parallel> [Parallel reason: ...]
    ...
  </Parallel> (range: Lxx-Lyy)
</Parallel> [Parallel reason: ...]
<Thread>
S5: [Description of step 5]
S6: [Description of step 6]
</Thread> (range: Lxx-Lyy)
...
</Parallel> (range: Lxx-Lyy)
...
...
```

****Guidelines:****

- Format the output **strictly** according to the output format specified above.
- The only XML tags allowed are <Parallel> and <Thread>.** A <Parallel> </Parallel> block should consist of multiple <Thread>'s. <Thread> can only exist inside <Parallel>.
- A <Parallel> block may form at either the main step level, or the substep level.
- Ensure **there are no <Parallel> tags nested inside one another.** Do not create structures like <Parallel> ... <Parallel> ... </Parallel> ... </Parallel>.
- **Max depth of substep nesting is 2.** Ensure there is no step number beyond Sx and Sx.y. For example, `Sx.y.z` is not allowed.
- **Do not add additional tags for sequential structures.** If an XML node's children are purely sequential, list them in order.
- **Tag parallel blocks.** Wrap parallelizable units in ``<Parallel>...</Parallel>``. Wrap each parallelizable unit (including sequential blocks and (sub)steps) in ``<Thread>...</Thread>``. List the sequential steps and substeps in order inside each ``<Thread>`` block.
- Provide the line range for each <Thread> block after the closing tag (`</Thread>` (range: Lxx-Lyy)).
- Provide the line range for each <Parallel> block after the closing tag (`</Parallel>` (range: Lxx-Lyy)).
- Keep the description for every step and substep as annotated in Task 1 and 2 (preserve as much as you can).
- **Do not create significantly imbalanced parallel blocks.** If most of an XML node's children are sequential and only a small number of them can run in parallel, you may leave the group untagged or only create a parallel block using the children that are genuinely parallel.
- Do not start your output with "````xml`". Indentation (i.e., spaces before Sx.y) indicates substeps, not XML structures; do not add tags other than <Parallel> and <Thread> around them.
- **Group parallelizable sets.** You may bundle several independent units into a single ``<Parallel>`` block when they share no dependencies.

Task 6: Consistency verification across contexts. Task 6 revisits the complete context (the original transcript plus all prior task outputs) to double-check that every instruction was satisfied. By reusing the same line numbering and dependency notes, it validates that no steps were skipped, that parallel reasoning is justified, and that references respect the allowed-versus-forbidden context policy introduced earlier in the prompt.

Task 6 Prompt for Step 1 Processing in ThreadWeaver

Task 6: Double-check Every Parallel Block and Refine the Structured Outline
Please check and revise the structured outline for parallel execution you generated in Task 5 by ensuring the following:

- **Coverage & Contiguity**: Make sure the line range (Lxx-Lyy) of each thread exactly spans from the first included (sub)step to the last. Make sure the line range (Lxx-Lyy) of each parallelizable block exactly spans from the first included thread to the last.

- ****Parallelism Validity****: Verify that there are no direct or indirect dependencies among the children of each parallel block. Specifically, confirm that each thread in the block does not depend on *any* of the threads before it in the same block. Add a note after the closing tag of each thread (`</Thread>`) indicating the result of this check.
 - If the check passes, add the note ``check passed`` and confirm the reason (`</Thread> [range: Lxx-Lyy, check passed: reason]``).
 - If the check fails (i.e. dependency is detected within the parallel block), refactor the problematic block into `Sequential(...)` or separate groups, and update the line ranges accordingly. If you cannot determine how to refactor, leave a note stating ``check failed`` along with the reason (`</Thread> [range: Lxx-Lyy, check failed: reason]``).
- ****Output Format****: Keep the outline format as specified in Task 5. Only the tags `<Parallel>` and `<Thread>` are allowed, and `<Thread>` tags can only appear inside `<Parallel>` blocks.

****Output format:****

...

```
S1: [Description of step 1]
  <Parallel>[Parallel reason: ...]
  <Thread>
    S1.1: [Description of substep 1.1]
    S1.2: [Description of substep 1.2]
  </Thread> (range: Lxx-Lyy, check passed: first thread)
  <Thread>
    S1.3: [Description of substep 1.3]
  </Thread> (range: Lxx-Lyy, check passed: S1.3 has no dependencies on S1.1 and
  S1.2)
  </Parallel> (range: Lxx-Lyy)
</Parallel>[Parallel reason: ...]
<Thread>
S2: [Description of step 2]
</Thread> (range: Lxx-Lyy, check passed: first thread)
<Thread>
S3: [Description of step 3]
</Thread> (range: Lxx-Lyy, check failed: S3 depends on S2)
</Parallel> (range: Lxx-Lyy)
S4: [Description of step 4]
  S4.1: [Description of substep S4.1]
  <Parallel> [Parallel reason: ...]
  ...
  </Parallel> (range: Lxx-Lyy)
</Parallel> [Parallel reason: ...]
<Thread>
S5: [Description of step 5]
S6: [Description of step 6]
</Thread> (range: Lxx-Lyy, check passed: first thread)
...
</Parallel> (range: Lxx-Lyy)
```

...

Step 2 The second step is a deterministic restructuring pass that converts the free-form sequences emitted above into canonical parallel markup. It sweeps over every annotated trajectory, infers the boundaries of each reasoning branch, and wraps the contiguous spans inside explicit thread containers that sit beneath the appropriate parallel block. During this pass the tool enforces ordering constraints, ensures that adjacent steps remain contiguous after threading, and drops in the dependency notes that certify why two segments may or may not execute simultaneously. Crucially, the script preserves the inline `Lxx` prefixes that Step 1 injected into every reasoning line, so the downstream prompts can continue to reference exactly the same line-number ranges even after the text is reorganized into `<Thread>` blocks. Because the logic consists solely of programmatic rewrites plus structural validation, no additional prompting is necessary.

Step 3 Next, each candidate trajectory undergoes a linguistic compliance review that compares the text under inspection against immediate predecessors and lateral sections. The reviewer model receives a slice of “allowed” context plus a sibling “forbidden” context, scans the target passage for implicit carry-overs, and proposes fine-grained edits to excise any leakage from the forbidden branch. The output takes the form of minimal substring replacements in JSON, which allows automated application while keeping every other token untouched. Running this step across all paths eliminates hidden cross-references, making each branch self-contained before outline regeneration.

Prompt for Step 3 Processing in ThreadWeaver

In this task, your goal is to check the English language of a given text segment against two other text segments from the same article:

- * The "allowed context" segment comes from the immediate preceding section of the text segment under review. The text segment under review may reference content from this previous section.
- * The "forbidden context" segment is from a section **parallel** to the text segment under review. The text segment under review must not reference any content from this parallel section.

Specifically, you will be provided with the following:

1. Allowed Context - contains information the text under review may reference.
2. Forbidden Context - contains information the text under review must not reference.
3. Text Under Review - the text segment to be checked.

Your task is to decide whether the text under review refers to any information in the Forbidden Context.

If it does, output a JSON array of substring replacements that remove or rewrite the referencing language in the text under review, following the output format specified below.

If it does not, output an empty array: `[]``.

Output the raw JSON only. Do not enclose the output in ````json...````.

Output format:

```

[
  {
    "src": "<the substring to be removed or rewritten (exact match, properly
    escaped)>",
    "dest": "<the replacement text which removes the reference to the Forbidden
    Context (use \"\" to delete)>"
  },
  ...
]

```

****Guidelines:****

- * Replace the minimal substrings needed to eliminate the reference to the Forbidden Context while keeping the text under review grammatical.
- * Escape quotes and newlines in JSON (use ``\"`` and ``\n``).
- * A proposed replacement will affect all occurrences of the "src" field in the text under review. If you do not intend to modify all the occurrences, make the substring in "src" more specific.
- * Do not introduce new information from the Forbidden Context in your replacements.
- * References to the Allowed Context are permitted. Do not remove any references to the Allowed Context.
- * Do not add explanations or code fences.
- * Do not output any text outside the JSON.

Special cases to watch for:

- * Remove or rewrite any occurrences of:
 - Anaphora/Deictics: Phrases like "as above," "as before," "same as earlier," "that result," "those values," which implicitly point to the Forbidden Context. Keep these when they refer to the Allowed Context.
 - Implicit carry-over: "unchanged from the previous section," "as we computed," when the "previous" information is from the Forbidden Context.
 - Shorthand references: Symbols, variable names, or equation labels defined only in the Forbidden Context (e.g., "that z," "the value from (3)").
 - Cross-references: "see previous," "see above," footnote markers, or inline citations that point to the Forbidden Context.
 - Paraphrased pointers: Rewrites that still rely on the Forbidden Context even without verbatim overlap.
- * Overlapping matches: When multiple candidate substrings overlap, prefer one replacement of the longest meaningful phrase rather than many small ones.
- * Case/whitespace variants: Handle differences in capitalization or spacing exactly by matching the literal substring present in the path.

- * Multiline fragments: If the referencing text to be replaced spans lines, include ``\n`` in `src` and `dest` as needed.
- * Grammar preservation: If removing a piece of referencing text leaves dangling punctuation or grammar issues, include the surrounding words/punctuation in `"src"` to keep the sentence clean.
- * Nothing salvageable: If an entire clause or sentence is a forbidden reference (e.g., "Which is the same as before."), delete it by setting `"dest": ""`.

Allowed Context:

...

[5 lines before the `<Parallel>` block]

Forbidden Context:

[First 5 lines at the beginning of the prior `<Thread>` block]

...

[Last 5 lines at the end of the prior `<Thread>` block]

Text Under Review:

[Full text of the `<Thread>` block to check]

Step 4 Once the sanitized paths are in place, a fresh model call rebuilds the planning scaffolds that will ultimately supervise downstream learners. Using only the contextual window that precedes each parallel block, the model drafts concise yet actionable outlines for every branch, restating the tactical moves a mathematician would plan before executing the proof. Each outline mirrors the ordering of the paths, isolates the reasoning intent without revealing conclusions, and keeps the different branches independent so later steps can treat them as interchangeable schedules. Because the outlines are derived strictly from the provided context, they serve as high-quality teacher plans for training planning modules or evaluating interpretability.

Prompt for Step 4 Processing in ThreadWeaver

In this task, your goal is to generate outlines for some mathematical reasoning procedures. Specifically, you will be given

1. A coherent math reasoning trace segment tagged with parallel structures (``<Parallel> ... <Thread> ... </Thread> ... </Parallel>``) that indicate reasoning steps which can be executed in parallel.
2. A group of placeholders placed at the beginning of the segment

...

`<Outlines>`
`<Outline>1: [Outline to be generated for path 1 below]</Outline>`
`<Outline>2: [Outline to be generated for path 2 below]</Outline>`
 ...
`<Outline>X: [Outline to be generated for path X below]</Outline>`
`</Outlines>`
 ...

3. A segment of text immediately preceding the target reasoning trace segment (``Context:``) to serve as the context for deriving the outline.

Using only the provided context (``Context:``), generate the outline for each `<Thread>` in the `<Parallel>` segment. Place the generated outline in the markers "[Outline to be generated for path X below]".

****Output format:****

...

`<Outlines>`

`<Outline>1: [Outline to be generated for path 1 below]</Outline>`

`<Outline>2: [Outline to be generated for path 2 below]</Outline>`

...

`<Outline>X: [Outline to be generated for path X below]</Outline>`

`<Outlines>`

...

****Guidelines:****

- * Imagine that a mathematician is thinking and using the outlines to structure and guide their thinking process in each corresponding path (``<Outline>X:...`

- `<Thread>X:...`).

- * Frame the outlines so the mathematician could logically plan them based solely on the context, without access to the content in the parallel segment or any additional information.

- * Make the outline directly and easily derivable from the context - neither too abstract nor too specific - and strictly aligned with the order of the paths.

- * Each outline should capture the gists and logical structure of its path without excessive details.

- * Thinking in each path is done independently. Ensure that the outlines remain distinct and do not depend on one another.

- * Do not reveal exact calculations, numerical results, or the final conclusion from its path in the outline. Instead, indicate the steps/methods/directions that lead to them.

- * Do not introduce a substantial amount of reasoning in the outlines.

- * Avoid phrases like "conclude that ...", "rule out that ...", or anything that cannot be directly derived from the context without significant reasoning. Do not frame the outline as if the conclusion is already known.

- * A reader should be able to write the path from the outline plus the context alone with some reasoning effort, without additional information.

- * Keep the outline concise and clear: one sentence for short content; at most two sentences for longer content.

- * Output plain text within each outline block: no bullet points, numbering, headings, or extra commentary.

- * Do not start your output with "`````xml`". Start your output with `<Outlines>` and end your output with `</Outlines>`. The only allowed tags in a `<Outlines>` block are `<Outline>` tags.

Step 5 The final step is a comprehensive filtering pass that operates on the model-generated artifacts. It traverses every outline file, enforces formatting contracts, checks that parallel

tags are balanced, and discards trajectories that still violate structural constraints after the preceding corrections, ensuring that only high-quality, well-structured demonstrations enter the supervised learning pipeline.

F.7 Discussion: Reward Hacking

A potential concern in RL with parallel rollout is *reward hacking*, where the model artificially inflates the acceleration reward by generating numerous redundant branches. In our experiments, we did not see clear evidence of this behavior. Our model benefits from strong reasoning priors established during supervised fine-tuning on high-quality, parallel-annotated trajectories, which guide it towards meaningful decomposition rather than excessive branching. Moreover, our RL rollout framework enforces a strict 40k total token limit per trajectory, which also discourages excessive branching (e.g., enumerating thousands of trivial cases). The acceleration ratio reward itself is intentionally conservative, by capping the reward and assigning a small scaling factor to it, so that correctness dominates the overall learning signal.

One solution for this type of reward hacking, which may happen in future search or application, is to apply a length penalty on the token latency. While seemingly more effective than acceleration ratio, it introduces normalization challenges, as the magnitude of the token counts is not naturally comparable to correctness rewards, and poorly normalized penalties can degrade performance by over-penalizing legitimate long reasoning traces. In contrast, the acceleration ratio is normalized relative to each trajectory’s structure (i.e., comparing the token latency with total number of tokens) and bounded, maintaining stability and fairness across tasks. For future extensions, more complicated length-based penalties may be applied to the token latency to better account for the potential reward hacking, but in our experiments, the clipped acceleration reward proved sufficient to keep acceleration high without compromising reasoning quality.

F.8 Future Directions

Topology-aware Parallelized Reasoning

The policy used in ThreadWeaver is *topology-agnostic*: it decides how many <Thread>s to spawn based on problem structure, not on available accelerators or interconnect characteristics. This limits scalability and efficiency. In contrast to CPU multithreading, where runtimes expose core counts and schedulers adapt thread pools, the model currently has no visibility into the number of GPUs, memory limits, or link bandwidth/latency. As a result, it may under-utilize large clusters or over-subscribe small ones, incurring overheads without reducing the critical path latency.

A natural extension is to condition the policy on a *hardware profile* at inference time, e.g., a concise header with (GPUs, communication bandwidth and latency, topology map). The

orchestrator (§8.3) can pass this profile via lightweight control tokens along with optional SLOs. A *topology-aware* policy would then adapt its fork–join plan, number of <Thread>s, granularity of subproblems, and join cadence, to the exposed resources, much like manually tuned tensor/pipeline parallelism, but learned and input-conditional.

Training can mirror this via *domain randomization* over hardware profiles and SLOs, coupled with a latency- and cost-aware variant of P-GRPO reward function. The acceleration bonus (§8.5) can be normalized by an estimate of the topology’s maximum attainable concurrency, while penalties account for scheduling overheads and stragglers. Ultimately, this points to a resource-conditioned planner that maps problem DAGs to device graphs, automatically discovering the best parallelization strategy for the hardware at hand.

Parallelizing Interactions Beyond Reasoning

Our framework can be interpreted as a multi-agent system, where the model autonomously determines when to branch its reasoning into multiple concurrent threads. Each thread functions as an independent agent that explores a subproblem and later synchronizes with others through structured aggregation. This implicit coordination mechanism allows the model to decompose reasoning tasks dynamically, but the same principle could extend far beyond text-based reasoning.

An exciting future direction is to generalize parallelization from internal reasoning to interactive decision-making in an environment. In such a setting, multiple agents could act concurrently within a shared environment, each pursuing complementary objectives or subtasks, towards one goal. For instance, in software engineering applications, distinct agents might analyze separate components of a complex system, refactor interconnected modules, and verify consistency through communication protocols. Similar frameworks could emerge in domains such as scientific discovery, robotics, or large-scale planning, where agents must balance autonomy with coordination.

Enabling such forms of parallelized interaction introduces new challenges in communication and synchronization. Simple message passing may be insufficient when agents share mutable contexts or overlapping goals. Mechanisms such as hierarchical coordination, role-based delegation, or dynamic resource locking could provide more robust foundations for scalable multi-agent reasoning. Exploring how these interaction protocols can be learned end-to-end rather than manually specified remains an open and promising direction for future research.