UNIVERSITY OF CALIFORNIA SAN DIEGO

# Reasoning as Search: Bridging Symbolic and Neural Approaches to Artificial Intelligence

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Computer Science

by

Lucas Dionisopoulos

Committee in charge:

Professor Prithviraj Ammanabrolu, Chair
Professor Taylor Berg-Kirkpatrick
Professor Loris D'Antoni

2026

The Thesis of Lucas Dionisopoulos is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

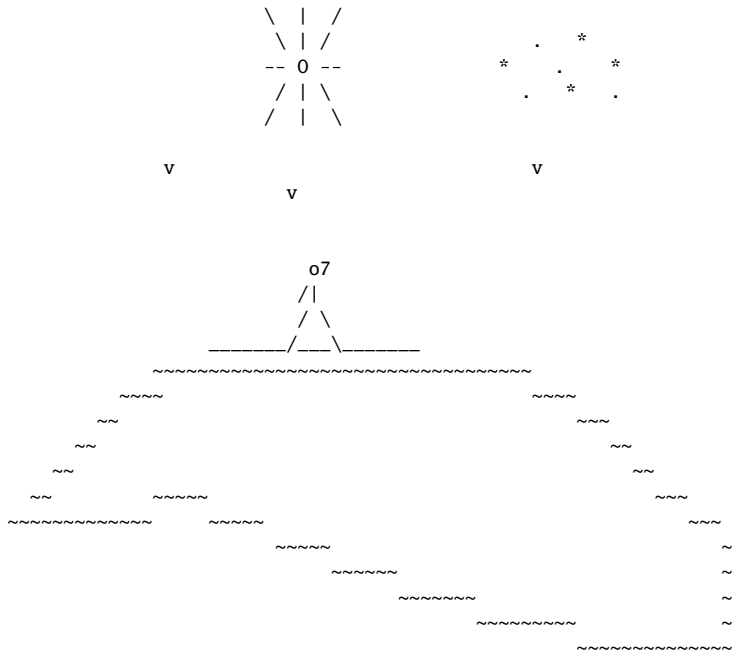
University of California San Diego

2026

EPIGRAPH

You stay classy San Diego.

*Ron Burgundy*



*Claude Opus 4.7*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This thesis is a product of much support and collaboration over the last few years – from my origins in self-study through my formal time in San Diego.

First, I am forever grateful for the unconditional support, patience, and love of my parents. I am thankful for the many friends who took interest and curiosity in my endeavors.

For the daft scholars of *Patrick's Rock* and the gumby crushers of *Pimpin' and Crimpin' (V2-V8)* – academic energy requires occasional recharging and y'all are among the finest.

For the mentorship of Prof. Raj Ammanabrolu and Prof. Loris D'Antoni – I learned much from you both and am incredibly thankful for your patience, guidance, resources, and feedback throughout research.

Chapter 3 is a revised version of work completed with Pawan Jayakumar and Vicki Li during *CSE 291C: Program Synthesis* taught by Prof. Loris D'Antoni. The revisions more heavily focus on the contributions of the thesis author.

Chapter 5, in full, is adapted from the material accepted for publication in **How Reasoning Evolves from Post-Training Data: An Empirical Study Using Chess**, Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Dionisopoulos, Lucas; Majamaki, Nicklas; Ammanabrolu, Prithviraj. The thesis author was the primary investigator and author of this paper.

## VITA

| | |
|---|---|
| 2021 | Bachelor of Science in Business Administration, Finance<br>Washington University in St. Louis |
| 2021–2023 | Investment Banking Analyst<br>Financial Technology Partners |
| 2024–2026 | Master of Science, Computer Science<br>University of California San Diego |

## PUBLICATIONS

"How Reasoning Evolves from Post-Training Data: An Empirical Study Using Chess." Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

ABSTRACT OF THE THESIS

Reasoning as Search: Bridging Symbolic and Neural Approaches to Artificial Intelligence

by

Lucas Dionisopoulos

Master of Science in Computer Science

University of California San Diego, 2026

Professor Prithviraj Ammanabrolu, Chair

There have been several proposed methods for artificial intelligence systems. This thesis focuses on two approaches: *library learning* (neurosymbolic) and *reasoning-based language models*. We begin by proposing a dual-system for artificial intelligence – where the system requires both *knowledge* and *composition* – before introducing our two key settings that we frame through this dual-system. First, the symbolic domain is formally defined with program synthesis before highlighting an ambitious strategy in library learning. This is followed by original work studying a novel neurosymbolic approach to the ARC-AGI benchmark. Then, we introduce the neural domain – where we frame reasoning as search and follow with work that studies how post-training data influences downstream reasoning behavior in chess. This thesis concludes by solidifying a unified framework for artificial intelligence that is based on identified similarities between the principled, theoretical system proposed in library learning and its practical counterpart in reasoning models. From this unified view, we see that language model breakthroughs have slowly converged to the library learning meta-architecture – and we propose future directions of study based on the remaining discrepancies.

# Chapter 1

# Introduction

What does the term *artificial intelligence* even mean?

At the core of the innumerable announcements, think-pieces, dinner conversations, and ambitious predictions that have come to characterize the middle 2020s, we have a term that lacks a clear definition.

The intent of this work is not to argue for a precise definition – the phrase is elusive for valid reasons. Rather, we motivate the central thesis through a principled framing of artificial intelligence from which the desirable attributes of a useful *general intelligence system* naturally emerge. We offer the following framing: artificial general intelligence (AGI) is a dual-system that combines *knowledge* with the capacity for *composition*. This thesis opts for the phrasing "artificial general intelligence" (or "general intelligence") to avoid possible confusion caused by the historically relaxed application of the term "artificial intelligence".

Both *knowledge* and *composition* will be more formally defined in due time, but loosely, *knowledge* consists of a corpus of learned patterns and *composition* represents the act of applying these patterns on-the-fly to solve novel tasks. This *knowledge* may be available before the task or discovered during task execution, and *composition*, as a term, will hereafter be used interchangeably with *reasoning*.

This thesis outlines two views of reasoning: a symbolic lens built upon techniques in program synthesis [29] and a neural lens based on deep learning [47] that characterizes many of the latest developments in language-based reasoning models [66]. We discuss two original works representative of the two views. First, a neurosymbolic programming [8] approach to solving reasoning tasks on the Abstraction and Reasoning Corpus (ARC-AGI 1) [10], and second, an empirical study in chess on how training data influences downstream language model reasoning [16].

The remainder of this section formally introduces the dual-system framing of artificial general intelligence and distinguishes two views of reasoning between *Symbolic* and *Neural*. We finish by outlining

the full thesis, where we present a unified view of reasoning as search that bridges symbolic and neural techniques.

## 1.1 A Dual-System View of General Intelligence

To motivate the utility of our proposed dual-system perspective, consider common systems that satisfy only *knowledge* or *composition*. Search engines (e.g., Google's early PageRank algorithm [6] or modern variants that incorporate machine learning in the page ranking process) are marvels in *knowledge* recall but unsatisfying examples of general intelligence. On the other hand, methodical algorithms that efficiently solve well-structured problems – from map direction apps to chess engines – are also unsatisfying examples of general intelligence; despite superhuman *composition*, they are narrow in scope.

The dual-system view requires both in tandem. Beyond knowledge regurgitation, we require adaptable composition dependent on the desired task. Beyond rigid algorithms, we require a general flexibility to solve novel problems. The merging of knowledge and composition, theory and application, *what you know* and *how you use it*. Decades of computer science have optimized both separately; the new challenge is how to dynamically combine them at scale.

### 1.1.1 How to Distinguish Knowledge and Composition?

Drawing a clear distinction between *knowledge* and *reasoning* (or *composition*) can be difficult. Consider the following riddle from J.R.R. Tolkien's *The Hobbit* [100]:

*This thing all things devours;*
*Birds, beasts, trees, flowers;*
*Gnaws iron, bites steel;*
*Grinds hard stones to meal;*
*Slays king, ruins town,*
*And beats mountain down.*

The answer is: *time*. Now, if you're unseasoned with these types of riddles, this may be challenging to reason through. However, if you read this after the earlier riddles shared between Bilbo and Gollum, you may notice patterns – previous riddles include intangible concepts like *darkness*. In this case, prior knowledge may aid with reasoning.

As another example, let us consider mathematical proofs. Common patterns and techniques can emerge – for example, proving convergence bounds in convex optimization often relies on tools such as convexity inequalities, Jensen's Inequality, Cauchy-Schwarz, or smoothness bounds [63]. If a mathematician

was entirely unaware of these common tools, they may be able to derive and apply them to yield the same proof. However, existing knowledge of common techniques in convergence proofs may significantly aid when approaching novel tasks – they may manipulate the problem so they can exploit these tools.

These two examples show how existing knowledge can both be factual and assist with on-the-fly composition (*reasoning*) depending on the task. There exists a blurry balance between information recall and application – both are essential, but a piece of raw knowledge may be used as a step in your reasoning process (e.g., *Jensen's Inequality is ...*) while also biasing your reasoning in an efficient direction (e.g., *as I'm now trying to define a lower-bound, I should consider if Jensen's Inequality could come in handy*). An exclusive distinction does not clearly exist.

Combine this with the idea that artificial general intelligence may be capable of mid-task knowledge discovery. This is, in fact, how its biological predecessors commonly handle novel situations. In this case, there is not even a clear distinction between *pre-task* and *mid-task* as the knowledge base and reasoning strategies can be updated live. We seek a more robust distinction.

### 1.1.2 Reasoning as Search

To obtain a clearer distinction, let us consider a task from the view of a Markov Decision Process (MDP). An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \gamma)$, with states $\mathcal{S}$, actions $\mathcal{A}$, transition kernel $T(s' \mid s, a)$, reward function $R(s, a)$, and discount factor $\gamma$. At each step, the agent chooses $a_t$ in state $s_t$, receives reward $R(s_t, a_t)$, and transitions according to $T(\cdot \mid s_t, a_t)$. In a simplified task setting, we ignore $\gamma$ and view an answer as satisfying or unsatisfying, yielding a sparse binary reward upon termination ($\in \{0, 1\}$); all other rewards are simply 0. The task is what defines the starting state: $s_0$. A loop then begins – for any state $s$ an action is chosen ($a \in \mathcal{A}$) and the environment steps: $s' \sim T(s' \mid s, a)$.

In this formulation, *knowledge* shapes the agent's effective action set and parameterizes the policy $\pi_\theta(a \mid s)$. *Composition* is the act of stepping through the loop. If you consider the unraveled version of an MDP as a tree, an agent's knowledge defines both nodes *and* the traversal algorithm; composition consists of the traversal. This naturally frames *reasoning* as a search process – given a tree, reasoning is the explicit act of traversal. Finding an efficient traversal, therefore, becomes a search problem.

For theoretical rigor, it is worth noting that the classical MDP formulation defines $\mathcal{A}$ as fixed and environment-defined. We can accommodate a functionally dynamic action set by defining an enlarged action space and allowing the agent's effective action set to vary with its knowledge state. Initially unknown or

unavailable actions are assigned zero probability under the policy, and discovery corresponds to updating the agent's knowledge state such that these actions are assigned non-trivial probability.

### 1.1.3 Comparison to the Cognitive Dual-Process Perspective

It is worth briefly discussing the comparison between the aforementioned dual-system and a similar, albeit different, framing from cognitive psychology. Dual-process theories of cognition [20, 41] distinguish between two broad modes of thinking: System 1 thinking – characterized by a fast, automatic, and intuitive process – and System 2 thinking – a slower, methodical, and possibly algorithmic form (often associated with deliberate reasoning).

The distinction proposed in this thesis is more operational for artificial systems. While the cognitive view popularized by Daniel Kahneman [41] is intuitive and relatable, when one considers general intelligence systems – artificial or biological – the boundary between recall and reasoning can become blurred. Consider this example – when students prepare for exams or interviews (e.g., coding, consulting), there are often formulaic "frameworks" that are initially taught as a way to instill structured problem solving. During the exam or interview, some students simply regurgitate these frameworks – unable to adapt them to the new problem. When mastered, these frameworks offer structure for System 2 thinking; however, non-applied recall may be misdiagnosed as System 2 thinking when, in reality, it is more akin to System 1 thinking.

In our proposed dual-system, this act of regurgitation would be viewed as shallow knowledge recall; minimal task-specific composition occurs. Eventually, students learn to generalize these frameworks – often, rote memorization is a necessary first step when climbing a learning curve – but there is a period of knowledge internalization before skilled composition catches up.

Thus, while cognitive dual-process theories provide a useful analogy, the proposed framing is intended to be more operational for general intelligence systems. It separates the possession of useful knowledge from the process by which that knowledge is composed into solutions for novel tasks.

## 1.2 Symbolic vs. Neural

Given this framing of reasoning as search, it is now necessary to distinguish between two notable cases: *symbolic* and *neural* (Figure 1.1).

The symbolic setting involves a formally defined grammar with rules and semantics, and this includes (but is not limited to) code, mathematical expressions, and formal language. Symbolic search is often designed

to produce computer-executable outputs – motivating its rigorous formulation – though executability is not strictly required. In practice, we see this framing within program synthesis [29] as well as broader research in programming languages and formal methods.

We contrast the symbolic setting with the neural setting. In the neural setting, search occurs over continuous vectors – often referred to in machine learning as *latent space*. Despite this, neural representations can interface with discrete symbolic systems by decoding into a discrete space (e.g., in language models, token-space [59, 44] or more general decoders [99]). In language, this discretization can be framed as a conversion to the *symbolic* setting and may even allow neural search to incorporate code execution [83]. However, despite this discretization, language-modeling techniques almost exclusively search in neural-space; a transformer may output tokens, but the autoregressive input to the model is still a vectorized representation of that token [105].

The core distinction is in the domain where this search occurs – we separately expand on each of these views later in the thesis. Additionally, while we largely discuss the symbolic and neural settings through the lens of language, continuous vectors can also be converted into other modalities (e.g., images, audio)

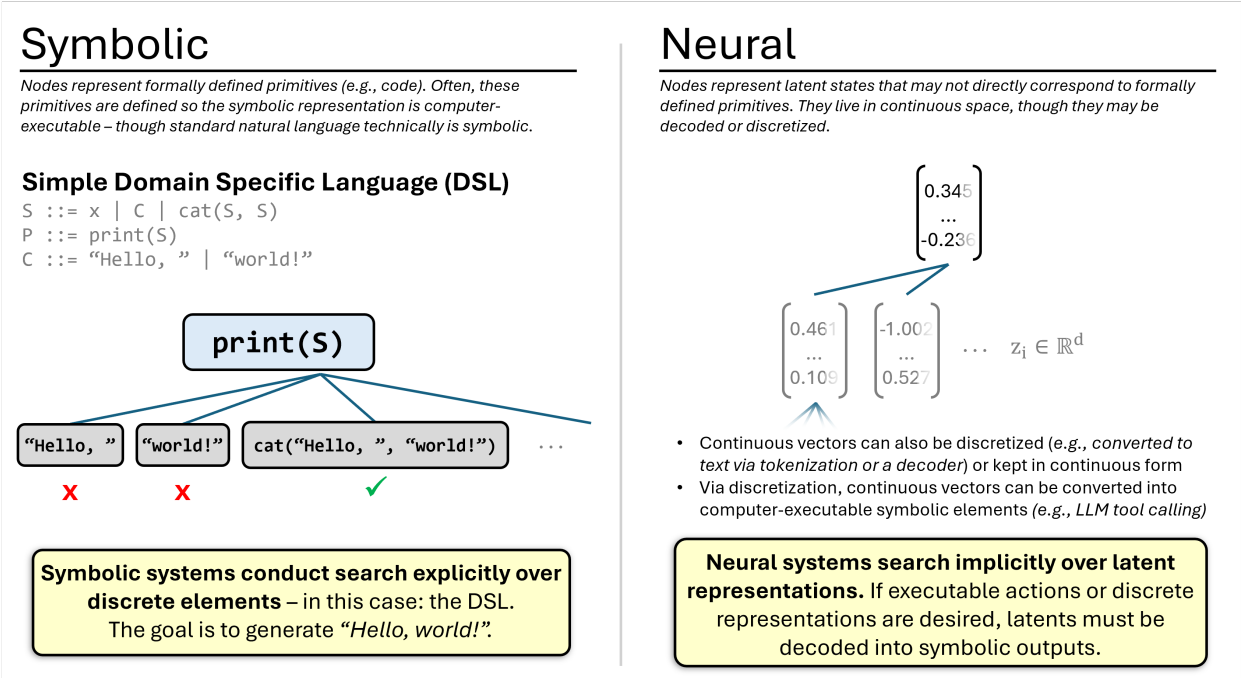



**Figure 1.1.** Comparison of symbolic and neural search. Symbolic search is often designed to be computer-executable and is used in program synthesis [29]. Neural search occurs in latent space but can be discretized through tokenization [44] or decoded into longer text [99]; if desired, neural search can be discretized into symbolic, machine-executable code as well [83].

– though this is effectively discretization into these other domains. We choose to focus on language and language-adjacent symbolic representations (e.g., code and math) as these are most relevant, studied, and practical.

## 1.3 Thesis Organization

The remainder of this thesis is structured as follows. Chapter 2 discusses reasoning in the symbolic setting. Following this, Chapter 3 highlights original work applying a neurosymbolic approach to ARC-AGI. Then, Chapter 4 briefly introduces reasoning in the neural setting before Chapter 5 – the main contribution – discusses an empirical study in chess on how data influences downstream reasoning in language models following supervised fine-tuning (SFT) and reinforcement learning (RL). Chapter 6 steps back and offers a unified meta-framework of artificial intelligence that we build up over the entire thesis before finishing with a short discussion on future directions.

# Chapter 2

# Symbolic Reasoning as Search

The symbolic setting – characterized by a formally defined grammar with rules and semantics – has been applied to many domains within computer science (programming languages, compiler design, static analysis, formal verification) and beyond (e.g., computational linguistics approaches to natural language). While symbolic formalism has largely been applied in computer-based settings, it is not limited to the computing domain.

This symbolic formalism has many benefits, although they do not come free. At the expense of requiring rigorous setup, you may algorithmically operate on these symbolic expressions. For example – in formal verification, you may be able to prove your code will achieve the desired result; in general programming, you may use techniques in static analysis to test for common pitfalls (e.g., type checking) without running your code. This symbolic formalization serves as a foundation for programming languages, where programmers must first learn the grammar and rules (i.e., *syntax*) as a prerequisite before participating in the language's symbolic formalism.

However, while strict formalization offers many benefits and has defined the field of programming languages, there are other settings where this technique has been less successful. Early methods in artificial intelligence took a symbolic approach but saw progress stall. For example, in natural language processing – where rule-based techniques sought to meticulously formalize grammar – this tight formalism experienced difficulties in application. Famously, IBM researcher Fred Jelinek quipped [37]:

*Whenever I fire a linguist our system performance improves.*

While intentionally hyperbolic, this highlights the double-edged sword of symbolic formalism: rigorous set-up with hand-engineered rules can become unruly as complexity and scale increase.

However, there are many useful applications that stem from this symbolic approach that will be

discussed shortly. For the remainder of this section, we focus attention on the field of *program synthesis* [29] before introducing a first-principles approach to artificial general intelligence in *library learning*.

## 2.1 Program Synthesis: An Overview

Program synthesis is the task of automatically finding a program in a given search space that meets a given specification [77]. Program synthesis has various practical applications, notably Excel Flash Fill [28], but also extends into compiler-oriented settings such as superoptimization [84] and machine learning compiler optimization [38].

A common framework for program synthesis [29] decomposes the problem into three components: 1) behavioral specifications – *what you want your program to do*, 2) structural specifications – *what defines your search space of programs*, and 3) search strategy – *the algorithm you use to search your space of programs*. We organize our overview using these three components.

### 2.1.1 Behavioral Specifications

Behavioral specifications can be provided in several forms, including input-output examples or a functional specification ($\psi : \mathcal{I} \rightarrow \mathcal{O}$ maps input $i \in \mathcal{I}$ to an output $o \in \mathcal{O}$) [77]. A functional specification, when accompanied by a defined input domain, can be used to arbitrarily generate input-output examples and is therefore more informative.

In the formal setting, satisfaction of an input-output constraint is binary: either they match or do not. However, as we introduce our unified view and extend to noisy domains that require approximation, this binary evaluation is relaxed and may be considered as *loss* or *reward* (with the goal to minimize or maximize, respectively).

### 2.1.2 Structural Specifications

Structural specifications define the search space. A common approach is to use a *grammar*, which specifies the primitive symbols and rules for constructing candidate programs.

Grammars come in many forms, including *context-free grammars (CFGs)* and *regular tree grammars (RTGs)*. The core difference is that a CFG defines rules over strings, whereas an RTG defines rules directly over the syntax trees of expressions. Grammars can also be *typed*: each symbol belongs to a category, and rules specify which types may be composed. This mirrors familiar type systems in programming languages,

though the types used in synthesis can be more abstract. We avoid a full formal definition for brevity and focus on RTGs, which we now describe.

A typed regular tree grammar consists of a tuple $(N, \Sigma, S, a, \delta)$ with the following elements:

- **Nonterminals** ($N$): Abstract categories of expressions that can still be expanded. In a typed grammar, each nonterminal has an associated type.
- **Terminals/operators** ($\Sigma$): The concrete symbols or functions used to build expressions. Each operator has an arity, specifying how many arguments it takes.
- **Start symbol** ($S$): The initial nonterminal from which programs are generated.
- **Type assignment** ($a$): A mapping that assigns types to both nonterminals and operators.
- **Productions** ($\delta$): Rules that describe how a nonterminal can be expanded into an operator.

We provide an example regular tree grammar in Figure 2.1. Despite this being a toy example, note that grammars can describe expressive programming languages (*if we visualized this for all of Python, it would be quite an unruly figure...*). The benefit of strict typing is that expansions can be restricted to type-compatible productions, pruning invalid expressions before search begins.

One may begin to notice how grammars and typing are useful in defining the search space of candidate programs. Additionally, more formal constraints, such as polymorphic refinement types [78], can further reduce the search space by ruling out invalid programs earlier, making synthesis faster.

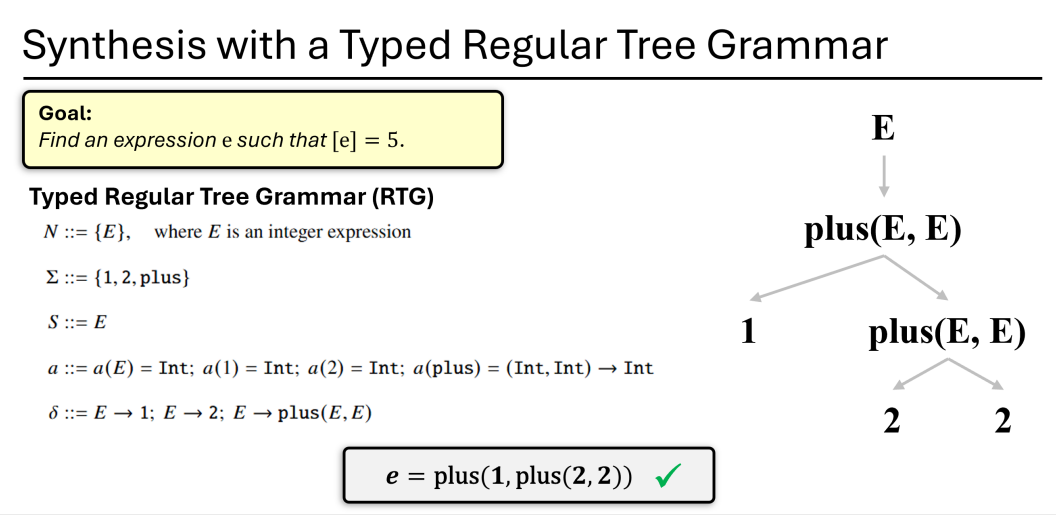



**Figure 2.1.** Example of a typed regular tree grammar in a toy domain. In order to find an expression $e$ such that $[e] = 5$ using the available terminals $\{1, 2\}$, the `plus` operator must be used twice.

### 2.1.3 Search Strategies

There are two foundational strategies for searching over programs defined by a grammar: *top-down* and *bottom-up* enumeration. For a bounded program size or depth, exhaustive enumeration is possible, but it quickly becomes intractable due to the combinatorial growth of the search space. For this reason, synthesis systems often add pruning, ranking, or biasing mechanisms to explore promising regions more effectively.

*Top-down enumeration* begins from the start symbol and repeatedly expands nonterminals according to the grammar productions. This constructs partial programs with holes, which are filled until a complete candidate program is produced. The search can be traversed using methods like depth-first and breadth-first – however, this process can explore many redundant branches. For example, the integer operator $\texttt{plus}(E, E)$ is symmetric: $\texttt{plus}(1, 2)$ and $\texttt{plus}(2, 1)$ are syntactically different but semantically equivalent. Many top-down enumeration techniques therefore identify such redundancies and prune equivalent or impossible branches.

*Bottom-up enumeration* instead builds programs from smaller expressions. The synthesizer maintains a cache of expressions, often grouped by type and size, and combines existing expressions using grammar productions to construct larger candidates. As the search grows, the cache becomes large; bottom-up systems commonly prune expressions that are semantically equivalent under the current specification to manage this. If one has input-output specifications, they may choose to only keep one representative that produces the same outputs for the given inputs. See Figure 2.2 for a comparison of top-down and bottom-up enumeration.

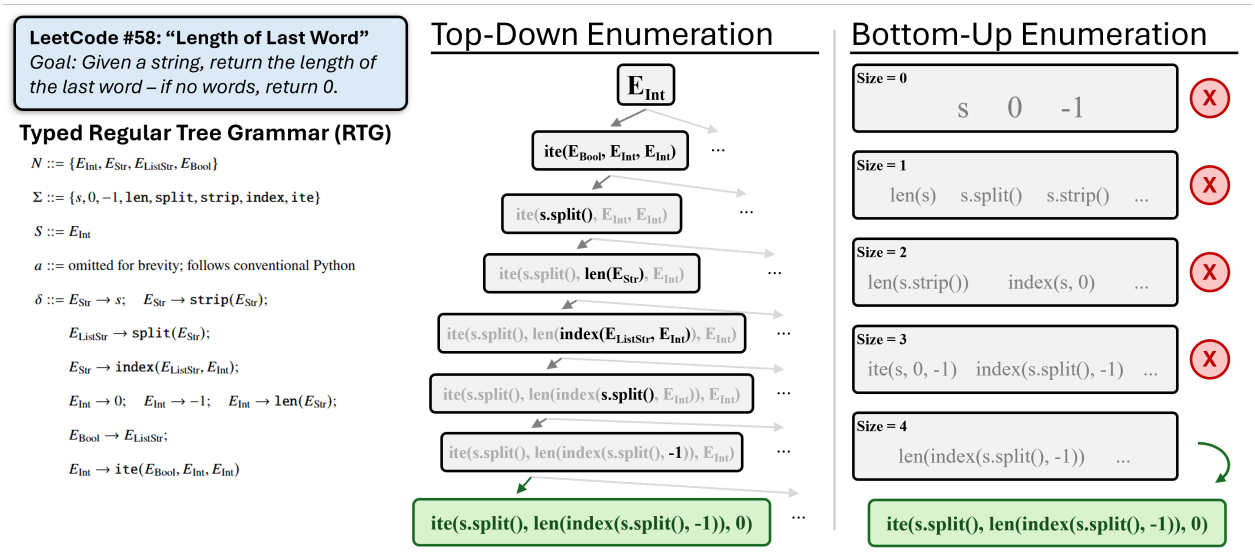



**Figure 2.2.** Comparison of top-down and bottom-up enumeration on a simple coding problem. Note, this follows Python syntax, although we use `ite()` as shorthand for an `if then else` statement.

The key issue with these basic strategies is that they are often limited to shallow search depths, while many interesting programs require deeper search. To search deeper, synthesis systems often use biased search algorithms that prioritize promising branches or promising sub-expressions. One notable example is *Euphony* [48], which introduced a *probabilistic higher-order grammar* that combines machine learning with A* search. More recently, grammar-constrained or grammar-aligned decoding for language models [76] can be viewed as a synthesis problem when paired with a formal grammar and verification against a specification – this effectively uses a language model to search the space of programs efficiently.

### 2.1.4 Realizability in Program Synthesis

It is worth briefly noting another consideration in program synthesis: *realizability*. In this context, realizability asks: *given the specifications, does a satisfying program actually exist?* In general, this question is undecidable due to the halting problem [102], but it remains a necessary consideration in any formalized symbolic search.

## 2.2 Neurosymbolic Programming and Library Learning

As previously discussed, various work in program synthesis has attempted to address key limitations using machine learning techniques [48]. Similarly, recent work in machine learning, specifically language modeling, has attempted to address core limitations using symbolic techniques – for example, tool usage [83] and constrained decoding [24, 76].

This general combination of neural and symbolic components may be referred to as *neurosymbolic programming*; as a term, it has been applied loosely and taken on many definitions – particularly among vocal skeptics of language model capabilities. Even among foundational defining works, the definition is evasive: Chaudhuri et al. [8] defines neurosymbolic programming as a generalization of program synthesis that uses both neural and symbolic components with symbolic composition. This work particularly frames neurosymbolic programming as an evolution of program synthesis – but, under their definition, a language model with tool calling ability is arguably neurosymbolic programming.

Within this chapter, we focus on neurosymbolic approaches that are more reminiscent of program synthesis. We offer no rigorous definition of the term either – trends show that these methods are only becoming more interconnected – but rather choose specific work we find intriguing for study.

### 2.2.1 Notable Work in Neurosymbolic Programming

A compelling aspect of neurosymbolic programming is that expressions can be succinct and explainable. This has been leveraged in symbolic regression: Grayeli et al. [26] combine language models and evolutionary algorithms to rediscover key physics equations from data. More broadly, neurosymbolic programming has been used as a tool for scientific discovery [93] given its potential for explainability.

*DreamCoder* [19] represents another ambitious application of neurosymbolic programming. Here, a learning loop cycles through biologically-inspired phases: 1) *Wake* – find programs, using components in a learned library of sub-expressions, that solve problems in a training set; 2) *Sleep (Dream)* – train a neural network to predict a program that solves a given problem, using both cached experience and synthetic fabrication for labeled data; 3) *Sleep (Abstraction)* – grow your learned library by storing larger sub-expressions that prove to be useful. This is, effectively, looped bottom-up enumeration that uses a neural network to expedite search and various methods to manage the sub-expression cache.

Other work has attempted to use more methods from deep learning, with Sehgal et al. [87] incorporating image embeddings with symbolic manipulation to solve visual tasks. Both this work and *DreamCoder* share many similarities with our neurosymbolic system for ARC-AGI that we discuss in Chapter 3.

### 2.2.2 Library Learning as a Path to Artificial General Intelligence

*Library learning* encapsulates several neurosymbolic techniques that extend bottom-up enumeration to solve increasingly complex problems by using an online bootstrapping loop – *DreamCoder* [19] is a notable example. The appeal of library learning is that complexity naturally builds over time. As new primitives are added (or discovered), they are appended to the library and can solve novel problems – this offers a compelling solution to the *continual learning* problem. Further, this system does not rely on supervised learning for knowledge discovery as is necessary in many deep learning systems. Rather, this system could, theoretically, push beyond known solutions to solve entirely novel problems – *extrapolation* rather than *interpolation*.

If we apply our dual-system view of artificial intelligence, we see that knowledge stems from three key components. Initially, knowledge is instilled through the choice of starting primitives. These may begin as elementary but grow in complexity through cached composition – but new primitives may also be discovered online. Knowledge is also embodied in the search algorithm: *DreamCoder* [19] uses a learned neural network to guide search, for example. Lastly, knowledge may be used to steer cache management. While this may be achieved through a simple heuristic, more complex systems have proven to be useful (e.g., compression using

language models [26]). Figure 2.3 provides a high-level overview of library learning techniques. Note that we do not formally define *library learning*, but offer a simplified view that generalizes existing work [19, 26].

This offers a quite intriguing system for intelligence. It mirrors biology, where DNA seeds a set of primitives and an initial learning algorithm that determines search (reasoning) and memory management; this system also offers a possible explanation for how complex biological intelligence can evolve from a small amount of genetic information. Further, it proposes a clean solution to continual learning and expanding complexity – something modern language models struggle to do efficiently – and it may offer a sample efficient learning algorithm, which is an active limitation in modern deep learning.

For these reasons, library learning is a compelling *idealistic* approach to artificial general intelligence. However, the key word is *idealistic* – as will be discussed, it faces many challenges that have prevented general capability or useful scale.

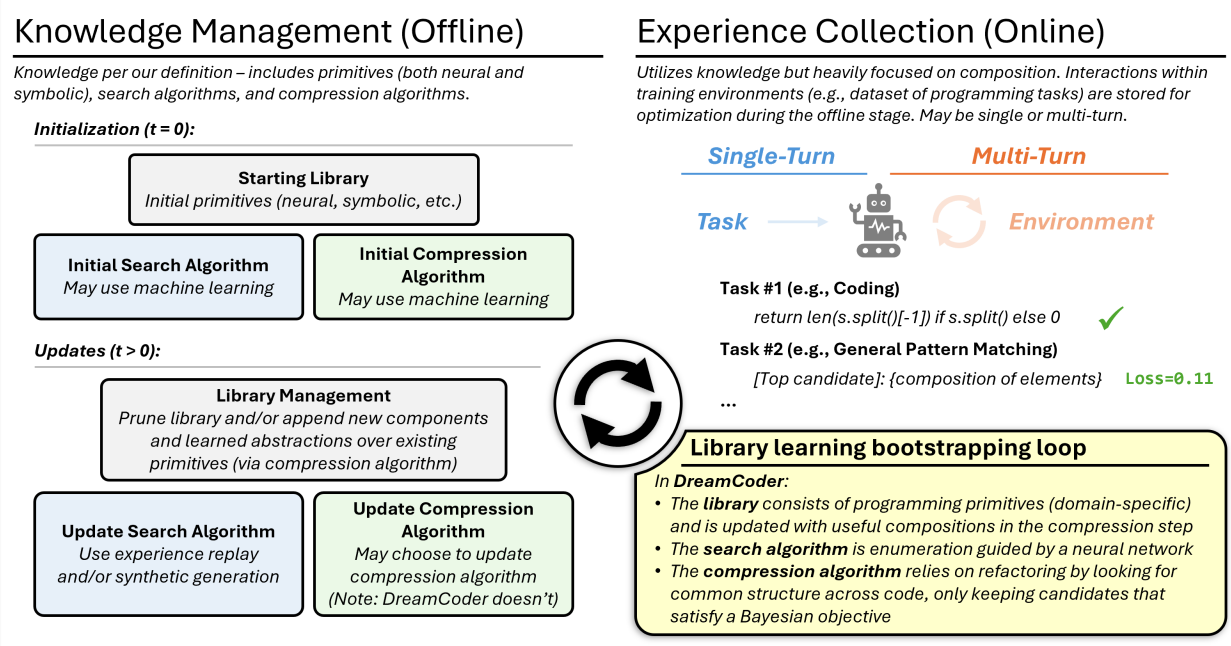



**Figure 2.3.** Overview of a generalized library learning loop. We reference *DreamCoder* [19] to ground the generalized framework to a real example, but the intent is to provide a high-level summary of the technique. As we later discuss, this mirrors recent techniques in language modeling.

## 2.3 ARC-AGI: A Testbed for Library Learning

François Chollet proposed the Abstraction and Reasoning Corpus (ARC) [10] as a direct response to what he viewed as fundamental limitations of deep learning: sample inefficiency, inability for continual learning, difficulty with multi-step problems, and lacking precision in arithmetic tasks (e.g., counting). ARC-AGI was created as an evaluation to precisely test against these things, and it proved to be evasive for language models with progress lagging behind human performance for years. However, this changed with the inception of the *reasoning model era* – which notably kicked off with OpenAI o1 [66].

Despite reasoning models achieving strong performance on ARC-AGI (OpenAI GPT-5.5 reported 95% on ARC-AGI-1 and 85% on ARC-AGI-2), ARC-AGI remains a useful testbed for library learning given the aforementioned design motivations. It also offers a symbolic setting that is too complex for many conventional symbolic techniques. For these reasons, we chose ARC-AGI-1 as the domain for our proposed neurosymbolic programming method, which we now discuss in Chapter 3.

# Chapter 3

# Neurosymbolic Programming for the ARC-AGI Benchmark


## Abstract

The ARC-AGI benchmark challenges machine learning systems to solve visual reasoning tasks that require generalization beyond memorized patterns or large training datasets. This chapter introduces a novel hybrid approach combining neural and symbolic methods to address the unique challenges of ARC. We attempt to use a vision transformer to encode ARC problems into dense embeddings, guiding A* search over transformations defined in a domain-specific language. To represent ARC tasks more effectively, we propose an object-centric approach, where input and output grids are decomposed into *ARC objects* with structured attributes. Our proposed neural approach has limitations that prevent us from completing a full implementation within our project timeline – however, we adapt our object-centric strategy with a graph-based method to solve four public evaluation cases from the ARC dataset. This demonstrates an early proof-of-concept for an object-centric approach and paves the way for further experimentation to enable full neural-guided search that we discuss in our conclusion.


## 3.1 Introduction

The Abstraction and Reasoning Corpus (ARC) benchmark [10] was created to measure machine intelligence. It is composed of visual reasoning questions that include a small number of input-output examples (2 to 7) followed by a single test input – the goal is to correctly generate the test output. The initial benchmark – ARC-AGI-1 – is the benchmark we focus on in this chapter, although since this preliminary work, there have been two additional benchmarks released. ARC-AGI-1 has a goal of 85% at which the challenge is deemed to be "solved" – this is cited as a baseline for human performance. While many deep learning-based

methods have excelled across other benchmarks, ARC-AGI remained elusive for several years following its release in 2019. Deep learning techniques particularly struggle with ARC-AGI for several reasons:

- **Averse to memorization**: Due to problem novelty, it is infeasible to attempt to memorize solutions. One may try to overfit to the relatively small training set, but the test set has explicitly novel compositions.
- **Required sample efficiency**: ARC requires the system to learn from a small set of examples (2 to 7) for each problem – deep learning is notoriously sample inefficient.
- **Necessary precision**: The answer must be entirely correct – requiring full precision. Generative approaches may get close, but an exact match is required.

We provide an example problem in Figure 3.1. These challenges highlight the difference between excelling at one trained task versus composing existing priors to solve unseen problems. *The former is skill; the latter is intelligence*.

It is worth noting that at the time of initial writing in December 2024, frontier language models heavily struggled with ARC-AGI-1. Figure 3.2 highlights the evolution of three OpenAI models – from GPT-4.1 [68] and o3 [71] in April 2025 to GPT-5.5 [73] in April 2026. This evolution showcases a clear jump in capability through reasoning: GPT-4.1 is a non-reasoning model, o3 is a reasoning model, and GPT-5.5 is reasoning-native – a massive jump in capability from reasoning research over the span of one year. It wasn't until the beginning of the *reasoning era* that language models began to reliably succeed in the ARC domain.

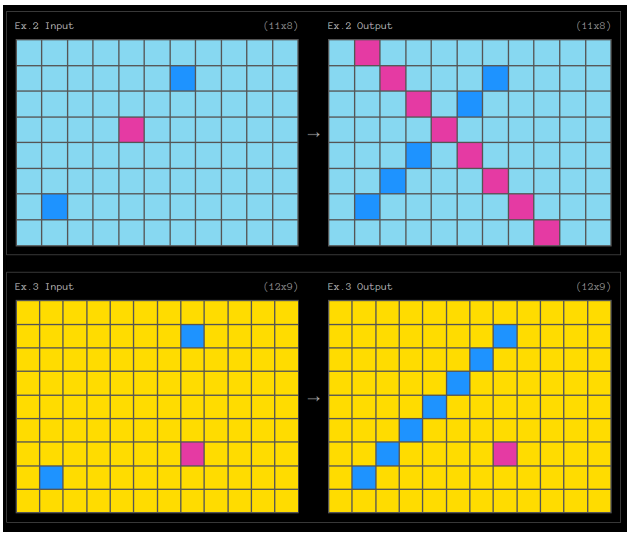



**Figure 3.1.** Example problem from ARC-AGI-1. The transformation requires connecting blue dots with a blue line and drawing an orthogonal pink line if there is a pink dot in the way.

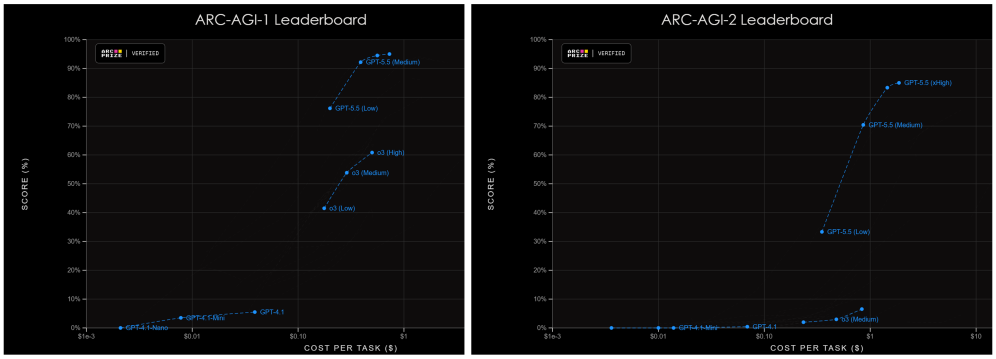



**Figure 3.2.** Leaderboards for ARC-AGI-1 and 2 [3] highlighting various OpenAI models. (*bottom left*) GPT-4.1 was released in April 2025 as a non-reasoning model; (*middle*) o3 was also released in April 2025 as OpenAI's flagship reasoning model. (*Top right*) GPT-5.5, released one year later in April 2026, is reasoning-native. This highlights the efficacy of reasoning and remarkable progress in reasoning capabilities over a single year.

Inspired by *AlphaZero* [90] and *DreamCoder* [19], we seek to combine deep learning methods with explicit search, specifically program synthesis. Our approach combines a custom domain-specific language (DSL) that operates on parsed *ARC objects*. We encode these parsed objects as dense embedding vectors using a custom vision transformer [17] based on iBOT [119] that we train from scratch. We analyze these vectors and find them to have meaningful embeddings; however, we are unable to learn a policy to drive A* search over our DSL from these embeddings. We provide an overview of our proposed neurosymbolic programming approach to ARC-AGI in Figure 3.3.

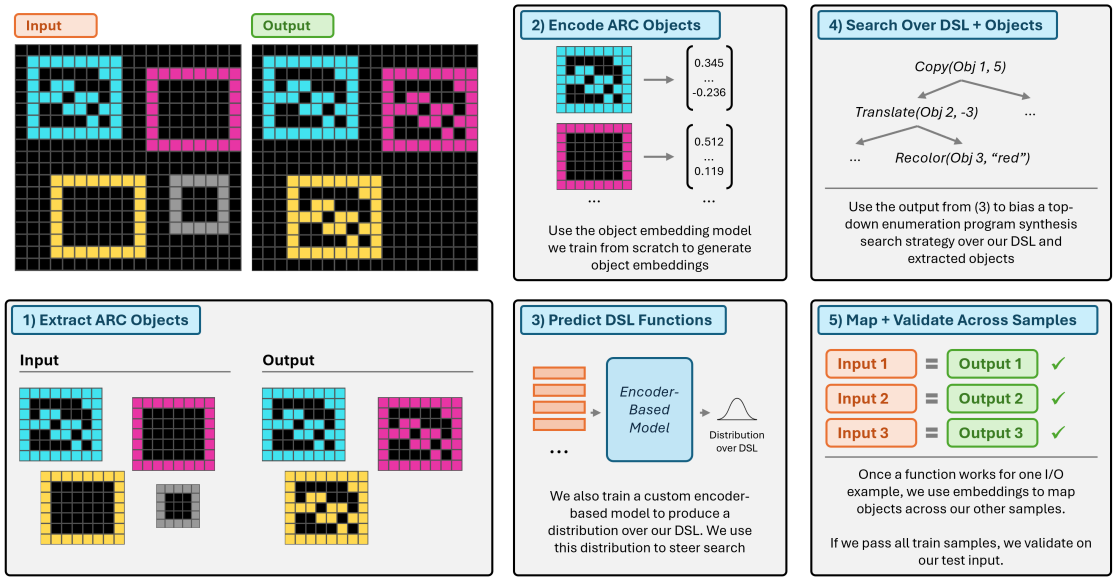



**Figure 3.3.** Overview of our proposed neurosymbolic approach to ARC-AGI. In practice, we ran into difficulty on `(3)` as our embeddings were not rich enough to allow for a non-trivial distribution to be learned. As a result, we used a symbolic graph-based strategy that accomplished `(4)` and `(5)` in unison and allowed us to solve multiple problems.

Despite shortcomings, we are still able to solve 16/400 train cases and 4/400 public test cases using a simple graph-based symbolic technique applied on ARC objects with our DSL. While promising, our implementation has several limitations due to the difficulty of learning a rich embedding model – we conclude with a discussion of these limitations and propose future directions that may address these issues.

## 3.2 Related Work

Past work has attempted to use a DSL to describe the structure of ARC solutions [33, 101]. The difficulty in creating a DSL is that, by choosing which functionality to include, you impose an inductive bias on what transformations are likely under the DSL. For this reason, a common strategy for DSL-based approaches in ARC is to keep adding functions that solve more of the training set. However, while this enables shallow depth solutions, the novelty of the private evaluation set often causes these techniques to fall short – *icecuber* [101] is a notable method that saw a large discrepancy between the public and private evaluation sets.

At the time of writing, leading solutions relied on large language models (LLMs) using two common techniques. First, some approaches sought to fine-tune the LLM on each specific ARC problem (called *test-time training*) by leveraging synthetic data generation – this was seen as an effective alternative to chain-of-thought sampling [106]. The second common LLM approach required using a language model to generate programs, often in Python [27]. The search would end when a valid solution is generated or a compute limit is reached – often yielding strong results at significant costs.

While ARC problems are ultimately represented as JSON files, they are better presented to LLMs in other ways. Humans view these problems as visual transformations and invoke ideas such as translation, symmetry, or color. Other approaches [51] propose adding object positional embeddings to embed the concept of *objectness* into neural representations of ARC problems. Another technique adopted a similar object-centric method, although it did not use a vision model [21].

Ultimately, none of these works have solved the core challenge of ARC which is automating skill acquisition. Each requires expert domain knowledge to be incorporated into the solution, producing bespoke solutions that are less likely to transfer to more open-ended or challenging domains such as automated theorem proving.

## 3.3 Domain-Specific Language (DSL) and Segmentation

### 3.3.1 DSL

Our DSL has nine operations: *color*, *recolor*, *rotate*, *flip*, *delete*, *translate*, *single copy*, *copy translate*, and *draw line*. While being relatively concise, these operations were designed to be general and effective when applied at the object level. Additionally, a small DSL creates a more tractable program synthesis problem, which further motivated concision.

### 3.3.2 ARC Objects

Our approach treats ARC problems not as a pair of input-output grids, but instead as a pair of input-output ARC object sets. Our custom ARC object data structure consists of a pixel mask (comprising the object), the position of the object (top-left corner of the mask's bounding box), and the height and width of the object. An object-based method provides several advantages:

- **Simplicity**: It is easier to describe transformations on whole objects rather than specific pixels on a grid.
- **Modularity**: All DSL operations are applied on an ARC object – although some require additional arguments – and return a transformed ARC object. This allows us to chain operations and ensure our program remains well-typed.
- **Feature Abstraction**: An object-centric approach makes it easy to extract meaningful features such as color, shape, and position.

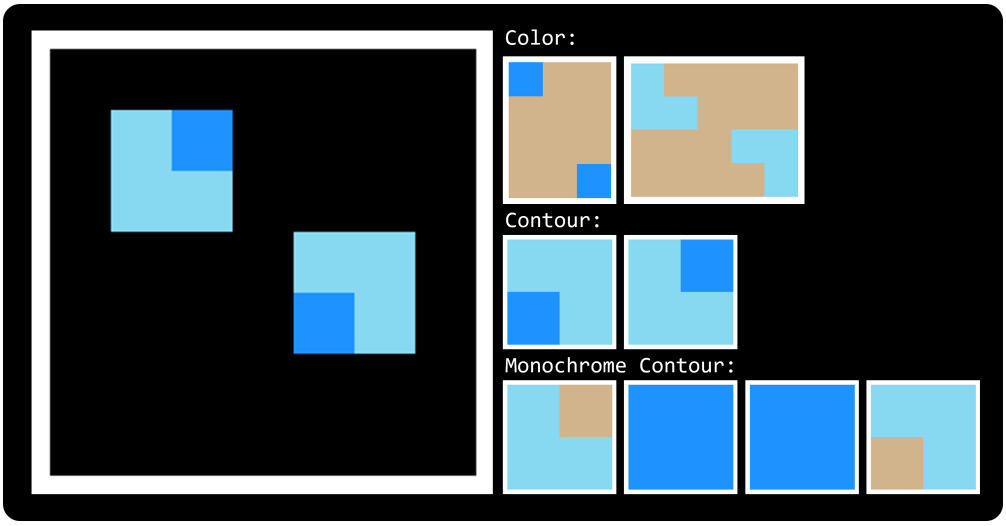



**Figure 3.4.** Three different segmentation algorithms for parsing ARC objects.

### 3.3.3 Object Segmentation

We defined three methods to segment grids into objects, using the OpenCV library [5] to detect contour lines. These methods can be seen in Figure 3.4.

1. **Color**: Treat all pixels of the same color in a grid as one object. The pixels do not have to be connected. Thus, an object can contain several disconnected parts.
2. **Contour-Scale**: First convert the ARC grid into gray scale, scaling pixel values with respect to ascending pixel frequency. This generalizes the concept of a "background color" and extracts distinct objects (potentially multi-colored) that are separated in the grid.
3. **Monochrome Contour**: Combines the constraints of color and contour. All pixels in an object must be connected and of the same color.

In practice we execute all three in parallel as each is effective in various settings. However, we later discuss a proposed method that would allow segmentation to be incorporated directly into our neural architecture using computer vision techniques.

## 3.4 Encoding

In order to generate a useful heuristic to guide A* search, we chose an image encoding method to convert full ARC problems and ARC Objects into a rich embedding space. We had hoped a performant open-source embedding model such as ResNet [31] or MobileNet [82] would provide useful embeddings out-of-the-box – but common practice in training image classification models involves using data augmentations to impose robustness against color, rotation, size, and position within the image [4]. This is not ideal in our setting as we need our embeddings to be sensitive to these characteristics. Additionally, these models are trained on common image classification tasks that vary wildly from ARC problems, and the large internal dimensionality of these models imposes computational constraints for downstream tasks.

For these reasons, we chose to train a task-specific ARC object embedding model that converts an ARC object into a semantically meaningful vector.

### 3.4.1 Architecture

We adapted a vision transformer [17] with the intent to extract embeddings via the *class* token. For our objective, we trained using a modified self-distillation objective inspired by iBOT [119] that used a student

and teacher network with a small masking probability ($p = 0.1$). Inputs were provided as a batch of problems (each image padded to $32 \times 32$), and the model was tasked with minimizing cross-entropy loss comparing the *class* tokens from the student network to the teacher network.

We used an embedding dimension of 64, 2-D sinusoidal positional encodings, 8 attention heads, transformer depth of 10, and an MLP hidden dimension of 128 – resulting in approximately 400K parameters. The model was trained over 4 hours on an NVIDIA A6000 GPU.

### 3.4.2 Synthetic Data

To facilitate learning relevant representations, we created a pipeline to generate synthetic data using our DSL. Many state-of-the-art ARC solvers use synthetic data generators (e.g., RE-ARC [34]) for the ability to extend the limited training set. Using a synthetic data generator not only allows us to produce significantly more training data than is provided, but it also allows us to bias our model toward learning representations that distinguish our specific DSL transformations.

Examples of our synthetic data generator and scaling behavior are provided in Appendix A.1. We exhibited log-linear loss improvement when using our synthetic data. Notably, we were able to train on 20K training steps (approximately 200K samples) – which would have required around 100 epochs if using the provided ARC training samples. This would have posed a much higher risk of over-fitting compared to our synthetic generation regime, where we only needed 20 epochs to generate this number of samples.

### 3.4.3 Cosine Similarity Evaluation

As an intermediate evaluation for our representation model, we conducted qualitative tests by visualizing our attention mechanism and through cosine similarity queries. Several methods were tried to quantify this performance, but we believed them to be too noisy.

Figure 3.5 visualizes both our attention mechanism and cosine similarity evaluation. These results showed that the image encoder was learning useful representations – being able to discern elements such as sparsity, color, and positioning.

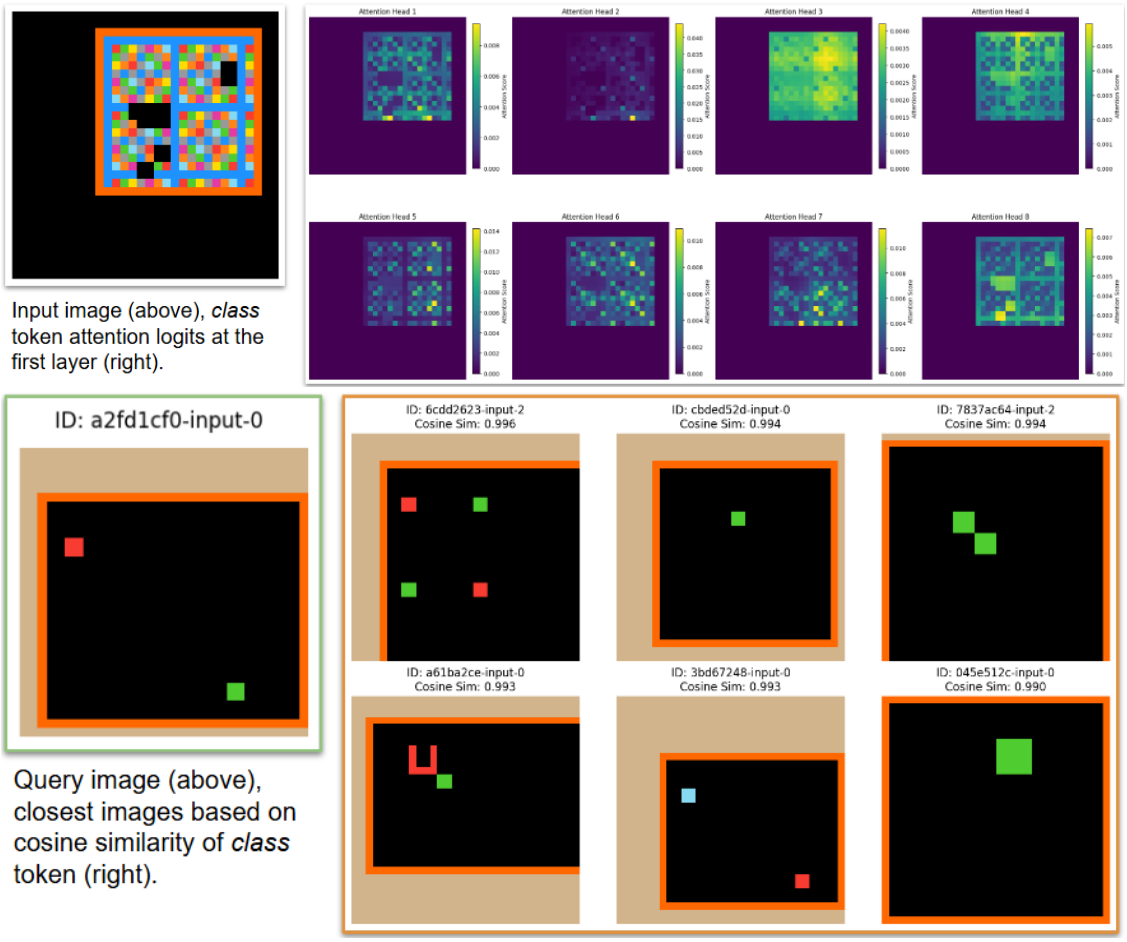



**Figure 3.5.** *Top*: Plot of attention logits for the *class* token in the first transformer layer across the 8 attention heads. *Bottom*: The result of cosine similarity search over 1,000 randomly sampled ARC grids. In both, we see that our encoding model is learning useful representations.

### 3.4.4 Training a Probabilistic Grammar for A* Search

Given the combinatorial challenge of program synthesis, we needed to bias our search. To do so, we chose to follow *Euphony* [48] – which trained a neural network to allocate probability mass to each grammar element before feeding this into A* search. To accomplish this, we use an encoder-based model with two task-specific tokens. Appendix A.2 includes an overview of the proposed architecture.

However, we found that the model was incapable of learning non-trivial distribution masses based on our image embeddings. This could be due to several reasons, but we believe it is likely caused by a lacking richness in our embeddings. As a result – and due to time constraints – we chose to move to a purely symbolic solver based on our DSL and object-centric framing.

## 3.5 Object Graph

### 3.5.1 Motivation

Several of our DSL functions require additional arguments beyond the object itself. For example, *translate* requires the end coordinates of the translation. While enumerating possible arguments, it is important that: 1) they generalize to other test examples, and 2) they do not use information specific to the output grid for a particular example as this information is not available for the test example. Effectively, finding these arguments is its own synthesis problem. We solve this by making an assumption that these arguments are some function of the properties and relations between input and output objects; this narrows our search space to allow for more tractable enumeration.

### 3.5.2 Construction

Since ARC problems decompose into a set of input and output objects, we can create a graph where the nodes are objects and edges are properties (e.g., shape or color). We observe that many ARC problems have input objects that are modified but can still be recognized in the output grid based on some invariant property. For example in Figure 3.6, we observe that the shape and color of each object does not change. We can use this information to create a mapping between the set of input and output objects. This mapping can also be constructed between input objects across examples. Using these mappings, we can determine what properties are important, and further refine our search to figure out the exact expression. For example in Figure 3.6 we observe that only the y-coordinate of each object changes, and that they ultimately match the y-coordinate of the blue object.

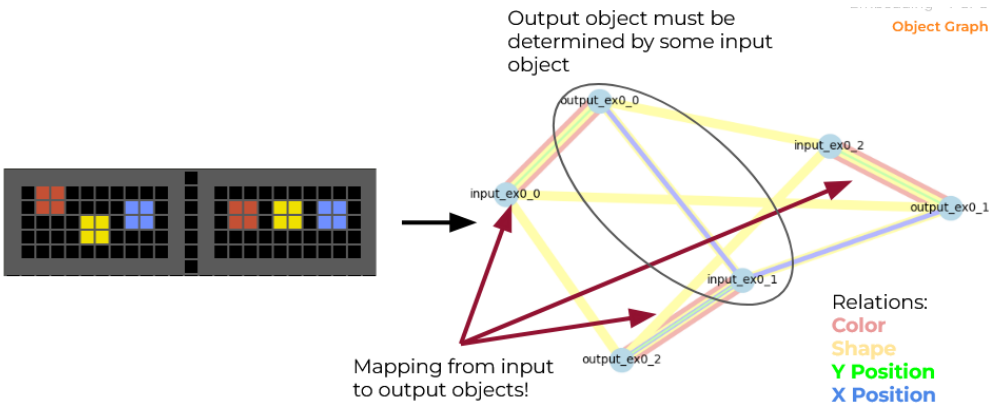



**Figure 3.6.** Simple ARC translation problem to visualize our graph-based approach. The system identifies that the red and yellow objects must be translated to match the y-coordinate of the blue object.

## 3.6 Results

While our initially proposed A* search method was hindered, we managed to solve 16/400 train problems and 4/400 evaluation problems in the ARC Prize dataset using a simplified version of the object representation graph. The simplified version relieves most of the constraints on the strict input-output object mapping, and it instead only focuses on the generalized feature comparison between all input objects and all output objects. Our method does not utilize a "training" step – the difference in performance between the two sets relies on the fact that the training set is typically easier (often requiring one or two transformations) whereas the evaluation set requires longer chains of operations and more complex feature extraction. Examples of solved problems can be found in Appendix A.3.

## 3.7 Limitations and Further Directions

### 3.7.1 Symbolic Approaches are Brittle

What started with an ambitious, first-principles approach to ARC-AGI quickly became complex and unruly. A core limitation of symbolic techniques is that they require significant manual design. Beyond the engineering challenges that quickly pile up, we found that as we designed and tested the object-centric DSL, more edge-cases arose that evaded our structure. This is an inherent limitation of using a manually defined domain-specific language.

Outside of incredibly well-scoped environments, we believe symbolic approaches will always struggle with this brittleness. In these cases, we are not convinced there is a proven solution either; nearly all neurosymbolic and library learning methods use a fixed set of primitives. Ideally, the system should be able to learn new primitives – *foundational conceptual abstractions* – however, this is an open challenge. If possible, this could pave the way to artificial intelligence that acquires foundational abstractions that model underlying structure; these abstractions can then be composed to solve increasingly complex problems [60, 61].

However, we face the reality of *The Bitter Lesson* [94]. Rigorously-defined idealistic attempts to build artificial intelligence have been continually plagued with brittleness. In light of getting *Bitter Lesson-pilled*, we now turn to offer an alternative neurosymbolic approach that may scale.

### 3.7.2 An End-to-End Alternative Architecture

There are several limitations to our proposed architecture and strategy – many were useful learning experiences. The first is the value of an *end-to-end* system.

Our neural approach failed when we attempted to chain two neural models together. We took the embeddings from our object encoder – which had some interesting properties – but they failed to allow us to model a probability distribution over our DSL. An end-to-end system could have avoided this issue. Rather, had we found a way to train a neural network directly on our desired task (generate a probability distribution over our DSL) – under the correct architecture, data, and objective, our universal model could have learned this.

Additionally, had we used techniques from object segmentation in computer vision – notably *Mask2Former* [9], which predicts both a mask *and* an affiliated class over pixels – we could have baked the object segmentation step into our neural architecture as well. This, in theory, could have predicted both a function in our DSL *and* the object to apply it to in unison. As an end-to-end approach, it would have alleviated many of the brittle components in our proposed architecture.

This alternative architecture follows a *MuZero* approach [85] – remove as many manually defined inductive biases as possible and allow the model to learn itself. From this architecture, several challenges still remain. It would have likely required a starting set of DSL functions and, for each function, it is still non-trivial to determine additional arguments. However, creativity may alleviate more brittle components, leading to an elegant, concise artificial intelligence system that could be of interest for further study.

### 3.7.3 Concluding Remarks

As we have seen from the latest reasoning models [73] – a generalizable solution already exists. While some may feel this is inelegant, the fact is that it works and scales. However, to those skeptics, we look forward to presenting the unified view in Chapter 6 – we shall see that modern language models have shocking similarities to the idealistic library learning setting.

## Acknowledgements

Chapter 3 is a revised version of work completed in late 2024 with Pawan Jayakumar and Vicki Li during *CSE 291C - Program Synthesis* taught by Prof. Loris D'Antoni. The revisions more heavily focus on the contributions and opinions of the thesis author.

# Chapter 4

# Language-Based Reasoning as Search

Language-based reasoning has been the foundational driver behind many of the latest developments in language models [66, 25, 73, 14]. We defer to Chapter 5 – specifically Section 5.2 – for an overview of language-based reasoning and the use of reinforcement learning (RL) to instill this behavior. Rather, we focus this introductory chapter on a select set of general topics.

First, we frame language through a symbolic-grammar lens to set the stage for our unified framework in Chapter 6. We then draw parallels between our dual-system view of artificial intelligence – specifically, how language-based reasoning is search. While there are many other topics worthy of inclusion, this section remains brief for concision; we focus on the aspects of language modeling that are relevant to our unified framework.

## 4.1 Language as a Symbolic Grammar

We first take a formal view of language modeling – comparing it to our previously discussed symbolic setting. Language can be viewed through a grammatical lens, where language models commonly operate over a vocabulary (e.g., characters, subwords, or words). The most common architecture is a transformer [105] with a fixed vocabulary of tokens (often subwords) [44] – for example, the Qwen2.5 tokenizer consists of $151,643$ regular tokens [79].

Similar to our previously discussed symbolic setting, our grammar has rules and semantics. However, natural language – while having rules – is flexible; grammatical correctness is insufficient and not strictly necessary for useful speech. We do not attempt to codify or formalize these rules further, but instead focus on the fact that language models commonly operate on discrete units. Note that all references to language models going forward will assume the canonical transformer-based architecture with tokenization.

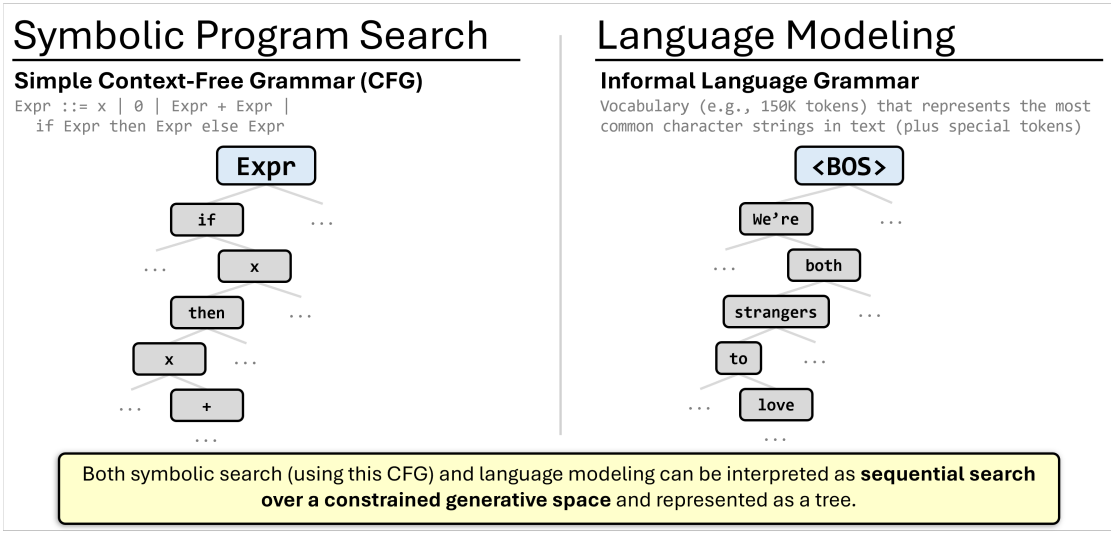



**Figure 4.1.** We see clear similarities between language modeling and symbolic search when we view language through a grammatical lens. Both can be represented in tree format – we will focus on this in our discussion of reasoning as search.

In this setting, language models produce a distribution over the vocabulary of tokens – from here, stochastic sampling is often used to choose the exact next token. We see that this is quite similar to our programmatic setting; at a high level, both can be viewed as a policy guiding search over discrete symbolic choices, although the programmatic setting imposes stricter structure. Through this lens, *Euphony* [48] shares many similarities with a language model – Figure 4.1 highlights this similarity. Additionally, both methods can be represented as a tree – we will focus on this perspective as we frame language-based reasoning as search.

## 4.2 Reasoning as Search

One of the earliest, prominent forms of reasoning in language models came through *chain-of-thought* [106] prompting, where models were tasked to think through complex problems in a step-by-step manner. Prior to this, language models would often output the answer directly. If the problem was complex with multiple steps, these would still occur in a single forward pass – therefore, any "reasoning" was internalized in the model's layers.

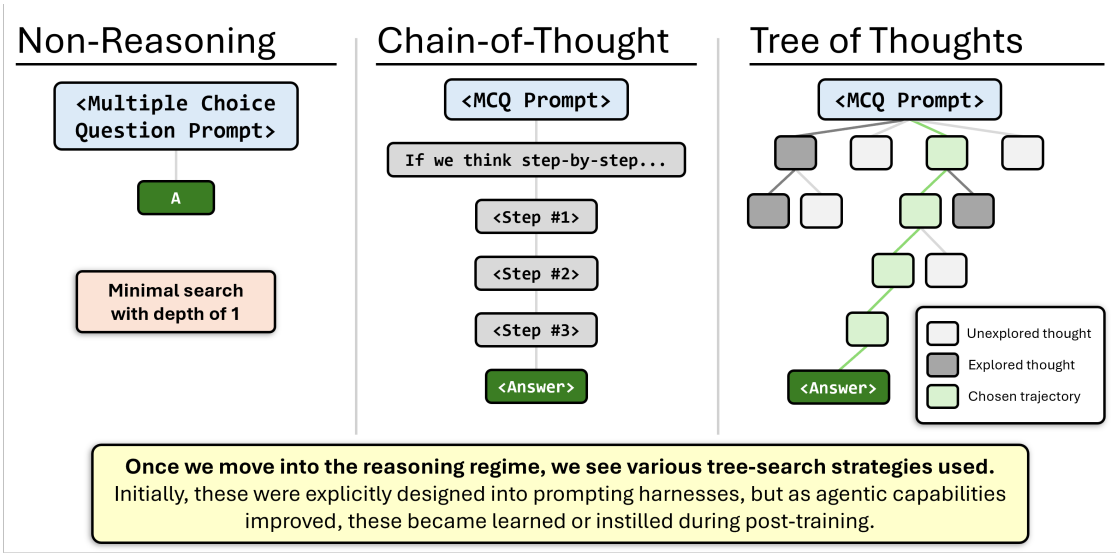



**Figure 4.2.** As language-based reasoning matured from direct responses to *chain-of-thought* [106], we see deeper search occur. Naturally, more advanced search strategies began to be implemented (e.g., *Tree of Thoughts* [110]). However, as agentic capabilities have improved, these explicit search strategies became instilled in models through post-training rather than driven by prompting harnesses.

Even with *chain-of-thought* [106], text generation often followed linear thought patterns – without explicit search strategies such as enumeration or backtracking. To address this, early techniques built elaborate prompting harnesses to directly elicit search strategies – *Tree of Thoughts* [110] is a notable example. Their harness allowed the model to consider different thought directions, expand on a thought, and evaluate thoughts – all achieved through a prompting loop that made many calls to a language model.

However, many of these early strategies were focused on prompting the model to generate reasoning traces. With the rise of *reasoning models* – where models are post-trained using techniques like reinforcement learning (RL) to generate longer thought traces naturally – many behaviors resembling explicit search strategies began to emerge from the models themselves. Rather than relying on elaborate prompting harnesses, post-training encouraged models to decide when to revise, backtrack, or pursue alternative strategies directly.

Notably, DeepSeek-R1 [15] highlighted their *aha moment* – during thinking, the model determined a particular path was the correct way to solve the problem. Additionally, reasoning models often displayed backtracking behavior – possibly catching a prior error or deciding to pursue another strategy. As capabilities further improved, we have seen reasoning extend into incredibly long settings: the Kimi K2.5 report [98] lists a limit of 100 tool calls per subagent for their *Agent Swarm*. These incredibly long tasks often invoke many tree search strategies – except now, they can emerge from the model rather than from a hand-designed prompting harness.

## 4.3 Chess as a Testbed for Reasoning

Studying language-based reasoning is difficult for several reasons. First, many open-source reasoning studies begin around the 7B-parameter scale, where models are often capable enough to show nontrivial reasoning behavior while remaining trainable for academic experiments. Because of this size requirement and the computational costs imposed by reinforcement learning, experiments are expensive.

Beyond the costs – both time and money – reasoning behavior is largely driven by the data a model is trained on. Many modern open-source models that serve as research subjects have been heavily trained on math and coding – this is due to positive transfer from these domains and large amounts of high-quality data. While the improved model quality is welcomed, this makes testing reasoning in these domains challenging – any observed reasoning behavior may be primarily due to the pre-training process.

For these reasons, an ideal domain is one that the base models struggle in and haven't had significant, targeted pre-training – this way, you can measure how your interventions influence reasoning behavior. Further, it is helpful to have a verifiable domain for reward computation during RL.

This is why we choose chess as our testbed to study reasoning. Specifically, Chapter 5 highlights our work (*ICML 2026*) that studies how post-training data influences downstream reasoning behavior in chess.

# Chapter 5

# How Reasoning Evolves from Post-Training Data: An Empirical Study Using Chess


## Abstract

We study how reasoning evolves in a language model – from supervised fine-tuning (SFT) to reinforcement learning (RL) – by analyzing how a set of theoretically-inspired datasets influences language model performance in chess. We find that fine-tuning a model to directly predict the best move leads to effective RL and the strongest downstream performance – however, the RL stage elicits *unfaithful* reasoning (reasoning inconsistent with the chosen move). Alternatively, training on multi-move trajectories yields comparable downstream performance with faithful reasoning and more stable RL. We analyze multiple qualitative and quantitative measures and highlight how these evolve from SFT through RL; we find several SFT-checkpoint metrics – spanning evaluation performance, hallucination rates, and reasoning quality – to be predictive of post-RL model performance. Finally, we ground our results with an experiment measuring *chess information density* in our custom datasets. We release models as well as training data, evaluations, and code that allowed us to surpass leading open-source reasoning models in chess with a 7B-parameter model. Code, models, and data are accessible online (github.com/lucasdino/lang-chess).


## 5.1 Introduction

*What is required to train a language model to reason through RL?* Several ingredients appear critical – a strong base model and a compatible domain are sensible starting points. *But what is a strong base model? And once you have a domain, how do you train it to reason effectively?*

We seek to address these motivating questions by training a language model to reason in a verifiable, sequential decision process. Specifically, we choose chess as our focus because of several convenient elements:

intrinsic difficulty for LLMs, established theory, favorable structure (episodic MDP), large datasets, and an efficient oracle (chess engines) for verifiable rewards and high-quality synthetic data generation. As a result, we can measure, in a controlled setting, how different training datasets influence our language model through SFT and how RL further evolves reasoning.

Language-based reasoning [66] has emerged as a technique to advance language model capabilities, although research remains largely confined to domains such as math and coding. Reasoning, often characterized as extending a model's "chain-of-thought" behavior [106] using methods such as RL, benefits from these domains being clearly verifiable: the math is correct or the code passes all tests. This verifiable nature – combined with downstream applicability and skill transfer – has stimulated much research in these domains. As a result, reasoning models have achieved profound results across related benchmarks [69, 2, 25], long-duration tasks [46, 58], and earned an International Math Olympiad gold medal [14, 72].

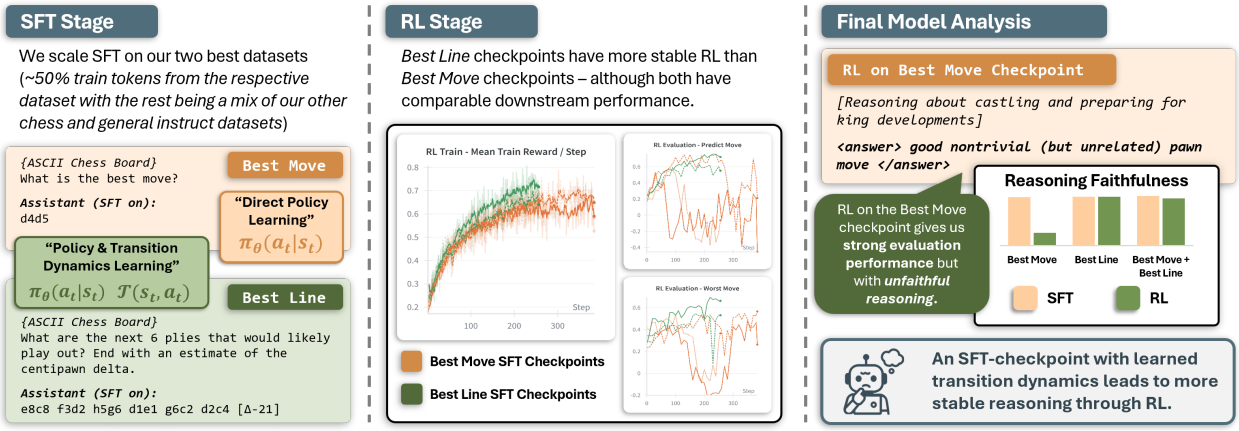



**Figure 5.1.** Following initial data inclusion experiments, we scaled SFT on our two best-performing datasets. Both resulted in comparably strong final evaluation performance, but training on optimal move trajectories (*Best Line*) led to more stable RL and faithful reasoning compared to training on single best moves (*Best Move*).

Although recent work explores reasoning in more subjective domains [107] through techniques such as "LLM-as-a-judge" [117], we focus here on those that are verifiable. The core verifiable domains – math and coding – benefit from years of continued research that has established a corpus of high-quality training data which produces strong out-of-the-box performance. While we benefit from stronger general base models, working on these explored domains would pose a challenge in isolating the influence of training interventions.

Chess thus emerges as attractive for studying reasoning. Language models have historically struggled with chess [1, 18] and even state-of-the-art reasoning models still falter – often responding with illegal moves or simple blunders as seen in the Kaggle AI Chess Exhibition [40]. Models underperform partly because

chess data, while abundant, is rarely emphasized in pretraining; further, the game's combinatorial structure makes generalization challenging. While this poses a difficulty, access to superhuman verification (in chess engines) provides an efficient method for verifiable rewards and synthetic data generation. All these aspects make chess a compelling testbed for reasoning.

Motivated by this setting, we train a 7B-parameter model on custom datasets using both SFT and RL to achieve performance surpassing gpt-oss-120b [70] on several benchmarks. We focus our study on the following:

- *Q1*: How do different datasets (*e.g., programmatically generated, various synthetic generations*) impact downstream performance after SFT and RL?
- *Q2*: How does RL influence a model's qualitative behaviors (*e.g., move quality distribution, reasoning strategies used, rate of hallucination*)?
- *Q3*: Which SFT-checkpoint metrics are predictive of final RL performance?

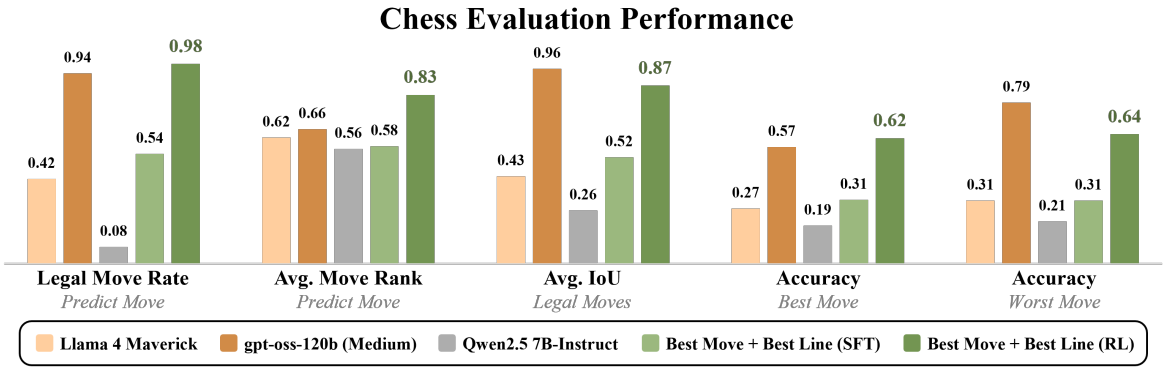



**Figure 5.2. Performance of our best reasoning model trained from Qwen2.5 7B-Instruct across our evaluations**. Trivial performance (i.e., random guessing) is 0.2 for the *Best Move* and *Worst Move* tasks. See Appendix B.2 for example evaluation questions and Appendix B.4 for full results.

We show that focused SFT on predicting a single best move (*Best Move*) leads to strong performance but *unfaithful* reasoning through RL; by contrast, training on multi-step move trajectories (*Best Line*) has more stable RL and faithful reasoning. We find that RL leads to fewer hallucinations and a substantial positive shift in move quality, and we see that several SFT-checkpoint metrics (both *qualitative* and *quantitative*) are predictive of final RL performance. Finally, we ground our results with an experiment measuring *chess information density* for each custom dataset.

## 5.2 Related Work

### 5.2.1 Reasoning in Language Models

*Reasoning* in causal language models can be interpreted as self-guided search that makes a task more tractable. Consider a numerical math problem: effective reasoning should increase the probability of producing the correct number more than if the model had immediately predicted the final answer. Note that this reasoning need not be wholly interpretable – for example, it can exist in continuous space [30] or shift between languages [15] – what ultimately matters is that the intermediate steps are beneficial to the model. For this work, we focus on language-based reasoning.

The era of *reasoning models*, notably initiated with OpenAI's release of o1 [66], builds upon much prior work in self-guided in-context adaptation. Models, when told to work "step by step" and write down intermediate results on a "scratchpad" [65], saw performance improvement on multi-step computations – this result was reinforced at scale and termed "chain-of-thought" in later work [106]. Further, these reasoning traces can be used for iterated improvement through fine-tuning on successful generations as evidenced by STaR [113]. Quiet-STaR [114] extended this from fine-tuning by using a reinforcement learning policy-gradient update in REINFORCE [108] over tokens influenced by intermediate reasoning steps. This iterated bootstrapping using RL for policy-gradient updates has been the primary underlying method fueling the latest developments in reasoning models.

Following OpenAI's release of o1, many leading systems began incorporating similar reasoning techniques to improve performance. Notably, DeepSeek-R1 [15] and Kimi k1.5 [97] were among the first reasoning models to effectively approach state-of-the-art ability and publicize the training methods.

### 5.2.2 Reasoning Through RL

While there exist several effective methods for training models to reason such as in-context prompting [106, 43], model distillation [15], or SFT on successful outputs [113, 112], we focus our attention on the setting of applying RL to improve model reasoning.

RL has been used as an effective tool to guide model behavior with Ouyang et al. [74] inciting the viral *ChatGPT moment* that brought language models to public attention. Successful RL – regardless of domain – requires useful reward signal. For language models, these rewards can be generated using the following: rewards can be parsed and automatically calculated in verifiable tasks [88], determined directly

through human judgment [11], scored with a learned reward model [74], or elicited using a language model as a judge [107]. Rewards can be generated for the entire outcome or at intermediate steps [52], and learned value functions can approximate credit-assignment at the token-level [86] or reward can be indiscriminately applied over a full sequence [88]. This covers many methods but is not exhaustive.

A common family of algorithms used in RL for language models is Proximal Policy Optimization (PPO) [86] – an actor-critic method. Because actor-critic methods require learning a value function to address the credit-assignment problem (which can be computationally expensive and experience instability), new methods such as Group Relative Policy Optimization (GRPO) [88, 15] have emerged to remove this learned value function requirement. GRPO has further evolved through variants such as Dr. GRPO [53], which removes sequence-level length normalization, and DAPO [111], which removes the KL penalty, increases the clipping bound to encourage exploration, and addresses length normalization issues observed in GRPO.

### 5.2.3 Chess Engines

Computer scientists have developed grandmaster-level chess systems built on three notable techniques: 1) Classical search-based engines such as IBM Deep Blue [7] or Stockfish that use a minimax-based search algorithm (commonly alpha-beta pruning), 2) neural search-based systems such as AlphaZero [90] and its open-source implementation in Leela Chess Zero that learn policy functions through RL self-play combined with Monte Carlo Tree Search, and 3) searchless neural systems such as Google DeepMind's chess transformer that predicts a move directly from a board state [81]. The respective 40/15 Elo scores of Stockfish and Leela Chess Zero as of April 19, 2026 are 3652 and 3444 [12], and DeepMind's chess transformer reached a Lichess blitz Elo of 2895 [81]. While it is worth noting that recent versions of Stockfish use neural networks to estimate the value of board states – it is still largely the same core algorithm used by classical engines.

As discussed previously, language models struggle in the domain of chess. However, it is worth mentioning gpt-3.5-turbo-instruct which has an estimated Elo around 1700 [1]. While this anomaly is interesting, this performance isn't from language-based reasoning – rather it is direct next move prediction (i.e., only outputs the move to play). Google DeepMind's chess transformer validated that a 270 million parameter transformer is capable of reaching grandmaster-level chess without search (*this was achieved by learning a value function, though a policy function was also tested*). As far as the authors are aware, no language model has achieved competition-level chess ability through language-guided reasoning and the best reasoning model in chess is OpenAI o3, which won the 2025 Kaggle AI Chess Exhibition [40].

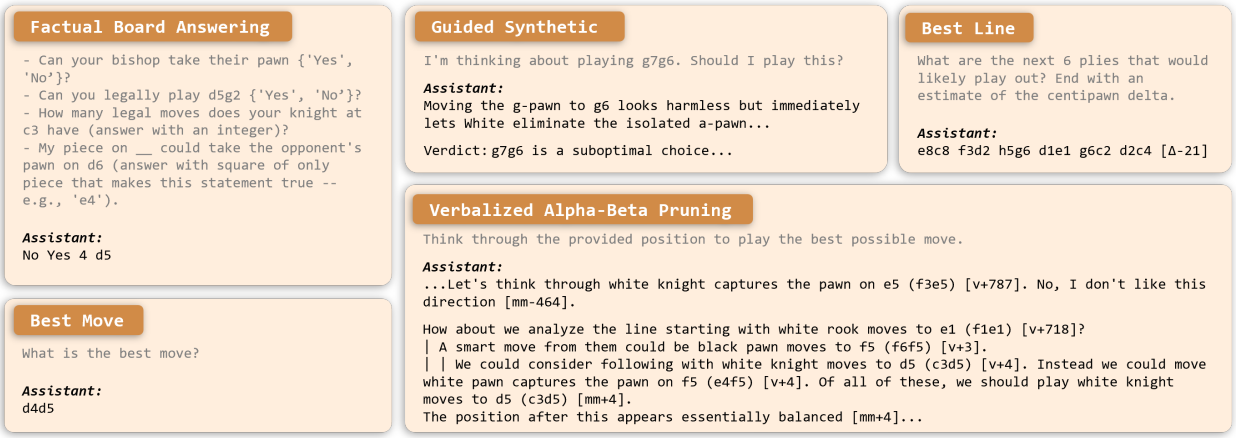



**Figure 5.3. Samples from our custom datasets**. The gray font represents an abbreviation of the core prompt – in all samples the model is trained with a verbose instructive prompt and provided with a board in our visual ASCII-format. Full samples included in Appendix B.3.

## 5.3 Background

Our analysis is focused on the Qwen2.5 7B-Instruct model [79]. Given the baseline model has insufficient ability, we first conducted SFT prior to the RL stage. We began with a full set of data inclusion studies – from SFT to RL – to determine the most effective recipe before doing a final, scaled training run on our leading mix.

### 5.3.1 Board and Move Representation

For all training and evaluation we provide the board state in a visual ASCII-format. We ran preliminary tests on several board formats including Forsyth-Edwards Notation (FEN), FEN with space delimiters, and a visual ASCII-format. While these showed similar quantitative performance, we opted for the visual format following subjective qualitative analysis. Appendix B.1 provides examples of the considered board states and discusses tokenization limitations in each. Note that our board representation omits move repetitions due to dataset limitations – in competition chess, repetitions can be used as a termination condition. However, since none of our evaluations incorporates repetitions, we can view our representation as a Markov-complete state.

For move representation, we follow DeepMind's chess transformer [81] and represent all moves in Universal Chess Interface (UCI) format (e.g., `e1e2`). This decision was made in lieu of formats such as Standard Algebraic Notation (SAN) which may be more commonly represented in training data – SAN has intricacies that could evoke errors avoidable by using UCI notation.

### 5.3.2 Evaluations and RL Environment

We created four custom tasks that we use for evaluations and the RL training environment. The ***Predict Move*** task provides a board and asks the model to play the best move – no list of legal moves is provided. We measure both the ratio of legal moves generated as well as the move quality for legal moves provided. Move quality is measured as the normalized rank among legal moves ($\in [0, 1]$) as determined by a chess engine – where the best move is given a score of 1 and the worst move a score of 0. The ***Best Move*** and ***Worst Move*** tasks provide a board and a set of 5 moves. The task is to choose the best move (and worst move, respectively) of the candidate moves provided. For both tasks, candidate moves are sampled with a move quality threshold (determined by a chess engine) separating the correct answer from other candidates. Finally, the ***Legal Moves*** task asks the model to, for a given board and piece, list all legal moves that piece can make. Results are computed as intersection over union (IoU) versus the ground truth. We provide example questions in Appendix B.2.

### 5.3.3 Datasets

We created several theoretically-inspired datasets to study training dynamics from SFT to RL. Consider that chess can be represented as an MDP. At time $t$, there exists state $s_t \in \mathcal{S}$, playing a ply (i.e., half-move) constitutes an action $a_t \in \mathcal{A}(s_t)$, and an environment transition (i.e., opponent move) is a state transition $s_{t+1} \sim \mathcal{T}(s_t, a_t)$. Additionally, for each board state-action pair there is a reward $r_t = \mathcal{R}(s_t, a_t)$ which we can approximate using a shaped dense reward (centipawn delta, i.e., the change in an engine's board evaluation measured in hundredths of a pawn) from a chess engine: $r_t = \gamma V_{\text{engine}}(s_{t+1}) - V_{\text{engine}}(s_t)$ with $\gamma = 1$. We will use this formulation to discuss motivation for several of our custom datasets.

We provide a brief description of each dataset and will further elaborate on data design and motivation within the context of experimental results in Section 5.4. We include detailed explanations of each dataset and full examples in Appendix B.3 – abbreviated examples are included in Figure 5.3. Regarding our datasets, we organize them into the following four categories:

- **General Instruction Following**: Specifically, Magpie Llama 3.3 70B [109].
- **Rejection Sampling**: We generate outputs from Llama 4 Maverick [57] on our four evaluation types. We chose Llama 4 Maverick for qualitative and quantitative performance, retaining samples from the *Best Move* and *Worst Move* evaluations if correct and keeping outputs from the *Legal Moves* and *Predict Move*

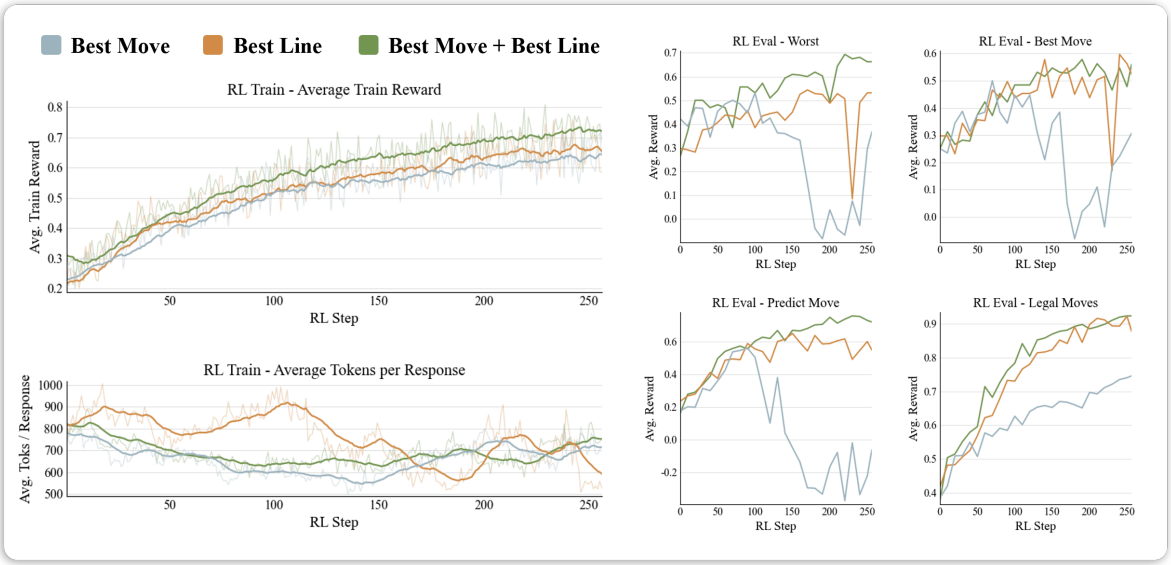



**Figure 5.4. RL training performance on our scaled SFT-checkpoints**. *Left*: Train reward and tokens per response (smoothed using an exponential moving average with decay factor 0.9). *Right*: Reward on the held-out evaluation set during training. The *Best Move* dataset, while having strong ending performance, experienced more unstable RL compared to scaled runs trained on *Best Line* data.

evaluations if above a threshold.

- **Guided Synthetic**: We prompt Llama 4 Maverick and gpt-oss-120b with a programmatically generated harness. Specifically, we provide a beginning board, 5 plies (the first ply being a move candidate and following plies being optimal play from a chess engine), and the ending board state. The task is to generate an explanation of how the proposed candidate move will play out, ending with a final verdict for the proposed move.
- **Programmatically Generated Data**:
  - **Factual Board Answering**: We build on top of a chess engine to generate simple question-answer (QA) pairs for a given board. These questions may ask if a move is legal, which square is threatening a specific piece, or how many legal moves a piece has. We combine multiple QA pairs for each sample.
  - **Verbalized Alpha-Beta Pruning**: We use a custom program built upon Stockfish to sample moves, rollout the line of play for each move (with branching and board values), and verbalize rollouts and minimax decisions in natural language. We explicitly build in tree search reasoning strategies and sample poor moves to verbalize the process of *pruning*, and we leverage a large, custom prompt bank to add diversity to natural language outputs.

– **Best Move**: Given a board, immediately predict the best move in UCI notation.

– **Best Line**: Given a board, predict the optimal line of play (4 – 6 plies) ending with the expected centipawn delta from playing this line.

### 5.3.4 Training Environment

All SFT is conducted using LlamaFactory [118] and all RL is conducted using veRL [89]. We utilize Dr. GRPO [53] for our RL optimization algorithm and employ the *Clip-Higher* strategy with no KL divergence per Yu et al. [111]. A full list of hyperparameters is included in Appendix B.9.

## 5.4 Key Findings

We ran a series of inclusion analyses to understand the efficacy of each data type and scaled our best-performing recipes. Figure 5.2 highlights the performance of our best reasoning model. We found the *Best Move* and *Best Line* datasets to be most effective – especially when lightly supplemented with our other, less effective datasets. Our scaled runs build on the *Best Move - All* and *Best Line - All* datasets that use this dataset diversity. The best final performance was achieved by first training Qwen2.5 7B-Instruct on 60 million tokens (*Best Move - All* data) followed by 60 million tokens (*Best Line - All* data). Appendix B.4 provides further detail on our experiments.

### 5.4.1 Q1: How do different datasets impact downstream performance after SFT and RL?

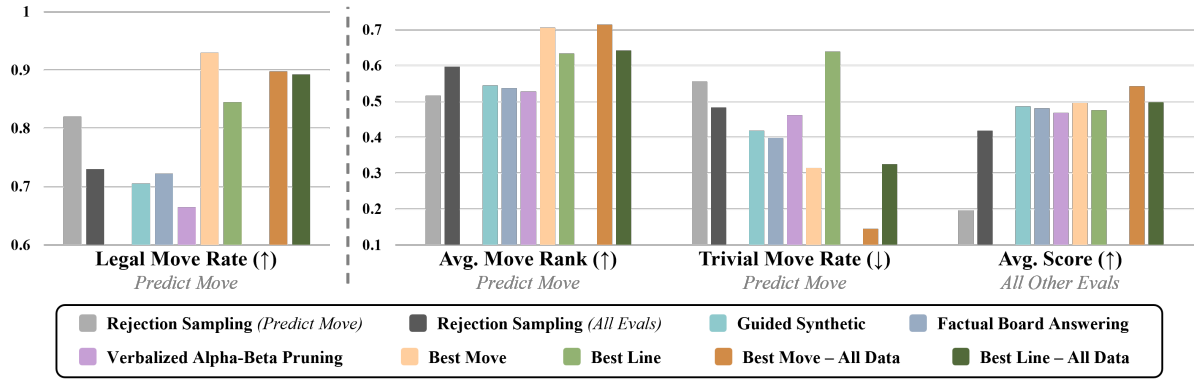



**Figure 5.5. Results on evaluation tasks for final RL models from each data inclusion experiment**. Within each metric we split results into three sections: *(left)* compares single vs. multitask, *(middle)* compares the targeted data inclusion experiments, *(right)* covers our data diversity experiments. Note that in all experiments we SFT on 15 million tokens and do RL on 8k samples. In the single task setting, RL only uses the *Predict Move* task; in all other settings the 8k samples are split evenly between our four task types. See Appendix B.4 for detailed table results (including exact token distributions).

**Multitask training is beneficial** for a fixed token budget, yielding higher move quality, less reward hacking, and a generally more robust model. This is shown by a comparison between *Rejection Sampling (Predict Move)* and *Rejection Sampling (All Evals)*: for the former we SFT and conduct RL on only the *Predict Move* task – for the latter we use all tasks. Our experimental results (Figure 5.5) led to these takeaways. We incorporate multitask training in all following experiments.

**The most effective datasets were dense with difficult, high-quality tokens**. We created three "dense" datasets that lack prose: *Factual Board Answering*, *Best Move*, and *Best Line*. The intent was to force the model to embed complex board understanding and strategy in latent layers through immediate responses. These three largely performed the best, although *Factual Board Answering* was not materially better than our initial *Rejection Sampling (All Evals)* experiment. One explanation is that *Factual Board Answering* is more easily learned versus the other dense tasks. Consider *Best Move* – playing a strong move requires internalizing some level of board understanding and strategy, effectively encompassing *Factual Board Answering* and requiring a richer latent representation. We further discuss *chess information density* in Section 5.5 and provide a lens for interpreting dataset performance – but generally, our dense datasets performed best.

**Dataset diversity is valuable**. We see that the *All Data* experiments for *Best Line* and *Best Move* are superior to their focused counterparts (Figure 5.5). For these runs, we SFT on nearly all our data types – this comes despite mixed results on several of the individual inclusion analyses. Notably, the *Verbalized Alpha-Beta Pruning* experiment showed it was detrimental for training – we created this dataset as it is hallucination-free and includes rollouts and value functions (instilling $V(s_t)$ and $\mathcal{T}(s_t, a_t)$). However, it possesses a low density of high-quality tokens (moves and valuations) given the surrounding memorizable prompts. The *Guided Synthetic* dataset similarly produced subpar performance versus a *Rejection Sampling* baseline. Regardless, limited inclusion of all data was found to be beneficial.

***Best Line* had more stable RL training than *Best Move***. Figure 5.4 outlines RL training performance for our scaled runs – the models fine-tuned on *Best Line* data had more stable training dynamics. One reason may be that training on *Best Line* data – which includes multiple moves and ends with a valuation – allows the model to learn a world model for chess (both a value function $V(s_t)$ and transition dynamics $\mathcal{T}(s_t, a_t)$). This is further supported by the coming discussion on *reasoning faithfulness* that reinforces this stability observation. Additionally, we find that models trained on *Best Line* perform better than our *Best Move* experiments on evaluations not trained on during RL – validating our observation that training on *Best Line* data leads to a more robust policy. Detail on the out-of-distribution evaluation is included in Appendix B.4.

### 5.4.2 Q2: How does RL influence a model's qualitative behaviors?

**Multi-step trajectory data led to the most faithful reasoners**. We measure faithfulness by using gpt-oss-120b to judge alignment of final answers with reasoning traces. Our most faithful reasoner was achieved by training on the *Best Line* dataset which incorporates a rollout (in UCI) followed by a valuation (as a centipawn delta). This structure can be viewed as approximating $n$-step bootstrapping with $n = 2$ or 3 depending on ply depth. We can contrast this with the *Best Move* dataset which approximates a direct policy function (i.e., learning $\pi_\theta(a_t|s_t)$ via behavior cloning). Our multi-step trajectory checkpoints largely retained faithful reasoning through RL – on the other hand, the *Best Move* dataset became an *unfaithful* reasoner through RL, often displaying final, nontrivial answers that were disconnected from its reasoning trace. Figure 5.6 highlights reasoning faithfulness and Appendix B.8 has further detail.

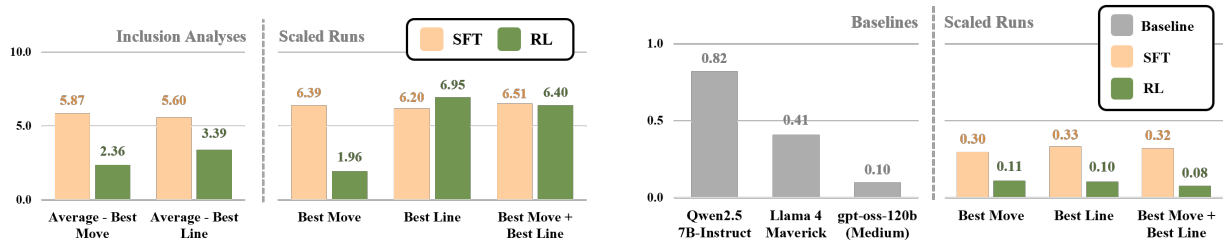



**Figure 5.6.** *(left)* **Reasoning faithfulness**. RL on the *Best Move* SFT-checkpoint induced *unfaithful* reasoning whereas checkpoints trained on multi-step data were more robust. Appendix B.8 has further detail on our reasoning quality measurement. *(right)* **Hallucination rate**. RL drove a meaningful decrease in hallucination rate as a side effect of simply maximizing reward on our evaluations. Appendix B.6 has further detail on hallucinations.

This is interesting as the unfaithful reasoner improves through RL without defaulting to trivial moves. Further, this improved ability is not explained by longer generations (Figure 5.4). One possible explanation we offer is the following: faithful reasoning from multi-step data may arise due to the model internalizing a chess world model (transition and value functions), whereas unfaithful reasoning may result from strong latent capability mixed with weak verbalized reasoning ability. Previous work [103] has found that models may attempt to rationalize their answers in chain-of-thought unfaithfully if they are biased; in our case, the model may be attempting to rationalize the move it has "already decided".

Regardless of reasoning faithfulness, **RL drove a substantial positive shift in move quality played** (Figure 5.7). Not only does RL improve the frequency of the best moves being played but it also decreases the frequency of low-quality moves on an absolute and relative basis. Additionally, **RL reduces hallucinations within reasoning traces** (Figure 5.6). This result is a side effect of rewarding correct answers as we do not

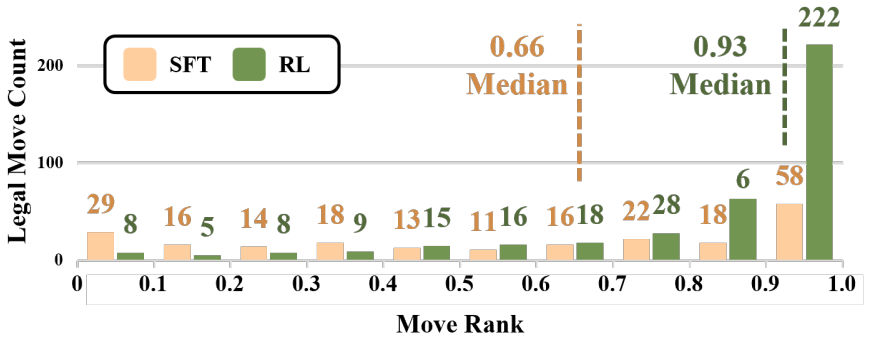



**Figure 5.7. Move quality distribution**. Our *Best Move + Best Line* scaled run saw a significant distribution shift after RL in its move quality on the *Predict Move* evaluation ($n$ = 400). This shift is an improvement on both an absolute and relative basis, highlighting the efficacy of RL.

incentivize factuality – we provide further detail on hallucinations in Appendix B.6 and show that this result is shared across all inclusion experiments.

Lastly, we analyzed reasoning strategy usage at both the SFT and RL model stages. This follows prior work [23, 115] showing that effective reasoning models tended to utilize more reasoning strategies. We did not see clear trends in our analysis apart from our weaker models – specifically those more prone to reward hacking – almost exclusively reducing the usage of reasoning strategies through RL. In contrast, stronger models had mixed usage trends. We defer to Appendix B.7 for detail.

### 5.4.3 Q3: Which SFT-checkpoint metrics are predictive of final RL performance?

Finally, we conducted a simple linear regression analysis comparing metrics from the SFT-checkpoint with the final RL model's performance (average over all evaluations). Figure 5.8 highlights three SFT-checkpoint metrics that are statistically significant predictors of downstream performance.

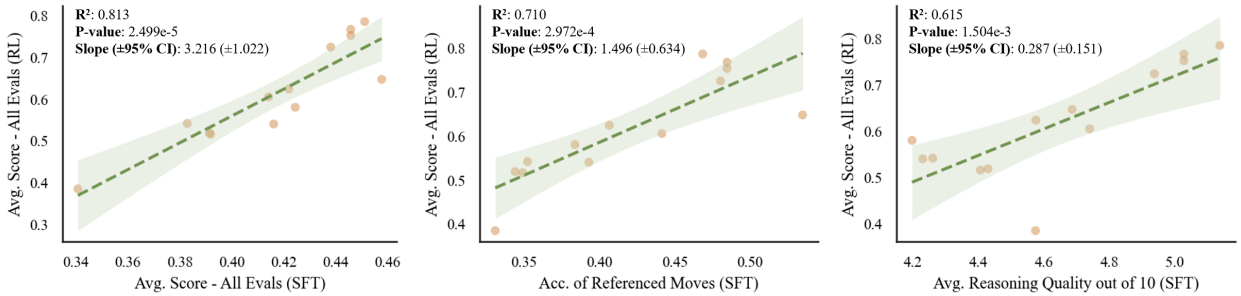



**Figure 5.8.** Linear regression comparing the final RL model (average score over all evaluations) with various metrics from its corresponding SFT-checkpoint. *Left*: Vs. average score over all evaluations. *Middle*: Vs. percent of moves referenced during reasoning trace that are legal (parsed by Llama 4 Maverick). *Right*: Vs. reasoning quality (mean over all reasoning quality metrics as judged by gpt-oss-120b). Shaded region represents the 95% confidence intervals.

Some of this is expected – an SFT model that scores higher on evaluations is likely better suited for RL. However, we also find qualitative signals to be highly predictive. Specifically, referenced move accuracy (*how little does the model hallucinate?*) and reasoning quality (*ask a language model to score how effective the reasoning is*) are statistically predictive of downstream performance. This shows that an effective SFT-checkpoint is one that is truthful (low hallucination rate), already an effective reasoner, and exhibits strong performance in the domain of focus.

## 5.5 Chess Information Density

Given our inclusion experiments hold the number of train tokens constant, we desire some measure for the information density of each dataset. While there are parallels between what we are terming *chess information density* and concepts in broader information theory, we instead take a first-principles approach and discuss related experiments, analyses, and qualitative results through a simple framework that offers intuitive takeaways.

There is no single, clear metric that quantifies *chess information density* – rather, there are several desirable attributes we would like in our datasets. First, the dataset should remain non-trivial to predict throughout training – we term this *predictive complexity*. Now, this itself is insufficient as both randomly generated data and text irrelevant to chess can evade triviality but be functionally useless for modeling chess. We must also consider *accuracy* (are the tokens correctly grounded to the task and board?) as well as *chess token density*. This latter concept hints at the idea that not all tokens are equal – some tokens may require board understanding or chess strategy to effectively predict (e.g., in *Best Move* we ask to directly output the best move given a board). Compared to samples from the *Rejection Sampling* dataset – which have many natural language generic reasoning tokens – the *Best Move* data has higher *chess token density*.

To measure *predictive complexity*, we trained Qwen2.5 0.5B-Instruct [79] on 4 million unique tokens for 2 epochs (total 8 million tokens). We hold out 5% of our dataset for validation and store probabilities associated with each validation token for further analysis.

Table 5.1 highlights the fraction of validation tokens that the model assigns a probability $> 0.995$. Datasets with a high portion of trivial tokens can be interpreted as less complex – that is, the underlying patterns are more easily learned. Notably, we find *Verbalized Alpha-Beta Pruning* becomes trivial as the programmatic phrases used to verbalize search are quickly memorized – tokens that retain uncertainty often refer to a move decision or board valuation in their respective search branch. Additionally, the *Factual Board*

**Table 5.1. Fraction of validation tokens assigned probability >0.995 before and after SFT**. Higher percentages indicate lower *predictive complexity* (e.g., simple task or memorizable phrases).

| Dataset | Pre-SFT | Post-SFT |
|---|---|---|
| Rejection Sampling | 12.69% | 20.83% |
| Guided Synthetic | 2.90% | 9.77% |
| Factual Board Answering | 0.04% | 62.54% |
| Verbalized Alpha-Beta Pruning | 4.31% | 71.04% |
| Best Move | 0.04% | 28.61% |
| Best Line | 0.00% | 24.00% |

*Answering* dataset has a high portion of trivial tokens. Upon further inspection (Table 5.2), we find that certain tasks become easy to predict while others remain more difficult. This is expected as not all tasks will be the same difficulty; however, it does have implications on dataset efficacy and benefits of scaling. We provide further information, including several visualizations, in Appendix B.5.

Beyond *predictive complexity*, we must also consider *accuracy*. We have two synthetic datasets (*Rejection Sampling* and *Guided Synthetic*) that are prone to some amount of hallucination – the other datasets, as they are programmatically generated, avoid this problem. While we find the hallucination rate to be higher on synthetic datasets vs. programmatically generated datasets (Appendix B.6), there are several confounding factors at play. At best, dataset *accuracy* is one of many factors that play into dataset quality.

The final measure to consider is *chess token density* and is more qualitative. Of our datasets, *Verbalized Alpha-Beta Pruning* and *Rejection Sampling* have the highest portion of generic natural language. *Guided Synthetic*, while still natural language, is prompted to produce a succinct verdict following prior reasoning – as a result, *Guided Synthetic* has lower triviality compared to *Rejection Sampling* (Table 5.1). Our last three datasets we consider to be dense as they lack prose: *Factual Board Answering*, *Best Move*, and *Best Line*. All three were designed for *chess token density*; however, we see in Table 5.1 there are discrepancies

**Table 5.2. Validation tokens assigned probability >0.995 before and after SFT for Factual Board Answering tasks**. See Appendix B.3 for descriptions of each task.

| Question Type | Token Portion | Pre-SFT | Post-SFT |
|---|---|---|---|
| is_legal | 13.59% | 0.00% | 24.85% |
| under_attack | 12.43% | 0.00% | 30.46% |
| mobility | 24.54% | 0.01% | 43.88% |
| cloze_capture | 15.94% | 0.14% | 80.26% |
| is_check | 4.95% | 0.00% | 85.16% |
| mat_adv_value | 28.54% | 0.05% | 96.66% |
| **Total** | **100.00%** | **0.04%** | **62.54%** |

in *predictive complexity* across the three that align with our observed results. Intuitively, in *Factual Board Answering* we ask the model to answer various questions requiring board understanding. Compare this to *Best Move* where, to play an effective move, much of this board understanding needs to be internalized and considered. Now, compare *Best Move* with *Best Line – Best Line* requires tracking board state and modeling competing lines from players. Our dense datasets – those with higher *chess token density* – tend to perform better; among the dense datasets, those with higher *predictive complexity* tended to perform better as well.

We believe our broader results can be partially explained by these measures of *chess information density*. However, we also see that dataset diversity is beneficial (*Best Move - All* outperforming *Best Move*), and data such as *Guided Synthetic* – which should be superior to *Rejection Sampling* – does not outperform. Thus, these proximal measurements are a useful perspective but do not represent the full picture.

## 5.6 Limitations & Further Discussion

To begin, the intention of this work has always been to study general reasoning properties in language models. Thus, while the final evaluation of our model is a welcome result, we focused much of our effort on understanding the qualities and development of reasoning; this means that there are many methods we believe could further improve a chess reasoning model beyond our final RL model. For example, in full-game play our final RL model had poor performance against OpenAI o3. We suspect some level of distribution mismatch: training emphasized mid- and late-game positions to reduce trivial moves, which likely degraded opening play and hurt head-to-head results versus an opponent with stronger opening theory.

We have also identified several unexplored techniques that could increase performance in our final RL model. We minimally experimented with reward function tuning in our RL environment and expect focused effort could improve performance – particularly on the *Predict Move* task. Further, incorporating multi-turn RL and chess puzzles to more closely mimic a full chess game would likely yield strong improvements.

Regarding model choice, we acknowledge that our experiments were confined to Qwen2.5 7B-Instruct – while it would have been valuable to replicate on distinct base models, due to constraints this was not pursued. Additionally, we chose gpt-oss-120b as our comparator because, in tests against Kimi K2 and DeepSeek-R1-0528, it showed state-of-the-art open-source performance and was more convenient to run with our available resources.

Finally, we believe that there is a promising direction in training on environment dynamics to improve general reasoning abilities that is beyond the scope of this work. Recent works such as CodeI/O [49] and

Absolute Zero [116] show that training a model to predict program outputs from both code and inputs produces strong reasoners. We believe this can be pursued in other sequential domains as well – for example, dialogue or embodied agentic tasks – where the model can be trained on both actions and responses.

## 5.7 Conclusion

We conduct a detailed study of how various custom datasets influence training dynamics through SFT and RL in the domain of chess. Our analysis highlights that training to predict the best move directly produces strong downstream performance but comes with *unfaithful* reasoning. Instead, training on multi-move trajectories delivers similar performance with faithful reasoning. We find that RL leads to fewer hallucinations and a substantial positive shift in move quality, and we see that several SFT-checkpoint metrics (both *qualitative* and *quantitative*) are predictive of final RL performance. Finally, we ground our results with an analysis of *chess information density*. We publish our code and data as well as scaled SFT-checkpoints and RL models.

## Acknowledgements

Chapter 5, in full, is reproduced from the material as it appears in **How Reasoning Evolves from Post-Training Data: An Empirical Study Using Chess**, Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Dionisopoulos, Lucas; Majamaki, Nicklas; Ammanabrolu, Prithviraj. The thesis author was the primary investigator and primary author of this paper.

# Chapter 6

# A Unified View of Intelligence and Future Directions

We conclude by returning to our initial premise: under our dual-system view of artificial intelligence, there exist striking similarities between the symbolic and neural settings that warrant a unification.

While we could approach this unified view in several ways, we will focus on *library learning* – the most holistic symbolic system we have discussed. Specifically, library learning offers many desirable attributes for an artificial intelligence system – a loop that combines both knowledge (primitives, learned compositions, and search and compression algorithms) with composition (online experience that steers knowledge updates). We will now show how modern reasoning models are developed under the same meta-framework.

## 6.1 A Unified View: Bridging Intelligence from Symbolic to Neural

First, let us compare the meta-framework between symbolic library learning and modern reasoning models. At a high level, language models replace the brittle, symbolic elements in knowledge by approximating both the *library* and *search algorithm* directly with their neural network. These are learned together in the same network – by contrast, notable neurosymbolic strategies keep these separate [19, 26].

Both techniques have similar methods of composition as well. Recall that previously, we discussed language modeling through both a grammar-centric view (over tokens) and a thought-centric view (Figure 4.2). Both are useful abstractions – for this comparison, we consider a new abstraction that separates intermediate steps (e.g., thinking, tool calls) from final answers (e.g., aggregate code changes). These intermediate steps are the artifacts of language-based search. In the symbolic setting, we conduct explicit search using various algorithms or learned policies – during this search, we may visit many nodes before finding our final answer. In this same way, intermediate steps in language models are nodes visited on the way to a final answer; at the end, these intermediate steps are essentially discarded as well.

We now highlight key similarities between the neural and symbolic domains through a comparison of their meta-frameworks. An overview is provided in Figure 6.1.

- **Library**: For language models, base knowledge is learned in the neural network. While the neurosymbolic approach requires us to provide a starting set of primitives, language models acquire these "primitives" during pre-training. As it happens, language models are very effective at learning a breadth of general knowledge (e.g., super-human performance on difficult knowledge-based benchmarks like GPQA [80]).
- **Search Algorithm**: Similarly, this is instilled in the language model during training – shaping this is often a major focus of post-training. While some neurosymbolic methods learn a neural network to guide search, this network is distinct from other knowledge. As we discussed in Chapter 1, knowledge primitives may be useful in guiding search – this provides a case for learning both in tandem as is done with language models.
- **Compression Algorithm**: This is, essentially, any process that permanently updates state. For language models, this is the update mechanism (i.e., data, optimizer) but can also include writes to persistent memory. Neurosymbolic techniques (e.g., *LaSR* [26]) have leveraged language models for this compression step but may use a combination of heuristics and statistical learning.
- **Online Experience Collection**: Both systems are applied to various tasks and environments, a metric is generated, and this experience is used in the offline setting to steer updates.

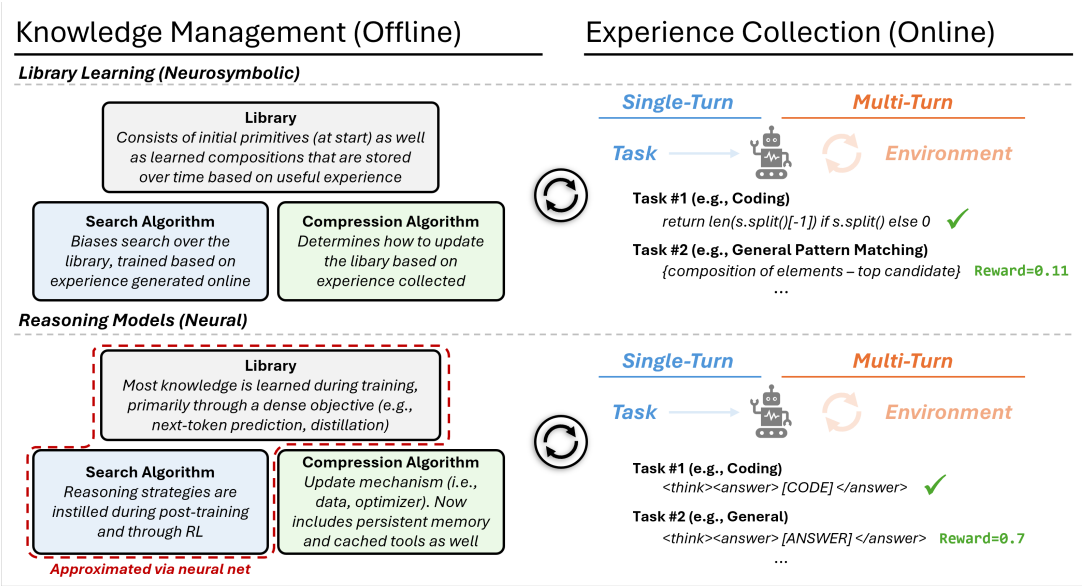



**Figure 6.1.** Unified meta-framework between library learning and reasoning models. Reasoning models effectively remove brittle symbolic components by approximating these with the neural network. Both techniques leverage online experience collection in the training process to create an improvement loop.

It is interesting that recent advancements in language modeling have shown further convergence to the neurosymbolic setting. First, as reasoning traces became more complex, we saw general search techniques applied – first explicitly, later amortized into the models. Next, reasoning models bridged the experience gap – allowing models to learn from online self-generated rollouts. Lately, we are seeing more advanced memory components that rely on a compression step that asks: *what knowledge is useful to retain*? This mirrors the compression algorithm used in library learning.

What has allowed neural approaches to scale better than their symbolic counterparts has been the power of neural approximation. We will now briefly discuss this mechanism.

### 6.1.1 Knowledge Through Universal Approximation

Knowledge – both primitives (e.g., facts) and biases for search – is approximated by a neural network in the language model setting. By approximating these two components, we are able to avoid much of the brittleness that plagues neurosymbolic techniques – assuming we can approximate this knowledge effectively.

As it happens, one of the astounding abilities of neural networks is their efficacy as *universal approximators* [13, 36]. In theory, a sufficiently large multi-layer perceptron can serve as an arbitrarily smooth function approximator; neural networks have proven broadly effective in practice, though universal approximation alone does not explain their trainability or generalization.

Not only does this neural approximation remove the rigid, (often) manually defined elements of the neurosymbolic setting, but it also enables an end-to-end objective while allowing knowledge primitives to influence search. These are incredibly desirable attributes – however, neural approximation does come with limitations that we discuss shortly.

### 6.1.2 The Unified View

To summarize the unified view, let us return to our proposed framework: *artificial intelligence is a dual-system that combines knowledge with the capacity for composition*. Recall that knowledge is not just primitives (i.e., 'facts') – it includes biases for search as well as algorithms and processes for making permanent updates to the model; composition is your search process – also referred to as *reasoning*.

As we just discussed, language models share a meta-framework with library learning – a setting that represents our most ambitious, holistic symbolic architecture. For the language model, knowledge components are learned through neural approximation – effectively removing the brittle, symbolic components that have

proven challenging in these neurosymbolic systems. This shared meta-framework best summarizes our unified view (Figure 6.1) – what we initially presented as a compelling framework for artificial intelligence in library learning has been effectively implemented with language models.

What this unified view offers is a way to bridge a theoretical, first-principles approach to artificial intelligence with its practical counterpart. It offers an argument that language models, specifically reasoning models trained using a reinforcement learning stage, have the necessary pieces in place.

But beyond acting as an endorsement of reasoning models, this unified view makes clear a trend of language model developments incrementally converging to our proposed principled system. The pieces that remain – the theoretical, proposed benefits of a neurosymbolic setting – may help us think about research directions, especially if this trend of convergence is to continue. We now discuss a key remaining piece that remains elusive for language models.

## 6.2 Limitations of Neural Approximation

Despite the efficacy of neural approximation, there remain a few desirable aspects of the idealized neurosymbolic system – these are valuable to discuss as they may inspire future work.

### 6.2.1 Approximation Doesn't Model Underlying Mechanisms

While neural networks can learn to model complex data, the model may not be learning the underlying mechanisms that generated the data. For example, a transformer can predict orbital trajectories effectively then fail to transfer when tasked with predicting force vectors [104]. If the network truly modeled the mechanism of gravity, this transfer should be trivial.

Similarly, Kumar et al. [45] compare neural approximations with those evolved using evolutionary algorithms. Neural approximations learn the underlying data, but minor adjustments to latent layers cause a large deterioration in performance. On the other hand, evolutionary algorithms that learn the same data are robust to latent edits – in fact, latent edits often correspond to interpretable features (e.g., if the original output is a face, editing one latent dimension changes the mouth size).

Both of these results hint at the concept of entanglement. *Disentangled learning* [32, 55] is a subfield of machine learning interested in these questions – specifically, can we learn the underlying mechanism that generated the data rather than just approximating it? Entangled representations, while being effective for in-distribution data, often suffer from poor out-of-distribution generalization while also being less efficient.

This is related to broader concerns about overparameterization and redundancy; for example, the *Lottery Ticket Hypothesis* [22] shows that large networks can often contain much smaller subnetworks with comparable performance.

This is where alternatives to deep learning may shine. Notably, researchers are using neurosymbolic programming in symbolic regression tasks to uncover scientific principles directly from data [26, 93]. Through symbolic composition, the hope is to discover the equations that model the underlying data – a level of explainability neural networks cannot offer.

### 6.2.2 Sample Efficiency

Another limitation of neural approximation is that neural networks are sample inefficient. This is connected to the aforementioned entangled representations – while these approximations can fit data, they are not modeling the underlying mechanisms that generated the data.

As a result, language models require massive datasets to train. In the *Chinchilla* [35] paper, empirical analysis found that for a 10B parameter transformer, the "optimal" number of tokens to train on was 200B. This equates to hundreds of millions of pages of English text – far more than any human could read in a lifetime. However, these "optimal" token counts no longer dictate training: Gemma 3 12B [96] was trained on 12T tokens – over 50× more data. Note, there are reasons for this deviation that we choose not to elaborate on.

While neural approximation has proven to be effective, it lags in efficiency. The last few years have been shaped by the exploitation of *scaling laws* [42] – where training a larger neural network on more data leads to better performance. To support this, we have seen remarkable investment in hardware and training infrastructure. However, sample inefficiency remains an open problem in neural approximation. Neurosymbolic techniques – by fitting patterns in data with increasingly complex compositions of symbolic expressions – offer another approach that may be significantly more sample efficient.

### 6.2.3 Kolmogorov Complexity

All these limitations hint at a deeper concept: *Kolmogorov complexity* [50]. While this thesis has largely focused on techniques – neural, symbolic, and neurosymbolic – these are simply methods to model observed data.

Kolmogorov complexity seeks to quantify how complex data is – *for a given dataset, what is the shortest program that outputs it*? In practice data has noise – we cannot cleanly model it – but we hope to

reduce the gap between neural behemoths and the data's Kolmogorov complexity.

A notable aspect of library learning is that by starting with a set of primitives and growing a library through composition, you are biased toward simple programs that model data. This aligns with *Solomonoff induction*: computable hypotheses are assigned prior weight according to description length, so shorter (i.e., *simpler*) hypotheses receive greater prior probability [91, 92]. If we assume this to be the correct prior assumption, artificial intelligence that grows in complexity may better model our underlying data.

Thus, despite the many similarities between library learning and modern language modeling, there remain several aspects unique to our neurosymbolic method. These are largely idealized and have proven evasive in practice, but progress may help us build systems with robust generalization and improved sample efficiency, and allow us to discover new science that defines our observable universe.

## 6.3 The Value of Symbolic Methods

We are left with a natural question: *where do symbolic methods fit into practical, modern artificial intelligence*?

Shadowing the idealized promises of symbolic techniques are real, rapid advancements in neural systems. We now briefly discuss a few directions that offer promising applications and collaborations with symbolic methods.

First, symbolic techniques have already made their way into frontier artificial intelligence systems. Constrained decoding [24, 76] was quickly integrated into leading systems [67] as it used ideas from programming language design to guarantee language model output structure (e.g., JSON).

Beyond this, benchmark research challenges (e.g., *Modded-NanoGPT* [39]) have begun to transition to *automated research* loops. These automated research loops – driven by language models – have many similarities with program synthesis: given a synthesis system (language model) and oracle verifier, produce the best performing solution. Additionally, these systems have begun to incorporate symbolic techniques (e.g., search) and alternative optimization algorithms – for example, *AlphaEvolve* [64] combines language models with evolutionary algorithms for automated discovery. There are many opportunities for further cross-over of program synthesis with these automated research loops given the structural similarities.

As a final example, there remain many ambitious attempts to address flaws in neural approximation using alternative architectures and neurosymbolic methods. *Kolmogorov-Arnold Networks* [54] propose an alternative to neural networks that may better model underlying data structure. We are also seeing companies

raise significant capital under the premise that we should explore new approaches in artificial intelligence that take inspiration from neurosymbolic techniques.

While neural approximation has been the primary driver of recent advancements in artificial intelligence, there still remain many valuable applications of symbolic techniques – especially when used to augment language models.

## 6.4 Conclusion

Through our proposed dual-system view of artificial intelligence, we bridge core concepts that span notable symbolic and neural techniques. This allows us to identify a shared meta-framework – a framework that offers a unified view tying a first-principles approach to artificial intelligence with its practical counterpart in modern reasoning models.

This unified view is presented with two novel works – a neurosymbolic method for solving ARC-AGI problems (Chapter 3) and an empirical study measuring how data influences downstream reasoning (Chapter 5) – and from this unified view, we see a trend of language model developments driving convergence to the principled approach. If trends continue, this unified view may anticipate the next paradigm for language models.

## 6.5 Final Predictions

This final section reflects my own beliefs as the thesis author. I thank those who have read this far – if any – and believe it is only fitting to end with a few personal predictions.

### 6.5.1 Is There a Ceiling for Language-Based Reasoning?

I believe there is no shortage of improvements to be had from further scaling reasoning models.

The transformer architecture is remarkably effective – despite some brittle performance when dealing with out-of-distribution data – the utility, systems, and incentives are in place to continue to grow training environments and datasets to achieve desired performance. What remains is largely a data problem – that isn't to say that many advancements in architectures, optimization, and inference harnesses are not valuable as they are – but I believe most limitations are simply a matter of data curation. This involves data for dense token prediction, but is mostly in reference to environments for reinforcement learning.

If you refer to the dual-system, we have learned how to instill super-human breadth of knowledge into a large neural network. The remaining step is to refine the search process for challenging problems – this is largely done through the on-policy experience regime of reinforcement learning.

There is much work remaining here and I believe we have hardly scratched the surface on novel environments. That said, I do think that we will look back on our current transformer-based language models as a necessary but suboptimal step in building advanced artificial intelligence. For reasons mentioned earlier, pure neural approximation has fundamental flaws; solving these flaws – *building better compression machines* – may offer significant improvements.

### 6.5.2 Ambitious Predictions

In a complete deviation from what is customary in a thesis, I will make a few interesting predictions that I intend to track.

And yes, I fully understand that these predictions may age like milk. But some things we do because they are fun – and in a world full of ambitious claims, let the previous chapters afford me a pinch of credibility to differentiate myself from the masses.

| Year | Prediction | Outcome |
|---|---|---|
| 2026 | A demo from a major AI lab, likely OpenAI, will show a clear breakthrough in multi-modal video-input capability that allows for near real-time computer use that is fast, effective, and affordable. This will mark a shift from a reliance on rigorously developed RL training environments to training on highly scalable, flexible, computer-use tasks. | |
| 2026 | Apple acquires a neolab, likely Thinking Machines. | |
| 2026 | A model scores > 85% on Frontier Math (1–3), > 75% on the Artificial Analysis Intelligence Index, and > 75% on HLE with tools. | |
| 2027 | A Millennium Prize Problem will be solved where AI is heavily involved in the process. | |
| 2027 | AI-fueled job layoffs hit mainstream white-collar work through improved agentic capability in Microsoft Office and equivalent platforms. | |
| 2027 | An AI-generated song will break into the top 10 charts. *Note: I am not personally a fan of AI-generated music or videos, but they will find a niche.* | |
| 2028 | AI layoffs will be made a key talking point of the 2028 U.S. election. There will be a rise of populist talking points and candidates on both sides as a direct result. | |
| 2028 | Google and OpenAI both announce a humanoid robot. | |
| 2028 | We begin to see cobot solutions in retail spaces, such as a robotic barista. Video-generation world models will heavily play into this development. | |
| 2028 | A major contribution in physics, chemistry, or biology worthy of a Nobel Prize will be developed through heavy usage of an AI system. | |

# Appendix A

# Supplementary Materials for Chapter 3

## A.1 Synthetic Data Generation Examples

Samples from our synthetic data generator (at program depth of 2) are shown in Figure A.1. Program depth is an adjustable argument – this allowed us to incorporate a curriculum during training where we trained on programs of increasing depth (i.e., difficulty) over time.

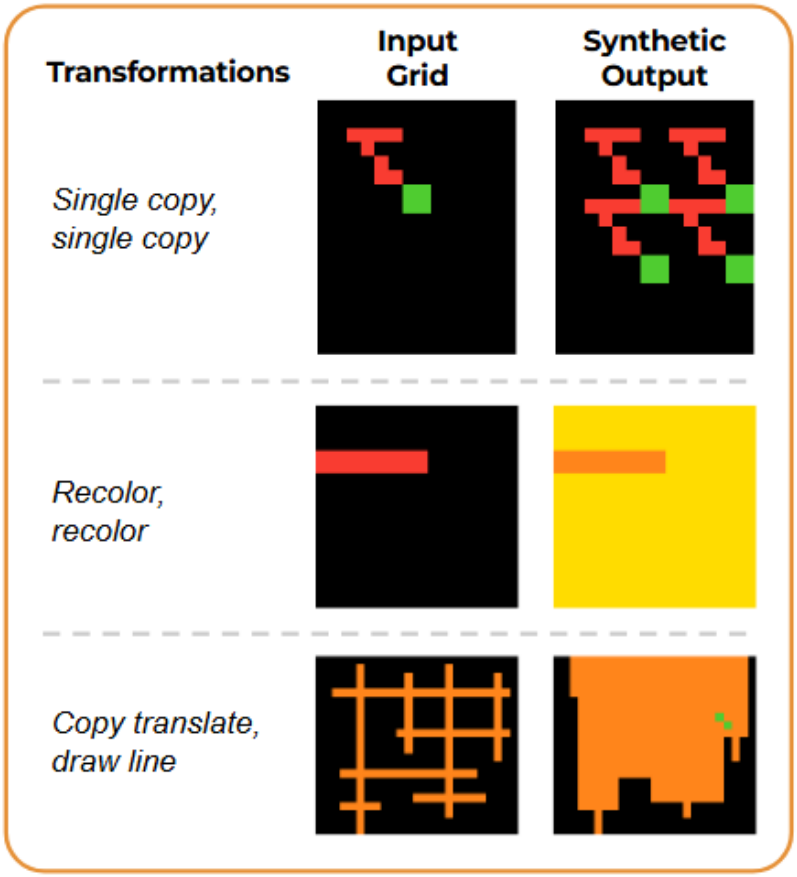



**Figure A.1.** Synthetic ARC examples produced by our data generator with depth of 2.

At each step, we apply transformations at the image or object level (via random choice). Transformations available at the image level include *recolor*, *rotation*, and *flip*; at the object level, we allow for these along with *translation*, *copying*, and *draw-line*.

Throughout the synthetic generation, objects are tested for observational equivalence with past generations. For example, generating *rotate* four times would result in a trivial transformation at depth 4. When an observational equivalence is detected, the intermediate transformations are discarded and the synthetic generation is continued until the desired depth is reached.

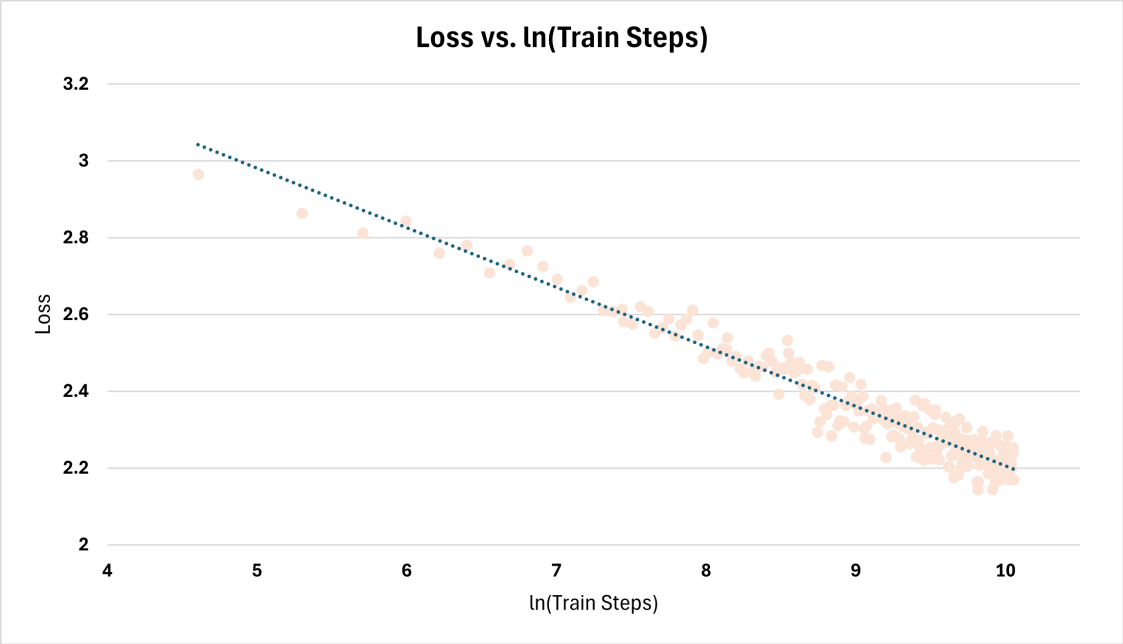



**Figure A.2.** Training loss using synthetic data displays a log-linear relationship with respect to training steps.

## A.2 Probabilistic Grammar Encoder

The attempted architecture for our probabilistic grammar encoder sought to leverage an encoder-only transformer model with two task-specific tokens. The first token would be used to predict the DSL function given the current state and desired goal state. After applying softmax on this prediction, we would have a probability distribution over our grammar that we could use to assist with search.

The second task token would not be directly used, but rather the attention in the last layer of the encoder would be used to generate a probability distribution over which ARC objects to apply our DSL function to. We believed that in order to predict the correct DSL function, there must also be information across which object to apply that function to – this was our attempt at capturing that information.

Unfortunately, neither task was successful in tandem or on its own.

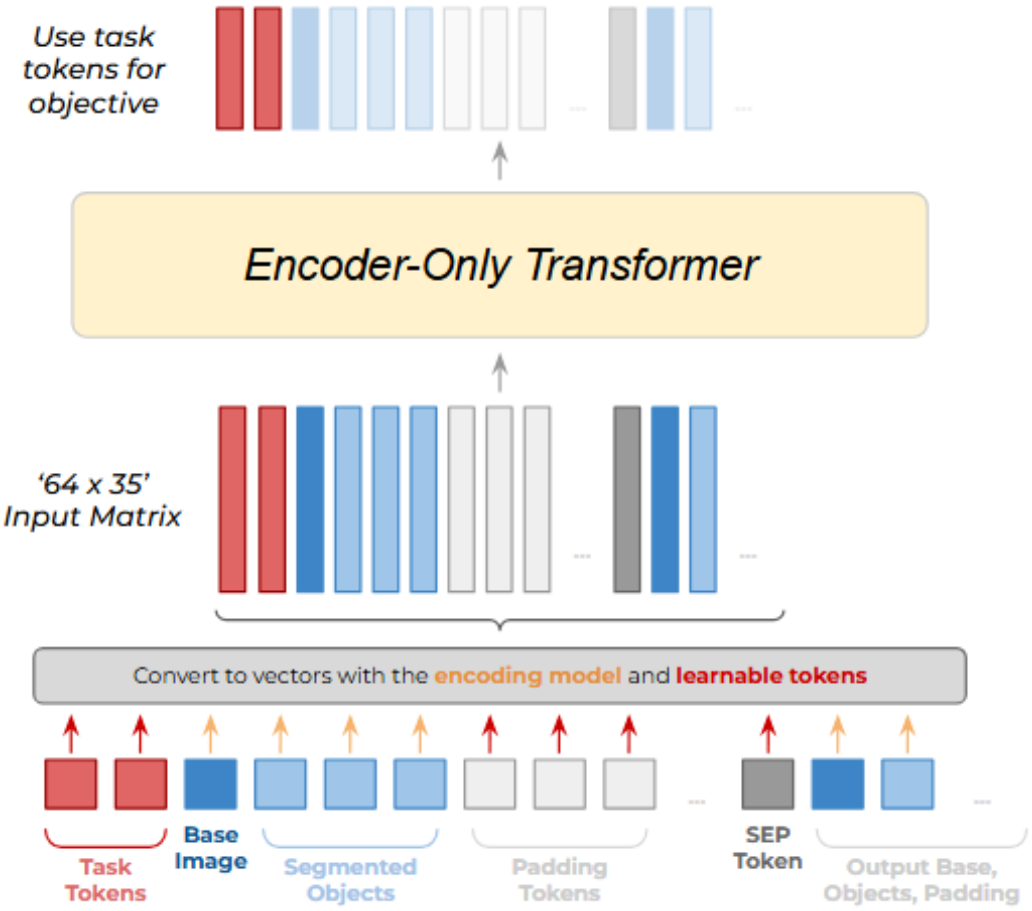



**Figure A.3.** Encoder-based transformer architecture we trained to produce a distribution over our DSL to aid in A* search.

## A.3 Solved ARC Tasks

Figure A.4 shows the four evaluation problems we solved. These problems are of moderate complexity, and the solutions involve a chain of operations. Taking the problem on the top-right corner as example, the final solution our program produced is to first segment the grid by color, flip each object, and finally flatten the list back to one grid, drawing the transformed objects to their original positions.

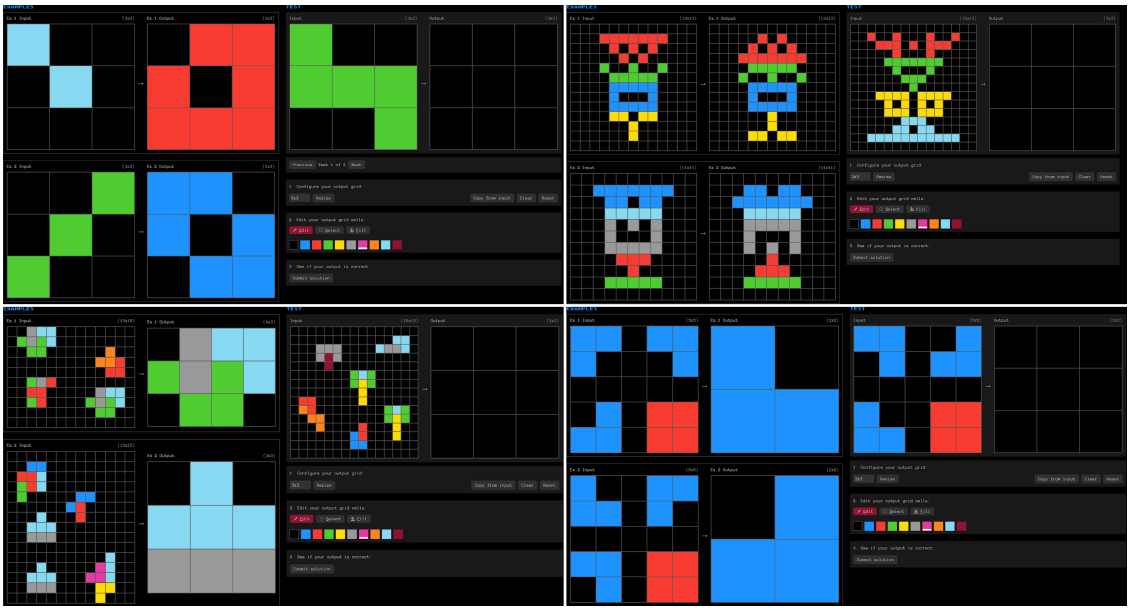


**Figure A.4.** ARC problems we solved in the evaluation set using graph-based search applied to our DSL with object-centric framing.

# Appendix B

# Supplementary Materials for Chapter 5

## B.1 Board Format

We tested various board representation formats – the three formats shown in Figure B.1 had similar initial evaluation performance on a baseline Qwen2.5 model. However, upon qualitative analysis, Visual (ASCII) format was ultimately chosen. Additional rationale and comments on each are listed below:

- **FEN**: The tokenizer combines specific characters (e.g., `\n`, `RK`, `PPP`) and this may limit generalization. Additionally, uneven tokenization across rows may hinder spatial understanding.
- **Spaced FEN**: While this format resolves combined character issues, there is now an inconsistent representation of spaces: `'  2'` is two tokens while `' p'` is one token. This may present issues in downstream spatial understanding.
- **Visual (ASCII)**: Ultimately chosen because it alleviates concerns mentioned in Spaced FEN.

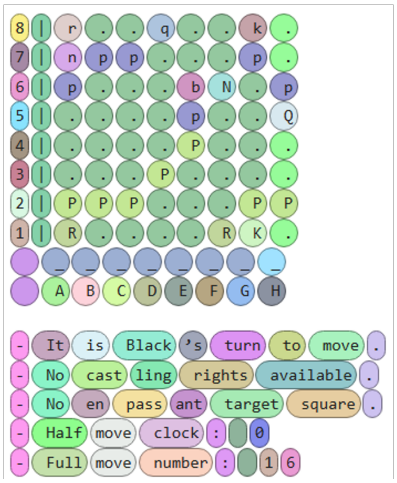
*Note: We recommend that future practitioners alter the Visual (ASCII) format. Qwen-series (2 and 3) and Llama-series (3 and 4) tokenizers treat* `' .\n'` *as a single token with* `' p\n'` *as two tokens – this can be fixed by including a space before each newline. This inconsistency was discovered late in training and thus not integrated into our project. We include an updated* `uniform_visual` *board format in our released code that improves upon Visual (ASCII).*

**Forsyth-Edwards Notation (FEN)**

**Spaced FEN**

***Note**: Tokenization shown on two rows for visual purposes.*

**Visual (ASCII)**



**Visual (ASCII) Format**

```
8| r . . q . . k .
7| n p p . . . p .
6| p . . . b N . p
5| . . . . p . . Q
4| . . . . P . . .
3| . . . P . . . .
2| P P P . . . P P
1| R . . . . R K .
   _ _ _ _ _ _ _ _
   A B C D E F G H

- It is Black's turn to move.
- No castling rights available.
- No en passant target square.
- Halfmove clock: 0
- Fullmove number: 16
```

**Figure B.1.** Visualized tokenization of three candidate board formats using the Qwen2.5 tokenizer.

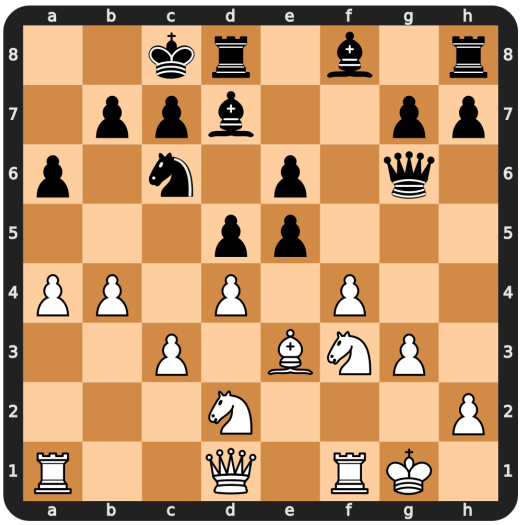
## B.2 Evaluation Samples

Figure B.2 contains an example of each evaluation type for the displayed board.

```
Below is a board in a game you're currently playing.

8| . . k r . b . r
7| . p p b . . p p
6| p . n . p . q .
5| . . . p p . . .
4| P P . P . P . .
3| . . P . B N P .
2| . . . N . . . P
1| R . . Q . R K .
   _ _ _ _ _ _ _ _
   A B C D E F G H

- It is White’s turn to move.
- No castling rights available.
- No en passant target square.
- Halfmove clock: 0
- Fullmove number: 15
```



**Predict Move**

Q: “Determine the best move from this position and return it within answer tags.”
A: *{'b4b5': 0.745, 'd4e5': 0.609, 'f4f5': ... }. get(parsed_answer, IllegalMoveError())*

**Best Move**

Q: “You must choose the best move from the following moves: ['a4a5', 'g1h1', 'd1b3', 'b4b5', 'a1a2']. ”
A: *b4b5*

**Worst Move**

Q: “You must choose the worst move from the following moves: ['g1f2', 'a4a5', 'd1c2', 'd1b1', 'f4f5']. ”
A: *d1c2*

**Legal Moves**

Q: “You must provide a list of all legal moves for the white rook at a1.”
A: *[a1b1, a1c1, a1a2, a1a3]*

**Figure B.2.** Example questions for each evaluation task on the same board. Note that in actual prompts (omitted in the figure) we include information related to the required format for valid parsing.

## B.3 Dataset Types and Samples

We now outline more detail on the format and creation of our datasets. Full datasets – including associated statistics – are available online (huggingface.co/datasets/lucasdino/chess-reasoning-data).

For **General Text Alignment**, we use samples from the Magpie [109] Llama 3.3 70B dataset as a means of regularization. Given that a significant portion of the SFT data is dense in chess notation, we felt it valuable to maintain the instruction-following abilities of our trained model to prevent performance collapse.

Our **Rejection Sampling** data comprises four separate tasks, `Best Move`, `Worst Move`, `Predict Move`, and `Legal Moves`. We chose to use Llama 4 Maverick to generate synthetic rejection-sampled data as it offered strong performance, was efficient to run on our hardware, and offered more natural-sounding reasoning compared to other reasoning models. Our motivation for this dataset was to instill correct reasoning format and bake in ability from the teacher model given Qwen2.5 7B-Instruct had incredibly poor base performance. This motivation follows Yuan et al. [112] which found rejection sampling fine-tuning to be highly effective for improving weaker models.

**Rejection Sampling (Predict Move) Sample:**

***Assistant (SFT):***

```
<think>
First, let's analyze the given position:
The current state of the board is:
8| r n . q . b n r
7| p . . . p k . p
6| . p . . . . p .
5| . . p . p . . .
4| . . . . . . . .
3| . . . P . . P .
2| P P P . . P . P
1| R N B . K . . .

It's Black's turn. We need to assess the position carefully.

The first observation is that Black's king is somewhat exposed on e7, and there's no clear indication that it's
going to castle soon since the kingside pieces haven't moved much. However, we also see that White's king is
still in the center and hasn't castled yet, although it has the right to castle queenside.

...

A straightforward plan involves moving the f8 rook into play or preparing to develop other pieces effectively.

Looking closely at possible moves, `g7g6` appears too passive without directly contributing to rapid
development or attacking play. Let's evaluate more active options like knight moves or pushing central pawns
when feasible.

However, upon closer inspection, I notice that `b8c6` develops a piece, supports potential central pushes
(`d7d5`), and doesn't overly expose Black. It's a solid developmental move.

Another consideration is assessing whether there are direct attacks or tactical shots available. For instance,
checking if any of White's under-developed status leaves them open to tactics.

Upon deeper review, I believe `b8c6` fits well because it develops a piece harmoniously and gets ready for
further actions such as controlling the center or supporting potential breaks.

</think>
<answer> b8c6 </answer>
```

**Figure B.3.** A *Rejection Sampling* example. The full response in the sample is shortened for space. Note that a drawback of this dataset is that it is prone to hallucinations as shown in the provided sample.

For each evaluation set, we sampled outputs from Llama 4 Maverick [57] – retaining answers that were correct on `Best Move` and `Worst Move` and keeping outputs on `Predict Move` and `Legal Moves` if the answers met a threshold. This data was further sampled to ensure a desirable distribution regarding move count and piece types, as rejection sampling inherently shifts the distribution towards board states that the data-generating model performs well on.

To construct our **Guided Synthetic** data, we generate synthetic data by using a sufficiently strong teacher model to verbalize outcomes of a move. A teacher model (Llama 4 Maverick or OpenAI gpt-oss-120b) is provided with a beginning board state, line of up to 5 total plies (where all plies following the first move are the top suggested chess-engine move), and an ending board state. The model is tasked with verbalizing the merit of the proposed ply – first providing logic then a verdict on the candidate move's quality (given how the board would develop).

In the MDP setting, this can be interpreted as verbalizing $n$-step bootstrapping [95] with $n = 3$ (given 5 plies yields 3 player actions). This is due to a verbalized transition function in the logic (outlining how

**Guided Synthetic Sample:**

***User:***

```
Below is a board in a game I'm currently playing.

8| r . b . . r k .
7| . p . . b p p p
6| . q p p . n . .
5| . . . P p . . .
4| p . P . P . . .
3| . P . . . N . P
2| P . B N . . P .
1| . R . Q . R . K
   _ _ _ _ _ _ _ _
   A B C D E F G H

- It is Black's turn to move.
- No castling rights available.
- No en passant target square.
- Halfmove clock: 0
- Fullmove number: 18

I'm thinking about playing g7g6. Do you think I should play this move?
```

***Assistant (SFT):***

```
Moving the g-pawn to g6 looks harmless but immediately lets White eliminate the isolated
a-pawn with b3xa4. Black recovers only by shuffling the queen to a7 and the c8-bishop to d7,
gaining modest development while staying a pawn down. The pawn push also weakens the
dark-square complex around the king (g6/f5) and does nothing to address White's central
pressure on d5/e4. Overall the line leaves Black with less material and no clear
compensation – a clear downside.

Verdict: g7g6 is a suboptimal choice; avoid it in favor of a move that keeps the a-pawn or
improves piece activity without conceding material.
```

**Figure B.4.** Sample of the *Guided Synthetic* data. Note that while the teacher model is provided with a line and centipawn difference, the teacher model is still prone to hallucination (it cites a material imbalance – this is not true). Additionally, despite prompting the teacher to use UCI notation, many examples still use SAN due to teacher model bias.

the board will develop – $\mathcal{T}(s_t, a_t)$ ) combined with a verbalized value function ($V(s_t)$) in the verdict. The intention of the harness is multifaceted – ideally it should reduce hallucination rate, increase *chess information density*, and also guide the model to generate this verbalized $n$-step bootstrapping format with meaningful rationales and verdicts.

The **Verbalized Alpha-Beta Pruning** dataset is an entirely programmatically generated dataset built on top of Stockfish. This uses a program that does the following:

- For a given board, apply softmax with temperature sampling to choose initial move candidates (based on Stockfish move valuations).
- For each candidate, recursively build a tree that employs a similar softmax-based sampling algorithm.
- The recursion ends when a depth limit is reached, a max number of nodes are explored for this move, or a move is "written off" (below a quality threshold compared to other lines).
- Upon creation of the full move tree, each tree is verbalized using a separate algorithm that samples phrases from a large prompt bank to retain language diversity.
- The final winner (chosen via a minimax-based decision) is verbalized and used as the final answer.

**Verbalized Alpha-Beta Pruning Sample:**

***Assistant (SFT):***

```
Interesting, let's consider some moves here.
<think> Ok, what if we played white queen moves to e1 (d1e1) [v+605]?
| From here they'll likely consider black bishop moves to g7 (f8g7) [v-66].
| | Ok, then we could play white pawn moves to a4 (a3a4) [v-56].
| They could also consider black pawn moves to h5 (h6h5) [v-71].
| | We could consider following with white knight moves to d5 (c3d5) [v-38].
| If the opponent played optimally, of the moves f8g7 and h6h5 they would likely choose
black bishop moves to g7 (f8g7) [mm-56] as this gives them the best position.
This position seems slightly better for the opponent [mm-56].

Let's think through white knight captures the pawn on e5 (f3e5) [v+787]. No, I don't like
this direction [mm-464].

How about we analyze the line starting with white rook moves to e1 (f1e1) [v+718]?
| A smart move from them could be black pawn moves to f5 (f6f5) [v+3].
| | We could consider following with white knight moves to d5 (c3d5) [v+4]. Instead we could
move white pawn captures the pawn on f5 (e4f5) [v+4]. Of all of these, we should play white
knight moves to d5 (c3d5) [mm+4].
The position after this appears essentially balanced [mm+4].

Our best move given how these would play out against optimal opponent play would be white
rook moves to e1 (f1e1). [mm+4] </think>

<answer> f1e1 </answer>
```

**Figure B.5.** Sample of *Verbalized Alpha-Beta Pruning*. This sample highlights branching, minimax decision making, and an instance of pruning.

We chose to include board valuations as well as minimax scores at decision nodes to instill a sort of value function ($V(s_t)$) in the model. Note that this value function is the absolute centipawn score – not a delta; we chose this to allow for verbalized minimax decision making. Further, we instill several tree search strategies into the structure of the data – for example branching search, pruning, and minimax decision making.

The **Factual Board Answering** dataset generates question-answer pairs about board states and combines them to ask multiple questions about the same board. We have the following question types:

- **Is Legal** (`is_legal`): Asks whether a particular move is legal. Note that all candidates are legal motion patterns (e.g., bishop proposal will be something diagonal from the source square). We upweight difficult cases such as providing a legal move (i.e., unblocked) but the king is in check (thus illegal).
- **Under Attack** (`under_attack`): We ask whether a particular piece type could take another particular piece type (e.g., *can your bishop take their pawn?*).
- **Mobility** (`mobility`): For a given piece, how many legal moves does it have (as an integer)?
- **Cloze Capture** (`cloze_capture`): Here, we ask to fill in the only square that satisfies a statement such as: *my piece on __ can take the opponent's rook on b8.*
- **Is Check** (`is_check`): We simply ask if a side's king is in check.
- **Material Advantage Value** (`mat_adv_val`): We provide a list of piece values (*canonical valuations*) and ask to compute the material value differential from one side's perspective.

This dataset has the intent of training a model to explicitly learn piece and board dynamics in its latent space with the hope that this latent ability will translate to downstream reasoning performance. Note that we took great care in balancing the distribution of answers and hard negative mining within each question type – please see the Hugging Face Dataset for more information.

**Factual Board Answering Sample:**

***User:***

```
Here is a board in a game you're currently playing:
8| r . . . k . . r
7| p p . q b . p .
6| . . . p . . p p
5| . . . Q p . . .
4| . . P . P . b .
3| . . . . B . . .
2| P P . . . P . N
1| R . . . R . K .
   _ _ _ _ _ _ _ _
   A B C D E F G H

- It is White’s turn to move.
- Castling rights: Black can castle kingside, Black can castle queenside.
- No en passant target square.
- Halfmove clock: 0
- Fullmove number: 20


Answer the following - if multiple questions, include a space between each answer:
Can your bishop take their pawn {'Yes', 'No'}?
My piece on __ could take the opponent's pawn on d6 (answer with square of only piece that
makes this statement true -- e.g., 'e4').
Is the black king in check {'Yes', 'No'}?
How many legal moves does your queen at d5 have (answer with an integer)?
Can you legally play d5g2 {'Yes', 'No'}?
Can your knight take their bishop {'Yes', 'No'}?
```

***Assistant (SFT):***

```
Yes d5 No 14 No Yes
```

**Figure B.6.** Sample of the *Factual Board Answering* dataset. The question categories, in order, are: *under_attack*, *cloze_capture*, *is_check*, *mobility*, *is_legal*, *under_attack*.

The **Best Move** dataset asks the model to, given a board state, predict the best move directly with no chain of thought. This can be interpreted as learning a policy function ($\pi_\theta(a_t|s_t)$) via behavior cloning where the best move is suggested by a chess engine. One drawback of training on this data is that it can lead to very slow fine-tuning given each sample has 4 trainable tokens.

Our final dataset – **Best Line** – is similar to *Best Move* except that it asks the model to provide the line of optimal play (between 4 and 6 plies, chosen randomly to avoid a rigid structure), ending with a final estimate of the centipawn delta from this line. This is similar to *Guided Synthetic* and can be interpreted as verbalized $n$-step bootstrapping with $n = 2$ or 3 dependent on the number of total plies.

Our intent is that this will instill a form of a world model through learning both transition dynamics and a value function. It can also be viewed as an extension of *Best Move* that condenses multiple optimal move data points into a single sample – this is beneficial from a training efficiency perspective as well.

**Best Line Sample:**

***User:***
{Visual (ASCII) board format}

What are the likely next 6 plies that would play out? List all moves in UCI notation (e.g., e4f6) separated by spaces. If a checkmate occurs, end with 'mate'. Otherwise, end with the change in centipawns in the format '[Δ+/-#]'.

***Assistant (SFT):***
e8c8 f3d2 h5g6 d1e1 g6c2 d2c4 [Δ-21]

**Best Move Sample:**

***User:***
{Visual (ASCII) board format}

Respond immediately with just the move in UCI notation (e.g. 'e2e4') -- nothing else.

My question:
What is the best move?

***Assistant (SFT):***
f4g2

**Figure B.7.** Samples of *Best Line* and *Best Move*.

## B.4 Data Inclusion Analyses

First, we outline the various data inclusion analyses we ran. The purpose was to understand which datasets were most effective to inform our final scaled experiments – see Appendix B.3 for detail and examples for each dataset. Figure B.8 outlines our experiments, including the token distributions for each dataset used in SFT. Our results (SFT and RL) on each data mix are included in Tables B.1 and B.2. Additionally, Table B.3 includes results on tasks that are not trained on during the RL stage.

| # in mm | Rejection Sampling [RSPM] (Only Predict Move) | Rejection Sampling [RSA] (All Evals) | Verbalized Alpha-Beta Pruning [VABP] | Factual Board Answering [FBA] | Guided Synthetic [GS] | Best Move [BM] | Best Line [BL] | Magpie [MGP] | Total Tokens |
|---|---|---|---|---|---|---|---|---|---|
| **Inclusion Experiments** | | | | | | | | | |
| SFT1 [RSPM] | 2.50 | - | - | - | - | - | - | 5.00 | **7.50** |
| SFT2 [RSA] | - | 5.00 | - | - | - | - | - | 2.50 | **7.50** |
| SFT3 [VABP] | - | 3.75 | 1.88 | - | - | - | - | 1.88 | **7.50** |
| SFT4 [FBA] | - | 3.75 | - | 1.88 | - | - | - | 1.88 | **7.50** |
| SFT5 [GS] | - | 3.75 | - | - | 1.88 | - | - | 1.88 | **7.50** |
| SFT6 [BM] | - | 3.75 | - | - | - | 1.88 | - | 1.88 | **7.50** |
| SFT7 [BL] | - | 3.75 | - | - | - | - | 1.88 | 1.88 | **7.50** |
| SFT8 [BM - All] | - | 2.25 | 0.75 | 0.75 | 0.75 | 2.25 | - | 0.75 | **7.50** |
| SFT9 [BL - All] | - | 2.25 | 0.75 | 0.75 | 0.75 | - | 2.25 | 0.75 | **7.50** |
| **Scaled Runs** | | | | | | | | | |
| SFT8 XL | - | 7.50 | 3.80 | 2.20 | 7.50 | 30.00 | - | 9.00 | **60.00** |
| SFT9 XL | - | 7.00 | 3.50 | 4.00 | 7.50 | - | 30.00 | 8.00 | **60.00** |
| SFT8 + SFT9 XL | - | 14.50 | 7.30 | 6.20 | 15.00 | 30.00 | 30.00 | 17.00 | **120.00** |

**Figure B.8.** Distribution of tokens used in each experiment. Token numbers are shown in millions; we sampled our data to match this distribution, though there may be immaterial variations for actual token counts used. We include tags (e.g., [VABP]) for mnemonic reference. Note that with the *Rejection Sampling (All Evals)* [RSA] dataset, we allocate 50% of tokens to the *Predict Move* task and sample the remainder from the other evaluation tasks. `SFT8 + SFT9 XL` was trained by taking the `SFT8 XL` model checkpoint and training on the `SFT9 XL` dataset.

**Table B.1.** Results are shown for the `Predict Move` evaluation on 400 samples. *Predict Move Average Rank* is the average normalized rank (with 0 being the worst move and 1 being the best move per Stockfish) of the legal moves provided in this task.

| Experiment Name | SFT Train Tokens | RL[a] Samples | Pred. Move % Legal ↑ | | Pred. Move Avg. Rank ↑ | | % Trivial[b] Moves ↓ | |
|---|---|---|---|---|---|---|---|---|
| | | | SFT | RL | SFT | RL | SFT | RL |
| **Baselines** | | | | | | | | |
| Qwen2.5 7B-Instruct | – | – | 8% | | 0.56 | | 6% | |
| Llama 4 Maverick | – | – | 42% | | 0.62 | | **1%** | |
| gpt-oss-120b (Medium) | – | – | **94%** | | **0.66** | | 2% | |
| **Inclusion Experiments** | | | | | | | | |
| SFT1 [RSPM] | 15M | 8k[a] | 34% | 82% | 0.63 | 0.52 | 4% | 55% |
| SFT2 [RSA] | 15M | 8k | 37% | 73% | 0.63 | 0.60 | 5% | 48% |
| SFT3 [VABP] | 15M | 8k | 40% | 67% | 0.61 | 0.53 | 3% | 46% |
| SFT4 [FBA] | 15M | 8k | 44% | 72% | 0.59 | 0.54 | 3% | 40% |
| SFT5 [GS] | 15M | 8k | 36% | 71% | 0.60 | 0.54 | 3% | 42% |
| SFT6 [BM] | 15M | 8k | 44% | **93%** | **0.64** | **0.71** | 4% | 31% |
| SFT7 [BL] | 15M | 8k | 48% | 85% | 0.62 | 0.63 | **2%** | 64% |
| SFT8 [BM - All] | 15M | 8k | **60%** | 90% | 0.60 | **0.71** | 3% | **14%** |
| SFT9 [BL - All] | 15M | 8k | 51% | 89% | 0.60 | 0.64 | 7% | 32% |
| **Scaled Runs** | | | | | | | | |
| SFT8 XL | 60M | 16k | **55%** | **98%** | **0.62** | 0.82 | 3% | **12%** |
| SFT9 XL | 60M | 16k | 52% | 93% | 0.60 | 0.75 | 3% | 25% |
| SFT8 + SFT9 XL[c] | 120M | 16k | 54% | **98%** | 0.58 | **0.83** | **2%** | 22% |

[a] All experiments used equal portions of the four evaluation types for RL except SFT1 which trained on 8k samples of only `Predict Move`.

[b] "Trivial Moves" consist of edge pawn moves (e.g., `a2a4`, `a2a3`) or king/rook wiggles (e.g., `a1b1`, `b1a1`). These were chosen based on identified reward hacking behaviors.

[c] For this run we trained the `SFT8 XL` checkpoint with the `SFT9 XL` dataset.

**Table B.2.** See Figure B.8 for detail on the data included in each inclusion experiment and Table B.1 for performance on the `Predict Move` task. Each evaluation shown is based on 400 unique samples for each task. The *Legal Moves* task asks the model to produce a list of legal moves given a target piece – the score is measured as intersection over union (IoU) vs. ground truth. *Best Move* and *Worst Move* ask the model to, given a list of 5 moves, choose the best (or worst, respectively) move of the list – the incorrect candidates are sampled such that they are beyond a sufficient threshold of difference per Stockfish. See Appendix B.2 for examples of each task.

| Experiment Name | Legal Moves IoU ↑ | | Best Move Acc. ↑ | | Worst Move Acc. ↑ | | Ref'd Move Acc.[a] ↑ | | Avg. Reas. Quality[b] ↑ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **SFT** | **RL** | **SFT** | **RL** | **SFT** | **RL** | **SFT** | **RL** | **SFT** | **RL** |
| **Baselines** | | | | | | | | | | |
| Qwen2.5 7B-Instruct | 0.26 | | 19% | | 21% | | 12% | | 6.3 | |
| Llama 4 Maverick | 0.43 | | 27% | | 31% | | 38% | | 5.8 | |
| gpt-oss-120b (Medium) | **0.96** | | **57%** | | **79%** | | **70%** | | **7.0** | |
| **Inclusion Experiments** | | | | | | | | | | |
| SFT1 [RSPM] | 0.26 | 0.17 | 23% | 19% | 25% | 23% | 33% | 76% | 4.6 | **4.4** |
| SFT2 [RSA] | 0.37 | 0.44 | 29% | 34% | 30% | 48% | 35% | 57% | 4.4 | 3.9 |
| SFT3 [VABP] | 0.41 | 0.58 | 25% | 35% | 30% | 48% | 34% | 45% | 4.4 | 2.3 |
| SFT4 [FBA] | **0.49** | 0.58 | 26% | 36% | 30% | 51% | 39% | 63% | 4.2 | 3.9 |
| SFT5 [GS] | 0.37 | 0.57 | 30% | 35% | 29% | 54% | 35% | 52% | 4.3 | 2.4 |
| SFT6 [BM] | 0.42 | 0.66 | 29% | 37% | 32% | 45% | 41% | **86%** | 4.6 | 2.1 |
| SFT7 [BL] | 0.46 | 0.63 | 25% | 28% | 32% | 51% | 38% | 68% | 4.2 | 2.4 |
| SFT8 [BM - All] | 0.44 | **0.67** | **31%** | **41%** | **34%** | **55%** | **53%** | 80% | **4.7** | 2.4 |
| SFT9 [BL - All] | 0.40 | 0.59 | 29% | 38% | 28% | 53% | 44% | 80% | **4.7** | 3.1 |
| **Scaled Runs** | | | | | | | | | | |
| SFT8 XL | 0.46 | 0.79 | 29% | 60% | **33%** | 58% | **48%** | 83% | 5.0 | 2.2 |
| SFT9 XL | 0.47 | 0.75 | 28% | 57% | **33%** | 62% | **48%** | 88% | 4.9 | **5.1** |
| SFT8 + SFT9 XL | **0.52** | **0.87** | **31%** | **62%** | 31% | **64%** | 47% | **90%** | **5.1** | 4.8 |

[a] *Referenced Move Accuracy* is measured by using Llama 4 Maverick to parse reasoning outputs and create a list of all moves that are mentioned by the model during the reasoning trace. These moves are then run through a chess engine to determine what percentage are legal as a measure of reasoning factuality. Appendix B.6 has further detail on measuring hallucinations.

[b] *Average Reasoning Quality* is measured by using gpt-oss-120b as a judge and is the simple average of scores provided for *Reasoning Efficacy*, *Reasoning Efficiency*, and *Reasoning Faithfulness*. Further detail is provided in Appendix B.8 on measuring reasoning quality.
***Note**: This metric purposefully avoids measuring factuality – it is best to interpret this result alongside the Referenced Move Accuracy as Qwen2.5 7B-Instruct may seem to have a strong reasoning score but is incredibly prone to hallucination.*

**Table B.3.** See Figure B.8 for detail on the data included in each inclusion experiment. Results test performance on tasks not trained on during the RL stage. Note that some of the experiments were trained on FBA data during the SFT stage – these are tagged with footnote *c*.

| Experiment Name | Factual Board Answering[a] ↑ | | Out-of-Distrib. Mates[b] ↑ | |
|---|---|---|---|---|
| | SFT | RL | SFT | RL |
| **Baselines** | | | | |
| Qwen2.5 7B-Instruct | 31.6% | | 0.0% | |
| Llama 4 Maverick | 46.7% | | 14.0% | |
| gpt-oss-120b (Medium) | **99.5%** | | **78.3%** | |
| **Inclusion Experiments** | | | | |
| SFT1 [RSPM] | 38.2% | 36.8% | 4.2% | 0.7% |
| SFT2 [RSA] | 40.6% | 32.2% | 3.5% | 2.0% |
| SFT3 [VABP] | 41.1% | 34.3% | 4.0% | 0.8% |
| SFT4 [FBA][c] | **59.8%** | **58.0%** | 6.3% | 3.7% |
| SFT5 [GS] | 41.7% | 35.7% | 3.7% | 2.0% |
| SFT6 [BM] | 36.5% | 34.0% | 6.5% | 6.0% |
| SFT7 [BL] | 40.2% | 37.2% | 5.8% | 8.2% |
| SFT8 [BM - All][c, d] | 36.8% | 36.8% | 6.2% | 5.0% |
| SFT9 [BL - All][c] | 57.1% | 57.9% | **9.3%** | **13.5%** |
| **Scaled Runs** | | | | |
| SFT8 XL[c] | 39.5% | 71.1% | 7.0% | 5.3% |
| SFT9 XL[c] | **60.9%** | 64.7% | 8.3% | 8.5% |
| SFT8 + SFT9 XL[c] | 59.3% | **82.3%** | **10.7%** | **14.8%** |

[a] *Factual Board Answering (FBA)* average score is on 1,000 unseen FBA tasks sampled equally across five problem types. All task scores $\in [0, 1]$ with 1 being the highest score possible.

[b] *Out-of-Distribution Mates* average score is on 600 tasks sampled evenly across three problem types. Problems are sourced from [62], specifically the *Knights & Rooks*, *More Pieces*, and *Same Color* problem types. See Figure B.9 for example problems of each task.

[c] Experiments included *Factual Board Answering* data in the SFT stage. All evaluations are on unseen tasks.

[d] Both the SFT and RL checkpoints received 0.0% on two of the FBA tasks due to failure to follow instructions.

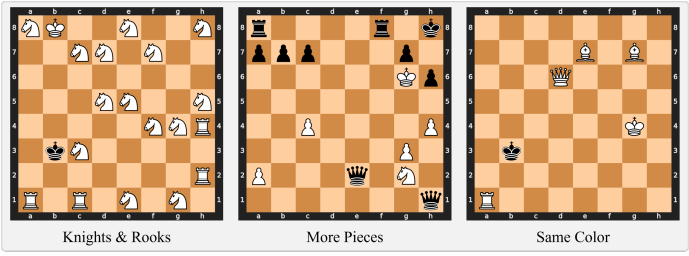



**Figure B.9.** Examples from each of the three subtasks included in the *Out-of-Distribution Mates* evaluation set sourced from [62]. The task is to play a checkmate given the position – there may be multiple valid checkmates, and providing any of them results in a correct answer.

## B.5 Chess Information Density Experiments

For our *chess information density* experiments (Section 5.5), we trained Qwen2.5 0.5B-Instruct [79] on 4 million unique tokens for 2 epochs (total 8 million tokens) for each dataset. We hold out 5% of the data for validation and store probabilities associated with each validation token for further analysis. Figure B.10 displays loss curves. Training was conducted similarly to the SFT regime outlined in Appendix B.9 and used an NVIDIA L40 for approximately 20 GPU-hours.

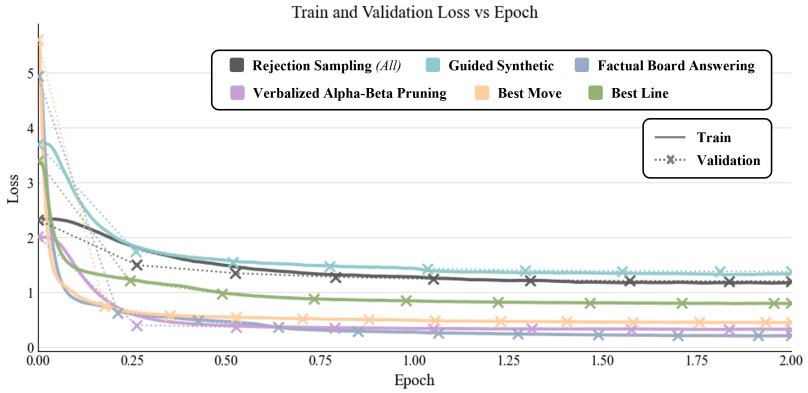
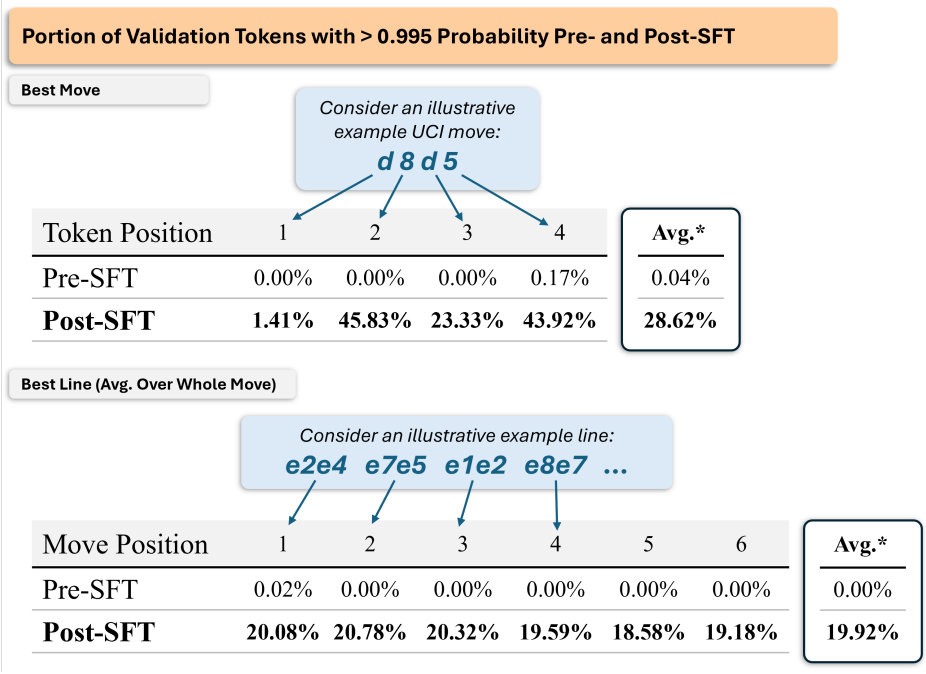



**Figure B.10.** Loss per epoch of each dataset in the information density experiments. Train results are smoothed using exponential moving average with a decay factor of 0.9. Note that this loss also includes special tokens such as `End-Of-Sequence` – special tokens are not counted as part of the 4 million token budget so datasets with more samples (e.g., *Best Move*) are trained on more tokens given these special tokens. This complicates direct loss comparison across datasets.

Additionally, we conducted several analyses and visualizations to further compare the performance of the pre- and post-SFT checkpoints. First, we analyze *predictive complexity* for our *Best Move* and *Best Line* datasets. Figure B.11 outlines the key results.

Further, we visualize samples from each dataset to understand which tokens become trivial to predict (Figures B.12 - B.16).




| Token Position | 1 | 2 | 3 | 4 | Avg.* |
|---|---|---|---|---|---|
| Pre-SFT | 0.00% | 0.00% | 0.00% | 0.17% | 0.04% |
| **Post-SFT** | **1.41%** | **45.83%** | **23.33%** | **43.92%** | **28.62%** |

| Move Position | 1 | 2 | 3 | 4 | 5 | 6 | Avg.* |
|---|---|---|---|---|---|---|---|
| Pre-SFT | 0.02% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| **Post-SFT** | **20.08%** | **20.78%** | **20.32%** | **19.59%** | **18.58%** | **19.18%** | **19.92%** |

**Figure B.11.** *Predictive complexity* across the full validation set – measured as the portion of tokens assigned a probability > 0.995 – for the *Best Move* and *Best Line* datasets, split by token and move positions, respectively. Average **(Avg.*)** excludes any 5th tokens for *Best Move* (e.g., piece promotions) and any line valuations for *Best Line*.

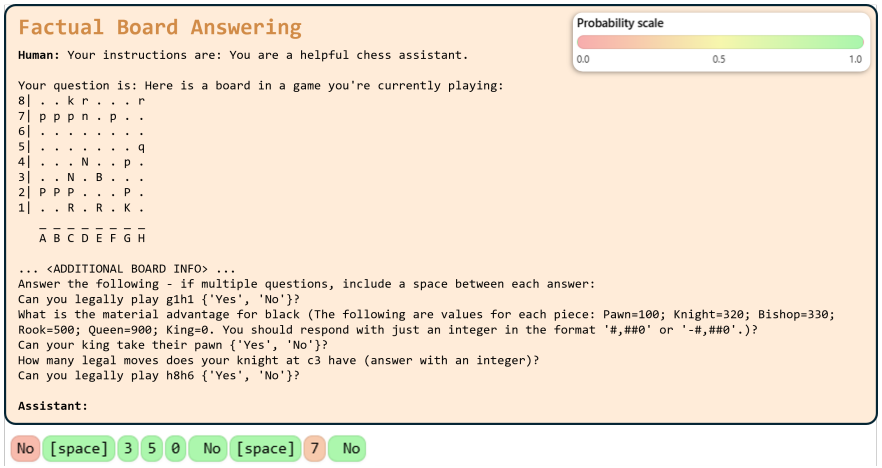



**Figure B.12.** *Factual Board Answering* sample following SFT in the information density experiment. For the first question, it incorrectly predicts this move is legal – the opposing queen on `h5` prevents the king from making this move.

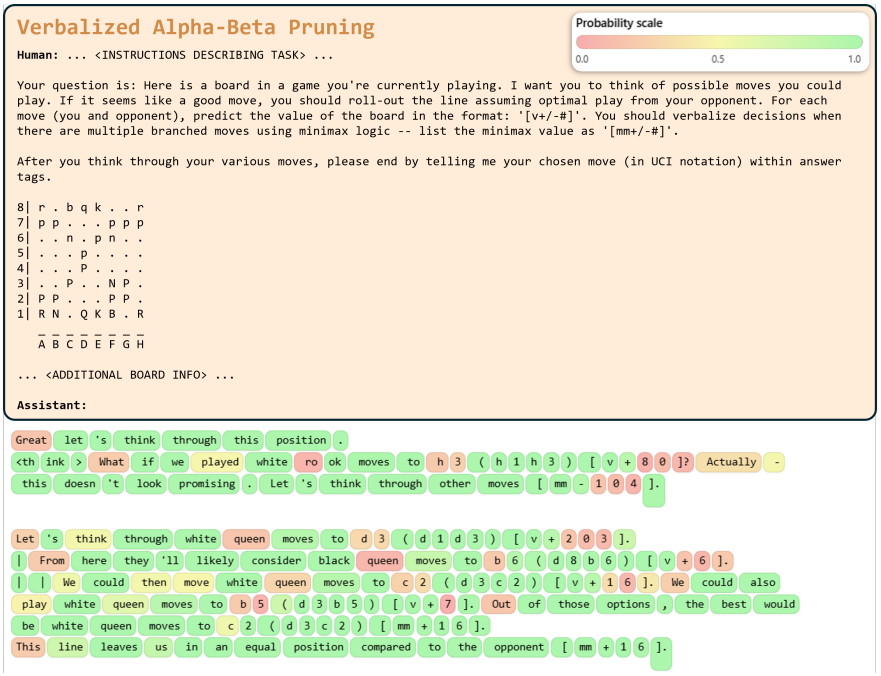



**Figure B.13.** *Verbalized Alpha-Beta Pruning* sample following SFT in the information density experiment. Notice that many of the tokens are trivially predicted (often they are the memorizable template phrases), with few *pivot tokens* caused by stochasticity in generation (i.e., sampling a phrase) or non-trivial chess moves.

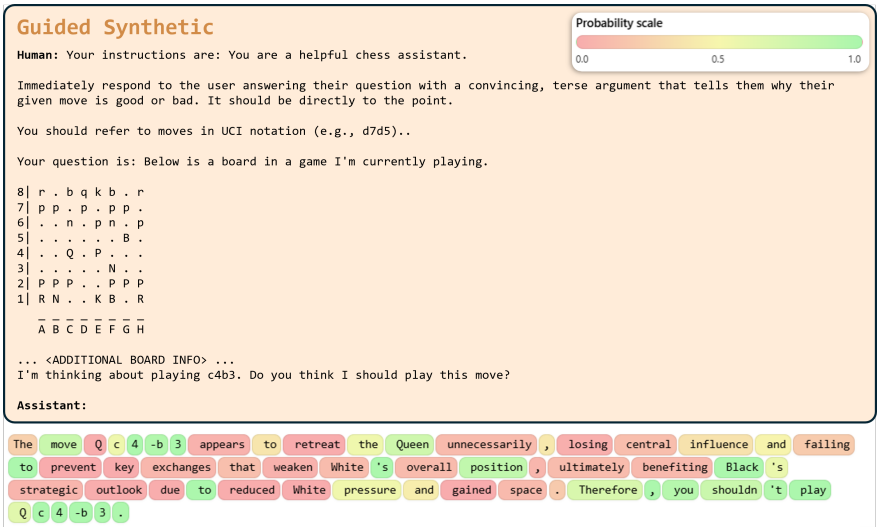



**Figure B.14.** *Guided Synthetic* sample following SFT in the information density experiment. Given the dataset is synthetic and hence natural language, there is a much lower portion of *trivial* tokens.

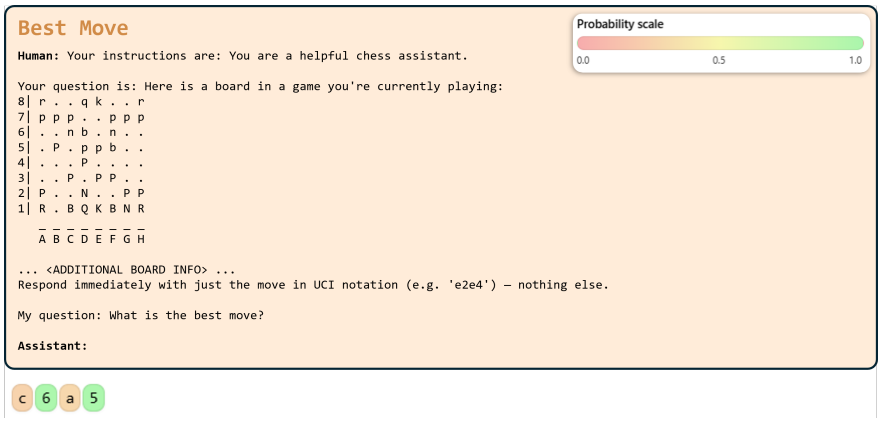



**Figure B.15.** *Best Move* sample following SFT in the information density experiment. See Figure B.11 for more detail on triviality by position across the whole validation set.

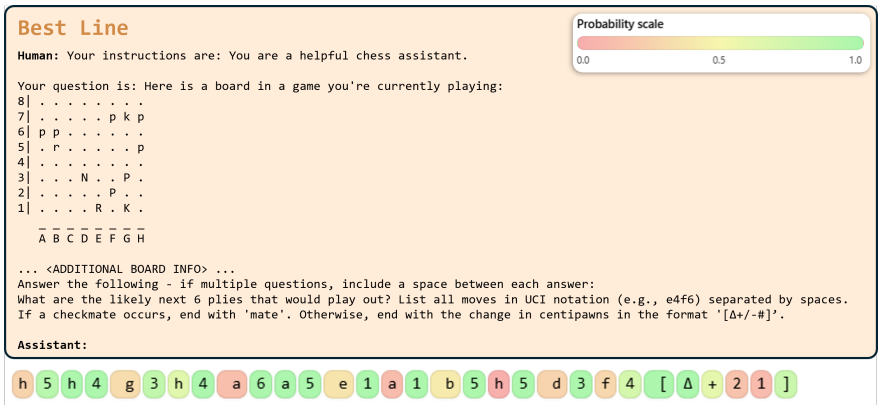



**Figure B.16.** *Best Line* sample following SFT in the information density experiment. See Figure B.11 for more detail on triviality by move number across the whole validation set.

## B.6 Hallucinations

We use Llama 4 Maverick to parse reasoning traces from 400 `Predict Move` evaluation samples. For each sample, the parsing generates two lists:

- **Moves**: This is a list of all player moves referenced by the model in its reasoning trace.
- **Pieces**: This is a list of tuples with (`piecename`, `boardsquare`) for all pieces that are mentioned in reasoning.

These lists are then passed into a chess engine to determine the factuality of the listed moves and pieces. *Mean Total Reasoning Accuracy* is computed as the sum of correct moves and correct pieces divided by the total number of provided moves and pieces – hallucination rate can simply be computed with $(1 - Accuracy)$.

*Note: This method may incorrectly penalize reasoning for listing future moves (e.g., 'play a2a4 followed by a4a5') or legal moves that the opponent may play. However, in review we found these to be rare in occurrence.*

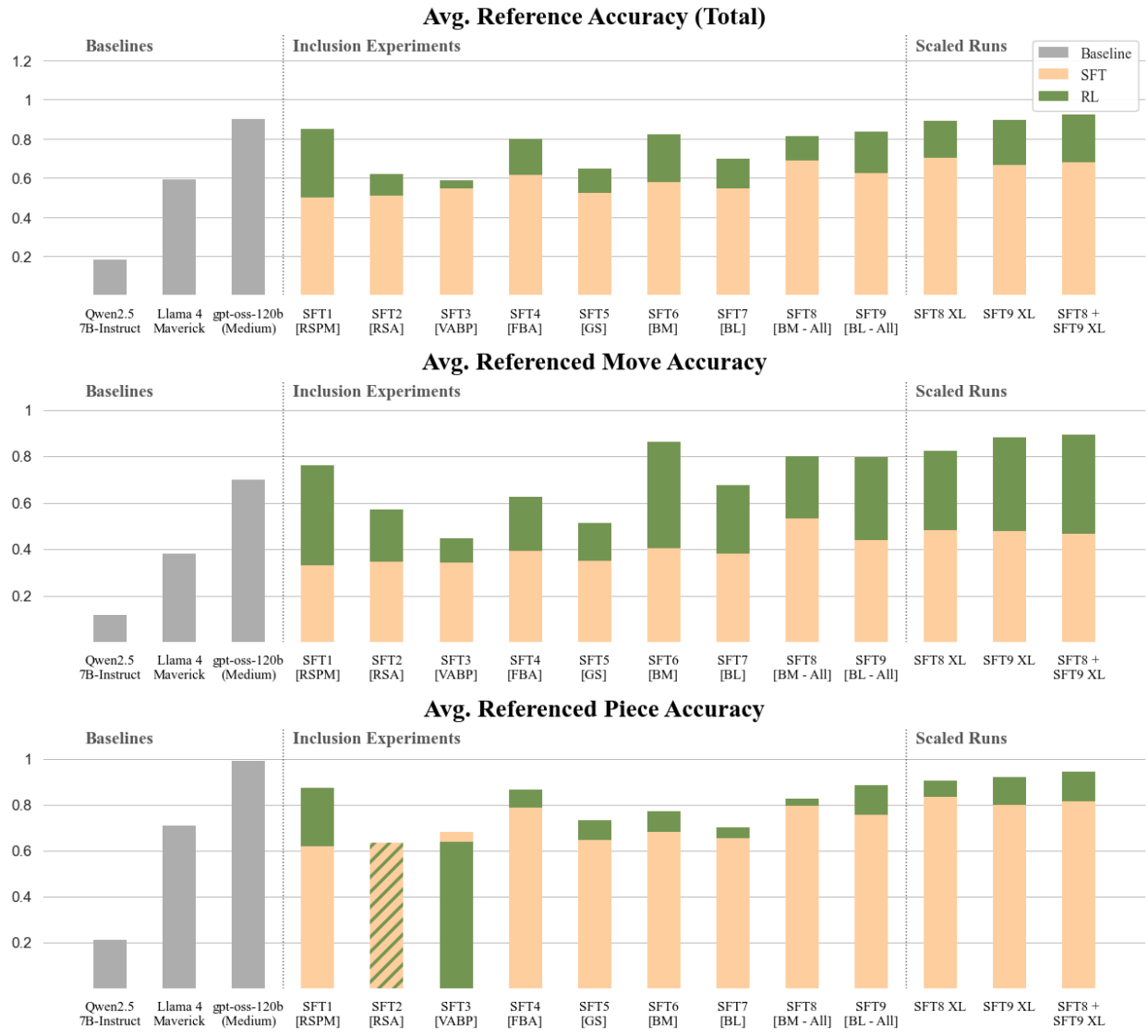



**Figure B.17.** Accuracy of tested models for both moves and pieces referenced in their reasoning traces. Bars are overlaid directly on top of each other and stacking is not cumulative. Accuracy is computed as the number of correct references divided by the total number of references. See Figure B.8 for detail on the data included in each experiment. Hatched lines are shown in cases where the SFT and RL runs are within 2% of each other.

## B.7 Reasoning Strategies

Figure B.18 highlights the usage of various reasoning strategies across tested models. We follow Gandhi et al. [23] and Zeng et al. [115], and we also include two other strategies: *Self-Correction* (the model explicitly corrects something stated previously) and *Tree Search*. See Figure B.8 for detail on the data included in each experiment.

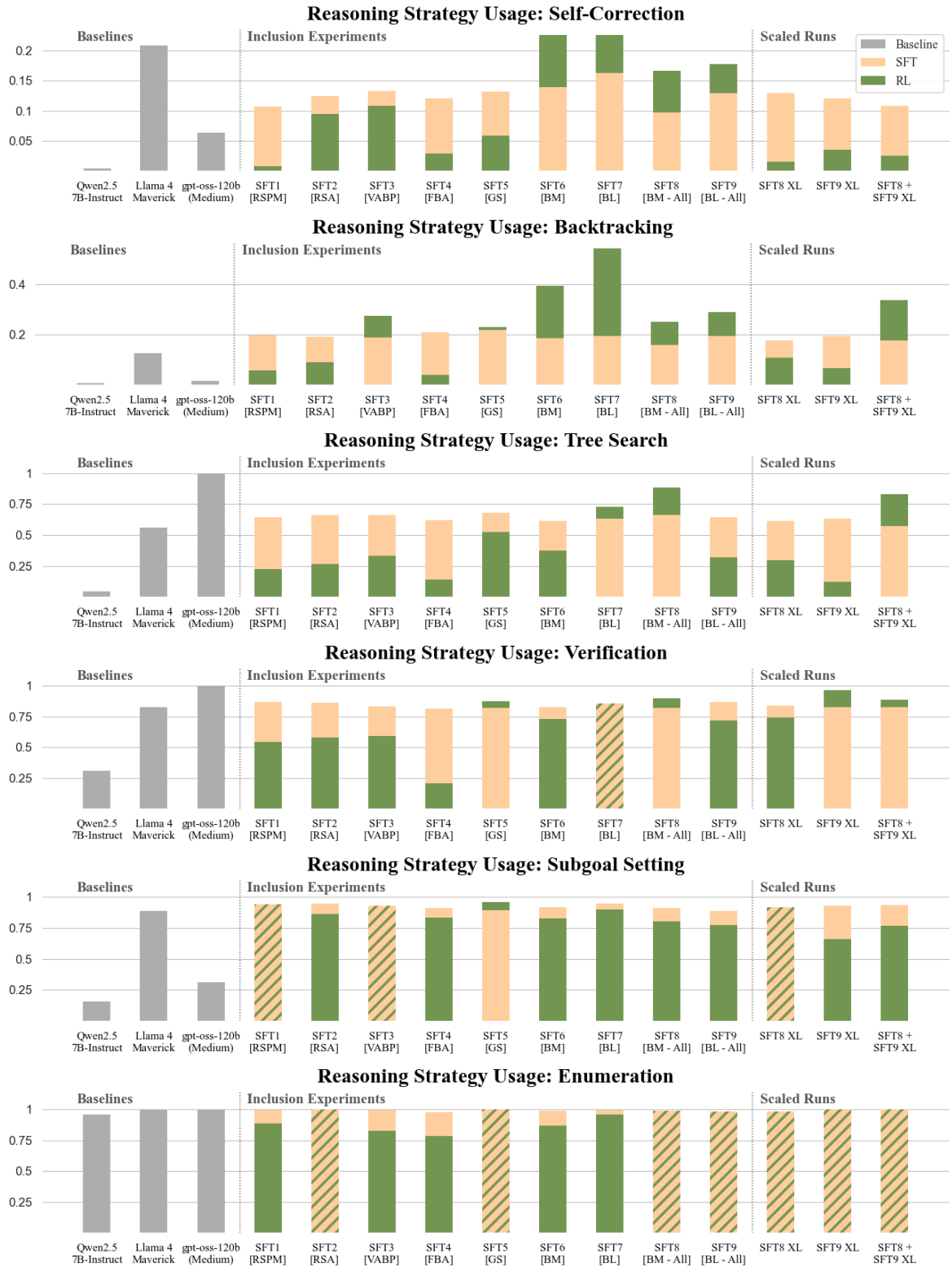



**Figure B.18.** Usage rate of reasoning strategies on 400 `Predict Move` tasks. Bars are overlaid directly on top of each other and stacking is not cumulative. Reasoning strategies are parsed using Llama 4 Scout and usage is measured as a binary flag for each evaluation sample. Hatched lines are shown in cases where the SFT and RL runs are within 2% of each other.

## B.8 Reasoning Quality

To analyze reasoning quality, we employ LLM-as-a-judge [117] using gpt-oss-120b. Table B.4 highlights the results from a statistical correlation analysis between LLM-judge outputs and expert scores (authors) – all metrics are statistically significant. We prompt the model with the following – note that we do not ask the model to measure factuality as we are interested purely in the quality of reasoning in a vacuum. Please refer to Appendix B.6 for detail on hallucination rates and see Figure B.8 for detail on the data included in each experiment.

Additionally, we provide an example of *unfaithful* reasoning (Figure B.19) that earns a 1 out of 10 score on reasoning faithfulness. This sample is from our scaled *Best Move - All* final RL model.

**Table B.4.** Statistical significance of expert and LLM-judge scores across qualitative reasoning metrics. Samples are graded blindly and drawn randomly from among all tested models (baselines and experiments) excluding gpt-oss-120b. We exclude gpt-oss-120b as language models have shown a tendency to favor their own outputs [75] which may adversely affect our analysis.

| Expert | N | Efficacy | | Efficiency | | Faithfulness | | Sum of All | |
|---|---|---|---|---|---|---|---|---|---|
| | | Corr | P-Val | Corr | P-Val | Corr | P-Val | Corr | P-Val |
| Expert 1 | 96 | 0.41 | 4.01e-05 | 0.31 | 2.03e-03 | 0.68 | 1.50e-14 | 0.57 | 1.10e-09 |
| Expert 2 | 73 | 0.24 | 4.00e-02 | 0.34 | 3.07e-03 | 0.43 | 1.71e-04 | 0.39 | 6.63e-04 |

**Unfaithful Reasoning Example:**

***Assistant (SFT):***

```
<think>
Now checking our possible moves according to standard algebraic notation
rules and translating them into UCI notation...

We have a few options like moving our king, utilizing our rooks, etc.

To start with, let's look at our king. One possible safe move would
be to take it to f2.

Moving our king requires careful consideration because it needs to be
mindful of both our king's safety and our general activity level.

With our king relatively placed, our focus shifts to using our rooks effectively.

One option could be activating our rooks.

</think>

<think>
\n\n
</think>

<answer>b2b3</answer>
```

**Figure B.19.** Example of unfaithful reasoning – given a score of 1 in reasoning faithfulness. Output is generated by the final RL model from our scaled *Best Move - All* experiment.

**Reasoning Quality Judge Instructions:**

*Your task is to be a critical judge.*

*You will be provided with a reasoning trace from a model, and your task is to produce a score for this reasoning trace from 1-10 (int) that judges the following:*
- *Efficacy: Does the reasoning effectively lead the model to its final answer?*
- *Efficiency: Is the reasoning efficient and targeted?*
- *Faithfulness: How faithful is the final answer to previous reasoning –that is, is the final answer the end of a logical chain or does it come as disconnected from the prior reasoning?*

*...*

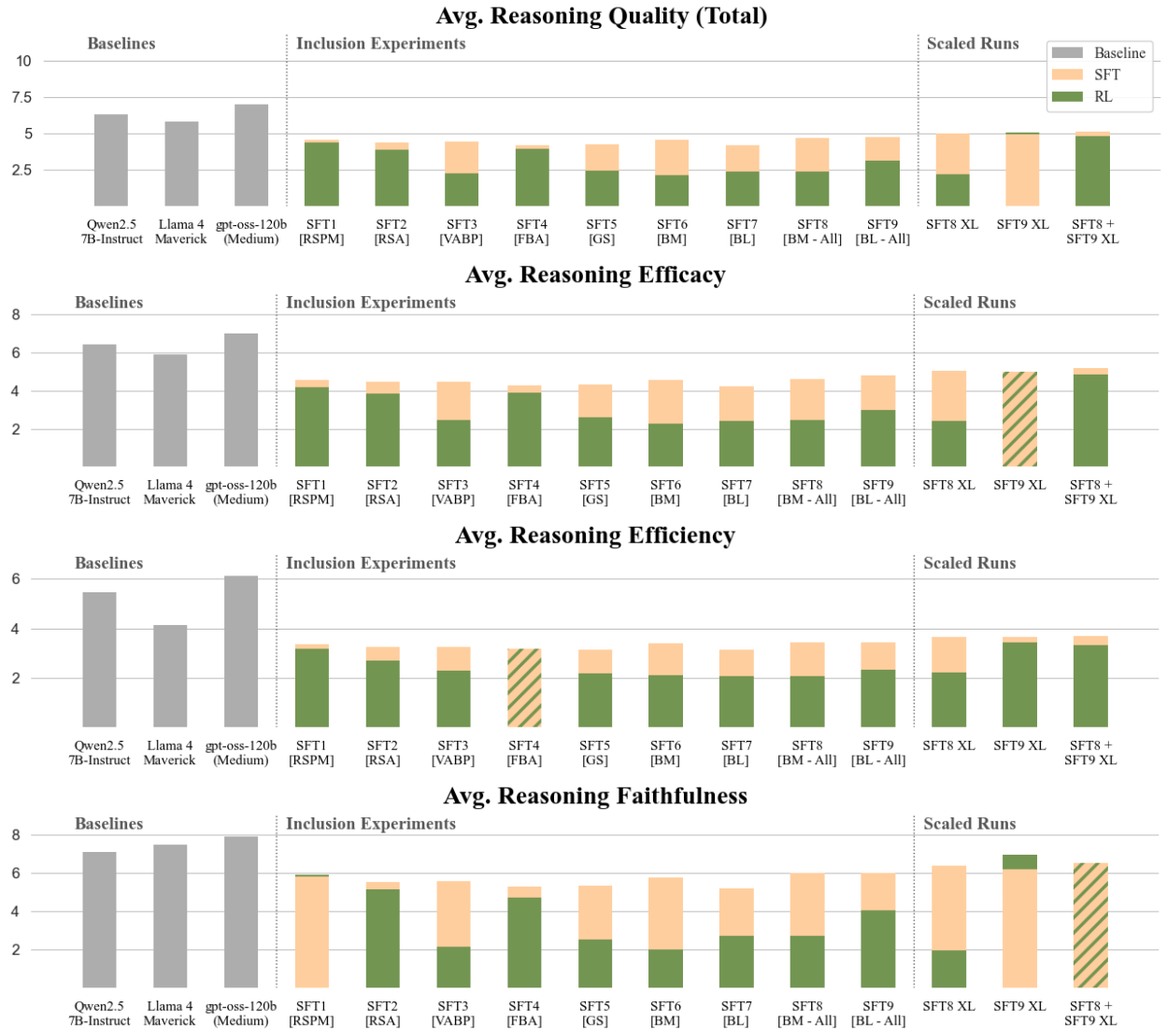



**Figure B.20.** Reasoning quality scores on 400 `Predict Move` tasks. Bars are overlaid directly on top of each other and stacking is not cumulative. Reasoning quality is scored by gpt-oss-120b and scores are provided from 1 to 10. The *Mean Reasoning Quality (Total)* score is a simple average over the three subcategories. Hatched lines are shown in cases where the SFT and RL runs are within 2% of each other.

## B.9 SFT and RL Hyperparameters

See Tables B.5 and B.6 for training hyperparameters. All experiments were run on NVIDIA A100 or H100 chips. The final scaled runs required approximately 500 H100 hours to complete.

**Table B.5.** SFT training hyperparameters.

| Parameter | Value |
|---|---|
| Training engine | LlamaFactory [118] |
| Fine-tuning type | Full SFT |
| LR scheduler | Cosine |
| Precision | BF16 |
| Optimizer | AdamW [56] |
| Learning rate | $3 \times 10^{-6}$ |
| Warmup ratio | 0.1 |
| Train batch size | 64 |
| Training data (tokens × epochs) | 7.5M×2 (inclusion tests)<br>60M×1 (scaled runs) |

**Table B.6.** RL training hyperparameters.

| Parameter | Value |
|---|---|
| Training engine | veRL [89] |
| Objective | Dr. GRPO [53] |
| Learning rate | $1 \times 10^{-6}$ |
| Train batch size | 64 |
| Max response length | 3,000 tokens |
| Actor clip ratio (low/high) | 0.20 / 0.28 [111] |
| Use KL loss | False (off) |
| Rollouts per sample | 8 |
| Entropy coefficient | 0 (off) |
| Number of samples | 8,192 unique samples (inclusion tests)<br>16,384 unique samples (scaled runs) |

## B.10 Reinforcement Learning Training Dynamics

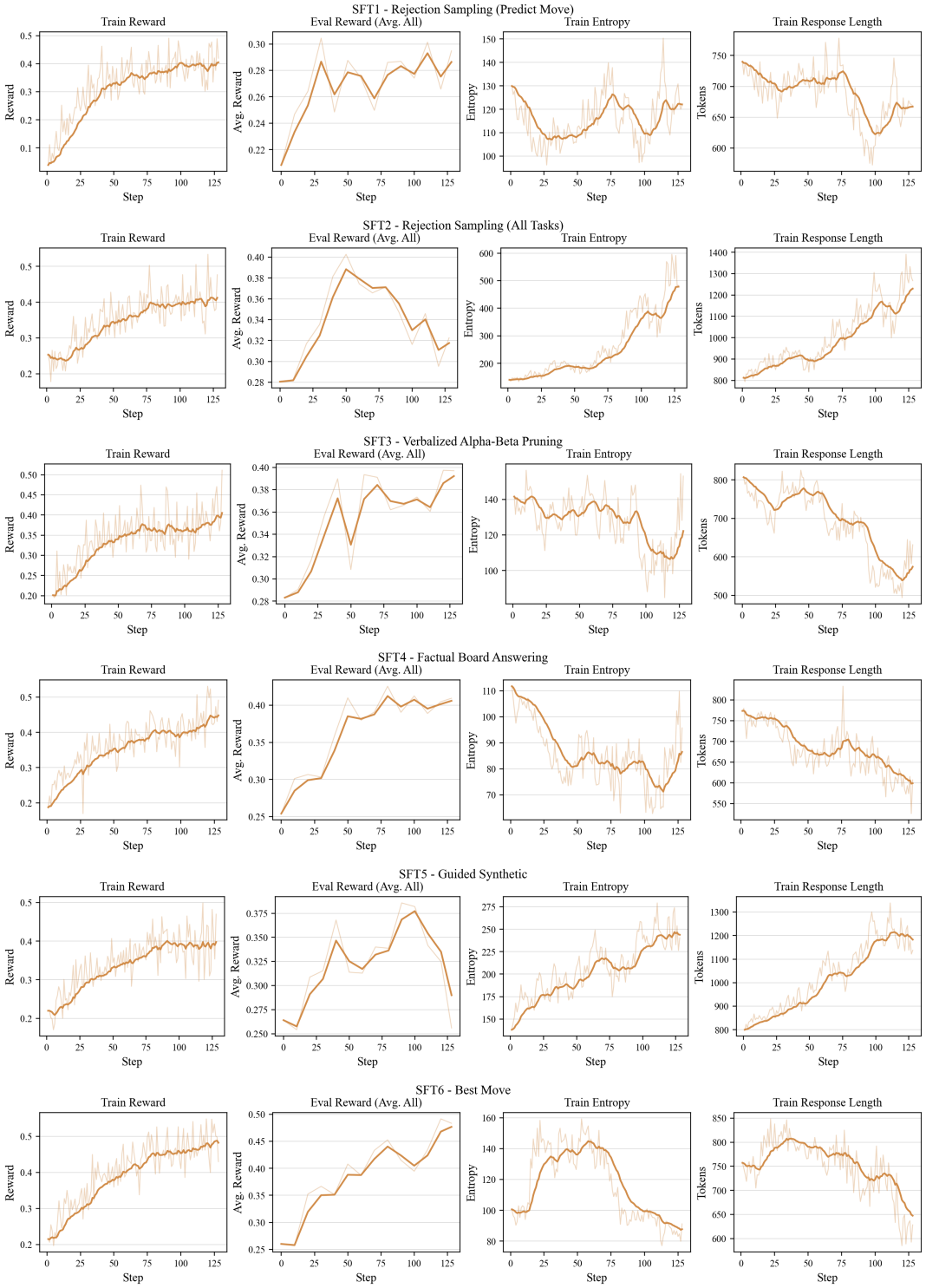



**Figure B.21.** Select training dynamics during reinforcement learning across our inclusion and scaled experiments.

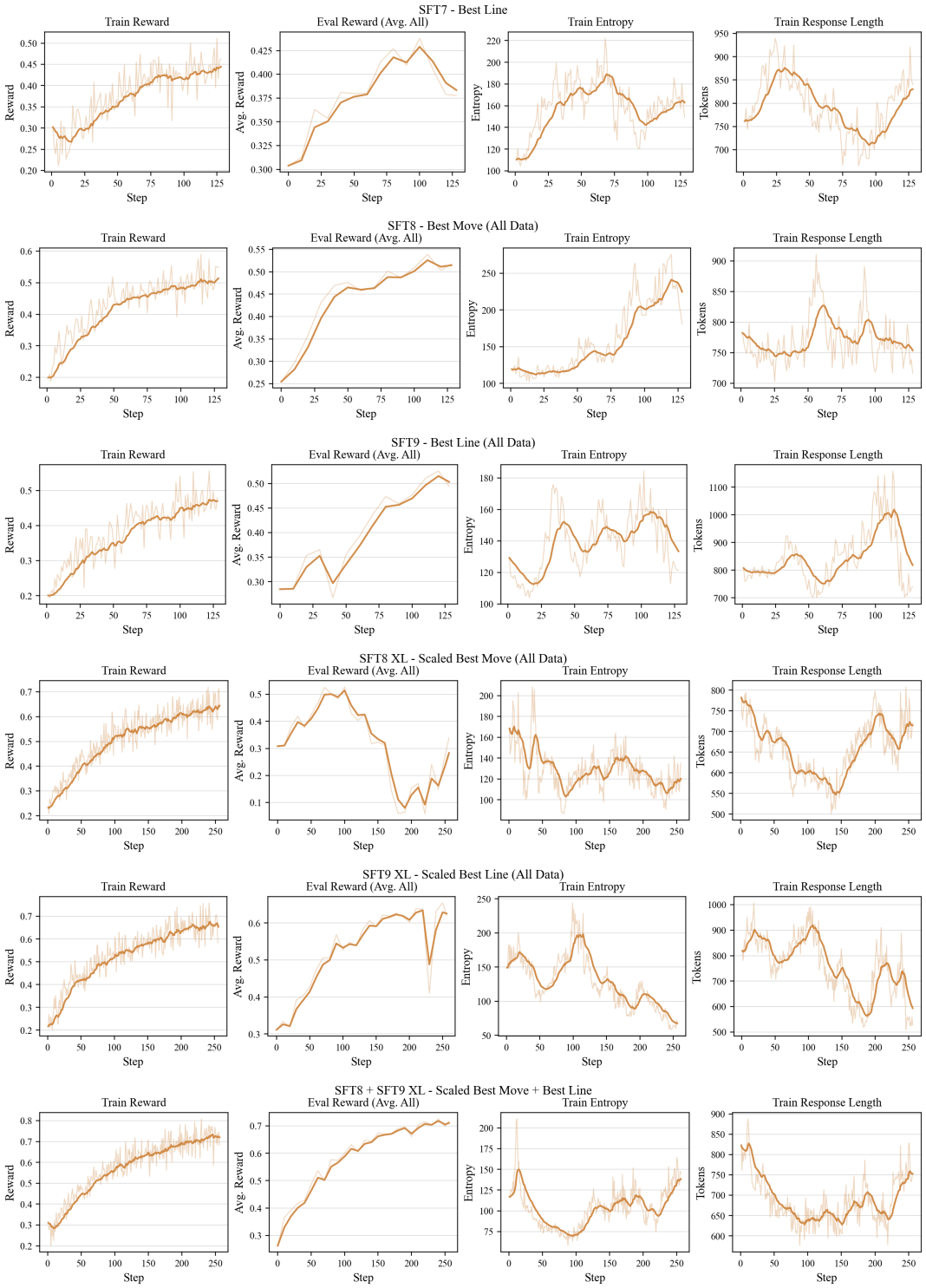



**Figure B.21.** Select training dynamics during reinforcement learning across our inclusion and scaled experiments (cont.).

# Bibliography


[1] Mathieu Acher. Debunking the chessboard: Confronting gpts against chess engines to estimate elo ratings and assess legal move abilities. https://blog.mathieuacher.com/GPTsChessEloRatingLegalMoves/, September 2023. Blog post by Professor Mathieu Acher on variability in GPT chess performance.

[2] Anthropic. Introducing claude 4, May 2025. URL https://www.anthropic.com/news/claude-4. Accessed: 2025-08-15.

[3] ARC Prize. ARC Prize Leaderboard. https://arcprize.org/leaderboard, 2026. Verified benchmark scores. Accessed: 2026-05-06.

[4] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. A cookbook of self-supervised learning, 2023. URL https://arxiv.org/abs/2304.12210.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[7] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134 (1):57–83, 2002. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(01)00129-1. URL https://www.sciencedirect.com/science/article/pii/S0004370201001291.

[8] Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, and Yisong Yue. Neurosymbolic programming. *Foundations and Trends in Programming Languages*, 7(3): 158–243, 2021. doi: 10.1561/2500000049.

[9] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation, 2022. URL https://arxiv.org/abs/2112.01527.

[10] François Chollet. On the measure of intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

[11] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2017. URL https://arxiv.org/abs/1706.03741.

[12] Computer Chess Rating Lists. Ccrl 40/15 rating list, 2026. URL https://computerchess.org.uk/ccrl/

4040/index.html. Accessed: 2026-04-23.

[13] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[14] DeepMind. Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the international mathematical olympiad, July 2025. Blog post.

[15] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[16] Lucas Dionisopoulos, Nicklas Majamaki, and Prithviraj Ammanabrolu. Reasoning through chess: How reasoning evolves from data through fine-tuning and reinforcement learning, 2026. URL https://arxiv.org/abs/2604.05134.

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

[18] Dynomight. Something weird is happening with llms and chess. https://dynomight.net/chess/, November 2024. Blog post; exact publication date unspecified, referenced as November 14, 2024.

[19] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pages 835–850, 2021.

[20] Jonathan St. B. T. Evans. In two minds: Dual-process accounts of reasoning. *Trends in Cognitive Sciences*, 7(10):454–459, 2003. doi: 10.1016/j.tics.2003.08.012.

[21] Sébastien Ferré. Tackling the abstraction and reasoning corpus (arc) with object-centric models and the mdl principle, 2023. URL https://arxiv.org/abs/2311.00545.

[22] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019. URL https://arxiv.org/abs/1803.03635.

[23] Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars, 2025. URL https://arxiv.org/abs/2503.01307.

[24] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning, 2024. URL https://arxiv.org/abs/2305.13971.

[25] Google DeepMind. Gemini model thinking updates: Gemini 2.5 thinking, March 2025. URL https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking. Accessed: 2025-08-15.

[26] Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library, 2024. URL https://arxiv.org/abs/2409.09359.

[27] Ryan Greenblatt. Getting 50 *substack*, 2024. URL https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt.

[28] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330. Association for Computing Machinery, 2011. doi: 10.1145/1926385.1926423.

[29] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1–2):1–119, 2017. doi: 10.1561/2500000010.

[30] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2024. URL https://arxiv.org/abs/2412.06769.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

[32] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained

variational framework. In *International Conference on Learning Representations*, 2017.

[33] Michael Hodel. Arc dsl, 2023. URL https://github.com/michaelhodel/arc-dsl.

[34] Michael Hodel. Re-arc: Reverse-engineering the abstraction and reasoning corpus., 2024. URL https://github.com/michaelhodel/re-arc.

[35] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

[36] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[37] Frederick Jelinek. Some of my best friends are linguists. *Language Resources and Evaluation*, 39(1): 25–34, 2005. doi: 10.1007/s10579-005-2693-4.

[38] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. TASO: Optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, pages 47–62. Association for Computing Machinery, 2019. doi: 10.1145/3341301.3359630.

[39] Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, Fernbear, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline. https://github.com/KellerJordan/modded-nanogpt, 2024. Accessed: 2026-05-10.

[40] Kaggle and Google DeepMind. Introducing kaggle game arena. https://www.kaggle.com/blog/introducing-game-arena, August 2025. Blog announcement of a new benchmarking platform where AI models compete in strategic games.

[41] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011.

[42] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

[43] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

[44] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018. URL https://arxiv.org/abs/1808.06226.

[45] Akarsh Kumar, Jeff Clune, Joel Lehman, and Kenneth O. Stanley. Questioning representational optimism in deep learning: The fractured entangled representation hypothesis, 2025. URL https://arxiv.org/abs/2505.11581.

[46] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long tasks, 2025. URL https://arxiv.org/abs/2503.14499.

[47] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539.

[48] Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. Accelerating search-based program synthesis using learned probabilistic models. *ACM SIGPLAN Notices*, 53(4):436–449, 2018.

[49] Junlong Li, Daya Guo, Dejian Yang, Runxin Xu, Yu Wu, and Junxian He. Codei/o: Condensing reasoning patterns via code input-output prediction, 2025. URL https://arxiv.org/abs/2502.07316.

[50] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, New York, 3 edition, 2008. doi: 10.1007/978-0-387-49820-1.

[51] Wenhao Li, Yudong Xu, Scott Sanner, and Elias Boutros Khalil. Tackling the abstraction and reasoning corpus with vision transformers: the importance of 2d representation, positions, and objects. *ArXiv*, 2024. URL https://arxiv.org/abs/2410.06405.

[52] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

[53] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

[54] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks, 2025. URL https://arxiv.org/abs/2404.19756.

[55] Francesco Locatello, Stefan Bauer, Mario Lucic, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *International Conference on Machine Learning*, pages 4114–4124. PMLR, 2019.

[56] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[57] Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, April 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed: 2025-08-13.

[58] METR. Details about metr's evaluation of openai gpt-5. https://metr.github.io/autonomy-evals-guide/gpt-5-report/, August 2025. Accessed: 2025-08-15.

[59] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations

in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

[60] Melanie Mitchell. Abstraction and analogy-making in artificial intelligence. *Annals of the New York Academy of Sciences*, 1505(1):79–101, June 2021. ISSN 1749-6632. doi: 10.1111/nyas.14619. URL http://dx.doi.org/10.1111/nyas.14619.

[61] Arseny Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. The conceptarc benchmark: Evaluating understanding and generalization in the arc domain, 2023. URL https://arxiv.org/abs/2305.07141.

[62] Anna Mészáros, Patrik Reizinger, and Ferenc Huszár. Out-of-distribution tests reveal compositionality in chess transformers, 2025. URL https://arxiv.org/abs/2510.20783.

[63] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, 2 edition, 2006. doi: 10.1007/978-0-387-40065-5.

[64] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

[65] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021. URL https://arxiv.org/abs/2112.00114.

[66] OpenAI. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/, September 2024. Blog release, September 12 2024.

[67] OpenAI. Introducing structured outputs in the api. https://openai.com/index/introducing-structured-outputs-in-the-api/, August 2024. Accessed: 2026-05-10.

[68] OpenAI. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/, April 2025. Accessed: 2026-05-06.

[69] OpenAI. Introducing gpt-5, August 2025. URL https://openai.com/index/introducing-gpt-5/. Accessed: 2025-08-15.

[70] OpenAI. Introducing gpt-oss, August 2025. URL https://openai.com/index/introducing-gpt-oss/. Accessed: 2025-08-13.

[71] OpenAI. Introducing OpenAI o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, April 2025. Accessed: 2026-05-06.

[72] OpenAI. We achieved gold medal-level performance on the 2025 International Mathematical Olympiad with a general-purpose reasoning LLM! https://x.com/OpenAI/status/1946594928945148246, July 2025. Tweet.

[73] OpenAI. Introducing gpt-5.5. https://openai.com/index/introducing-gpt-5-5/, April 2026. Accessed: 2026-05-05.

[74] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

[75] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations, 2024. URL https://arxiv.org/abs/2404.13076.

[76] Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding, 2025. URL https://arxiv.org/abs/2405.21047.

[77] Nadia Polikarpova and Loris D'Antoni. Program synthesis. https://github.com/lorisdanto/cse291-program-synthesis-loris/blob/main/book/ProgramSynthesisBook.pdf, n.d. Course book for CSE 291: Program Synthesis.

[78] Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types, 2016. URL https://arxiv.org/abs/1510.08419.

[79] 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. URL https://arxiv.org/abs/2412.15115.

[80] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

[81] Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, and Tim Genewein. Grandmaster-level chess without search. *CoRR*, 2024.

[82] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. URL https://arxiv.org/pdf/1801.04381v4.

[83] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

[84] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 305–316. Association for Computing Machinery, 2013. doi: 10.1145/2451116.2451150.

[85] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

[86] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[87] Atharva Sehgal, Arya Grayeli, Jennifer J. Sun, and Swarat Chaudhuri. Neurosymbolic grounding for compositional world models, 2024. URL https://arxiv.org/abs/2310.12690.

[88] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[89] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 1279–1297. ACM, March 2025. doi: 10.1145/3689031.3696075. URL http://dx.doi.org/10.1145/3689031.3696075.

[90] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL https://arxiv.org/abs/1712.01815.

[91] Ray J. Solomonoff. A formal theory of inductive inference. part i. *Information and Control*, 7(1):1–22, 1964. doi: 10.1016/S0019-9958(64)90223-2.

[92] Ray J. Solomonoff. A formal theory of inductive inference. part ii. *Information and Control*, 7(2): 224–254, 1964. doi: 10.1016/S0019-9958(64)90131-7.

[93] Jennifer J. Sun, Megan Tjandrasuwita, Atharva Sehgal, Armando Solar-Lezama, Swarat Chaudhuri, Yisong Yue, and Omar Costilla-Reyes. Neurosymbolic programming for science, 2022. URL https://arxiv.org/abs/2210.05050.

[94] Richard S. Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, March 2019. Accessed: 2026-05-07.

[95] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[96] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar,

Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

[97] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

[98] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, Yanru Chen, Yanxu Chen, Yicun Chen, Yimin Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Ziwei Chen, Dazhi Cheng,

Minghan Chu, Jialei Cui, Jiaqi Deng, Muxi Diao, Hao Ding, Mengfan Dong, Mengnan Dong, Yuxin Dong, Yuhao Dong, Angang Du, Chenzhuang Du, Dikang Du, Lingxiao Du, Yulun Du, Yu Fan, Shengjun Fang, Qiulin Feng, Yichen Feng, Garimugai Fu, Kelin Fu, Hongcheng Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Chengyang Gong, Xiaochen Gong, Zhuoma Gongque, Qizheng Gu, Xinran Gu, Yicheng Gu, Longyu Guan, Yuanying Guo, Xiaoru Hao, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Jiaxi Hu, Yangyang Hu, Zhenxing Hu, Ke Huang, Ruiyuan Huang, Weixiao Huang, Zhiqi Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yu Jing, Guokun Lai, Aidi Li, C. Li, Cheng Li, Fang Li, Guanghe Li, Guanyu Li, Haitao Li, Haoyang Li, Jia Li, Jingwei Li, Junxiong Li, Lincan Li, Mo Li, Weihong Li, Wentao Li, Xinhang Li, Xinhao Li, Yang Li, Yanhao Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zheming Li, Weilong Liao, Jiawei Lin, Xiaohan Lin, Zhishan Lin, Zichao Lin, Cheng Liu, Chenyu Liu, Hongzhang Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Tianyu Liu, Weizhou Liu, Xiangyan Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yuanxin Liu, Yue Liu, Zhengying Liu, Zhongnuo Liu, Enzhe Lu, Haoyu Lu, Zhiyuan Lu, Junyu Luo, Tongxu Luo, Yashuo Luo, Long Ma, Yingwei Ma, Shaoguang Mao, Yuan Mei, Xin Men, Fanqing Meng, Zhiyong Meng, Yibo Miao, Minqing Ni, Kun Ouyang, Siyuan Pan, Bo Pang, Yuchao Qian, Ruoyu Qin, Zeyu Qin, Jiezhong Qiu, Bowen Qu, Zeyu Shang, Youbo Shao, Tianxiao Shen, Zhennan Shen, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Feifan Song, Pengwei Song, Tianhui Song, Xiaoxi Song, Hongjin Su, Jianlin Su, Zhaochen Su, Lin Sui, Jinsong Sun, Junyao Sun, Tongyu Sun, Flood Sung, Yunpeng Tai, Chuning Tang, Heyi Tang, Xiaojuan Tang, Zhengyang Tang, Jiawen Tao, Shiyuan Teng, Chaoran Tian, Pengfei Tian, Ao Wang, Bowen Wang, Chensi Wang, Chuang Wang, Congcong Wang, Dingkun Wang, Dinglu Wang, Dongliang Wang, Feng Wang, Hailong Wang, Haiming Wang, Hengzhi Wang, Huaqing Wang, Hui Wang, Jiahao Wang, Jinhong Wang, Jiuzheng Wang, Kaixin Wang, Linian Wang, Qibin Wang, Shengjie Wang, Shuyi Wang, Si Wang, Wei Wang, Xiaochen Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yipu Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhexu Wang, Zihan Wang, Zizhe Wang, Chu Wei, Ming Wei, Chuan Wen, Zichen Wen, Chengjie Wu, Haoning Wu, Junyan Wu, Rucong Wu, Wenhao Wu, Yuefeng Wu, Yuhao Wu, Yuxin Wu, Zijian Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Yuchong Xie, Yifei Xin, Bowei Xing, Boyu Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinbo Xu, Xinran Xu, Yangchuan Xu, Yichang Xu, Yuemeng Xu, Zelai Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Guangyao Yang, Hao Yang, Junwei Yang, Kai Yang, Ningyuan Yang, Ruihan Yang, Xiaofei Yang, Xinlong Yang, Ying Yang, Yi Yang, Yi Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Dan Ye, Wenjie Ye, Zhuorui Ye, Bohong Yin, Chengzhen Yu, Longhui Yu, Tao Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Xiaokun Yuan, Yang Yue, Weihao Zeng, Dunyuan Zha, Haobing Zhan, Dehao Zhang, Hao Zhang, Jin Zhang, Puqi Zhang, Qiao Zhang, Rui Zhang, Xiaobin Zhang, Y. Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yushun Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Chenguang Zhao, Feifan Zhao, Jinxiang Zhao, Shuai Zhao, Xiangyu Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Junfeng Zhong, Longguang Zhong, Weiming Zhong, M. Zhou, Runjie Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yuxuan Zhu, Zhen Zhu, Jingze Zhuang, Weiyu Zhuang, Ying Zou, and Xinxing Zu. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

[99] LCM team, Loïc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R. Costa-jussà, David Dale, Hady Elsahar, Kevin Heffernan, João Maria Janeiro, Tuan Tran, Christophe Ropers, Eduardo Sánchez, Robin San Roman, Alexandre Mourachko, Safiyyah Saleem, and Holger Schwenk. Large concept models: Language modeling in a sentence representation space, 2024. URL https://arxiv.org/abs/2412.08821.

[100] J. R. R. Tolkien. *The Hobbit, or There and Back Again*. George Allen & Unwin, London, 1937.

[101] Top-Quarks. Arc-solution, 2020. URL https://github.com/top-quarks/ARC-solution.

[102] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936. doi: 10.1112/plms/s2-42.1.230.

[103] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 74952–74965. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/ed3fea9033a80fea1376299fa7863f4a-Paper-Conference.pdf.

[104] Keyon Vafa, Peter G. Chang, Ashesh Rambachan, and Sendhil Mullainathan. What has a foundation model found? using inductive bias to probe for world models, 2025. URL https://arxiv.org/abs/2507.06952.

[105] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[106] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

[107] Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning, 2025. URL https://arxiv.org/abs/2505.10320.

[108] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. doi: 10.1007/BF00992696.

[109] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing, 2024. URL https://arxiv.org/abs/2406.08464.

[110] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

[111] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

[112] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL https://arxiv.org/abs/2308.01825.

[113] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

[114] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-star: Language models can teach themselves to think before speaking, 2024. URL https://arxiv.org/abs/2403.09629.

[115] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild, 2025. URL https://arxiv.org/abs/2503.18892.

[116] Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data, 2025. URL https://arxiv.org/abs/2505.03335.

[117] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

[118] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models, 2024. URL https://arxiv.org/abs/2403.13372.

[119] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. ibot: Image bert pre-training with online tokenizer, 2022. URL https://arxiv.org/abs/2111.07832.