Title

Enhancing the Reasoning Capabilities of Large Language Models: From Game Playing to Embodied Planning

ISBN

9798265483263

Author

WANG, SHU

Publication Date

2025-12-11

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Enhancing the Reasoning Capabilities of Large Language Models: From Game Playing to Embodied Planning

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Statistics

by

Shu Wang

2025

ABSTRACT OF THE DISSERTATION

Enhancing the Reasoning Capabilities of Large Language Models: From Game Playing to Embodied Planning

by

Shu Wang

Doctor of Philosophy in Statistics

University of California, Los Angeles, 2025

Professor Ying Nian Wu, Chair

Reasoning is a fundamental capability of human intelligence that enables complex problem-solving and decision-making. Recent advances in Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse tasks, yet their reasoning abilities in complex, long-horizon scenarios remain inadequately understood. This thesis systematically investigates and enhances the reasoning capabilities of LLMs across three progressively complex domains: constrained game playing, task and motion planning, and sequential manipulation in partially known environments.

First, we explore LLM reasoning in the structured domain of chess, where we demonstrate that language explanations combining long-term strategic thinking and short-term tactical analysis significantly enhance model performance. Through the MATE dataset of 1 million annotated chess positions, we show that fine-tuned models incorporating strategy and tactic annotations outperform state-of-the-art commercial LLMs by 24.2%, establishing that explicit reasoning guidance improves decision-making quality.

Second, we present LLM^3, a framework that leverages LLMs as domain-

independent interfaces between symbolic task planning and continuous motion planning. By categorizing motion planning feedback into collision and unreachability failure modes, LLM^3 enables iterative refinement of action sequences and parameters. Experimental results demonstrate a 91.3% success rate across realistic scenarios, with 36.1% reduction in travel distance compared to baselines, highlighting the effectiveness of LLM-based failure reasoning.

Third, we introduce EPoG, which integrates exploration and sequential manipulation planning on graph-based scene representations for partially known environments. By employing LLMs for both informed exploration prioritization and situated replanning, EPoG naturally combines information gathering with task execution. Across 46 household scenes and 5 long-horizon tasks, EPoG achieves superior performance while reducing exploration overhead by 40% and travel distance by 36.2%.

Collectively, this thesis makes three primary contributions: (1) demonstrating that language-based explanations enhance LLM reasoning across domains of increasing complexity, (2) establishing effective methods for integrating LLM reasoning with classical planning algorithms through structured feedback mechanisms, and (3) showing that LLMs can provide domain-independent heuristics that improve both planning efficiency and adaptability to environmental uncertainty. These findings advance our understanding of LLM reasoning capabilities and provide practical frameworks for deploying LLMs in complex embodied AI applications.

The dissertation of Shu Wang is approved.

Demetri Terzopoulos

Guang Cheng

Wei Wang

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2025

To my family, for their unwavering support and encouragement throughout my Ph.D. journey. When I stepped away for two years to pursue my dream of becoming a professional chess player—an unconventional choice—my family fully supported my decision.Through my friendship with multiple-time chess world champion Yifan Hou, I came to understand that my path lay in a different direction. Though I ultimately returned to my Ph.D. studies, I remain grateful for having had the courage to pursue that dream and for my family's steadfast support through my every decision.

Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to all those who have contributed to the completion of this thesis.

First and foremost, I am profoundly grateful to Prof. Ying Nian Wu, for his invaluable guidance, continuous support, and insightful feedback throughout my doctoral studies. I will carry his words with me always: "Surrender and trust." I am deeply grateful to Prof. Song-Chun Zhu, who taught me to be resilient, be strong, be confident, be tough, and to maintain passion in life.

I thank my committee members, Prof. Demetri Terzopoulos, Prof. Wei Wang, and Prof. Guang Cheng, for their time and constructive feedback.

I would also like to thank my co-authors and lab-mates: Wenxiao Zhao, Xiaofeng Gao, Lifeng Fan, Tengyu Liu, Zhixiong Nan, Ran Gong, Yizhou Zhao, Tianmin Shu, Lei Ji, Renxi Wang, Haokun Liu, Yifan Hou, Muzhi Han, Ziyuan Jiao, Zixia Jia, Haoyi Wu, Jiapeng Li, Lixing Niu, Hangxin Liu, Xiaoyuan Yi, Han Jiang, Deqian Kong, Minglu Zhao, Dehong Xu, Yasi Zhang, Edouardo Honig, Jianwen Xie, Caiming Xiong, Benjamin Yao, Jacky Yi, Pan Lu, Kewei Tu, Tianyang Zhao, Shuwen Qiu, Zilong Zheng, Bo Pang, Puhao Li, Qian Long, Zhi Li, Rocky Wang.

Thanks to my good friends for supporting me to go through all the journey. I am especially grateful to those who offered their help and companionship during the most challenging periods—their friendship will always hold a special place in my heart.

Finally, I would like to thank my family for their unconditional love, patience, and support throughout this journey. This work would not have been possible without them.

Chapter 1

INTRODUCTION

Rational thought and deliberate cognition rely heavily on reasoning, a core component of human intelligence.

— Alan Garnham and Jane Oakhill, 1994

1.1 Motivation

Reasoning is a central capability of human intelligence, enabling us to solve complex problems, make decisions, and plan actions in diverse and uncertain environments. Given sufficient information, humans can logically progress through sequences of steps to achieve long-term goals while adapting to unexpected situations. In the field of artificial intelligence, developing systems with robust reasoning capabilities has been a persistent objective, as reasoning is essential for both problem-solving and decision-making processes Garnham and Oakhill [1994], Russell and Norvig [2016].

The past few years have witnessed remarkable advances in Large Language Models (LLMs), which have demonstrated extraordinary aptitude in tasks requiring reasoning capabilities Brown et al. [2020], Wei et al. [2022a], Kojima et al. [2022], Bubeck et al. [2023]. Pre-trained on vast amounts of text data, these models encode rich world knowledge and exhibit emergent abilities to understand context, generate coherent responses, and perform multi-step reasoning. However, despite these impressive capabilities, LLMs still show significant limitations in planning and reasoning for complex, long-horizon tasks Xu et al. [2023], Dziri et al. [2024], Srivastava et al. [2022], Wang et al. [2024b], Mirzadeh et al. [2024].

This thesis investigates three fundamental questions about LLM reasoning capabilities:

1. **Can language explanations enhance LLM reasoning?** While LLMs can generate action sequences, it remains unclear whether providing explicit reasoning explanations—such as strategic goals and tactical analysis—can improve their decision-making quality in complex domains.

2. **How can LLMs integrate with classical planning methods?** Traditional planning algorithms offer guarantees of completeness and optimality but require extensive domain-specific engineering. Can LLMs serve as domain-independent interfaces that reduce manual effort while maintaining planning quality?

3. **Can LLMs handle uncertainty and adapt to dynamic environments?** Real-world scenarios often involve incomplete information and unexpected situations. Can LLMs reason about environmental uncertainty and generate situated plans that adapt to new observations?

To address these questions, this thesis presents three interconnected studies that progressively explore LLM reasoning across domains of increasing complexity:

- **Chess as a reasoning testbed** (Chapter 3): We use chess—a structured game requiring both strategic and tactical thinking—to study how language explanations enhance LLM reasoning. By annotating chess positions with expert-provided strategies and tactics, we demonstrate that LLMs can learn to make better moves when guided by explicit reasoning.

- **Task and motion planning with failure reasoning** (Chapter 4): We develop LLM^3, a framework that uses LLMs to interface between sym-

bolic task planning and continuous motion planning. By categorizing and reasoning about motion failures, LLM^3 iteratively refines plans to overcome geometric constraints without requiring domain-specific programming.

- **Integrated exploration and manipulation in unknown environments** (Chapter 5): We present EPoG, which seamlessly combines exploration with task execution using graph-based scene representations. By leveraging LLMs for informed exploration and situated replanning, EPoG efficiently operates in partially known, dynamic environments.

1.2 Challenges and Opportunities

The integration of LLMs into complex reasoning and planning tasks presents both significant challenges and exciting opportunities.

1.2.1 Challenges

Long-horizon reasoning: LLMs struggle with tasks requiring extended sequences of interdependent decisions. As the planning horizon increases, the number of possible action sequences grows exponentially, making it difficult for LLMs to maintain consistency and coherence across long chains of reasoning.

Spatial and geometric reasoning: Language models, trained primarily on text, have limited understanding of 3D spatial relationships, geometric constraints, and physical feasibility. This limitation becomes critical in embodied AI applications where actions must respect physical laws and environmental constraints.

Handling uncertainty: Real-world environments are partially observable and dynamic. LLMs must reason about uncertain information, adapt plans

based on new observations, and handle unexpected situations—capabilities that require more than pattern matching on training data.

Domain-specific knowledge: While LLMs possess broad commonsense knowledge, they may lack the specialized domain knowledge required for expert-level performance in specific fields like chess, robotics, or scientific reasoning.

Evaluation and verification: Unlike classification or generation tasks with clear metrics, evaluating reasoning quality is challenging. Plans may appear reasonable but fail in execution due to subtle logical errors or overlooked constraints.

1.2.2 Opportunities

Commonsense knowledge: LLMs encode vast amounts of world knowledge that can serve as powerful heuristics for planning. This knowledge can guide exploration, suggest plausible actions, and help interpret sensory observations in meaningful ways.

Language as a reasoning medium: Natural language provides a flexible and interpretable medium for expressing reasoning steps, explaining decisions, and communicating with humans. This makes LLM-based systems more transparent and easier to debug compared to purely neural approaches.

Few-shot adaptation: LLMs can adapt to new tasks and domains with minimal examples, reducing the need for extensive task-specific training data and enabling rapid deployment in diverse scenarios.

Integration with classical methods: Rather than replacing traditional planning algorithms, LLMs can complement them by providing domain-independent heuristics, handling exceptions, and bridging symbolic and continuous representations.

Situated reasoning: LLMs' ability to process contextual information and

generate appropriate responses based on situational cues makes them well-suited for situated decision-making in dynamic environments.

1.3 Research Contributions

This thesis makes the following contributions to the field of AI reasoning and planning:

1. **Framework for language-enhanced reasoning:** We establish that explicit language explanations combining long-term strategic goals and short-term tactical analysis significantly improve LLM reasoning performance across domains.

2. **Motion failure reasoning taxonomy:** We categorize motion planning failures into semantically meaningful types (collision, unreachability) and demonstrate that LLMs can effectively reason about these failures to generate improved plans.

3. **Integrated exploration-planning paradigm:** We show that exploration and task execution can be naturally integrated through graph-based representations and LLM-based belief updates, eliminating the need for separate exploration and planning phases.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 provides background on LLM capabilities, planning methods, and related work in reasoning, task planning, and embodied AI. We review relevant literature on LLM reasoning, TAMP approaches, and planning in uncertain environments.

Chapter 3 presents our work on chess as a testbed for studying LLM reasoning. We describe the MATE dataset, our approach to integrating strategy and tactic annotations, and experimental results demonstrating the effectiveness of language-guided reasoning.

Chapter 4 introduces LLM^3, our framework for task and motion planning with motion failure reasoning. We detail the system architecture, failure categorization approach, and experiments showing improved planning efficiency.

Chapter 5 describes EPoG, our framework for integrated exploration and sequential manipulation in partially known environments. We explain the graph-based representation, informed exploration strategy, and situated replanning mechanism.

Chapter 6 concludes the thesis by summarizing our findings and discussing limitations.

1.5 Publications

The work presented in this thesis has been published or submitted to peer-reviewed conferences and journals:

- **Chapter 3:** Shu Wang, Lei Ji, Renxi Wang, Wenxiao Zhao, Haokun Liu, Yifan Hou, Ying Nian Wu. "Explore the Reasoning Capability of LLMs in the Chess Testbed." *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2025. Wang et al. [2025]

- **Chapter 4:** Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, Hangxin Liu. "LLM^3: Large Language Model-based Task and Motion Planning with Motion Failure Reasoning." *IEEE*

International Conference on Intelligent Robots and Systems (IROS), 2024. Wang et al. [2024b]

- **Chapter 5:** Heqing Yang, Ziyuan Jiao, Shu Wang, Yida Niu, Si Liu, Hangxin Liu. "Integrated Exploration and Sequential Manipulation on the Graph-Based Representation with Situated Replanning." *IEEE International Conference on Robotics and Automation(ICRA)*, 2026.

1.6 Summary

This introductory chapter has motivated the study of LLM reasoning capabilities through progressively complex domains, outlined the key challenges and opportunities, and summarized the main contributions of this thesis. The following chapters will present detailed technical approaches, experimental results, and analyses that collectively advance our understanding of how LLMs can reason effectively in complex, real-world scenarios.

Chapter 2

BACKGROUND AND RELATED WORK

This chapter provides the necessary background and reviews related work relevant to this thesis. We begin by introducing Large Language Models and their reasoning capabilities, then discuss classical planning methods, and finally review related work in embodied AI and robotics applications.

2.1 Large Language Models

2.1.1 Transformer Architecture and Pre-training

Large Language Models are built upon the Transformer architecture, which uses self-attention mechanisms to process sequential data. The key innovation of Transformers is the ability to capture long-range dependencies in sequences through multi-head attention, enabling models to understand context across entire documents.

Recent models have scaled to hundreds of billions of parameters and have been trained on trillions of tokens from diverse web sources, resulting in impressive capabilities across a wide range of tasks Wang et al. [2024a], Jiang et al. [2024a], Lin et al. [2025], Chen et al. [2025], Niu et al. [2025], Kong et al. [2025], Liang et al. [2025].

2.1.2 Emergent Reasoning Capabilities

As LLMs have grown in scale, researchers have observed emergent capabilities that were not explicitly programmed or trained for Wei et al. [2022a]. These include:

In-context learning: The ability to adapt to new tasks based on a few examples provided in the prompt, without updating model parameters Brown et al. [2020].

Chain-of-thought reasoning: When prompted to show their reasoning steps, LLMs can break down complex problems into intermediate steps, improving performance on tasks requiring multi-step reasoning Wei et al. [2022b], Kojima et al. [2022].

2.1.3 LLMs for Planning and Decision Making

Recent work has explored using LLMs for planning tasks:

Direct planning: Using LLMs to directly generate action sequences for robot tasks Huang et al. [2022b], Song et al. [2023]. While promising for simple tasks, this approach struggles with long-horizon planning and geometric constraints.

PDDL generation: Using LLMs to translate natural language task descriptions into formal planning languages like PDDL Zhou et al. [2024], Chen et al. [2023]. This leverages classical planners' guarantees while using LLMs for domain specification.

Heuristic guidance: Using LLMs to provide heuristics or value estimates to guide classical planning algorithms Shah et al. [2023].

Code generation for planning: Using LLMs to generate executable code that implements planning logic Liang et al. [2023], Singh et al. [2023].

Our work contributes to this line of research by demonstrating effective integration strategies between LLMs and classical planning methods.

2.2 Classical Planning Methods

2.2.1 Symbolic Task Planning

Classical task planning operates on discrete, symbolic representations of states and actions. The most common formalism is STRIPS or PDDL, which represents:

- **States:** Sets of ground predicates describing the world
- **Actions:** Operators with preconditions and effects
- **Goals:** Desired state configurations

Planning algorithms search for action sequences that transform the initial state into a goal state. Common approaches include:

Forward search: Starting from the initial state, apply actions to generate successor states until reaching the goal (e.g., A*).

Backward search: Starting from the goal, work backwards to find actions that achieve it.

Heuristic search: Use domain-specific or domain-independent heuristics to guide search.

While symbolic planning offers completeness guarantees and interpretable plans, it requires careful domain engineering and struggles to incorporate geometric and continuous constraints.

2.2.2 Motion Planning

Motion planning addresses the problem of finding collision-free trajectories for robots in continuous configuration spaces. Common approaches include:

Sampling-based planners: Algorithms like RRT and PRM randomly sample configurations and build a roadmap of feasible paths.

Optimization-based planners: Methods that formulate motion planning as an optimization problem subject to constraint.

Learning-based planners: Neural network approaches that learn to generate feasible motions from data.

Motion planning is computationally expensive, especially in high-dimensional spaces or cluttered environments. Our work addresses this by using LLMs to select action parameters that are more likely to result in feasible motions, reducing the number of motion planning queries needed.

2.2.3 Task and Motion Planning (TAMP)

TAMP integrates symbolic task planning with geometric motion planning Garrett et al. [2021]. The key challenge is interfacing between the discrete symbolic level and the continuous geometric level.

Common TAMP approaches include:

Hierarchical refinement: Generate a symbolic plan, then refine it by solving motion planning problems for each action Kaelbling and Lozano-Pérez [2011].

Integrated search: Interleave symbolic and geometric reasoning during search Dantam et al. [2016], Garrett et al. [2020a].

Constraint-based: Encode geometric constraints in the symbolic domain Toussaint [2015].

Learning-based parameter sampling: Learn to sample action parameters that are likely to result in feasible motions Chitnis et al. [2016], Wang et al. [2018].

Traditional TAMP methods require extensive domain-specific engineering to specify symbolic domains, action parameters, and geometric constraints. Our LLM^3 framework reduces this burden by using LLMs as domain-independent

interfaces.

2.3 Planning in Uncertain Environments

2.3.1 Partially Observable Environments

When complete environmental information is unavailable, robots must reason about uncertainty:

POMDP: Partially Observable Markov Decision Processes provide a formal framework for planning under uncertainty, but are often computationally intractable for realistic problems.

Belief space planning: Maintain a belief distribution over possible world states and plan in belief space Garrett et al. [2020b].

Information gathering: Actively plan actions to reduce uncertainty through observation Xu et al. [2022].

2.3.2 Exploration and Mapping

In unknown environments, robots must explore to build maps and locate objects:

Frontier-based exploration: Identify unexplored boundaries and move towards them.

Information-gain exploration: Plan actions that maximize expected information gain.

Task-driven exploration: Integrate exploration with task execution to minimize overall cost Xu et al. [2022].

Traditional exploration strategies use hand-designed heuristics that may not generalize across domains. Our EPoG framework leverages LLM commonsense knowledge to prioritize exploration more intelligently.

2.4 Embodied AI and Robotics

2.4.1 VR Environment and Scene Representation

Robots require rich representations of the environment to support reasoning and planning.

Virtual Environment: One of the main challenges of advancing task-oriented learning such as visual task planning and reinforcement learning is the lack of realistic and standardized environments for training and testing Embodied AI agents. There have been recent advances in virtual systems built with 3D physics engines and photo-realistic rendering for indoor and outdoor environments. Gao et al. [2019, 2020], Nan et al. [2020], Gong et al. [2023], Gao et al. [2022]

3D Scene Graphs: Hierarchical graph structures that represent spatial and semantic relationships between entities in a scene Armeni et al. [2019], Rosinol et al. [2020].

Our work builds on graph-based scene representations in virtual environment, using them as the foundation for planning and reasoning.

2.4.2 Robot Task Planning with LLMs

Recent work has explored various ways to integrate LLMs into robotic systems:

Task planning from language: Translating natural language commands into robot action sequences Huang et al. [2022a], Singh et al. [2023].

Code generation for robot control: Using LLMs to generate executable code that controls robots Liang et al. [2023], Han et al. [2024].

Interactive replanning: Using LLMs to adapt plans based on execution feedback Huang et al. [2023], Ding et al. [2023].

Our work contributes to this area by demonstrating effective methods for

situated replanning and exception handling.

2.4.3 Sequential Manipulation Planning

Sequential manipulation involves reasoning about sequences of pick-and-place operations:

Rearrangement planning: Finding sequences of object movements to achieve desired configurations.

Graph-based planning: Using graph representations to reason about object relationships and dependencies Jiao et al. [2022].

Learning-based approaches: Using machine learning and reinforcement learning Li et al. [2024]to predict good action sequences.

Our EPoG framework addresses sequential manipulation in partially known environments, combining exploration with task execution.

2.5 Chess as a Reasoning Testbed

Chess has a long history as a benchmark for AI systems:

Classical chess engines: Programs like Stockfish use minimax search with alpha-beta pruning and sophisticated evaluation functions.

Neural chess engines: AlphaZero Silver et al. [2017] and Leela Chess Zero use deep reinforcement learning and Monte Carlo Tree Search.

Transformer-based chess: Recent work trains Transformers on game datasets to play chess without explicit search Ruoss et al. [2024].

However, little work has explored whether LLMs can learn to reason about chess through language explanations of strategy and tactics. Our work fills this gap by showing that explicit reasoning guidance improves LLM chess performance.

2.6 Summary and Research Gaps

While significant progress has been made in LLM reasoning, classical planning, and embodied AI, several gaps remain:

1. **Language-guided reasoning:** Limited understanding of how language explanations can enhance LLM reasoning in complex domains beyond simple question-answering.

2. **Integration with planning:** Most work treats LLMs either as standalone planners or as simple preprocessing steps, missing opportunities for deeper integration with classical methods.

3. **Handling uncertainty:** Few approaches effectively leverage LLM commonsense knowledge for reasoning about environmental uncertainty and prioritizing exploration.

4. **Situated adaptation:** Limited work on how LLMs can perform situated replanning to handle unexpected situations in real-world scenarios.

This thesis addresses these gaps through three complementary studies that progressively explore LLM reasoning capabilities. The following chapters present our approaches and findings in detail.

Chapter 3

EXPLORING REASONING IN CONSTRAINED DOMAINS: THE CHESS TESTBED

Strategy without tactics is the slowest route to victory. Tactics without strategy is the noise before defeat.

— Sun Tzu

3.1 Introduction

Reasoning is a fundamental component of human intelligence that enables complex problem-solving and decision-making across diverse situations. While recent large language models have demonstrated impressive capabilities in tasks ranging from natural language understanding to code generation, their reasoning abilities in complex, structured domains remain inadequately understood and require systematic investigation. In this chapter, we explore LLM reasoning through the lens of chess—a domain that requires both long-term strategic thinking and short-term tactical calculation, providing an ideal controlled environment for studying reasoning capabilities.

Chess provides an ideal testbed for studying reasoning for several compelling reasons that make it particularly well-suited for controlled experimentation. First, it represents a pure reasoning task where all rules are clearly defined and all board positions are fully observable, eliminating uncertainty and randomness that could confound analysis of reasoning capabilities. Second, expert chess players employ dual reasoning modes, using both intuitive strategic reasoning to evaluate positions and analytical tactical calculation to

compute precise move sequences. Third, chess has developed a rich vocabulary and linguistic framework for describing strategic plans, tactical patterns, and positional evaluations, enabling natural integration with language models. Fourth, centuries of accumulated chess theory and expert knowledge provide well-established principles for position evaluation and move selection, offering reliable ground truth for validating reasoning quality.

Drawing inspiration from how expert chess players think—combining rapid strategic intuition with deliberate tactical analysis, both expressed in clear language—we propose enhancing LLMs' chess-playing capabilities by incorporating language explanations of both strategy and tactics. This approach aims to bridge the gap between pure pattern matching and genuine reasoning by making the reasoning process explicit and linguistically grounded.

3.1.1 Chess Reasoning: Strategy and Tactics

Good chess players employ a dual approach to decision-making that integrates two complementary modes of thinking, each serving different purposes in the evaluation and selection of moves.

Long-term Strategy: Strategic thinking relies on rapid, intuitive pattern recognition of the chess position, drawing on accumulated experience to assess overall position quality and formulate plans. It involves understanding and applying fundamental positional principles that guide long-term decision-making. Material count focuses on maintaining or achieving material advantage, as having more pieces generally provides greater tactical flexibility and winning potential. Piece activity emphasizes positioning pieces on effective squares where they control important areas of the board and coordinate well with other pieces. Pawn structure analyzes the configuration and quality of pawn formations, which tend to be relatively permanent features of positions

that create lasting strategic advantages or weaknesses. Space concerns controlling a larger portion of the board, which restricts opponent options and provides room for piece maneuvering. King safety prioritizes protecting the king from attack, as an exposed king creates immediate tactical vulnerabilities that can override other considerations.

Short-term Tactics: Tactical play involves slow, analytical calculation that explicitly computes move sequences looking ahead multiple moves into the future. Expert players typically calculate 1-6 moves ahead depending on their skill level and the complexity of the position, though in sharp tactical positions they may calculate even deeper forcing sequences. This analytical mode includes recognizing specific tactical patterns and motifs such as pins (where a piece cannot move without exposing a more valuable piece behind it), forks (where one piece attacks two or more enemy pieces simultaneously), skewers (forcing a valuable piece to move and exposing a less valuable piece behind it), and numerous other forcing sequences that create concrete threats or win material.

Notably, experienced chess players naturally "think out loud" during their decision-making process—they verbally articulate strategic plans in clear language, explaining their understanding of the position's key features, and evaluate resulting positions after calculating precise tactical sequences. This observation that expert reasoning is inherently linguistic motivates our approach of using explicit language explanations to guide LLM reasoning in chess.

Figure 3.1 illustrates the fundamental difference between strategic and tactical thinking through concrete examples.

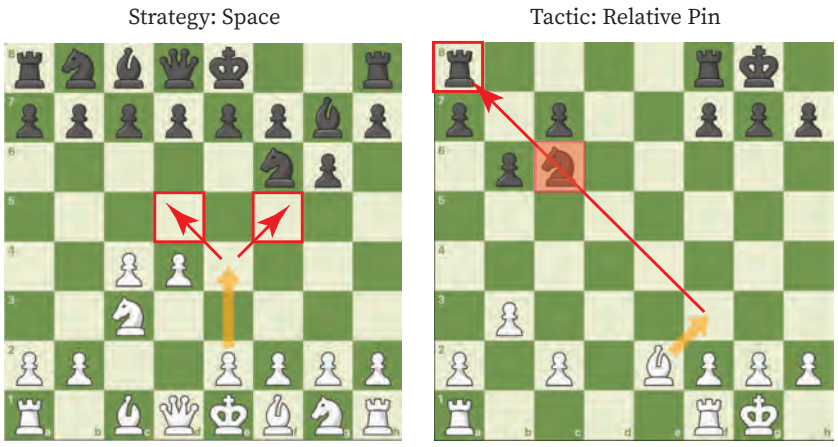


Figure 3.1: Strategy and Tactic examples. (a) Strategy: White's E2 pawn moves to E4, taking more space in the center and exerting pressure on Black. (b) Tactic: White's E2 bishop moves to F3 and pins the knight on C6.

3.1.2 Research Questions

This chapter addresses three fundamental research questions that probe different aspects of how language explanations can enhance LLM reasoning capabilities in structured domains.

First, we investigate whether language explanations of strategy and tactics can improve LLM reasoning in chess. This addresses the fundamental question of whether making reasoning explicit through language provides measurable benefits for decision-making quality, or whether language serves primarily as a post-hoc description of decisions made through other mechanisms.

Second, we examine how strategic and tactical annotations independently contribute to performance. By isolating these two types of explanations, we can understand which mode of reasoning provides greater benefit and whether the benefits are additive or synergistic when combined.

Third, we explore whether fine-tuned LLMs trained with these annotations can outperform state-of-the-art commercial models that rely primarily on general pre-training. This tests whether domain-specific training with ex-

pert annotations can overcome the advantages of scale and general capabilities present in frontier commercial models.

3.2 The MATE Dataset

3.2.1 Dataset Overview

We introduce MATE (**M**ove on str**A**tegy and **T**actics dat**as**E**t**), a comprehensive dataset specifically designed for studying language-guided chess reasoning. MATE consists of approximately 1 million chess positions, each with candidate moves that have been carefully annotated by expert chess players including world champion-level annotators who bring the highest level of chess understanding to the annotation process.

The dataset is designed according to several key principles that ensure it provides comprehensive and rigorous evaluation of reasoning capabilities. First, comprehensive coverage ensures that positions span all phases of the game including opening theory, complex middlegame positions, and technical endgames, providing a complete picture of chess reasoning across different game stages. Second, diverse scenarios guarantee that the dataset includes positions exhibiting various strategic themes and tactical patterns, from quiet positional maneuvering to sharp tactical complications. Third, multiple difficulty levels range from beginner-accessible positions to grandmaster-level complications, enabling assessment of reasoning capabilities across the full spectrum of chess skill. Fourth, clear comparisons are enabled by ensuring each position contains exactly two candidate moves with significantly different evaluation scores from chess engines, making the better move objectively identifiable while maintaining realistic difficulty.

3.2.2 Chess Notation

We employ standard chess notation formats that enable precise and compact representation of positions and moves, facilitating both human interpretation and computational processing.

Forsyth-Edwards Notation (FEN): This provides a compact text representation of complete chess positions in a single line of ASCII characters. FEN systematically encodes all information necessary to reconstruct a position: piece positions are represented using capital letters for White pieces (K for king, Q for queen, R for rook, B for bishop, N for knight, P for pawn) and lowercase letters for Black pieces, with digits indicating consecutive empty squares. The active color indicates whose turn it is to move (w for White, b for Black). Castling rights specify which players can still castle kingside or queenside based on whether the relevant kings and rooks have moved. The en passant target square identifies if a pawn has just advanced two squares and can be captured en passant. The halfmove clock counts moves since the last pawn move or capture, used for the fifty-move draw rule. The fullmove number indicates how many complete moves have been played since the game's start.

Universal Chess Interface (UCI): This standard format encodes moves using algebraic notation that specifies the starting and ending squares of a piece movement. For example, "e2e4" indicates moving a piece from square e2 to square e4, providing an unambiguous and compact representation of moves that is widely used in chess software and engines.

3.2.3 Strategy Annotation

We categorize chess strategy into five fundamental types that capture the most important positional considerations in chess evaluation. These categories provide a structured framework for expressing strategic reasoning through lan-

guage.

Material Count This strategic theme focuses on maintaining or gaining material advantage, which is typically the most concrete and measurable aspect of position evaluation. Each piece type has a conventionally assigned relative value: pawns are worth 1 point, knights and bishops are worth approximately 3 points each, rooks are worth approximately 5 points, and queens are worth approximately 9 points. Kings have infinite value as their loss ends the game. This strategic consideration is most relevant in several types of positions. Material count becomes paramount when there exists a material imbalance between the two sides, as the player with more material generally has greater tactical flexibility and winning chances. It is particularly important when both kings are relatively safe from immediate threats, allowing players to focus on converting material advantages through piece exchanges and simplification. Additionally, material considerations dominate in positions that are not particularly dynamic, where tactical complications do not override the fundamental advantage of having more pieces.

Piece Activity This strategic theme emphasizes the placement and effectiveness of pieces in terms of the squares they control and their coordination with other pieces. Centralized pieces positioned near the board's center are typically more powerful than pieces on the edges or corners, as central pieces control more squares and have greater flexibility to influence different areas of the board. This strategic consideration becomes crucial in several position types. Piece activity is most important when there exists a marked difference in piece positioning between the two sides, where one player's pieces are well-coordinated and actively placed while the opponent's pieces are passive or poorly coordinated. It becomes decisive in dynamic positions with tactical

opportunities where piece coordination can create immediate threats. Additionally, piece activity is critical when one side is mounting an attack, as effective piece placement determines whether the attack will succeed or be repelled.

Space This strategic theme concerns controlling a larger portion of the board through pawn advances and piece placement. Spatial advantage limits the opponent's options for piece placement and maneuvering while creating opportunities for the player with more space to launch attacks or improve piece positions. This consideration is particularly important in several contexts. Space control matters most in opening and middlegame phases when players are developing their pieces and establishing the position's character, as early spatial advantages can persist and grow throughout the game. It is especially relevant when many pawns remain on the board, as pawn chains and formations define the available space for piece maneuvering. Space advantage becomes decisive for restricting opponent piece mobility, as cramped positions limit tactical options and make it difficult to coordinate defensive resources.

Pawn Structure This strategic theme analyzes the configuration and quality of pawn formations, which tend to be relatively permanent features of positions since pawns cannot move backward. Strong pawn structures featuring connected pawns, pawn chains, or passed pawns can create lasting strategic advantages that provide long-term winning chances. Conversely, weak pawn structures with doubled pawns (two pawns of the same color on the same file), isolated pawns (pawns with no friendly pawns on adjacent files), or backward pawns (pawns that cannot advance without becoming weak) become enduring liabilities. This consideration deserves careful attention in several types of positions. Pawn structure is critical when there are clear structural issues

such as doubled or isolated pawns that will influence the position's character for many moves. It becomes important in typical opening structures where theoretical pawn formations emerge, as these structures guide the appropriate plans for both sides. Pawn structure considerations are essential for long-term planning, as structural advantages or weaknesses often determine which side has better prospects in the resulting endgame.

King Safety This strategic theme prioritizes protecting the king from attack, as the king's safety directly determines whether other plans can be executed or whether defensive resources must be diverted to address immediate threats. A secure king positioned behind a solid pawn shield and supported by defensive pieces enables other strategic plans to proceed confidently. Conversely, a vulnerable king exposed to attack creates immediate tactical threats that must be addressed before other considerations. King safety should always be carefully considered in several types of positions. It becomes paramount when the king is exposed without protective pawn cover, as this creates immediate tactical vulnerabilities. King safety is critical when opponent pieces are well-coordinated for attack, with multiple pieces targeting the king's vicinity. Additionally, it demands immediate attention when open files or diagonals point toward the king's position, providing avenues for heavy pieces (rooks and queens) to invade and create threats.

For each strategic category, we developed approximately 20 distinct linguistic expressions to describe the corresponding plans and evaluations. This rich variation in language annotations ensures that models learn the underlying strategic concepts rather than memorizing specific phrasings, promoting robust understanding and generalization.

3.2.4 Tactic Annotation

Chess tactics encompass numerous specific patterns and motifs including pins, forks, skewers, discovered attacks, deflections, decoys, and many others. Rather than attempting to enumerate and categorize all possible tactical types, which would require extensive manual annotation effort and chess expertise, we adopt a pragmatic approach that captures tactical information through concrete sequences and outcomes.

Our tactical annotations consist of three components that together provide clear tactical understanding. First, we list the specific sequence of moves constituting the tactical idea, limiting the length to 3-5 moves to focus on short-term analysis rather than deep strategic planning. Second, we provide a factual description of the resulting position after the tactical sequence completes, explaining what material or positional gains have been achieved. Third, we focus on describing concrete consequences rather than labeling patterns with technical names, as the outcomes matter more than the classification for reasoning purposes.

Tactical sequences are generated algorithmically using the Stockfish chess engine, which is widely recognized as the strongest chess playing program and provides reliable evaluation of positions and moves. We evaluate move quality using Stockfish's centipawn scores, which measure position evaluation in units of one-hundredth of a pawn. For each position in the dataset, we select two candidate moves whose Stockfish score differences exceed a specified threshold (typically 100-200 centipawns, equivalent to approximately one to two pawns of advantage), clearly indicating that one move is objectively superior to the other while maintaining realistic difficulty where both moves might seem plausible to human players.

3.2.5 Dataset Subsets

Based on the complete MATE dataset, we create four specialized sub-datasets that isolate different combinations of annotation types. This design enables systematic study of how strategic and tactical annotations independently and jointly contribute to reasoning performance.

MATE-No-Explanation (MATE-N): This subset contains chess positions with candidate moves but no strategic or tactical annotations, providing a baseline that tests pure pattern recognition without explicit reasoning guidance. This subset comprises 40.8% of the total dataset, representing positions where high-quality annotations could not be reliably generated or where the position is straightforward enough to not require detailed explanation.

MATE-Strategy (MATE-S): This subset contains positions with strategic explanations annotated for each candidate move, articulating the long-term positional considerations and plans associated with each move. This subset comprises 39.2% of the dataset, representing positions where clear strategic themes can be identified and described linguistically.

MATE-Tactic (MATE-T): This subset contains positions with tactical descriptions for each candidate move, providing concrete move sequences and their consequences. This subset comprises 10% of the dataset, reflecting the fact that sharp tactical positions requiring precise calculation are less common than strategically complex positions.

MATE-Strategy&Tactic (MATE-ST): This subset contains positions with both strategic and tactical annotations for each candidate move, providing comprehensive dual-mode reasoning guidance. This subset comprises 10% of the dataset, representing positions that feature both significant strategic considerations and concrete tactical opportunities.

Figure 3.2 illustrates the distribution of positions across these subsets and

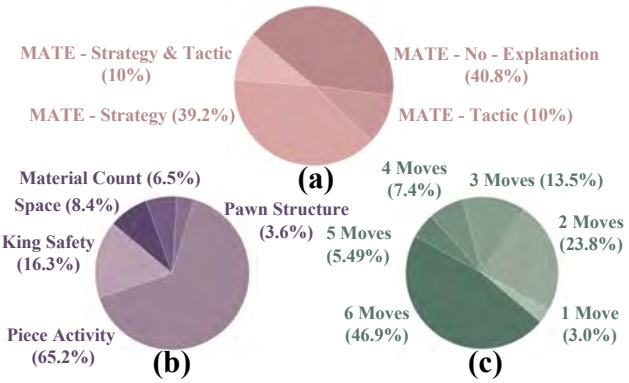


Figure 3.2: Distribution of MATE dataset across different subsets and strategic categories.

breaks down the strategic categories within the strategy-annotated subsets.

The distribution reflects the natural character of chess positions. Most positions lend themselves better to long-term strategic planning rather than immediate tactical calculations. Many positions are not particularly sharp or forcing, lacking immediate tactical opportunities that require precise calculation, but strategic plans based on positional principles can still be meaningfully formulated and evaluated. Consequently, MATE-Strategy is significantly larger than MATE-Tactic and MATE-ST, reflecting the prevalence of strategic complexity over tactical sharpness in typical chess positions.

3.2.6 Data Example

Figure 3.3 presents a complete example drawn from the MATE-Strategy&Tactic subset, illustrating how we annotate positions with both types of reasoning to provide comprehensive guidance for move evaluation.

The example demonstrates how strategic explanations articulate the positional themes and long-term plans associated with each move, while tactical descriptions provide concrete move sequences showing the immediate consequences of each choice. This dual annotation provides rich context that enables

models to reason about moves from multiple complementary perspectives.

3.3 Method

3.3.1 Model Architecture and Training

We fine-tune LLaMA-3-8B Dubey et al. [2024] as our base model, leveraging its strong general capabilities while specializing it for chess reasoning through training on the MATE dataset. The LLaMA-3 architecture provides an excellent foundation due to its strong performance on reasoning tasks and its tractable size for fine-tuning on academic hardware.

Training Framework: We employ LLaMAFactory Zheng et al. [2024], a comprehensive framework for efficiently fine-tuning large language models that handles data processing, training loops, and distributed training coordination.

Optimization: The training process uses carefully tuned hyperparameters to ensure stable convergence and optimal performance. We employ a cosine learning rate scheduler with 3% warm-up, which gradually increases the learning rate during the first 3% of training steps before smoothly decreasing it following a cosine curve, helping the model escape local optima and converge stably. The maximum learning rate is set to 5×10^{-6}, which is relatively conservative to avoid catastrophic forgetting of the pre-trained knowledge while still enabling effective adaptation to the chess domain. We train for 5 epochs over the dataset, allowing the model multiple passes to learn the complex patterns while monitoring for overfitting. The batch size is set adaptively based on available GPU memory, maximizing training efficiency while respecting hardware constraints.

Infrastructure: We leverage DeepSpeed ZeRO Stage 3 Rajbhandari et al. [2020], an advanced distributed training framework that partitions optimizer

states, gradients, and model parameters across multiple GPUs to enable training of large models. The training runs on 4× H100 GPUs, providing substantial computational power while remaining within reasonable academic computing budgets.

Special Tokens: To better represent chess positions in the language model's vocabulary, we incorporate special tokens into the FEN format representation. A `<line>` token separates board rows, making the two-dimensional board structure more explicit in the linear text representation. A `<color>` token clearly indicates whose turn it is to move, helping the model track this critical state information. These special tokens help bridge the gap between the spatial nature of chess positions and the sequential nature of language model input.

We train four separate models, one for each MATE subset, to isolate the effects of different annotation types. The MATE-N model trains on positions without explanations, establishing a baseline for pure pattern recognition. The MATE-S model trains on positions with strategic annotations, learning to reason about long-term positional factors. The MATE-T model trains on positions with tactical annotations, learning to follow and evaluate concrete move sequences. The MATE-ST model trains on positions with both types of annotations, learning to integrate strategic and tactical reasoning.

3.3.2 Prompt Design

The prompt structure for our models is carefully designed to provide clear context and elicit structured reasoning. The complete prompt template consists of several components that guide the model's response.

The system-level instruction establishes the model's role and task:

```
You are an expert chess player. You are given a chess board
```

```
with FEN format. Your goal is to choose a better move given
two candidate moves with their strategy explanations and
tactics.
```

The position specification provides the current board state in standard format:

```
The FEN of the given chess board is: [FEN string]
```

For each candidate move, we provide a structured presentation that includes the move itself and its associated annotations:

```
Move A: [move in UCI format]
Strategy: [strategic explanation]
Tactic: [tactical sequence and description]

Move B: [move in UCI format]
Strategy: [strategic explanation]
Tactic: [tactical sequence and description]
```

Finally, the query explicitly asks for the model's judgment:

```
Which move is better?
```

For models trained on subsets without certain annotations (such as MATE-N which lacks all explanations, or MATE-S which lacks tactical annotations), we omit the corresponding sections from the prompt. This ensures consistency between training and evaluation formats while testing the specific contribution of each annotation type.

3.3.3 Baseline Models

We compare our fine-tuned models against a comprehensive set of state-of-the-art commercial LLMs to establish the effectiveness of our approach relative to frontier systems with vastly more parameters and training compute.

GPT models from OpenAI: We evaluate four models from the GPT family representing different capabilities and training approaches. gpt-4-0613 represents OpenAI's flagship model from mid-2023, known for strong reasoning capabilities. gpt-4o-2024-08-06 represents an optimized variant focusing on efficiency. o1-preview-2024-09-12 and o1-mini-2024-09-12 represent OpenAI's reasoning-optimized models that use chain-of-thought during inference, making them particularly relevant for comparison to our reasoning-enhanced approach.

Claude models from Anthropic: We evaluate two models from Anthropic's Claude family. claude-3.5-sonnet represents a balanced model optimizing for both capability and efficiency. claude-3-opus represents Anthropic's most capable model, optimized for complex reasoning tasks.

Gemini models from Google: We evaluate two models from Google's Gemini family. gemini-1.5-pro represents Google's most capable model with extensive context windows. gemini-1.5-flash represents a faster variant optimized for efficiency while maintaining strong capabilities.

This diverse set of baselines ensures that our comparisons reflect the current state of commercial LLM capabilities across multiple organizations and architectural approaches.

3.3.4 Evaluation Settings

We evaluate models in two settings that probe different aspects of their reasoning capabilities and their ability to learn from examples.

Zero-shot evaluation tests models on their inherent reasoning abilities without providing any task-specific examples beyond the instruction and format specification. This setting measures whether models can understand and execute the task based solely on their pre-trained knowledge and the current prompt, representing the most challenging evaluation scenario.

Few-shot evaluation provides a small number of demonstration examples (typically 3-5) before the test case to illustrate the task format, expected reasoning style, and output format. This setting measures how effectively models can adapt their behavior based on examples, leveraging in-context learning capabilities that have proven powerful for many language model applications.

For each setting, we evaluate on 1,000 randomly sampled test cases from each test set, ensuring statistical reliability while maintaining manageable computational costs. Models score one point for each test case where they correctly output the optimal move from the two candidates, with accuracy computed as the percentage of correct selections.

3.4 Experimental Results

3.4.1 Main Results

Table 3.1 presents our main experimental results comparing the fine-tuned models with commercial LLMs across all four test sets and both evaluation settings, providing a comprehensive view of how different annotation types and model capabilities affect chess reasoning performance.

The results reveal several key findings about the effectiveness of language-guided reasoning and domain-specific training.

1. Fine-tuned models significantly outperform commercial LLMs: Our models achieve dramatically higher accuracy across all test sets compared

Table 3.1: Accuracy (%) on MATE test sets. Best results in **bold**, second-best underlined.

Model	Zero-Shot				Few-Shot			
	N	**S**	**T**	**ST**	**N**	**S**	**T**	**ST**
gpt-4	53.1	54.6	60.0	60.0	54.7	58.9	57.7	68.1
gpt-4o	46.4	52.8	54.8	60.1	48.5	54.3	52.7	63.1
o1-mini	51.5	58.8	64.1	69.2	50.4	58.3	62.0	65.9
o1-preview	56.4	65.4	77.2	76.6	59.0	65.4	76.2	78.6
claude-3.5-sonnet	49.6	54.9	56.9	54.9	51.9	63.7	59.9	66.1
claude-3-opus	48.3	54.5	53.7	57.3	51.0	55.8	53.2	60.2
gemini-1.5-pro	50.6	48.8	54.2	52.6	50.5	50.1	52.7	50.4
gemini-1.5-flash	46.1	50.8	54.2	52.9	49.7	48.2	53.8	55.6
Ours-no-explanation	**63.5**	–	–	–	**64.7**	–	–	–
Ours-strategy	–	**89.7**	–	–	–	**89.8**	–	–
Ours-tactic	–	–	**94.6**	–	–	–	**94.5**	–
Ours-strategy&tactic	–	–	–	**95.2**	–	–	–	**95.3**

to even the strongest commercial baselines. The Ours-strategy&tactic model achieves 95.2% accuracy on MATE-ST in the zero-shot setting, outperforming the best commercial model (o1-preview at 78.6%) by 16.6 percentage points, representing a 21% relative improvement. This gap is even more pronounced on other subsets: on MATE-T, our tactic-focused model achieves 94.6% compared to o1-preview's 77.2%, a 22.5% relative improvement. Even our baseline Ours-no-explanation model achieves 64.7% on MATE-N in the few-shot setting, surpassing o1-preview's 59.0%, demonstrating that domain-specific training provides substantial benefits even without explicit reasoning annotations.

2. Language explanations enhance reasoning across all models: Comparing performance across test sets provides clear evidence that language explanations improve reasoning for both fine-tuned and commercial models. For o1-mini, accuracy improves from 51.5% on MATE-N (no explanations) to 69.2% on MATE-ST (both explanations), representing a 34% relative improvement. For o1-preview, accuracy increases from 56.4% on MATE-N to 78.6%

on MATE-ST in the few-shot setting, a 39% relative improvement. These substantial gains demonstrate that even powerful commercial models with billions of parameters and extensive pre-training benefit significantly from explicit language explanations of strategic and tactical reasoning, rather than relying purely on implicit pattern matching.

3. Combined strategy and tactics yield best performance: Models generally perform best on MATE-ST, which includes both strategic and tactical annotations, compared to subsets with only one type of annotation. This pattern holds for both commercial models and our fine-tuned models. For example, o1-preview achieves 56.4% on MATE-N, 65.4% on MATE-S, 77.2% on MATE-T, and 78.6% on MATE-ST, showing consistent improvement as more reasoning guidance is provided. This suggests that the dual-mode reasoning approach combining strategic intuition with tactical calculation is most effective, as the two types of reasoning provide complementary perspectives that together enable better move selection.

4. Few-shot prompting provides modest gains: The improvement from zero-shot to few-shot settings is relatively small for most models, typically in the range of 2-5 percentage points. For instance, o1-preview improves from 76.6% to 78.6% on MATE-ST, and gpt-4 improves from 60.0% to 68.1%. This indicates that in-context learning from a few examples provides limited benefit for this task, likely because chess reasoning requires deep domain knowledge that cannot be easily conveyed through a handful of examples. The task's complexity and the need for genuine understanding rather than format matching limit the effectiveness of few-shot learning.

Table 3.2: Accuracy (%) on positions with 3 candidate moves (zero-shot).

Model	**N**	**S**	**T**	**ST**
gpt-4	37.4	40.1	61.7	56.3
gpt-4o	38.5	40.2	43.2	49.5
o1-mini	25.0	35.0	65.0	60.1
o1-preview	45.0	26.8	70.1	50.2
claude-3.5-sonnet	39.1	42.0	50.4	46.0
claude-3-opus	32.2	41.7	49.4	47.0
gemini-1.5-pro	30.9	41.5	38.1	40.5
gemini-1.5-flash	35.5	35.7	38.3	45.5
Ours (combined)	**40.0**	**56.1**	**57.2**	**54.8**

3.4.2 Ablation Studies

Multiple Candidate Moves

To test whether our approach scales to more challenging evaluation scenarios with larger choice sets, we evaluate models on positions with 3 candidate moves instead of 2, making the task more difficult as models must discriminate among more options. Table 3.2 presents results for this extended evaluation.

With more candidate moves, performance decreases substantially for all models, as expected given the increased difficulty of discriminating among three options rather than two. Random guessing would achieve only 33.3% accuracy compared to 50% in the two-move setting. However, models trained with strategy and tactic explanations show greater robustness to this increased difficulty, maintaining relatively better performance. The relative advantage of our fine-tuned models becomes even more pronounced in this more challenging setting, with our models achieving 54-57% accuracy on most subsets compared to commercial models typically achieving 25-50% accuracy. This demonstrates that the reasoning capabilities learned through training with explicit annotations generalize better to more difficult decision scenarios.

Table 3.3: Cross-evaluation of explanation quality. Rows indicate which model is being evaluated, columns indicate which explanations are used.

Evaluator	**MATE-gpt**	**MATE-claude**	**MATE-ours**
gpt-4o	–	48.6	51.0
claude-3.5-sonnet	52.7	–	56.7
Ours	**74.7**	**75.6**	–

Quality of Generated Explanations

To evaluate whether our fine-tuned models can generate high-quality strategic explanations that match or exceed those from commercial models, we conduct a cross-evaluation study. We create three test sets where the same positions are annotated with explanations from different sources. MATE-ours uses positions with explanations generated by our fine-tuned model, testing whether the model has learned to produce human-quality reasoning descriptions. MATE-gpt uses the same positions with explanations generated by GPT-4o, representing the quality of a leading commercial model. MATE-claude uses the same positions with explanations from Claude-3.5-sonnet, representing another top commercial system.

Each model is then evaluated on all three test sets, allowing us to compare how well different models can utilize explanations from different sources. Table 3.3 presents the cross-evaluation results.

Our model performs best regardless of which explanations are used, achieving 74.7% accuracy with GPT-4o's explanations and 75.6% with Claude's explanations, substantially outperforming the commercial models even when they evaluate their own explanations. This suggests two important findings. First, our training has instilled stronger chess understanding and reasoning capabilities that enable effective utilization of any reasonable explanations. Second, the quality of explanations generated by our model is competitive

with or exceeds those from commercial models, as evidenced by other models performing better on our explanations than on commercial explanations (for instance, Claude achieves 56.7% on our explanations versus 52.7% on GPT's explanations).

Difficulty Consistency Across Subsets

To verify that the performance differences observed across different subsets (MATE-N, MATE-S, MATE-T, MATE-ST) are genuinely due to the presence and type of explanations rather than variations in inherent position difficulty, we conduct a controlled difficulty assessment study.

We randomly sample 50 positions from each subset and ask human chess players (rated 1800-2200 Elo, representing strong club-level to expert-level players) to rate the difficulty of selecting the better move on a 5-point scale without seeing any explanations. We also test multiple commercial models on these same positions in a no-explanation setting to provide an objective measure of difficulty.

The results show that positions have similar difficulty distributions across all subsets, with average human ratings varying by less than 0.3 points on the 5-point scale and commercial model performance varying by less than 5 percentage points when evaluated without explanations. This confirms that performance differences observed in the main experiments are indeed due to the presence and quality of strategic and tactical annotations rather than systematic differences in inherent position difficulty across subsets.

3.4.3 Case Study

Figure 3.4 illustrates detailed responses from three representative models (GPT-4, o1-preview, Claude-3.5-sonnet) when evaluating the same chess position

with both strategy and tactic annotations for two candidate moves.

The case study reveals distinct reasoning patterns across different models. Claude-3.5-sonnet provides detailed qualitative analysis discussing strategic themes and positional factors in depth, demonstrating strong language generation capabilities and chess vocabulary. However, despite this articulate reasoning, it ultimately selects the suboptimal move, suggesting that verbal fluency and chess knowledge are insufficient without proper integration of strategic and tactical considerations.

o1-preview correctly identifies the better move through systematic reasoning that explicitly weighs strategic and tactical factors. The model's response shows step-by-step evaluation of both moves, considering material consequences, positional implications, and tactical sequences before reaching a conclusion. This demonstrates effective integration of the provided explanations into a coherent decision-making process.

GPT-4 provides concise reasoning that focuses on the most critical factors without extensive elaboration. Despite the brevity, it correctly selects the optimal move, suggesting efficient attention to the most relevant considerations. This demonstrates that different reasoning styles can be effective—extensive analysis is not always necessary if attention is properly directed to decisive factors.

This case study illustrates that different models employ different reasoning strategies with varying effectiveness. The most successful models demonstrate ability to integrate multiple types of information (strategic and tactical), maintain focus on decision-relevant factors, and properly weigh competing considerations to reach sound conclusions.

3.5 Discussion

3.5.1 Language Enhances Chess Reasoning

Our experimental results provide strong empirical evidence that language explanations substantially enhance LLM reasoning capabilities in chess. This finding has important implications for debates about the role of language in reasoning and cognition. The consistent improvement observed across all models when provided with strategic and tactical explanations—ranging from 34% to 39% relative improvement for reasoning-focused models like o1—challenges views suggesting that language is not centrally used for reasoning Fedorenko et al. [2024]. Instead, our results align with findings demonstrating that linguistic reasoning and explicit verbalization of thought processes can improve problem-solving across diverse domains Wei et al. [2022b].

The effectiveness of dual-mode annotations combining strategy and tactics is particularly noteworthy and reveals important insights about reasoning architecture. This suggests several key principles about how reasoning benefits from language. First, different types of reasoning—intuitive strategic pattern recognition versus analytical tactical calculation—are complementary rather than redundant. Strategic reasoning provides high-level goals and evaluation principles, while tactical reasoning grounds these principles in concrete sequences and consequences. Second, explicit verbalization of reasoning steps aids in decision-making by making implicit heuristics explicit and checkable. When reasoning is articulated linguistically, errors in logic or overlooked considerations become more apparent. Third, the combination of strategic and tactical annotations provides richer context for evaluation by offering multiple perspectives on the same position, enabling more robust and well-founded decisions.

3.5.2 Integrating Strategic and Tactical Thinking

Our approach of combining long-term strategic planning with short-term tactical calculations directly mirrors how human chess experts think about positions. Expert players do not rely exclusively on one mode of reasoning but fluidly integrate both depending on position characteristics. This dual approach provides several important benefits that explain its effectiveness.

First, it provides both high-level goals through strategic evaluation and concrete sequences through tactical calculation. Strategic thinking identifies what the position requires (improving piece activity, exploiting pawn weaknesses, launching an attack), while tactical thinking determines whether specific moves actually accomplish these goals or create unintended problems.

Second, this framework balances abstract principles with specific calculations. Strategic principles like "control the center" or "improve piece coordination" provide general guidance, but tactics determine whether particular implementations of these principles are actually sound given the specific piece placements and opponent possibilities.

Third, the dual approach offers multiple complementary perspectives for evaluating positions. A move might appear strategically attractive but tactically flawed, or tactically sharp but strategically dubious. Considering both perspectives enables more nuanced evaluation that avoids the pitfalls of relying exclusively on either mode.

This framework of integrating intuitive pattern recognition with systematic calculation may be applicable to other domains beyond chess that require both types of reasoning. Business strategy requires balancing long-term vision with short-term execution details. Military planning requires integrating strategic objectives with tactical operational realities. Game design requires balancing overall player experience goals with specific mechanic implementa-

tions. The principle of explicitly combining these complementary reasoning modes through language may benefit reasoning in these diverse domains.

3.5.3 Limitations of Commercial LLMs

While commercial LLMs demonstrate some chess reasoning capability, achieving accuracies in the 50-78% range on our benchmarks, their performance remains limited compared to specialized fine-tuned models. Several factors contribute to these limitations and reveal important constraints on general-purpose language model reasoning.

Lack of domain expertise: General-purpose models are trained on broad corpora covering countless topics, but they do not receive the concentrated exposure to expert chess annotations that would build deep domain knowledge. While they may have seen chess games and basic explanations during pre-training, they lack the systematic presentation of strategic principles and tactical patterns with clear links to optimal moves that our fine-tuned models receive. This suggests that depth of domain-specific knowledge matters for reasoning quality, not just general intelligence or language understanding.

Difficulty with spatial reasoning: Chess positions involve complex spatial relationships including piece placement, board geometry, movement patterns, and multi-step sequences that unfold across the 8×8 board. While language models excel at sequential text processing, they struggle with the implicit spatial computations required for chess. Encoding positions as FEN strings and moves as UCI notation creates an abstraction barrier that may hinder spatial understanding. This limitation suggests that purely text-based reasoning may be insufficient for domains with inherently spatial or geometric structure without specialized representations or training.

Limited training on annotated positions: Without extensive expo-

sure to expert-annotated positions during pre-training, general-purpose models struggle to learn the effective heuristics and evaluation principles that guide strong play. They may have seen game scores and general chess discussion, but systematic training on positions paired with expert explanations of why particular moves are superior appears necessary for learning robust reasoning strategies. This highlights the value of structured, expert-annotated training data for developing domain reasoning capabilities.

The dramatic improvement achieved through fine-tuning—from approximately 60% accuracy for the best commercial models to 95% accuracy for our specialized models—demonstrates that domain-specific training with appropriate supervision can overcome the limitations of general pre-training. This suggests that hybrid approaches combining the broad capabilities of pre-trained foundation models with targeted fine-tuning on expert-annotated domain data may be most effective for achieving strong reasoning performance in specialized domains.

3.5.4 Implications for Broader AI Reasoning

The success of language-guided reasoning in chess suggests several broader implications for how we approach reasoning in artificial intelligence systems across diverse domains.

Explicit reasoning improves performance: Making reasoning steps explicit through language consistently helps models make better decisions across different architectures and capability levels. This principle aligns with findings on chain-of-thought prompting Wei et al. [2022b], which shows that asking models to articulate their reasoning process before answering improves performance on complex reasoning tasks. Our results extend this finding by demonstrating that training with explicit expert reasoning annotations—rather

than just prompting for reasoning during inference—provides substantial and lasting benefits. This suggests that human-like verbalization of thought processes genuinely benefits AI systems rather than merely serving as interpretable outputs for humans.

Domain expertise matters: While general knowledge and capabilities are valuable for providing a foundation, domain-specific training with expert annotations provides substantial additional benefits that cannot be easily captured through scale or general pre-training alone. The 16.6 percentage point gap between our fine-tuned 8B parameter model and frontier commercial models with orders of magnitude more parameters demonstrates that targeted expertise can overcome sheer scale. This suggests that hybrid approaches combining pre-trained foundation models (providing general language understanding and reasoning capabilities) with specialized fine-tuning (providing domain-specific knowledge and evaluation heuristics) may be most effective for achieving strong performance in specialized reasoning domains.

Multi-modal reasoning: Combining different reasoning modes—strategic versus tactical, intuitive versus analytical, pattern-based versus calculation-based—improves overall performance beyond what either mode achieves independently. This principle of reasoning diversity mirrors findings in ensemble learning and suggests that cognitive architectures should incorporate multiple complementary reasoning strategies rather than relying on a single approach. This principle may extend to other domains requiring multiple types of reasoning, such as scientific problem-solving (combining theoretical principles with experimental design), software engineering (combining high-level architecture with low-level implementation), or medical diagnosis (combining pattern recognition from experience with systematic differential diagnosis).

3.6 Conclusion

This chapter explored LLM reasoning capabilities using chess as a controlled testbed for studying how language explanations enhance decision-making in structured domains. The work makes several key contributions to our understanding of language-based reasoning.

First, we introduced the **MATE dataset**, a large-scale resource comprising approximately 1 million chess positions with expert annotations for strategy and tactics. This dataset provides systematic coverage of chess reasoning across different game phases, strategic themes, and tactical patterns, enabling rigorous study of how language explanations affect reasoning quality. The dataset's design with multiple subsets isolating different annotation types enables precise ablation studies of strategic versus tactical reasoning contributions.

Second, we demonstrated a **dual-mode reasoning framework** showing that combining strategic explanations (articulating long-term positional principles) with tactical explanations (describing concrete move sequences and their consequences) significantly improves performance. This integration of intuitive pattern recognition with systematic calculation mirrors how human experts think about chess and proves more effective than either reasoning mode in isolation. The consistent superiority of combined annotations across all tested models validates the importance of multi-modal reasoning.

Third, we provided **comprehensive empirical validation** showing that fine-tuned models with language explanations substantially outperform state-of-the-art commercial LLMs. Our specialized 8B parameter models achieve 95.2% accuracy compared to 78.6% for the best commercial model (o1-preview), representing a 24.2% error rate reduction. This demonstrates that domain-

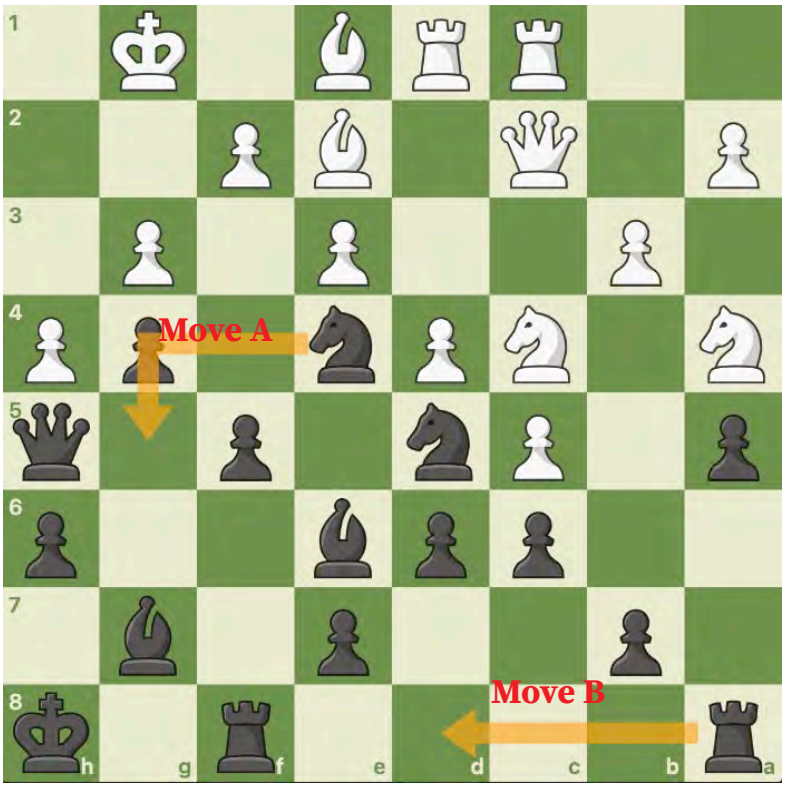
specific training with expert annotations can overcome the advantages of scale and general capabilities present in much larger frontier models.

Fourth, we established an important **theoretical insight** about the role of language in reasoning. The consistent and substantial improvements from language explanations—observed across models of different scales, architectures, and training approaches—provide strong evidence that language plays a functional role in reasoning rather than merely describing reasoning performed through other mechanisms. This challenges views suggesting language is epiphenomenal to reasoning and supports theories positing that linguistic representations genuinely enhance cognitive processes.

These findings provide a foundation for the subsequent chapters of this dissertation, which explore how language-based reasoning can be applied to increasingly complex and realistic domains. While chess represents a fully-observable domain with perfect information and deterministic dynamics, the lessons about dual-mode reasoning, explicit verbalization of reasoning steps, and the benefits of expert annotations extend to more challenging settings.

The next chapter extends these ideas to task and motion planning for robotic manipulation, where agents must reason about both symbolic action sequences and continuous geometric constraints. Unlike chess where all information is provided and all moves are legal subject to rules, robotic manipulation involves physical constraints, uncertain outcomes, and the need to coordinate high-level task planning with low-level motion planning. The principles of language-guided reasoning established in this chapter—particularly the value of making reasoning explicit and combining complementary reasoning modes—inform our approach to these more complex embodied reasoning challenges.

You are an expert chess player. You are given a chess board with FEN format. Your goal is to choose a better move given two candidate moves with their strategy explanations and tactics. The FEN of the given chess board is
r4r1k/1p2p1b1/2ppb2p/p1Pn1p1q/N1NPn1pP/1P2P1P1/P1Q1BP2/2RRB1K1 b - - 0 1



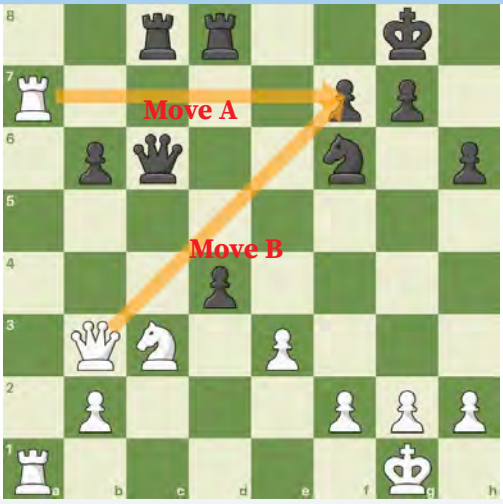
MoveA:e4g5.Black knight jumps into the g5, seduce h-pawn to take it, open the h-file, threaten the white king safety. After e4g5, h4g5, h6g5, black has a lot of initiatives.

MoveB:a8d8.Black rook moves to the center of the board with more activity and waits for the open of the center. After a8d8, e2d3,d5b4,e1b4,a5b4, black has the strong c3 square.

Considering the long-term strategy and short-term tactic, move A is a better move.

Figure 3.3: Example data point from MATE showing position, candidate moves, and annotations for both strategy and tactics.

You are an expert chess player. You are given a chess board with FEN format. Your goal is to choose a better move given two candidate moves with their strategy explanations and tactics. The FEN of the given chess board is 2rr2k1/R4pp1/1pq2n1p/8/3p4/1QN1P3/1P3PPP/R5K1 w - - 0 25. Which move is better?



MoveA:a7f7, Sacrifice a piece to clear a file or diagonal adjacent to the enemy king
TacticA: a7f7 c6c4 b3c4 c8c4 Trade the lower value piece for a higher value piece.

MoveB:b3f7, Offer a piece to unblock a file or diagonal around the enemy king
TacticB: b3f7 g8h8 f7g7 Checkmate! "

The response from Claude 3.5 Sonnet:

To evaluate which move is better, I'll analyze both moves and their proposed tactics:
Move A: a7f7 (Rook from a7 to f7)
Strategy: Sacrifice a piece to clear a file or diagonal adjacent to the enemy king
Tactic: a7f7 c6c4 b3c4 c8c4 (Trade the lower value piece for a higher value piece)
This move sacrifices the rook on a7 to open the a-file. However, the proposed tactic doesn't seem to lead to a significant advantage. After the exchange, Black would still have a material advantage (a rook for a queen), and White's position doesn't seem to have improved significantly.
Move B: b3f7 (Queen from b3 to f7)
Strategy: Offer a piece to unblock a file or diagonal around the enemy king
Tactic: b3f7 g8h8 f7g7 Checkmate!
This move is a queen sacrifice that leads to an immediate checkmate. The queen moves to f7, forcing the black king to h8. Then, the queen delivers checkmate on g7. This tactic is decisive and ends the game immediately.
Conclusion:
Move B (b3f7) is clearly the better move. It leads to an immediate checkmate, which is the ultimate goal in chess. While Move A attempts to create some tactical opportunities, it doesn't lead to a clear advantage and actually loses material. On the other hand, Move B, despite sacrificing the queen (the highest value piece), results in a checkmate, which is worth more than any material advantage.
Therefore, the better move is definitely Move B: b3f7.

Figure 3.4: Case study comparing model responses. The position requires choosing between two moves, each with strategic and tactical implications.

Chapter 4

LLM-BASED TASK AND MOTION PLANNING WITH MOTION FAILURE REASONING

The combination of symbolic reasoning with continuous action is at the heart of intelligent behavior.

— Leslie Kaelbling and Tomás Lozano-Pérez

4.1 Introduction

In Chapter 3, we demonstrated that language explanations enhance LLM reasoning in a structured, fully-observable domain. This chapter extends these ideas to a more challenging setting: robots operating in physical environments where they must plan both symbolic action sequences and continuous motion trajectories.

Sequential manipulation planning requires robots to autonomously perform diverse tasks in complex environments. To successfully complete such tasks, robots must generate executable motions that achieve long-term task objectives. This capability requires several key technical components. First, planning algorithms must be efficient enough to enable responsive operation in real-time scenarios where delays could impact task success. Second, the system needs sophisticated reasoning capabilities to anticipate how environmental changes will affect future actions. Third, there must be effective integration between symbolic task planning, which operates over discrete action sequences, and geometric motion planning, which computes continuous trajectories in configuration space.

Task and Motion Planning (TAMP) addresses this challenge by hierarchically decomposing the planning problem into two complementary stages. The first stage involves high-level symbolic task planning, which focuses on reasoning over long-horizon abstract action sequences such as "pick block A, then place it on block B." The second stage consists of low-level continuous motion planning, which computes feasible robot trajectories that satisfy geometric constraints such as collision avoidance and joint limits. This hierarchical decomposition allows the system to leverage efficient symbolic search for high-level reasoning while using specialized geometric algorithms for continuous planning.

However, a persistent challenge remains in traditional TAMP approaches: how to effectively interface between task and motion planners to generate action sequences that simultaneously satisfy both symbolic goals and continuous geometric constraints.

4.1.1 Motivation

Traditional TAMP approaches rely on manually designed modules to interface between symbolic and continuous domains, as illustrated in Figure 4.1. These interface modules serve two critical functions in the planning pipeline.

Generate action parameters: The first function is to sample real-valued parameters for symbolic actions. For example, when executing a symbolic action like `Place(object, x, y)`, the module must select specific continuous target locations (x, y) from the infinite space of possible placements. Appropriate parameter selection is crucial for motion feasibility, as poorly chosen parameters may lead to collisions, unreachable configurations, or unstable object arrangements.

Incorporate motion feedback: The second function is to translate mo-

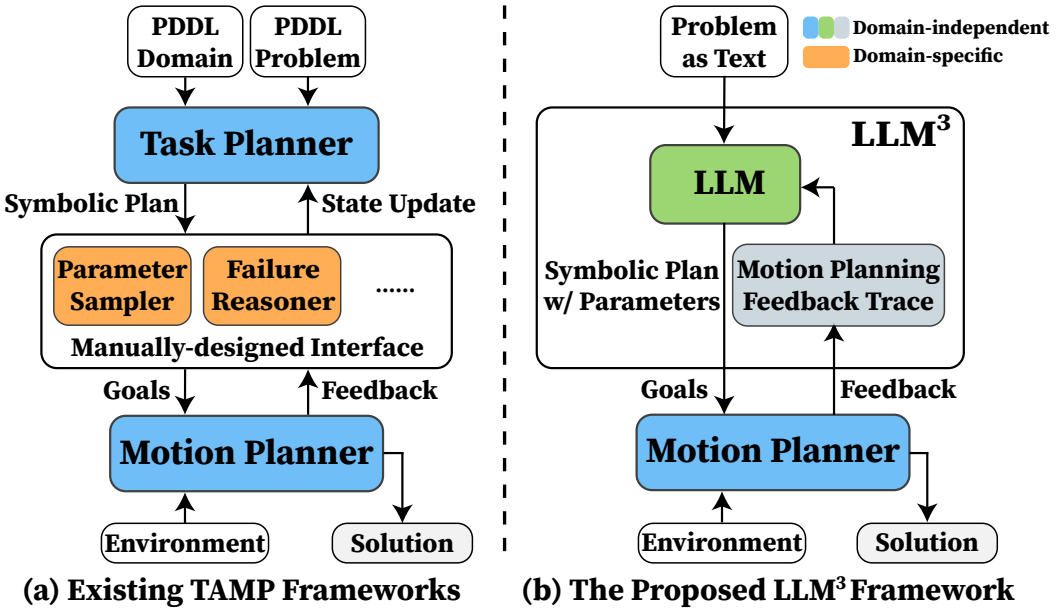


Figure 4.1: Comparison of traditional TAMP frameworks (left) with our LLM^3 framework (right). Traditional approaches require manually designed, domain-specific interface modules, while LLM^3 uses a pre-trained LLM as a domain-independent interface.

tion planning failures into updates to the symbolic state or constraints that guide future planning. When the motion planner reports that a particular action is infeasible due to geometric obstacles or kinematic limitations, the interface module must interpret this failure and communicate relevant information back to the task planner. This enables the task planner to generate improved plans that avoid previously failed action attempts.

Unfortunately, these manually designed interface modules suffer from several significant limitations. First, they are domain-specific and lack generalizability across different task environments or robot platforms. A module designed for tabletop manipulation may not transfer to mobile manipulation or assembly tasks. Second, these modules are labor-intensive to design and implement, requiring significant engineering effort from domain experts who understand both the symbolic planning formalism and the geometric motion planning algorithms. Third, the modules are difficult to adapt to new tasks

or environments, as each change may require reimplementing or substantially modifying the interface logic.

4.1.2 Key Insight: LLMs as Domain-Independent Interfaces

Recent Large Language Models demonstrate emergent capabilities that suggest they could serve as general-purpose interfaces between symbolic and continuous planning. Specifically, LLMs have shown remarkable performance in several relevant areas. They exhibit strong reasoning and planning capabilities, as demonstrated by their ability to decompose complex tasks and generate logical action sequences Huang et al. [2022b], Wei et al. [2022b]. They can generate continuous parameters based on natural language descriptions and spatial reasoning Mirchandani et al. [2023]. Additionally, they demonstrate the ability to process environment feedback and iteratively refine their outputs based on execution results Huang et al. [2023].

Our key insight is that these pre-trained LLM capabilities could provide a general, domain-independent approach to interfacing between symbolic and continuous planning domains. Rather than manually designing specialized modules for each domain, we can leverage the broad knowledge and reasoning capabilities already encoded in large-scale language models. This approach eliminates the need for domain-specific manual module design while potentially providing more flexible and generalizable solutions.

4.1.3 Contributions

We present LLM^3 (**L**arge **L**anguage **M**odel-based Task and **M**otion Planning with **M**otion Failure Reasoning), an LLM-powered TAMP framework that integrates language models throughout the planning pipeline. The framework makes three core technical contributions.

First, the system **proposes action sequences** by leveraging LLM commonsense knowledge and task understanding to generate symbolic action plans that work toward achieving specified task goals. The LLM reasons about object relationships, spatial constraints, and action dependencies to produce coherent high-level plans.

Second, the framework **selects action parameters** by using LLM spatial reasoning heuristics to generate continuous parameter values that are likely to result in feasible motion. Rather than randomly sampling from parameter spaces or using hand-crafted heuristics, the LLM applies implicit spatial understanding to suggest placements that avoid obvious obstacles and respect reachability constraints.

Third, the system **reasons about motion failures** by categorizing motion planning feedback into semantically meaningful failure modes and iteratively refining plans through explicit reasoning about why previous attempts failed and how to avoid similar failures in subsequent iterations.

These contributions provide several key advantages over traditional TAMP approaches. The framework requires no manually designed symbolic domain files or PDDL specifications, as the LLM can understand task goals and available actions from natural language descriptions. It provides domain-independent informed parameter sampling that applies across different manipulation tasks without task-specific tuning. Additionally, the approach offers planner-independent motion failure reasoning that works with any motion planning algorithm capable of providing basic feedback about collision and reachability.

4.2 Problem Formulation

4.2.1 TAMP Problem Definition

A Task and Motion Planning (TAMP) problem is formally defined as a tuple $\langle \mathcal{O}, \mathcal{S}, \mathcal{A}, \mathcal{T}, s_0, g \rangle$ where each component captures a different aspect of the planning problem.

Objects $\mathcal{O}$ represent the set of manipulable entities and environmental fixtures present in the scene. This includes both objects that the robot can grasp and move (such as blocks, tools, or containers) as well as static environmental elements (such as tables, shelves, or walls) that affect motion feasibility.

State space $\mathcal{S}$ is factorized with respect to the objects in the environment, meaning that each state $s \in \mathcal{S}$ can be decomposed into individual object states. Each object state comprises various attributes including 3D position and orientation in world coordinates, geometric dimensions and shape representations, physical properties such as mass and friction coefficients, and relational properties such as which objects are grasping or supporting other objects.

Actions $\mathcal{A}$ represent primitive robotic operations with low-level trajectory execution handled by the motion planning module. Each action $a \in \mathcal{A}$ is parameterized by two types of variables. First, object variables $\bar{o}$ specify which objects in the environment are involved in the action, such as which object to pick or where to place it. Second, continuous parameters θ provide real-valued specifications such as target positions, approach angles, or grasp configurations.

A ground action is obtained by instantiating the action template with specific objects $o = (o_1, \ldots, o_m)$ from the environment and concrete parameter values. For example, a symbolic `Place` action becomes grounded as

`Place(block, [0.1, 0.2])` when instantiated with a specific block object and target position coordinates.

The continuous parameters θ serve as goals for the motion planning algorithm, specifying desired end-effector poses or object configurations that the planner must reach. An action a is considered feasible when the motion planner successfully finds a collision-free trajectory τ that achieves the specified goal configuration. We denote feasible actions that have been validated with trajectories as $a(\tau)$ to distinguish them from abstract symbolic actions.

Transition function $\mathcal{T}$ models the dynamics of the environment by computing the next state s_{t+1} that results from executing a feasible action $a_t(\tau_t)$ in the current state s_t:

$$s_{t+1} = \mathcal{T}(s_t, a_t(\tau_t))$$

The transition function captures how robot actions change object poses, update support relationships, and modify other relevant state variables. We assume that $\mathcal{T}$ can be evaluated using a black-box physics simulator that accurately models rigid body dynamics, contact mechanics, and other relevant physical phenomena.

Initial state $s_0 \in \mathcal{S}$ specifies the starting configuration of all objects in the environment, including the robot's initial configuration, before any actions have been executed.

Goal function $g : \mathcal{S} \to \{0, 1\}$ is a boolean predicate that checks whether the task goal has been achieved in a given state. The goal function returns 1 if all goal conditions are satisfied (such as specific objects being in target locations or satisfying desired spatial relationships) and returns 0 otherwise.

The objective of TAMP is to derive a sequence of feasible actions that accomplishes two requirements simultaneously. First, executing this sequence must achieve the task goal, formally expressed as $g(s_{T+1}) = 1$ where s_{T+1} is

the final state after executing all actions. Second, the sequence must be physically executable, meaning that each state transition must follow the dynamics: $s_{t+1} = \mathcal{T}(s_t, a_t(\tau_t))$ for all time steps $t = 0, 1, \ldots, T$.

4.2.2 Motion Planning Preliminaries

We realize each ground action a by computing a collision-free trajectory τ using sampling-based motion planning algorithms such as Bidirectional Rapidly-exploring Random Trees (BiRRT) **?**. These algorithms provide probabilistic completeness guarantees while remaining efficient for high-dimensional configuration spaces typical in robotic manipulation.

The motion planner receives as **input** the current environment state, including the poses of all objects and obstacles, the robot's current joint configuration, and the goal pose of the robot's end-effector as specified by the action parameters θ. The planner must find a path through configuration space that transitions from the current robot state to a configuration that achieves the desired end-effector pose.

The motion planner produces as **output** either a collision-free trajectory τ in joint space that smoothly reaches the goal pose while avoiding collisions with environmental obstacles and respecting joint limits, or a failure indication when no such trajectory can be found within computational budget constraints.

By default, standard motion planning algorithms report only binary success or failure outcomes, providing insufficient information for higher-level reasoning about why failures occur. To enable effective failure reasoning, we enhance the motion planner to synthesize semantically meaningful feedback that categorizes different failure modes. This enriched feedback allows the LLM to understand not just that an action failed, but why it failed and what con-

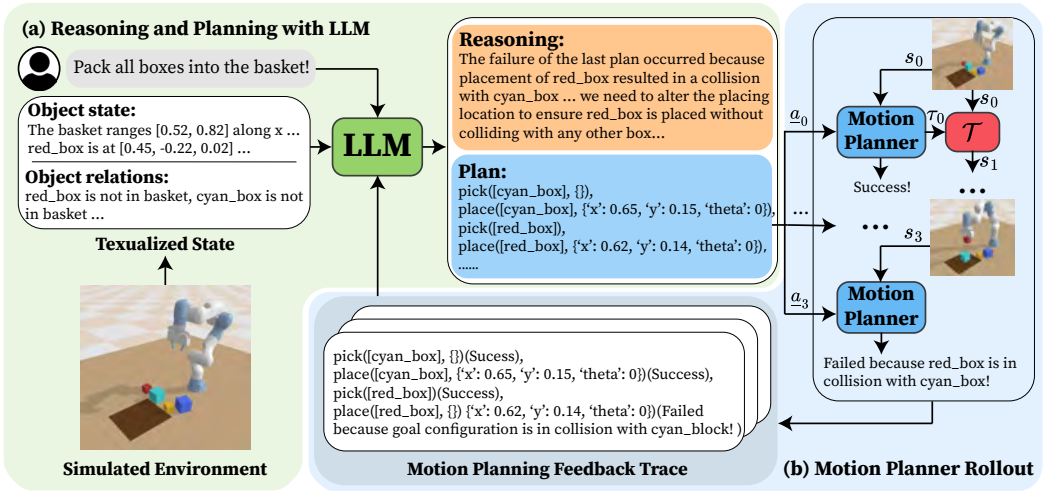


Figure 4.2: System diagram of LLM^3 framework. (a) LLM generates reasoning and action sequences. (b) Motion planner verifies feasibility and provides feedback.

straints were violated, enabling more informed replanning.

4.3 The LLM^3 Framework

Figure 4.2 illustrates the overall architecture of the LLM^3 framework, showing how the LLM-based reasoning module interacts with the motion planning module through an iterative refinement loop.

4.3.1 Framework Overview

Algorithm 1 outlines the complete LLM^3 framework, which operates through iterative cycles that alternate between two main phases.

The first phase involves **reasoning and planning with the LLM**, where the language model analyzes the current state and any accumulated failure feedback to generate a new action sequence. The LLM produces both explicit reasoning about the task and a concrete sequence of actions with parameters.

The second phase focuses on **verifying with the motion planner**, where each proposed action is validated by attempting to compute a collision-free

Algorithm 1 LLM^3 for TAMP

Require: Pre-trained LLM, transition function $\mathcal{T}$ with motion planner MP, goal function g, initial state s_0, max attempts N_{max}, max trace size k

Ensure: Success indicator, feasible action plan

$plan \leftarrow [], success \leftarrow False, feedback \leftarrow [], trace \leftarrow []$
$s \leftarrow s_0, iter \leftarrow 0$
while not $success$ and $iter < N_{max}$ **do**
  $reason, llm\_plan \leftarrow LLM(s_0, trace)$
  $iter \leftarrow iter + 1$
  for a in llm_plan **do**
    $\tau, mp\_feedback \leftarrow MP(s; a)$
    $feedback$.append$((a, mp_feedback))$
    if τ is None **then**
      break {Terminate rollout if infeasible}
    end if
    $s \leftarrow \mathcal{T}(s, a(\tau))$
    $plan$.append$(a(\tau))$
  end for
  $success, task\_feedback \leftarrow g(s)$
  if not $success$ **then**
    $feedback$.append$(task_feedback)$
  end if
  $trace$.append$(feedback)$
  $trace \leftarrow trace[-k:]$ {Keep last k feedbacks}
  $feedback \leftarrow []$
end while
return $success, plan$

trajectory. The motion planner provides detailed feedback about feasibility, including categorized information about why actions fail if they cannot be executed.

At each planning iteration, the framework executes a structured sequence of operations. First, the LLM receives the current state s and the accumulated feedback trace from previous failed attempts, then outputs both explicit reasoning about the task and a proposed action sequence. Second, the motion planner attempts to find collision-free trajectories for each action in the sequence, validating whether the proposed actions are geometrically feasible. Third, detailed feedback is synthesized for each motion planning query, cat-

egorizing the outcome as either successful, colliding, or unreachable. Fourth, if any action proves infeasible or if the final state does not satisfy the goal condition, the corresponding feedback is appended to the trace for use in the next iteration. Finally, the process repeats until either the task succeeds or the maximum number of planning attempts is exceeded.

This iterative structure naturally integrates exploration of the action space with progressive refinement based on execution feedback, enabling the system to handle complex long-horizon manipulation tasks that require multiple planning attempts to resolve geometric conflicts.

4.3.2 Reasoning and Planning with LLM

Following recent work demonstrating the effectiveness of explicit reasoning in language models Wei et al. [2022b], Huang et al. [2023], we prompt a pre-trained LLM to generate two types of outputs in sequence. First, the LLM produces explicit reasoning about motion failures from previous attempts, analyzing why certain actions failed and what constraints must be satisfied in future attempts. Second, based on this reasoning, the LLM generates concrete action sequences in a structured text format that can be parsed and executed by the motion planner.

We employ zero-shot prompting combined with Chain-of-Thought (CoT) reasoning to elicit this structured output. The LLM generates its response auto-regressively, first producing reasoning tokens that explain its understanding of the task and failures, then producing action tokens that specify the actual plan to execute.

Prompt Structure The prompt provided to the LLM consists of several carefully structured components that provide context and guide the generation

process.

The **system message** provides global context that activates the LLM's planning and reasoning capabilities. This message typically describes the LLM's role as a robotic task planner and sets expectations about the type of reasoning and output format required.

The **task description** specifies the manipulation environment, including which objects are present and their properties, what goal the robot must achieve, and what primitive actions are available for execution. This description is provided in natural language rather than formal planning languages, leveraging the LLM's ability to understand flexible task specifications.

The **initial state** provides a textualized representation of the current environment configuration, listing object poses, spatial relationships, and other relevant state information in a format that the LLM can process. This representation is automatically generated from the simulator state.

The **feedback trace** contains motion planning feedback from previous planning attempts, organized chronologically to show the progression of attempted solutions and their failure modes. This trace is the only component that updates across planning iterations, providing the LLM with information about what has been tried and why those attempts failed.

The **output format specification** describes the desired structure for the LLM's response, including how to format reasoning statements and how to specify action sequences with parameters. This ensures the output can be reliably parsed for execution.

Most prompt content remains fixed across planning iterations, providing stable context about the task and environment. Only the feedback trace component updates dynamically as new failed attempts are added, allowing the LLM to build on previous experience while maintaining consistent understand-

ing of the task goals and available actions.

Two Planning Strategies We implement two complementary strategies for generating improved action sequences after failures, each suited to different types of planning challenges.

The **backtrack variant** instructs the LLM to identify the last successfully executed action in the previous failed attempt, then backtrack to that point and continue generating a modified plan from there. This strategy preserves the successful prefix of the previous plan while only modifying the actions that led to failure. This approach is particularly appropriate when failure occurs late in the action sequence, as it avoids redundantly replanning actions that have already been validated as feasible.

The **from-scratch variant** instructs the LLM to generate an entirely new action sequence that learns from previous failures but does not necessarily preserve any specific actions from earlier attempts. This strategy provides more flexibility to explore fundamentally different solution approaches and is better suited to situations where early actions in the previous plan were problematic or when the overall strategy needs to be reconsidered.

Figure 4.3 presents the detailed prompt templates for both variants, with variant-specific instructions highlighted in different colors.

4.3.3 Synthesizing Motion Planning Feedback

Standard motion planning algorithms typically report only binary success or failure outcomes, which provides insufficient information for effective reasoning about geometric constraints. A simple "failure" indication does not distinguish between different failure causes that require different solution approaches. To enable more effective LLM reasoning, we synthesize categorized feedback that

explains why motion planning fails in semantically meaningful terms.

Motion Failure Taxonomy We identify and categorize the motion planning outcomes into three distinct cases that cover the most common scenarios in manipulation planning.

(A) Collision failures occur when the goal configuration specified by the action parameters places the robot or manipulated object in collision with environmental obstacles or other objects. Even though the goal configuration may be kinematically reachable, the motion planner cannot find a collision-free path to reach it. These failures indicate that the selected placement location is geometrically infeasible due to interference with the environment.

(B) Unreachability failures arise when the goal configuration has no feasible inverse kinematics (IK) solution, meaning it lies beyond the robot's reachable workspace. The target location may be too far from the robot base, require joint angles beyond mechanical limits, or violate kinematic constraints. These failures indicate fundamental limitations in where the robot can physically position its end-effector.

(C) Feasible outcomes represent successful motion planning where the goal configuration is both collision-free and reachable. The motion planner successfully computes a valid trajectory, and the action can proceed to execution.

Figure 4.4 illustrates representative examples of these three outcome categories in a manipulation scenario.

In practical implementation, we integrate the motion planning module with specialized tools to efficiently detect and categorize these failure modes. An inverse kinematics solver runs first to quickly determine if the goal configuration is even kinematically feasible before invoking the expensive trajectory planning

algorithm. A collision checker evaluates whether goal configurations or trajectory waypoints intersect with environmental geometry. Together, these tools enable fast categorization of failures into the taxonomy, providing structured feedback that the LLM can reason about effectively.

This design choice is practically effective because it covers the vast majority of failure modes encountered in manipulation planning while remaining simple enough for reliable detection. The categorized feedback provides actionable information for the LLM to generate improved plans that address specific geometric constraints.

4.4 Experiments

4.4.1 Experimental Setup

We evaluate LLM^3's performance in a simulated manipulation environment built using the PyBullet physics engine, which provides realistic rigid body dynamics and collision detection. The evaluation focuses on box-packing tasks that require reasoning about object placement, collision avoidance, and workspace reachability.

Task Description The core task requires the robot to place a set of objects from a tabletop into a basket that is also located on the table. These tasks systematically vary across several dimensions to test different aspects of the planning system. The number of objects ranges from small sets of 3-4 items to larger sets of 8-10 items, testing scalability to longer action sequences. The sizes of objects vary, with some sets containing many small objects that can be placed freely and others containing larger objects that require more careful placement to avoid collisions. The basket size and position also varies, with some baskets providing ample space and being centrally located, while

others are smaller or positioned at the edge of the robot's workspace where some regions may be unreachable. Finally, the likelihood of collisions varies depending on how densely packed the basket becomes as more objects are placed.

Figure 4.5 illustrates the simulation environment and the specific task configurations used in the evaluation.

Settings We design two complementary experimental settings that emphasize different planning challenges.

Setting 1 uses three different object sets with progressively increasing total volume, while keeping the basket size constant across all trials. The small object set (Easy) contains items that comfortably fit with minimal risk of collision. The medium object set (Medium) begins to crowd the basket, requiring more careful placement to avoid collisions between objects. The large object set (Hard) significantly fills the available basket space, necessitating precise reasoning about how to arrange objects to avoid collisions. This setting primarily tests the system's ability to reason about object-object collisions and spatial arrangement within limited space.

Setting 2 uses the largest object set from Setting 1 but varies the basket size across three conditions: small, medium, and large baskets. Importantly, the larger baskets are positioned such that some portions extend beyond the robot's reachable workspace, creating unreachability constraints. The small basket is fully reachable but provides very limited space. The medium basket offers more space but has some unreachable regions near its far edge. The large basket provides ample space but has substantial portions that the robot cannot reach. This setting primarily tests the system's ability to reason about robot workspace limits and reachability constraints.

Implementation Details The experimental system integrates several components to enable comprehensive evaluation. We use GPT-4 Turbo as the underlying language model, querying it through the API to generate reasoning and action plans. For motion planning, we implement the BiRRT algorithm, which provides reliable performance for manipulation tasks while remaining computationally efficient enough for iterative planning. The system runs 10 independent trials for each experimental condition to ensure statistical reliability of the results, with random variations in initial object positions across trials.

Evaluation Metrics We measure system performance using three complementary metrics that capture different aspects of planning effectiveness.

The **Success Rate (%SR)** measures the percentage of task instances where the robot successfully places all required objects into the basket and achieves the goal condition. This metric captures the overall reliability and robustness of the planning approach, with higher values indicating more consistent success across varied scenarios.

The **Number of LLM Calls (#LM)** counts how many times the system queries the language model during the planning process for a given task. Each query represents a complete planning iteration where the LLM generates reasoning and a new action sequence. Lower values indicate more efficient planning that finds feasible solutions with fewer iterations.

The **Number of Motion Planner Calls (#MP)** counts how many times the motion planning module is invoked to verify action feasibility. Since motion planning involves expensive geometric computation, this metric reflects the computational cost of the planning process. Lower values indicate that the system generates higher-quality action proposals that require less motion

planning validation.

Together, these metrics provide a comprehensive view of both effectiveness (success rate) and efficiency (planning iterations and computational cost).

4.4.2 Baseline Methods

We compare LLM^3 against several baseline approaches that ablate different components of the framework, allowing us to isolate the contribution of motion feedback and different planning strategies.

LLM^3 Backtrack represents the complete proposed framework using the backtracking strategy. This method includes motion feedback and iteratively refines plans by modifying actions after the last successful step.

Backtrack (no feedback) implements the backtracking strategy but provides the LLM with only binary success/failure information rather than categorized motion feedback. This ablation tests whether detailed feedback categories (collision vs. unreachability) provide additional value beyond knowing that an action failed.

LLM^3 Scratch represents the complete proposed framework using the from-scratch replanning strategy. This method includes motion feedback but generates entirely new action sequences after each failure rather than preserving successful prefixes.

Scratch (no feedback) implements from-scratch replanning with only binary feedback. The LLM generates a single initial plan without iterative refinement based on motion planning outcomes, testing the value of feedback-driven iteration.

These baselines allow systematic comparison of the contributions of detailed motion feedback (feedback vs. no feedback) and planning strategies (backtrack vs. from-scratch).

Table 4.1: Ablation study results for LLM^3 framework

Method	Setting 1 - Easy			Setting 1 - Medium			Setting 1 - Hard		
	%SR	#LM	#MP	%SR	#LM	#MP	%SR	#LM	#MP
LLM^3 Backtrack	**100**	**1.6**	**11.8**	**100**	**4.4**	**28.4**	**60**	**11.4**	**39.8**
Backtrack	100	1.8	12.6	90	6.3	32.0	40	15.1	55.3
LLM^3 Scratch	100	1.7	13.2	100	7.0	46.1	70	8.8	30.9
Scratch	100	2.4	17.6	60	12.3	45.3	50	13.7	42.8

Method	Setting 2 - Small			Setting 2 - Medium			Setting 2 - Large		
	%SR	#LM	#MP	%SR	#LM	#MP	%SR	#LM	#MP
LLM^3 Backtrack	**60**	**11.4**	**39.8**	**80**	**9.5**	**50.0**	**50**	**13.5**	**44.8**
Backtrack	40	15.1	55.3	30	14.6	16.0	30	15.8	48.0
LLM^3 Scratch	70	8.8	30.9	70	11.5	50.2	60	10.6	32.0
Scratch	50	13.7	42.8	30	16.2	45.7	40	13.2	24.0

4.4.3 Main Results

Table 4.1 presents comprehensive experimental results across both settings and all difficulty levels, showing success rates and efficiency metrics for each method.

Key Findings The experimental results reveal several important insights about the effectiveness of motion feedback and the comparison of planning strategies.

1. Motion feedback significantly improves success rates: The LLM^3 variants that incorporate categorized motion feedback consistently achieve higher success rates compared to their counterparts without feedback across all difficulty levels and settings. For example, in the most challenging Setting 1-Hard scenario, LLM^3 Backtrack achieves a 60% success rate compared to only 40% for Backtrack without feedback, representing a 50% relative improvement. Similarly, in Setting 2-Medium, LLM^3 Backtrack reaches 80% success versus 30% without feedback. These substantial improvements demonstrate that understanding why actions fail (collision vs. unreachability) enables much more

effective replanning than simply knowing that failures occurred.

2. Feedback reduces planning iterations and computational cost: Methods incorporating motion feedback require significantly fewer LLM calls (#LM) and motion planning queries (#MP) to achieve success, indicating more efficient planning processes. For instance, in Setting 1-Hard, LLM^3 Backtrack requires only 11.4 LLM calls compared to 15.1 for the version without feedback. The reduction in motion planning calls is even more dramatic: 39.8 versus 55.3, representing a 28% decrease in expensive geometric computations. This efficiency gain compounds over multiple tasks, substantially reducing total planning time in practical deployments.

3. No clear advantage of backtracking versus from-scratch strategies: Both planning strategies show comparable overall performance, though they exhibit different behavioral characteristics. The backtrack approach tends to adjust the continuous parameters of failed actions while preserving successful prefixes, leading to more conservative modifications that build incrementally on previous attempts. The from-scratch approach samples entirely new action sequences that may explore fundamentally different strategies, providing more flexibility but potentially discarding useful partial solutions. The choice between strategies may depend on task characteristics, with backtracking better for tasks where early actions are clearly correct and from-scratch better when the overall approach needs reconsideration.

4. Robust overall performance across diverse scenarios: Aggregating across all experimental settings and difficulty levels, LLM^3 achieves a strong 91.3% overall success rate. This demonstrates robust performance across varied manipulation challenges involving different types of geometric constraints, from object-object collisions in crowded spaces to workspace reachability limitations. The framework successfully handles both failure modes

Table 4.2: Comparison of parameter sampling strategies

Method	**#Iterations**	**#MP**
Random Samples	109.6	663.1
LLM	10.8	70.2
LLM + Feedback	**7.9**	**53.2**

through appropriate reasoning and replanning.

4.4.4 Action Parameter Selection Study

To specifically evaluate the LLM's capability as an informed action parameter sampler, we conduct a focused experiment comparing three different approaches to selecting continuous parameters for placement actions.

Random sampling serves as a baseline where placement locations are uniformly sampled from the available basket space without any intelligent guidance. This represents the traditional approach of uninformed parameter sampling that relies purely on the motion planner to validate feasibility.

LLM sampling uses the language model to select placement parameters based on its understanding of the task, object sizes, and spatial constraints. The LLM reasons about where objects should be placed to avoid collisions and respect workspace limits, but does not receive feedback about motion planning outcomes.

LLM + Feedback extends LLM sampling by incorporating categorized motion planning feedback, allowing the LLM to iteratively refine its parameter selection based on which placements actually succeed or fail.

Table 4.2 presents the results of this comparison, measured by the number of planning iterations and total motion planning calls required to complete tasks.

The results demonstrate dramatic differences in planning efficiency across

the three approaches. Random sampling requires an average of 109.6 iterations and 663.1 motion planning calls to find feasible action sequences, reflecting the inefficiency of uninformed parameter selection in high-dimensional continuous spaces. LLM-based sampling reduces this computational cost by an order of magnitude, requiring only 10.8 iterations and 70.2 motion planning calls. This demonstrates that the LLM's implicit spatial reasoning provides strong heuristics for parameter selection even without explicit feedback. Adding categorized motion feedback further improves efficiency to 7.9 iterations and 53.2 planning calls, representing a 24% improvement over LLM sampling alone and a 92% reduction compared to random sampling.

These results confirm that LLMs can serve as effective informed parameter samplers, dramatically reducing the number of motion planning queries needed to find feasible action sequences. The integration of feedback enables even more efficient parameter selection by allowing the LLM to learn from failures and focus its sampling on more promising regions of the parameter space.

4.4.5 Real Robot Experiments

To validate that LLM^3 transfers effectively from simulation to physical systems, we deploy the framework on a Franka Research 3 robotic manipulator operating in a real household environment. We evaluate performance on two tasks that present different challenges typical of real-world manipulation scenarios.

Task 1: Object localization and manipulation with partial information

This task requires the robot to place three objects—a basket, a cabbage, and a cup—onto a coffee table. The challenge is that the cup's location is initially unknown to the robot, requiring the system to infer its likely location

and adapt when predictions prove incorrect. The robot must first use the LLM to generate a hypothesis about where the cup might be located based on commonsense knowledge about typical household object placements. When the robot navigates to the predicted location and does not find the cup through visual perception, the system must update its belief about the environment and generate a new prediction. The robot continues searching predicted locations until it successfully locates the cup, then completes the task by placing all three objects on the coffee table. This scenario tests the framework's ability to handle incomplete state information and adapt its plans based on perception feedback.

Task 2: Complex exception handling with multiple obstacles

This more challenging task requires placing a tissue box onto a coaster, but multiple obstacles must be overcome first. The tissue box is located inside a closed closet, presenting an inaccessibility constraint. Additionally, a cup blocks access to the tissue box within the closet, and a tea can occupies space on the coaster that would cause collision with the placed tissue box. The robot must reason through this cascade of obstacles in an appropriate order. First, it must open the closet to gain access to its contents, resolving the inaccessibility constraint. Second, it must remove the blocking cup by temporarily placing it in an alternative location. Third, it must retrieve the tissue box from the now-accessible closet. Fourth, it must remove the tea can from the coaster to create space for the tissue box. Finally, it successfully places the tissue box on the cleared coaster. This scenario tests the framework's ability to handle complex exception cascades that require multi-step corrective actions.

Figure 4.6 shows photographic sequences of the robot executing both tasks.

The robot successfully completed both tasks without manual intervention, demonstrating that LLM^3 handles real-world challenges effectively. In par-

ticular, the system proved robust to perception noise from RGB-D sensors, execution uncertainty from contact dynamics and object slipping, and the accumulated effects of small errors across multiple manipulation actions. These real-world experiments validate that the framework's combination of LLM reasoning and motion planning feedback transfers effectively from simulation to physical manipulation, handling the increased uncertainty and complexity inherent in real robotic systems.

4.5 Discussion

4.5.1 Domain-Independent Interface

A primary contribution of LLM^3 is eliminating the need for manually designed, task-specific interface modules between symbolic task planning and continuous motion planning. Traditional TAMP systems require expert engineers to design separate modules for each new domain, encoding domain knowledge about appropriate parameter sampling strategies, failure interpretation logic, and replanning heuristics. In contrast, LLM^3 leverages the pre-trained language model to serve multiple roles simultaneously within a single unified interface.

As a task planner, the LLM leverages its broad commonsense knowledge to generate action sequences that progress toward task goals. It understands object relationships, typical manipulation strategies, and task decomposition without requiring explicit domain models or symbolic action specifications. As an informed parameter sampler, the LLM uses implicit spatial reasoning heuristics to suggest continuous parameter values that are likely to be geometrically feasible, considering factors like object sizes, collision avoidance, and workspace constraints. As a failure reasoner, the LLM processes catego-

rized motion planning feedback to understand what went wrong in previous attempts and generates improved plans that address specific geometric constraints violated by earlier action proposals.

This domain-independent approach significantly reduces the engineering effort required to deploy TAMP systems in new environments or for new tasks. Rather than spending weeks or months implementing specialized interface modules, practitioners can adapt the system to new domains primarily by modifying natural language descriptions in the prompts. This democratizes access to sophisticated task and motion planning capabilities.

4.5.2 Motion Failure Reasoning

The categorization of motion planning failures into semantically meaningful types—collision versus unreachability—enables the LLM to perform targeted refinement of action plans based on specific violated constraints. This structured feedback is substantially more informative than binary success/failure signals because it indicates what type of geometric problem must be addressed.

When the system identifies a **collision failure**, the LLM understands that the selected placement location interferes with environmental obstacles or other objects. The appropriate response is to adjust the placement location to avoid these obstacles, perhaps by selecting a different region of the target receptacle or reorienting the object. The LLM can reason about spatial arrangements to find alternative placements that maintain the symbolic goal while satisfying geometric constraints.

When the system detects an **unreachability failure**, the LLM understands that the target location lies beyond the robot's kinematic workspace. The appropriate response is fundamentally different: rather than fine-tuning the placement location, the LLM must select an entirely different region that

falls within the robot's reachable space. This might involve placing objects closer to the robot base or selecting alternative surfaces that are more accessible.

By distinguishing these failure modes, the framework enables more efficient replanning that addresses the actual constraint violations rather than blindly trying alternative parameters. The experimental results confirm this advantage: methods with categorized feedback consistently outperform those with only binary feedback, particularly in challenging scenarios where multiple types of geometric constraints must be satisfied simultaneously.

4.5.3 Efficiency Gains from Informed Parameter Sampling

By using LLM heuristics for action parameter selection, LLM^3 achieves substantial reductions in computational cost compared to uninformed random sampling approaches. The parameter selection study demonstrates that LLM-based sampling reduces the number of motion planning queries by approximately 90% compared to random sampling (from 663.1 calls to 70.2 calls on average).

This efficiency gain has important practical implications. Motion planning queries involve expensive geometric computations that can take hundreds of milliseconds to seconds per query, depending on environment complexity and the planning algorithm used. Reducing the number of queries by an order of magnitude translates directly to proportional reductions in total planning time, enabling more responsive robot operation. In resource-constrained settings such as battery-powered mobile manipulators, this computational efficiency also reduces energy consumption. For systems that must perform many manipulation tasks throughout the day, the compounding effect of more efficient planning enables completing substantially more tasks within the same

time budget.

The 36.1% reduction in motion planning calls achieved by adding feedback (from 70.2 to 53.2 calls) provides additional efficiency improvements beyond the benefits of informed sampling alone. This demonstrates that the combination of LLM spatial reasoning with iterative refinement based on motion feedback creates a virtuous cycle where the system progressively learns which parameter selection strategies work well for the current environment and task.

4.5.4 Limitations

Despite its advantages and demonstrated effectiveness, LLM^3 has several important limitations that suggest directions for future research and development.

Long-horizon task challenges: As task horizons increase beyond 15-20 actions, maintaining plan coherence and consistency becomes increasingly difficult for the LLM. With longer sequences, the language model may generate inconsistent actions that contradict earlier decisions, repetitive actions that cycle through the same failed approaches, or actions that lose sight of the overall task goal. This limitation stems from the LLM's bounded context window and the challenge of maintaining long-range dependencies in autoregressive text generation.

Geometric reasoning limitations: While LLMs demonstrate impressive spatial reasoning capabilities, they occasionally struggle with precise geometric reasoning required for tight manipulation constraints. The models may suggest placements that are infeasible due to subtle geometric conflicts not captured in the textualized state representation. They may also have difficulty reasoning about three-dimensional spatial relationships when these relationships are described textually rather than through visual or geometric

representations. This limitation becomes more pronounced in tasks requiring precise positioning or complex spatial arrangements.

Incomplete failure mode coverage: Our two-category motion failure taxonomy (collision and unreachability) covers the most common failure modes in manipulation planning but may miss specialized constraints that arise in particular domains. For example, the current taxonomy does not explicitly address dynamic stability constraints where an object placement would cause tipping or falling, force and torque limitations where the robot cannot generate sufficient force to move heavy objects or maintain grasps, or dexterous manipulation constraints involving finger placement and grasp quality. Extending the taxonomy to cover these additional failure modes would require more sophisticated feedback synthesis and potentially domain-specific failure detection modules.

Computational cost and latency: Each LLM query incurs API latency that can range from several seconds to tens of seconds depending on response length and API server load. For time-critical applications where rapid response is essential, such as reactive manipulation in dynamic environments or human-robot collaboration scenarios requiring low latency, this overhead may be prohibitive. The current system's iterative nature means that multiple queries may be needed to find feasible plans, multiplying the total latency. Future work could address this through model distillation to faster local models, prompt caching to reduce redundant computation, or parallel generation of multiple candidate plans.

4.6 Conclusion

This chapter presented LLM^3, a framework that leverages Large Language Models as domain-independent interfaces for Task and Motion Planning. The

work makes several key contributions to the integration of symbolic reasoning with continuous motion planning.

First, we introduced a **novel framework architecture** that represents the first TAMP system to holistically use language models throughout the planning pipeline—for high-level task planning, continuous parameter selection, and failure reasoning. This unified approach eliminates the need for separate handcrafted modules that traditionally interface between symbolic and geometric planning levels.

Second, we developed a **motion failure taxonomy** that categorizes planning outcomes into semantically meaningful types (collision, unreachability, success). This categorization enables effective LLM reasoning about why specific actions fail and how to generate improved plans that address violated geometric constraints. The structured feedback proves substantially more valuable than binary success/failure signals for iterative plan refinement.

Third, we provided **comprehensive empirical validation** demonstrating that LLM^3 achieves a 91.3% overall success rate across diverse manipulation scenarios while reducing motion planning calls by 36.1% compared to methods without feedback. These results confirm that language models can effectively serve as domain-independent interfaces when provided with appropriate structure and feedback mechanisms.

Fourth, we presented a **successful real-world demonstration** on a physical Franka manipulator operating in unstructured household environments. The system handled perception uncertainty, execution noise, and complex exception cascades, validating that the approach transfers effectively from simulation to physical robotic systems.

The progression from chess reasoning to embodied manipulation planning demonstrates increasingly complex applications of LLM reasoning capabilities.

While chess (Chapter 3) involved fully-known states with perfect information about all piece positions, LLM^3 addresses partially-observable geometric constraints where action feasibility must be discovered through interaction with a motion planner. The framework shows that language models can effectively reason about continuous constraints through categorical feedback, even though these constraints are not directly represented in the language domain.

The next chapter extends these ideas even further to environments where not only geometric constraints but also object locations themselves are initially unknown. This requires integrated exploration and manipulation planning, where the robot must actively gather information about where objects are located while simultaneously working toward task goals. This progression toward increasingly uncertain and partially-known environments continues to test the boundaries of how language models can support robot reasoning and planning.

System message: You are an AI robot that generates a plan of actions to reach the goal... You are expected to correct the plan incrementally (on top of the last plan) to avoid motion failure. This may involve sample new parameters for the failed action or reverse one or more succeeded actions for backtracking... You are expected to generate a plan from scratch.

Task description: A robot arm is tasked to pack boxes into a basket on a table. The robot sits at (0, 0), and faces the positive x-axis, while the positive z-axis points up. The robot is equipped with primitive actions, each taking a list of objects and continuous parameters as input:

- `pick`([`obj`], {}): pick up `obj`, with no parameters.
- `place`([`obj`], {"x": $[0.0, 1.0]$, "y": $[-1.0, 1.0]$, "$theta$": $[-3.14, 3.14]$}): place `obj` at location (x, y) with the planar rotation theta, where x ranges $(0.0, 1.0)$, y ranges $(-1.0, 1.0)$, and $theta$ ranges $(-3.14, 3.14)$.

Initial state: {init_state}

motion planning feedback trace: {trace}

Output format: Please generate output step-by-step, which includes your reasoning for the failure of the last plan as well as the generated plan... Answer the questions: (i) what is the cause of the failure of the last plan? (ii) can altering action parameters for the failed action solve the problem... (iii) do we need to reverse one or more succeeded actions executed before the failed action... (ii) what is your strategy to generate a new plan from scratch to accomplish the task goal? Please organize the output following the JSON format below: { "Reasoning": "My reasoning for the failure of the last plan is ...", "Full Plan": ["pick(['red_box'], {})", "place(['red_box'], {'x': 0.51, 'y': 0.02, 'theta': 0.00})", ...] }

Figure 4.3: Prompt templates for LLM^3. Orange text is specific to backtrack variant, blue text to from-scratch variant.

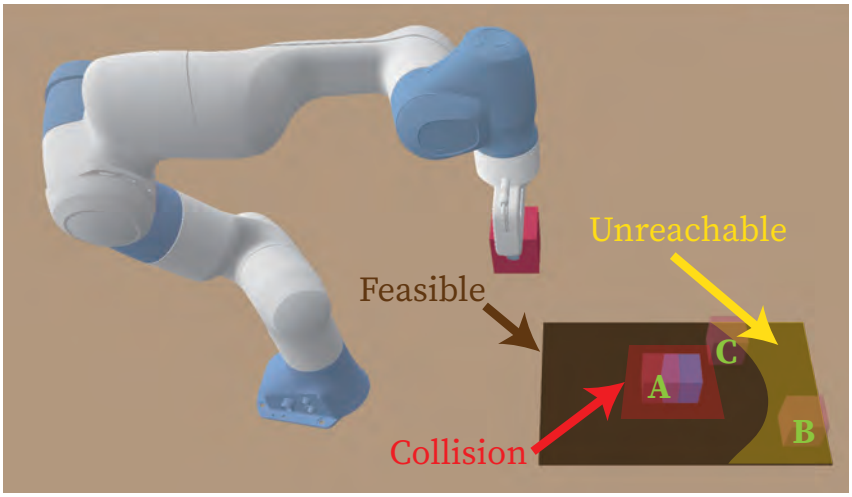


Figure 4.4: Three types of motion planning outcomes: (A) collision with objects, (B) unreachable location, (C) feasible placement.

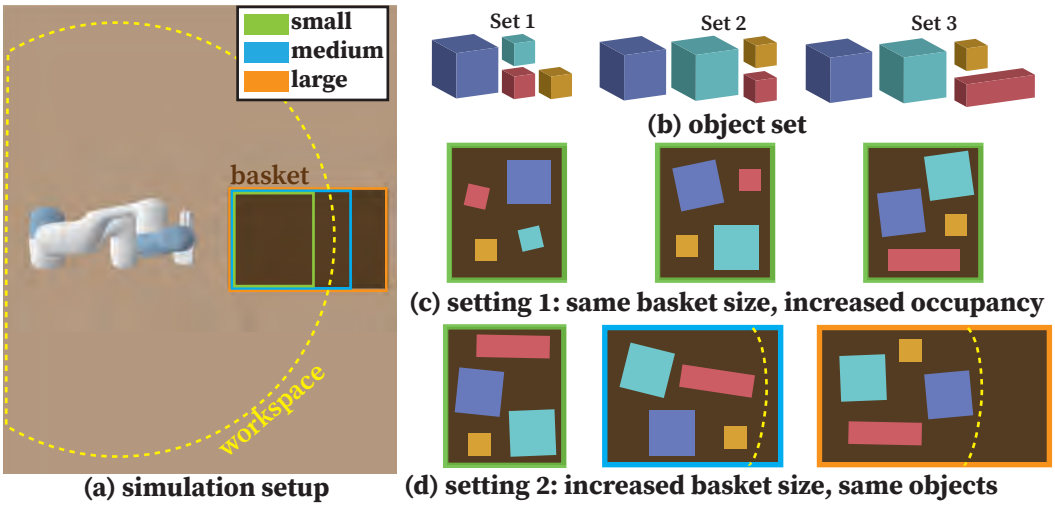


Figure 4.5: Box-packing task setup. (a) Simulation environment. (b) Three object sets of increasing size. (c) Setting 1: same basket size, increased occupancy. (d) Setting 2: increased basket size with some unreachable areas.

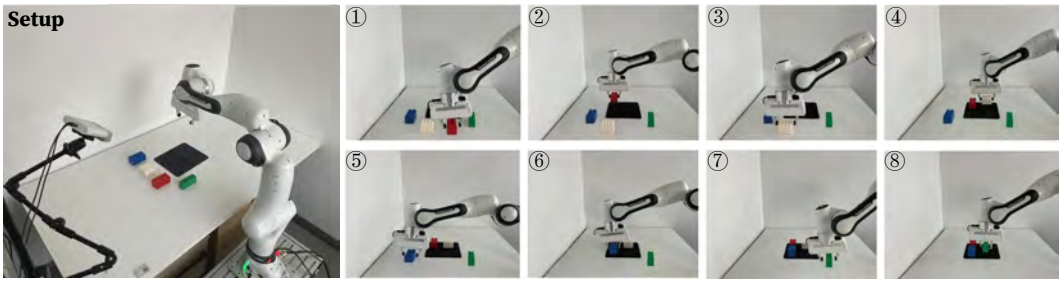


Figure 4.6: Real-world experiments on physical manipulator. (a) Task 1: Moving objects with unknown cup location. (b) Task 2: Handling blocked objects and closed containers.

Chapter 5

INTEGRATED EXPLORATION AND SEQUENTIAL MANIPULATION PLANNING

The map is not the territory.

— Alfred Korzybski

5.1 Introduction

In previous chapters, we explored LLM reasoning in fully-known chess positions (Chapter 3) and partially-observable geometric constraints (Chapter 4). This chapter addresses an even more challenging scenario: robots operating in environments where object locations are initially unknown, requiring integrated exploration to gather information alongside task execution.

In partially known environments, robots face a fundamental trade-off between pure exploration and pure exploitation. On one hand, mapping the entire environment before planning is inefficient and time-consuming, particularly when only a subset of the environment is relevant to the current task. On the other hand, acting without sufficient information about object locations and environmental state leads to failures and repeated attempts, ultimately wasting time and computational resources. Effective operation requires seamlessly integrating exploration (gathering information) with task execution (achieving goals), while continuously adapting to new observations.

5.1.1 Motivation

Consider a household robot tasked with "prepare breakfast." The robot has access to certain types of information about the environment. It knows the overall kitchen layout and the locations of major appliances, which are rarely moved by occupants. The robot also understands the goal state: it needs to place specific items on the dining table in appropriate configurations.

However, the robot lacks critical information about the current state of the environment. It does not know where specific objects are currently located, as these items are frequently moved by humans during daily activities. The robot also cannot determine whether containers hold relevant items without opening them to inspect their contents. Additionally, it remains unaware of which areas in the home have been recently rearranged, making prior knowledge of object locations potentially outdated.

This scenario presents three key challenges illustrated in Figure 5.1:

Challenge 1: Informed exploration - Without precise knowledge of object locations, the robot must intelligently prioritize which areas to explore first. Random or systematic exploration strategies are inefficient for long-horizon tasks, as they waste time examining irrelevant locations and delay task completion.

Challenge 2: Balancing exploration and execution - The robot should strategically interleave information gathering actions with goal-directed manipulation actions to minimize total effort. A pure exploration-first approach delays task progress, while a pure execution approach leads to repeated failures when objects are not where expected.

Challenge 3: Situated adaptation - New observations gathered during execution may invalidate previous plans and assumptions, requiring dynamic replanning based on environmental feedback. The system must gracefully

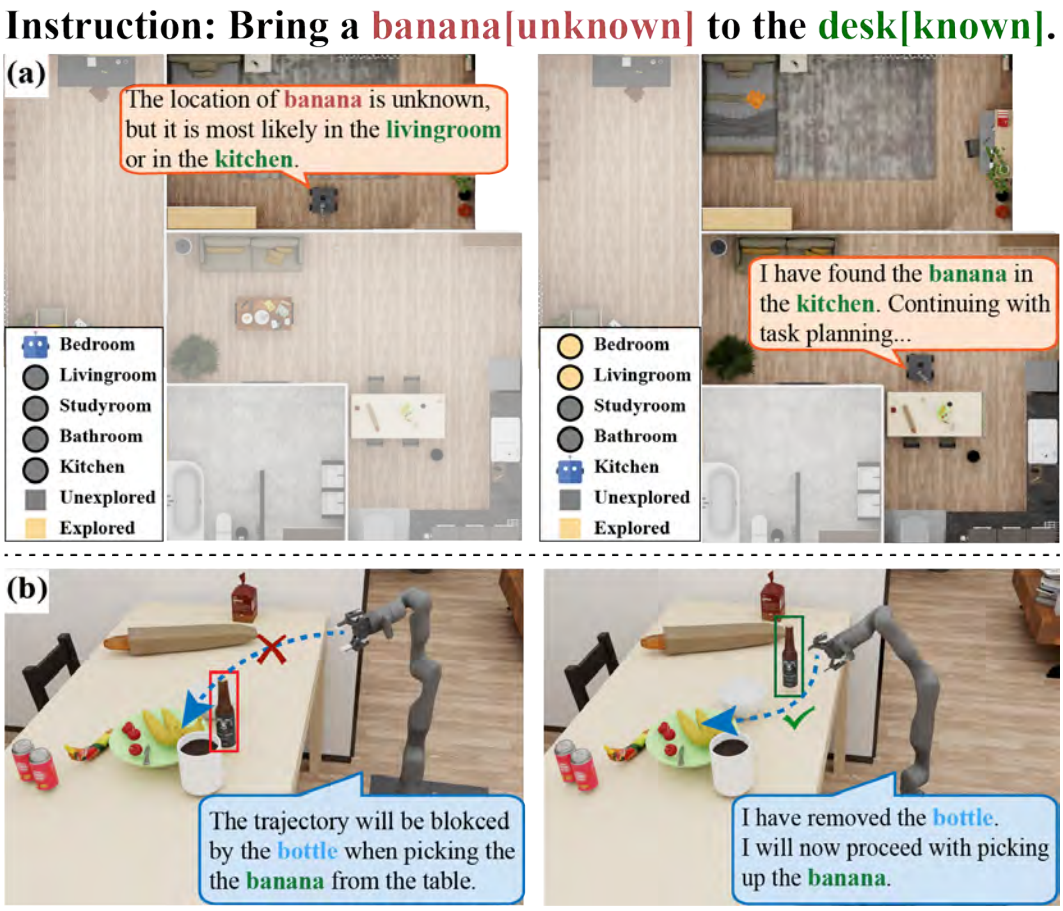


Figure 5.1: Challenges in integrated exploration and manipulation: (a) Prioritizing exploration locations without prior knowledge. (b) Balancing exploration with task execution. (c) Handling situated exceptions during execution.

handle situations where predictions prove incorrect and adapt its strategy accordingly.

Existing approaches to this problem suffer from several limitations. Some methods explore the entire environment before initiating any task planning Jiang et al. [2024b], which is inefficient for long-horizon tasks where only a subset of the environment is relevant. Other approaches rely on manually defined exploration heuristics Xu et al. [2022], which lack generalizability across different environments and task domains. A third category of methods separates exploration and task planning into distinct sequential phases Rosinol et al. [2020], preventing the opportunistic integration of information gathering

with goal-directed actions. These methods lack the flexibility and efficiency needed for complex, real-world tasks.

5.1.2 Key Insight: Graph-Based Belief Management

We propose leveraging graph-based scene representations combined with LLM commonsense knowledge to address these challenges. Our approach first uses LLMs to estimate probable object locations before physical exploration begins, drawing on semantic knowledge about typical object placements in household environments. The system then continuously updates these beliefs based on new observations gathered during task execution, maintaining an evolving representation of environmental knowledge. Finally, the framework generates plans that naturally integrate exploration actions with manipulation actions, eliminating the need for separate exploration and execution phases.

5.1.3 Contributions

We present EPoG (**E**xploration-based sequential manipulation **P**lanning **o**n **G**raph-based representations), a framework featuring three core technical contributions.

First, we introduce a belief graph construction method that uses LLM commonsense reasoning to estimate unknown object locations. This provides informed initial hypotheses about where task-relevant objects are likely to be found, based on semantic relationships and typical household organization patterns.

Second, we develop an integrated planning algorithm that computes graph edit operations while considering both task goals and movement costs. This optimization-based approach naturally determines when to explore for unknown objects and when to execute manipulation actions, avoiding the ineffi-

ciency of separate planning phases.

Third, we implement a situated replanning mechanism that employs LLM reasoning to handle exceptions during execution. When unexpected situations arise, such as blocked paths or inaccessible containers, the system generates corrective action sequences without requiring hand-crafted rules for each exception type.

This framework offers several key advantages over existing approaches. First, it eliminates the need for a separate exploration phase, allowing the robot to begin task execution immediately with informed predictions about object locations. Second, the system leverages domain-independent heuristics derived from LLM commonsense knowledge, avoiding the need for manually crafted rules specific to each environment or task domain. Third, it achieves natural integration of observation and action within a unified planning framework, where information gathering and goal achievement are considered jointly. Finally, the optimization-based planning approach ensures efficient execution by minimizing the total robot travel distance while accomplishing task objectives.

5.2 Graph-Based Scene Representation

5.2.1 Representation Overview

Following established work in 3D scene understanding and semantic mapping Armeni et al. [2019], Han et al. [2021], Jiao et al. [2022], we structure scene graphs as hierarchical trees that provide semantic abstraction over spatial environments. This representation enables efficient reasoning about object relationships and supports both high-level task planning and low-level motion planning.

A scene graph $G = (V, E, A)$ consists of three fundamental components: nodes representing entities, edges encoding spatial relationships, and attributes capturing functional properties.

Scene Nodes $v_i \in V$ represent physical entities in the environment, where each node is defined as:

$$v_i = \langle o_i, c_i, M_i, B_i \rangle$$

The node components capture different aspects of the entity. The unique object identifier o_i distinguishes this entity from all others in the scene. The semantic label c_i provides a category-level description of the object type, such as "cup," "table," or "refrigerator." The set of geometric primitives $M_i = \{m_j^i | j = 1, \ldots, |M_i|\}$ represents the detailed 3D geometry, which may include meshes, point clouds, or parametric shapes. The 3D bounding box B_i provides a compact geometric approximation used for collision checking and spatial reasoning.

Scene Edges $e_{i,j} \in E$ represent spatial relationships between entities, where each edge is defined as:

$$e_{i,j} = \langle v_i, v_j, t_{i,j}, c_{i,j} \rangle$$

The edge components encode both geometric and semantic relationships. The parent node v_i represents the supporter in the relationship, typically a surface or container. The child node v_j represents the supported object that rests on or is contained within the parent. The spatial transformation $t_{i,j}$ specifies the relative pose between the two entities, encoding position and orientation. The relationship type $c_{i,j} \in \{\text{on}, \text{contain}\}$ categorizes the spatial relationship, distinguishing between objects resting on surfaces ("on") and objects enclosed within containers ("contain").

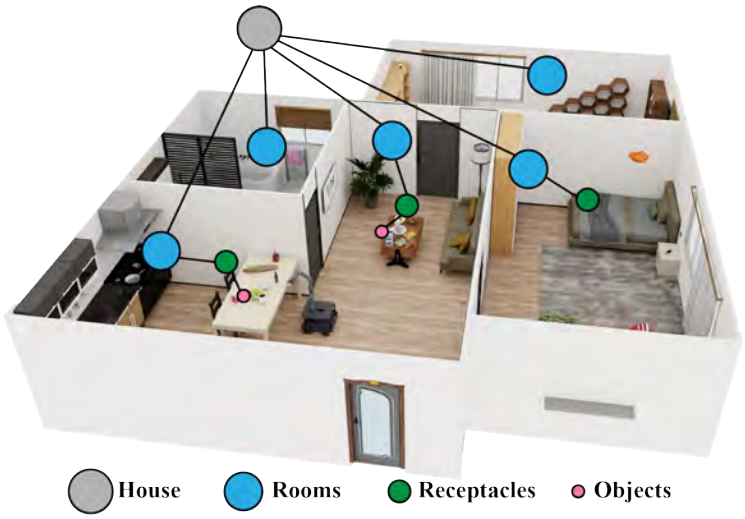


Figure 5.2: Hierarchical structure of indoor scene graphs: House → Rooms → Receptacles → Objects.

Attributes $A = \{(A_i^s, A_i^c) | i \leq |V|\}$ encode functional properties of nodes that affect planning feasibility. The supporting attribute A_i^s indicates whether a node can support other objects, which is true for surfaces like tables and shelves but false for small objects like cups. The container attribute A_i^c is defined for objects with doors or lids, tracking whether the container is currently open or closed, which affects the observability and accessibility of contained objects.

5.2.2 Hierarchical Structure

Indoor scenes naturally exhibit hierarchical organization that we explicitly encode in the graph structure. Our representation structures environments into four distinct levels, as illustrated in Figure 5.2:

At Level 0, the root node represents the entire house or building, providing the top-level spatial container for all other entities. Level 1 contains room nodes representing distinct functional spaces within the house, such as kitchen, bedroom, living room, and bathroom. These room-level nodes partition the environment into manageable subregions for exploration and plan-

ning. Level 2 consists of receptacle nodes representing furniture and fixtures within each room, including containers like cabinets and refrigerators, as well as surfaces like tables, countertops, and shelves. Finally, Level 3 and below contain object nodes representing manipulable items that are the targets of task specifications, such as cups, plates, utensils, and food items.

This hierarchical structure directly supports downstream planning algorithms by providing multiple forms of abstraction. At the spatial level, it organizes the environment into rooms and receptacles, enabling efficient navigation planning between semantically meaningful locations. For object search, the semantic grouping naturally partitions the search space, allowing the robot to focus exploration on likely locations based on object categories. Additionally, the explicit representation of support relationships enables the planner to reason about manipulation feasibility, ensuring that planned actions respect physical constraints such as object stacking and container accessibility.

5.3 Problem Definition

5.3.1 State Representation

The state $s \in \mathcal{S}$ of the environment is represented as a scene graph G, providing a structured encoding of all entities and their relationships. The initial state s_{init} typically contains partial information about the environment. It includes House, Room, and Receptacle nodes, which represent relatively static elements of the environment that change infrequently over time. The graph also contains edges between these nodes that encode their spatial relationships, such as which rooms belong to the house and which receptacles are located in each room. However, critically missing from the initial state are the Object nodes representing task-relevant items, as these objects are frequently moved by

humans and their current locations are unknown at the start of the task.

The goal state s_{goal} is also represented as a graph G_{goal}, but with a simplified structure that focuses on task-relevant constraints. The goal graph includes all task-relevant object nodes that must be manipulated to complete the task. It also specifies the desired relationships between these objects and receptacles, though these relationships are simplified as semantic descriptions rather than precise geometric transformations. For example, a breakfast preparation task might specify goals such as "apple on table, banana in basket," without requiring exact pose specifications for each object.

5.3.2 Actions

The action set $\mathcal{A}$ consists of primitive robotic actions that modify the scene graph structure. These actions form the basic building blocks for task execution.

The $\texttt{Pick}(v_i, v_j)$ action removes object v_j from its supporter v_i, deleting the corresponding edge from the scene graph and updating the robot's gripper state to hold the object. The $\texttt{Place}(v_i, v_j)$ action places the currently held object v_j onto supporter v_i, creating a new edge in the scene graph and freeing the robot's gripper. The $\texttt{Open}(v_i)$ action opens a container v_i, changing its container attribute A_i^c from closed to opened, which makes the container's contents observable and accessible. Conversely, the $\texttt{Close}(v_i)$ action closes container v_i, changing A_i^c from opened to closed. Finally, the $\texttt{Walk}(v_i)$ action navigates the robot to the vicinity of node v_i, which may be a room, receptacle, or object, updating the robot's position in the environment.

Each action is associated with a cost that reflects the computational and physical resources required for execution. For manipulation actions like `Pick` and `Place`, the cost is relatively small and uniform. For `Walk` actions, the cost

is proportional to the movement distance, encouraging plans that minimize robot travel.

5.3.3 Observations

The robot's sensors provide partial observations of the environment as it navigates and manipulates objects. Upon entering a room or approaching a receptacle, the robot observes:

$$obs = \{V_{obs}, E_{obs}\}$$

The observation consists of two components. The set V_{obs} contains all visible object nodes within the robot's sensor range, including both their semantic labels and geometric information. The set E_{obs} contains all observable relationship edges between visible objects and their supporters. However, the observation model has important limitations. Objects within closed receptacles are not observable until the container is opened, as the robot's sensors cannot penetrate opaque surfaces. Similarly, objects may be occluded by other objects or environmental structures, limiting the completeness of observations.

5.3.4 Objective

The planning problem is to find an action sequence $\pi = (a_0, a_1, \ldots, a_T)$ that achieves two objectives simultaneously. First, the sequence must transform the initial state s_{init} to satisfy the goal condition, formally expressed as $g(s_{T+1}) = 1$ where $g(\cdot)$ is a boolean goal-checking function that verifies whether all required object relationships are established. Second, among all valid action sequences that achieve the goal, the planner should select the sequence that minimizes total cost $\sum_{t=0}^{T} \text{cost}(a_t)$, where cost primarily reflects the cumulative robot

travel distance.

This formulation encourages efficient task execution by penalizing unnecessary movement while ensuring that all task objectives are ultimately achieved. The challenge lies in planning under uncertainty about object locations, requiring the integration of exploration actions to gather information with manipulation actions to achieve goals.

5.4 The EPoG Framework

5.4.1 Framework Overview

Algorithm 2 outlines EPoG's bi-level planning architecture, which combines global planning for overall task strategy with local planning for exception handling.

The framework operates through two complementary planning levels that address different aspects of the task execution problem.

Global Planner: The global planning level operates on the belief graph representation to generate overall task strategies. It begins by estimating a belief graph using LLM predictions about likely object locations based on commonsense knowledge. Next, it computes graph edit operations that transform the current belief state into the goal state, identifying which objects need to be moved and which containers need to be opened. Finally, it generates an optimized action sequence via topological sorting that respects action dependencies while minimizing robot travel distance.

Local Planner: The local planning level handles exceptions and unexpected situations that arise during plan execution. When the motion planner detects that an action cannot be executed as planned, it invokes the local planner to generate corrective actions. The local planner uses LLM reason-

Algorithm 2 EPoG Framework

Require: Initial graph G_{init}, Goal graph G_g
Ensure: Success indicator, feasible action plan

$G_b \leftarrow$ EstimateBeliefGraph(G_{init})
$P_g \leftarrow$ GraphBasedPlanner(G_b, G_g)
while P_g is not empty **do**
  $action \leftarrow$ Pop(P_g)
  $obs, exception \leftarrow$ RollOut($action$)
  $G_b, replan \leftarrow$ UpdateGraph(G_b, obs)
  if $replan$ is not none **then**
    $P_g \leftarrow$ GraphBasedPlanner(G_b, G_g)
  else if $exception$ is not none **then**
    LocalPlanner($exception$)
  end if
end while

Function LocalPlanner($exception$):
  $actions \leftarrow$ LLMPlanner($exception$)
  foreach $action$ in $actions$ **do**
    $obs, exception \leftarrow$ RollOut($action$)
    if $exception$ is not none **then**
      LocalPlanner($exception$) {Recursive resolution}
    end if
  end for

ing to understand the exception type and generate appropriate solutions, such as temporarily moving blocking objects or opening inaccessible containers. Importantly, the local planner operates recursively, allowing it to handle compound exceptions where corrective actions themselves encounter new obstacles.

Figure 5.3 illustrates the system architecture and information flow between these components.

5.4.2 Belief Graph Estimation

The `EstimateBeliefGraph`($\cdot$) function constructs initial beliefs about unknown object locations by leveraging LLM commonsense knowledge about typical household organization patterns.

For each task-relevant object that is missing from the initial graph G_{init}, the

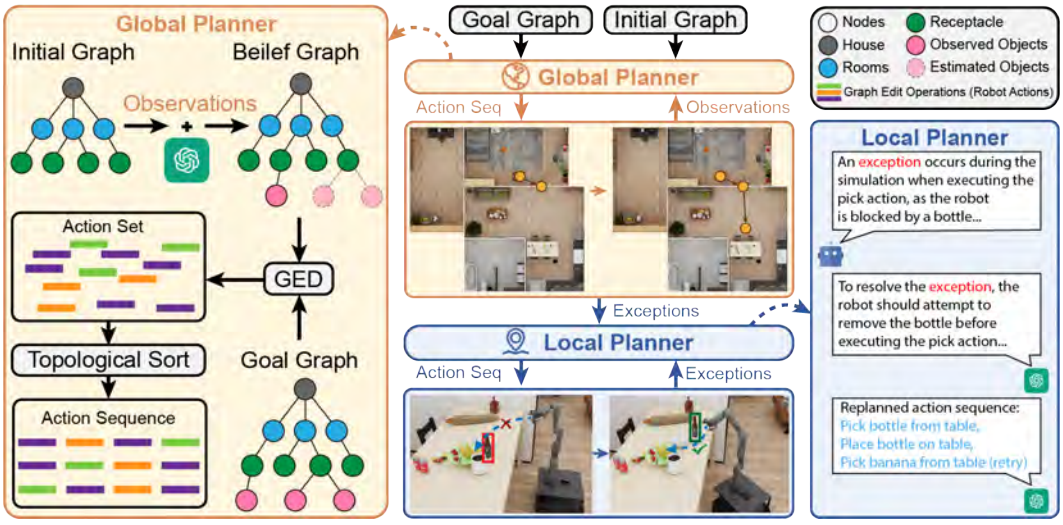


Figure 5.3: EPoG system architecture showing global planning with belief graph updates and local planning with exception handling.

system performs a two-stage estimation process to predict its likely location.

Step 1: Estimate Room location

The system first determines which room the object is most likely to be found in by querying the LLM:

```
Prompt: "In which room would you typically find a [object]
in a house? Respond with only the room name."
```

This leverages semantic knowledge about object-room associations, such as the fact that cooking utensils are typically found in kitchens and toiletries are typically found in bathrooms.

Step 2: Estimate Receptacle location

Once the room is identified, the system refines the prediction by determining which receptacle within that room is most likely to contain the object:

```
Prompt: "On which furniture or receptacle would you typically
find a [object] in a [room]? Respond with only the receptacle
name."
```

This second-level prediction narrows down the search space by identifying specific surfaces or containers, such as predicting that cups are likely on kitchen countertops or in cabinets.

The belief graph G_b is then constructed by systematically integrating these predictions with the known environment structure. The system first adds task-relevant object nodes V_{task} from the goal graph G_{goal} to represent all objects that must be manipulated. It then adds the corresponding object attributes A_{task} that specify properties like graspability and size. Finally, it creates estimated edges E_{est} based on the LLM predictions, connecting each object to its predicted receptacle with an appropriate relationship type ("on" or "contain").

Formally, the belief graph combines known and estimated information:

$$G_b = (V_{init} \cup V_{task}, E_{init} \cup E_{est}, A_{init} \cup A_{task})$$

An important design decision is that we do not attempt to predict exact spatial transformations $t_{i,j}$ for the estimated edges, since precise object poses remain unknown until direct observation. Instead, the estimated edges encode only the predicted support relationships, with geometric details to be filled in during execution when objects are actually located.

5.4.3 Graph-Based Global Planner

The global planner generates action sequences by computing graph edit distance (GED) between the belief and goal graphs, then optimizing the execution order to minimize movement cost.

Graph Edit Distance

Graph edit distance provides a principled method for identifying the minimum set of operations needed to transform one graph into another. For our planning problem, GED identifies which objects need to be moved and which containers need to be opened:

$$\text{GED}(G_1, G_2) = \min_{\{a_1,\ldots,a_k\} \in \mathcal{K}(G_1,G_2)} \sum_{i=1}^{k} \text{cost}(a_i)$$

We consider four graph edit operations that correspond directly to robot primitive actions. The delete operation delete($e_{i,j}$) removes an edge from the graph, corresponding to the robot picking object v_j from supporter v_i using the `Pick`(v_i, v_j) action. The insert operation insert($e_{i,j}$) adds a new edge to the graph, corresponding to the robot placing object v_j onto supporter v_i using the `Place`(v_i, v_j) action. The substitute operations on container attributes change the state of containers: substitute(A_i^c, opened) corresponds to `Open`(v_i), while substitute(A_i^c, closed) corresponds to `Close`(v_i).

This mapping produces an unsorted action set K along with a set of temporal constraints C that encode dependencies between actions.

Constrained Topological Sort

The actions in set K cannot be executed in arbitrary order, as they have temporal dependencies that form a partially ordered set. For example, picking an object from one location must precede placing it at another location, ensuring the robot is actually holding the object before attempting to place it. Similarly, opening a container must precede picking any objects from inside that container, as closed containers prevent access to their contents.

The constraint set $C = \{(a_i, a_j) | i \neq j\}$ encodes these precedence relation-

ships, where each tuple (a_i, a_j) specifies that action a_i must be executed before action a_j, denoted $a_i \prec a_j$.

Optimization-Based Search Rather than selecting an arbitrary topological ordering of the constrained actions, we formulate the problem as a search for the sequence that minimizes cumulative movement cost. This optimization is crucial because different execution orders can lead to dramatically different amounts of robot travel, even though they achieve the same final configuration.

Each search node in our algorithm is represented as $N = (K', C', \pi')$, containing three components. The set $K' \subseteq K$ represents actions that have not yet been scheduled in the current partial plan. The set $C' \subseteq C$ contains the remaining precedence constraints that still need to be satisfied. The partial action sequence $\pi' \subseteq \pi$ represents the actions scheduled so far in the current branch of the search.

The heuristic cost function estimates the total robot travel distance that would result from executing the current partial sequence π' followed by the remaining actions in K'. This estimation is challenging because some object poses are unknown until they are observed during execution. To handle this uncertainty, we insert `Walk` actions to navigate to the parent receptacles of unknown objects, using A* search to estimate the distances between navigation waypoints based on the known environment structure.

To improve search efficiency, we continuously update a cost upper bound as better solutions are found, using this bound to prune search branches that cannot possibly improve on the best solution found so far. This branch-and-bound approach significantly reduces the search space in practice.

Algorithm 3 details the complete optimized planning procedure.

This approach explicitly accounts for robot movement costs during plan-

Algorithm 3 Optimized Graph-Based Task Planner

Require: Belief graph G_b, Goal graph G_g
Ensure: Optimal plan π^*

$K \leftarrow \text{GED}(G_b, G_g)$ {Min-cost edit operations}
$C \leftarrow \text{GetConstraints}(K)$ {Temporal constraints}
stack $\leftarrow \emptyset$, min_cost $\leftarrow \infty$, $\pi^* \leftarrow \emptyset$
for $a \in \text{UnconstrainedActions}((K, C))$ **do**
  stack.push($([a], K \setminus \{a\}, C \setminus \{a\})$)
end for
while ¬stack.empty() **do**
  $(\pi, K_r, C_r) \leftarrow$ stack.pop()
  cost $\leftarrow \text{HeuristicCost}(\pi)$
  if cost $\geq$ min_cost **then**
    continue {Prune suboptimal}
  else if $\text{IsGoalReached}(\pi)$ **then**
    $\pi' \leftarrow \text{InsertWalkActions}(\pi)$
    if cost $<$ min_cost **then**
      min_cost $\leftarrow$ cost, $\pi^* \leftarrow \pi'$
    end if
    continue
  end if
  for $a \in \text{NextActions}(K_r, C_r)$ **do**
    $\pi_{new} \leftarrow \pi \oplus a$
    $K_{new} \leftarrow K_r \setminus \{a\}$
    $C_{new} \leftarrow C_r \setminus \{a\}$
    stack.push($(\pi_{new}, K_{new}, C_{new})$)
  end for
end while
return π^*

ning and seeks to minimize total travel distance. Importantly, it naturally integrates exploration (navigating to unknown object locations) with manipulation (performing task actions), eliminating the need for separate exploration and execution phases.

5.4.4 Belief Graph Update

The `UpdateGraph`$(\cdot)$ function maintains synchronization between the belief graph and the robot's observations, ensuring that the planner always operates on the most current information available.

When the robot executes an action and receives new observations $O = (V_{obs}, E_{obs})$ from its sensors, the system updates the belief graph according to three distinct cases.

Case 1: Object found as expected

When the robot navigates to a predicted location and successfully finds the expected object, the belief is confirmed and refined. The system updates the node v_i with actual pose information obtained from perception, replacing the abstract belief with concrete geometric data. It confirms that the predicted edge $e_{i,j}$ correctly represents the object's support relationship. Since the prediction was accurate, no replanning is needed, and execution continues with the next action in the plan.

Case 2: Expected object not found

When the robot searches a predicted location but does not observe the expected object, the initial belief was incorrect and must be revised. The system removes the incorrect estimated edge e_{err} from the belief graph, as it represents a false hypothesis about the object's location. It then generates a new location estimate e_{new} by querying the LLM again, potentially considering alternative rooms or receptacles based on updated context. Finally, it triggers replanning with the updated belief graph G_b to generate a new action sequence that searches the revised predicted location.

Case 3: New objects discovered

As the robot explores the environment, it may observe objects that were not part of the initial task specification or belief graph. The system adds these observed nodes V_{obs} to the belief graph G_b, maintaining a more complete representation of the known environment. It also adds the observed edges E_{obs} that encode the newly discovered object relationships. If any of these newly discovered objects are relevant to the current task, the system updates the

task plan accordingly, potentially creating shortcuts or identifying alternative solutions.

This continuous update process enables dynamic adaptation to environmental changes and incorrect predictions, ensuring that the robot's internal model remains consistent with observed reality.

5.4.5 Local Replanning with LLM

The local planner handles exceptions that arise during low-level motion planning and execution, using LLM-based situated reasoning to generate corrective action sequences.

Exception Taxonomy

We identify and categorize four fundamental types of motion planning exceptions that commonly occur during household manipulation tasks, as illustrated in Figure 5.4:

(a) Blocking exceptions occur when an object obstructs the manipulation path or prevents access to a target object. For example, a cup may block access to a plate behind it on a shelf, requiring the cup to be temporarily moved before the plate can be grasped.

(b) Inaccessibility exceptions arise when a target object is inside a closed container that must be opened first. For example, attempting to pick an object from inside a refrigerator fails if the refrigerator door is closed, requiring an open action before the pick can succeed.

(c) Collision exceptions occur when placing an object at the planned location would result in collision with the environment or other objects. For example, attempting to place a large bowl on a crowded countertop may fail because there is insufficient free space, requiring other objects to be moved

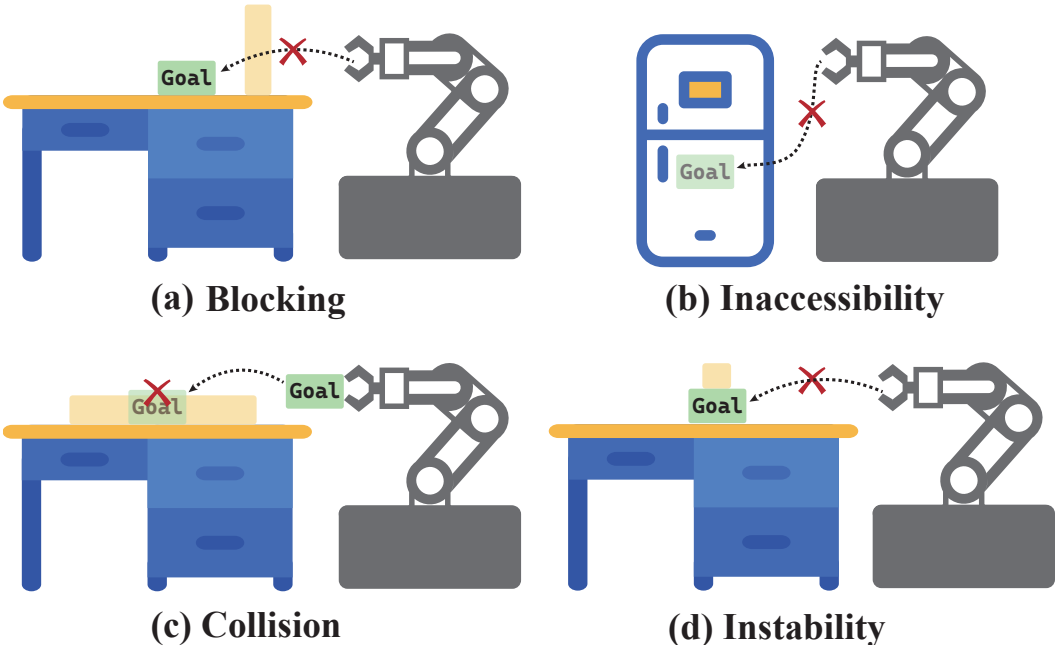


Figure 5.4: Four types of motion planning exceptions: (a) Blocking, (b) Inaccessibility, (c) Collision, (d) Instability.

first.

(d) Instability exceptions arise when a manipulation action would cause instability or toppling of object configurations. For example, attempting to remove a bottom plate from a stack would cause the plates above to fall, requiring the stack to be disassembled in the correct order.

LLM-Based Exception Handling

For each exception type, we prompt the LLM to generate appropriate corrective actions by providing structured context about the failure and available primitives.

The prompt template follows this structure:

```
System message: As an assistive robot, you must implement
corrective actions to address errors during task execution.
```

```
Task description: You have the following primitives:
Pick(x, y), Place(x, y), Open(x), Close(x). You may
temporarily place objects in a designated parking area.

Example: [Exception type and reasoning example]

Question: You attempted [failed_action], but it failed due
to [exception]. The parking place is [parking_place].
```

The LLM generates a corrective action sequence that typically follows a general pattern. First, if the robot is currently holding an object, it temporarily places that object in the designated parking area to free the gripper. Second, it removes or adjusts interfering objects that are causing the exception, such as moving blocking objects out of the way or opening closed containers. Third, it retries the original failed action, which should now succeed with the obstacles removed. Finally, if necessary, it restores the environment by returning temporarily moved objects to their original locations.

This approach provides flexible exception handling without requiring handcrafted rules for each specific situation. The LLM's commonsense reasoning allows it to understand the causal relationships between objects and generate appropriate multi-step solutions. Furthermore, the recursive structure of the local planner (as shown in Algorithm 2) allows it to handle compound exceptions where corrective actions themselves encounter new obstacles, invoking the exception handler recursively until all issues are resolved.

Table 5.1: Benchmark task descriptions and goals

No.	Task	# Scenes	Task Goals
1	Breakfast Prep.	10	apple on plate, bread on plate, fork on plate, plate on diningtable
2	Bedroom Work	10	alarmclock on desk, CD on desk, laptop on desk, pencil on desk
3	Movie & Snack	10	remotecontrol on sofa, bread on plate, plate on diningtable
4	Tea & Relaxation	10	kettle on countertop, cup on diningtable, remotecontrol on sofa
5	Bath Prep.	6	soapbottle on faucet, cloth on faucet

5.5 Experiments

5.5.1 Experimental Setup

We evaluate EPoG's performance on five long-horizon object transportation tasks executed across 46 realistic household scenes sampled from the ProcThor-10k dataset Deitke et al. [2022]. This dataset provides diverse indoor environments with varied layouts, furniture arrangements, and object distributions, enabling comprehensive evaluation of generalization capabilities.

Tasks Table 5.1 describes the five benchmark tasks used in our evaluation, each representing common household activities that require multi-object manipulation and navigation across rooms.

Each scene is configured to present realistic challenges by placing task-relevant objects at unknown initial locations throughout the environment. Additionally, to test exception handling capabilities, we introduce two randomly placed obstacles per scene that require corrective actions during execution, such as blocking objects or closed containers.

Metrics We evaluate system performance using three complementary metrics that capture different aspects of efficiency and effectiveness.

The **Success Rate (%SR)** measures the percentage of task instances that the robot successfully completes, defined as achieving all specified goal configurations without irrecoverable failures. This metric captures overall reliability and robustness.

The **Explored Nodes (%EN)** quantifies the average percentage of environment nodes (rooms, receptacles, and objects) that the robot newly explores during task execution, reported relative to EPoG's exploration count. Lower values indicate more efficient information gathering that focuses on task-relevant locations.

The **Travel Distance (%TD)** measures the average cumulative distance the robot travels during task execution, reported as a percentage difference relative to EPoG's travel distance. Lower values indicate more efficient navigation with less unnecessary movement.

Implementation The experimental system combines several components to enable realistic evaluation. We use the ProcThor simulation environment, which provides physics-based simulation with realistic object dynamics and sensor models. All methods use GPT-4 Turbo as the underlying language model for commonsense reasoning and exception handling. For low-level motion planning, we employ the VKC-based manipulation planning framework Jiao et al. [2021], which handles geometric constraints and collision avoidance. The system runs on commodity hardware with NVIDIA GPUs for perception processing.

5.5.2 Baseline Methods

We compare EPoG against three baseline approaches that represent different points in the design space of exploration and planning integration.

LLM Planner: This baseline uses pure LLM-based planning without any graph representation or structured belief management. The system simply prompts the LLM with the task specification and allows it to generate action sequences based on zero-shot reasoning. This tests whether modern LLMs can directly solve long-horizon manipulation tasks without specialized representations or algorithms.

Exploration+LLM Planner: This approach first explores the environment exhaustively to locate all objects before invoking LLM-based planning on the complete scene graph. This establishes an upper bound for sequential perception-planning pipelines by eliminating uncertainty about object locations at planning time, though at the cost of extensive upfront exploration.

Exploration+PoG: This method implements the traditional Planning on Graph framework from Jiao et al. [2022], which uses rule-based planning with hand-crafted heuristics for exception handling. Like the previous baseline, it explores the environment completely before planning, but uses graph-based optimization rather than LLM reasoning. This provides a reference point for predefined constraint-based approaches that do not leverage language model capabilities.

5.5.3 Main Results

Table 5.2 presents comprehensive experimental results across all five tasks and comparison methods, with metrics reported relative to EPoG's performance.

Table 5.2: Experimental results for EPoG framework compared to baseline methods

Method	Breakfast			Bedroom			Movie			Tea			Bath			Total		
	%SR	%EN	%TD	%SR	%EN	%TD	%SR	%EN	%TD	%SR	%EN	%TD	%SR	%EN	%TD	%SR	%EN	%TD
LLM	10.0	+0.0	+110	10.0	+0.0	+48.7	10.0	+0.0	+179	40.0	+10.3	+115	16.7	+0.0	+16.7	17.4	+2.05	+93.9
Exp.+LLM	40.0	+77.9	+80.6	40.0	+75.9	+84.0	40.0	+50.4	+248	60.0	+75.0	+124	16.7	+168	+63.2	41.3	+89.4	+120
Exp.+PoG	**100**	+77.2	+59.0	**100**	+50.4	+25.0	**100**	+66.5	+112	**100**	+67.9	+122	**100**	+104	+60.5	**100**	+73.2	+75.6
EPoG	**100**	52.1	35.8	**100**	50.8	52.1	80.0	52.9	33.0	90.0	52.1	23.5	83.3	46.4	43.6	91.3	51.9	37.8

Key Findings The experimental results reveal several important insights about the effectiveness of integrated exploration and planning.

1. High success rate with integrated approach: EPoG achieves an overall success rate of 91.3% across all tasks and scenes, demonstrating robust performance despite operating with initially unknown object locations. This success rate is comparable to the Exploration+PoG baseline (100%) that explores the entire environment before planning, showing that informed exploration can nearly match exhaustive exploration in terms of task completion while being significantly more efficient.

2. Significant efficiency gains: EPoG demonstrates substantial improvements in exploration and navigation efficiency compared to exhaustive exploration methods. Specifically, the framework reduces the number of explored nodes by an average of 40% compared to Exploration+PoG (51.9% vs. 73.2% relative exploration). Additionally, EPoG reduces travel distance by 36.2% on average compared to the same baseline (37.8 vs. 75.6 relative distance). These results demonstrate that EPoG effectively integrates exploration with task execution, focusing information gathering on task-relevant locations rather than exhaustively mapping the environment.

3. Pure LLM struggles with long-horizon tasks: The LLM Planner baseline achieves only a 17.4% success rate, confirming significant limitations in pure language model planning for complex robotics tasks. Analysis reveals that LLMs struggle with several key aspects: reasoning over complex graph

structures with many nodes and edges, maintaining plan consistency over long action sequences spanning dozens of steps, and spatial reasoning about object locations and movement distances. These limitations motivate the need for structured representations and specialized planning algorithms.

4. Exploration overhead matters: The Exploration+LLM baseline exhibits high exploration percentages (%EN) and travel distances (%TD), yet achieves only moderate success rates (41.3%). This demonstrates that exhaustive environment exploration before planning incurs substantial overhead without guaranteeing success, particularly when the LLM planner struggles with the resulting complex scene graphs. The results suggest that exploration and planning should be tightly integrated rather than separated into sequential phases.

5.5.4 Case Studies

Figure 5.5 illustrates actual robot trajectories for two representative task instances, providing qualitative insight into how EPoG integrates exploration with execution.

Case 1: Breakfast Preparation

In this scenario, EPoG demonstrates efficient execution by recognizing and exploiting task structure. The robot first collects multiple items (apple, bread, fork) and places them on a single plate in the kitchen (actions 1→2→3), recognizing the plate as a transport vehicle. It then transports the loaded plate in a single trip to the living room dining table (→4), minimizing navigation between rooms. In contrast, the LLM baseline method failed to recognize this transport opportunity, instead carrying objects separately one at a time. This resulted in multiple round trips between the kitchen and living room, substantially increasing travel distance and execution time.

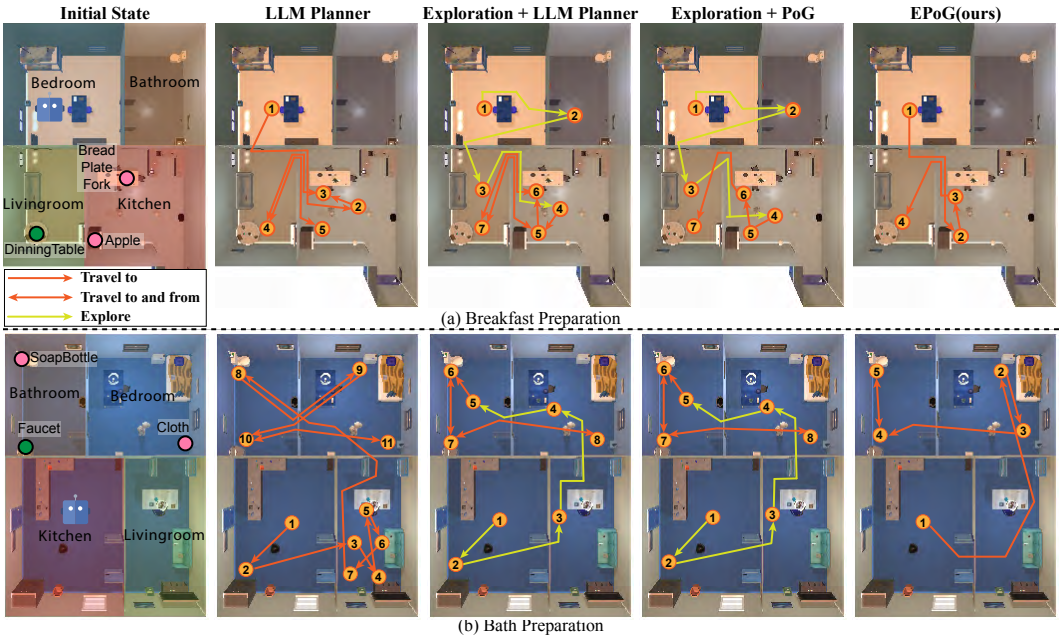


Figure 5.5: Case studies showing robot paths for (a) Breakfast Preparation and (b) Bath Preparation tasks. Numbers indicate action sequence order.

Case 2: Bath Preparation

This case demonstrates EPoG's dynamic adaptation to incorrect predictions. The system initially estimated that the cloth would be on the bedroom nightstand based on LLM commonsense reasoning. When the robot navigated to the nightstand and did not find the cloth, the system immediately updated the belief graph to remove the incorrect hypothesis. It then generated a new prediction that the cloth might be in the bathroom cabinet, regenerated the task plan to target this new location, and successfully located and retrieved the cloth. This example highlights how continuous belief updating enables graceful handling of prediction errors without requiring exhaustive exploration.

5.5.5 Real Robot Experiments

We validate EPoG's practical applicability by deploying it on a physical mobile manipulator in a real household environment, as shown in Figure 5.6.

Experiment 1: Object localization with unknown locations

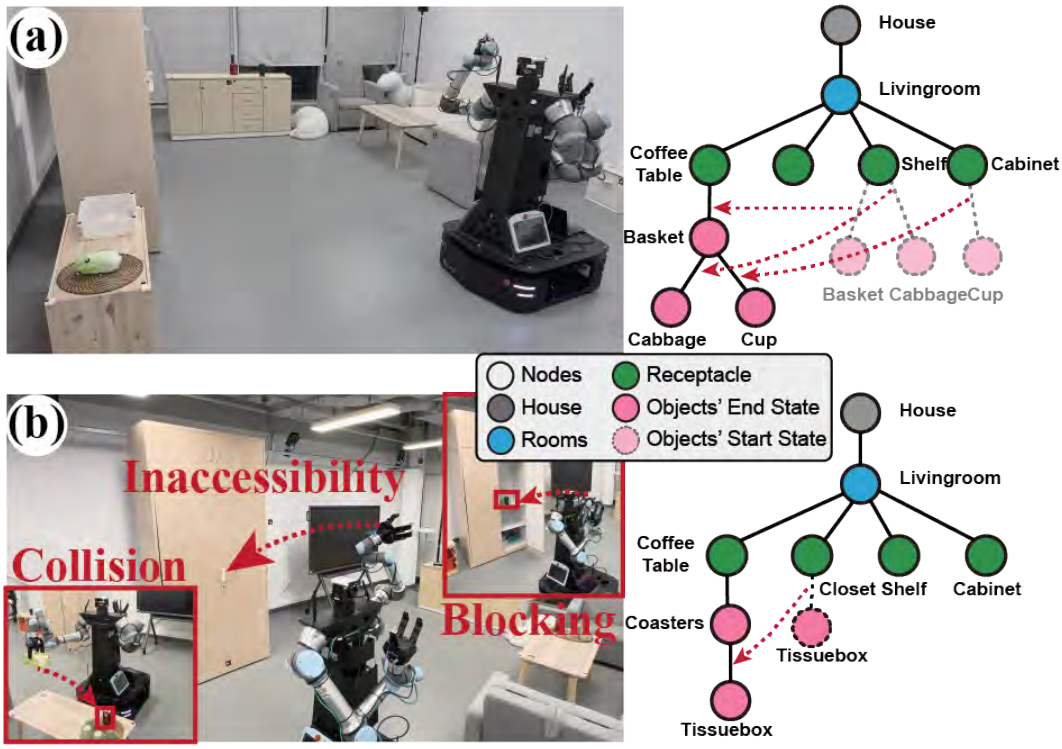


Figure 5.6: Real-world experiments on mobile manipulator demonstrating EPoG's practical applicability.

In this task, the robot must place three objects (bowl, apple, cup) at specified goal locations. The bowl and apple locations are known from prior knowledge, but the cup location is unknown. The robot uses the LLM to estimate that the cup is likely on the kitchen counter. It navigates to the predicted location and successfully finds the cup. However, upon attempting to place it at the goal location, the robot discovers the location is already occupied. The system updates its belief graph with this new observation, generates an alternative placement strategy, and successfully completes the task. This experiment demonstrates EPoG's ability to handle both prediction uncertainty and dynamic replanning in response to unexpected observations.

Experiment 2: Complex exception handling

This more challenging task requires retrieving a tissue box from inside a closed closet that contains several other objects. The robot encounters multi-

ple cascading exceptions during execution. First, it detects an inaccessibility exception because the closet is closed, and opens the closet door. Next, it identifies a blocking exception because a cup is preventing access to the tissue box, and temporarily moves the cup to a parking area. When attempting to place the tissue box at its goal location on a coaster, the robot detects a collision exception because a tea can is obstructing the coaster surface. The system generates a corrective plan to remove the tea can, then successfully places the tissue box. Throughout this sequence, the local planner recursively handles each exception as it arises, demonstrating robust exception handling in complex real-world scenarios.

Both experiments validate that EPoG's integration of exploration and exception handling transfers effectively from simulation to real robots, handling perception noise, execution uncertainty, and complex exception cascades that are common in real-world household environments.

5.6 Discussion

5.6.1 Integration of Exploration and Planning

EPoG's key innovation lies in seamlessly combining exploration with task execution within a unified optimization framework. By using LLM predictions to initialize probabilistic beliefs about object locations, the system begins with informed hypotheses rather than random search. By optimizing plans to minimize robot movement costs, the framework naturally balances exploration (navigating to unknown objects) with exploitation (executing manipulation actions). By dynamically updating beliefs with observations, the system continuously refines its environmental model without requiring separate exploration and planning phases.

The framework naturally determines when to explore and when to act through cost-based reasoning. When an object's predicted location has high confidence and low travel cost, the planner immediately generates actions to retrieve it. When predictions are uncertain or locations are distant, the planner may defer those actions until the robot is already nearby for other reasons. This opportunistic integration of exploration with execution emerges naturally from the optimization objective, without requiring explicit mode-switching or hand-crafted scheduling heuristics.

5.6.2 LLM as Exploration Heuristic

The use of LLM commonsense knowledge as an exploration heuristic represents a significant advantage over traditional approaches. Rather than exploring randomly or systematically (which waste time on irrelevant locations), or using hand-crafted rules (which require domain expertise and lack generalization), EPoG leverages semantic knowledge about typical object placements: cups are typically in the kitchen, remotes are often on coffee tables, toiletries are usually in the bathroom.

This domain-independent heuristic provides several benefits. It is more versatile than hand-crafted rules because it draws on broad commonsense knowledge learned from large-scale pretraining data. It adapts across different household configurations because the LLM can reason about context-specific placements based on room types and furniture arrangements. It gracefully handles uncommon scenarios through probabilistic reasoning, generating plausible alternatives when initial predictions prove incorrect.

Importantly, while LLM predictions are not always correct, they provide useful guidance that dramatically reduces the search space compared to uninformed exploration. The continuous belief update mechanism ensures that

incorrect predictions are quickly detected and corrected during execution.

5.6.3 Handling Environmental Uncertainty

EPoG addresses environmental uncertainty through multiple complementary mechanisms that work together to maintain robust operation.

Belief representation: The graph structure explicitly distinguishes between known information (observed environment structure) and estimated information (predicted object locations). This separation allows the planner to reason appropriately about uncertainty, for example by considering multiple possible locations or generating contingency plans.

Dynamic updates: The continuous belief update process refines the environmental model as new information becomes available. When predictions are confirmed, the system fills in geometric details. When predictions prove incorrect, the system generates alternative hypotheses. When unexpected objects are discovered, the system incorporates them into the representation. This ongoing synchronization ensures the planner always operates on the most current information.

Situated replanning: The LLM-based exception handling provides flexible adaptation to unexpected situations that cannot be anticipated at planning time. Rather than using fixed contingency plans or aborting when surprises occur, the system reasons about the specific situation and generates appropriate corrective actions. This situated reasoning is more flexible than assuming complete knowledge or using predetermined responses to predefined exception types.

Together, these mechanisms enable EPoG to operate effectively in partially known, dynamic environments where complete information is unavailable and the world state changes during execution.

5.6.4 Efficiency vs. Completeness Trade-off

The experimental results reveal an interesting trade-off between efficiency and completeness in exploration strategies. While the Exploration+PoG baseline achieves a perfect 100% success rate by exhaustively exploring the environment before planning, it incurs substantial costs: exploring 40% more nodes than necessary, traveling 36% farther than needed, and taking significantly longer total execution time.

EPoG's 91.3% success rate with dramatically lower exploration and travel costs represents a favorable trade-off for practical applications where efficiency matters. The small reduction in success rate (8.7% task failures) comes from cases where initial LLM predictions are very inaccurate and the robot exhausts its search budget before finding all required objects. For most real-world scenarios, this trade-off is acceptable because the efficiency gains translate directly to faster task completion, lower energy consumption, and reduced wear on robot hardware.

Furthermore, the efficiency advantage compounds over time. For systems that perform multiple tasks throughout the day, reducing exploration overhead by 40% and travel distance by 36% can mean completing several additional tasks within the same operating period, significantly improving overall productivity.

5.6.5 Limitations

Despite its effectiveness, EPoG has several limitations that suggest directions for future work.

LLM prediction accuracy: When initial location estimates are substantially incorrect, the robot may explore many wrong locations before finding required objects. This is particularly problematic for unusual object placements

that violate typical household organization patterns. While the dynamic update mechanism eventually corrects these errors, doing so requires additional exploration time. Future work could improve prediction accuracy by conditioning on more detailed environmental context or learning from previous task instances in similar environments.

Exception handling brittleness: Compound exceptions involving multiple simultaneous problems can confuse the LLM-based local planner, leading to suboptimal or incorrect solutions. For example, when an object is simultaneously blocking another object and located in a closed container, the LLM may generate action sequences that address only one problem or that violate action dependencies. Improving the robustness of exception handling for complex compound scenarios remains an open challenge.

Computational overhead: Each LLM query adds latency to the planning and execution pipeline. For time-critical applications where response time is paramount, this overhead may be prohibitive. Current experiments use GPT-4 Turbo with query latencies of several seconds per call. Optimizing inference speed through model distillation, caching common queries, or using faster models could reduce this overhead.

Scalability: Very large environments with hundreds of objects may exceed LLM context windows or produce overwhelming complexity in the scene graph representation. The current system is designed for typical household tasks involving 5-15 objects across 2-5 rooms. Scaling to larger environments like offices, warehouses, or multi-story homes would require hierarchical decomposition strategies or more sophisticated attention mechanisms to focus on relevant subgraphs.

5.7 Conclusion

This chapter presented EPoG, a framework for integrating exploration and sequential manipulation planning using graph-based scene representations. The work makes several key contributions to embodied AI and robotic task planning.

First, we introduced a belief-based planning approach that uses LLM predictions to construct initial beliefs about unknown object locations, leveraging commonsense knowledge to guide exploration toward likely locations rather than exhaustively searching the environment.

Second, we developed a cost-aware optimization algorithm that generates plans minimizing robot movement while achieving task goals. This optimization naturally determines when to explore for unknown objects and when to execute manipulation actions, seamlessly integrating information gathering with goal achievement.

Third, we implemented a situated adaptation mechanism that employs LLM reasoning for flexible exception handling during execution. This approach generates corrective actions for various failure modes without requiring handcrafted rules for each exception type.

Fourth, we provided empirical validation demonstrating 91.3% task success rate with 40% reduction in exploration overhead and 36.2% reduction in travel distance compared to exhaustive exploration baselines. These results confirm that informed, integrated exploration can nearly match the success rate of complete environment mapping while being substantially more efficient.

Fifth, we demonstrated successful deployment on a physical mobile manipulator, validating that the approach transfers effectively to real-world scenarios with perception noise, execution uncertainty, and complex exception cascades.

EPoG extends the progression of reasoning capabilities explored throughout this dissertation:

- Chapter 3 investigated LLM reasoning in fully-known states with perfect information, using language-guided search to improve chess playing through explicit state evaluation and move generation.

- Chapter 4 explored reasoning with known environment layouts but geometric uncertainties, using LLMs to diagnose and explain motion planning failures in manipulation tasks.

- This chapter addresses reasoning with unknown object locations requiring integrated exploration, using LLMs both for predictive modeling of likely states and for situated exception handling during execution.

Together, these studies demonstrate that LLMs can provide valuable reasoning capabilities across the spectrum from abstract strategic reasoning to grounded physical interaction, though the specific role of language models must be carefully matched to the requirements of each domain. The next chapter synthesizes insights across all three studies, analyzing common themes in how LLMs contribute to reasoning and discussing broader implications for the future of intelligent robotic systems.

Chapter 6

CONCLUSION

This chapter synthesizes insights from the three studies presented in Chapters 3, 4, and 5, examining common themes for LLM reasoning research, summarizing the thesis contributions, and reflecting on limitations.

6.1 Progressive Complexity Across Studies

The three studies presented in this dissertation represent a carefully designed progression of complexity, systematically exploring how Large Language Models can reason effectively across increasingly challenging domains. This progression enables us to isolate different aspects of reasoning while building toward realistic autonomous systems operating in complex, uncertain environments.

6.1.1 Increasing Environmental Complexity

The progression across the three studies systematically introduces additional sources of complexity and uncertainty, allowing us to understand how different reasoning challenges affect LLM capabilities.

The chess study (Chapter 3) operates in a fully observable, deterministic game environment where all information is available to the decision-maker at every step. The board state is completely known, all legal moves can be enumerated, and the consequences of actions are deterministic and predictable. This controlled setting enables pure reasoning without the confounding factors of perception uncertainty or environmental variability, providing a clean testbed for understanding how language explanations enhance decision-making

Table 6.1: Comparison of problem characteristics across three studies

Characteristic	**Chess**	**LLM^3**	**EPoG**
Environment	Fully known	Known layout	Partially known
State space	Discrete	Hybrid	Hybrid
Reasoning	Strategic/Tactical	Failure	Situated
Uncertainty	None	Geometric	Location
Horizon	Medium (3-6)	Medium-Long	Long (10+)
Physical embodiment	No	Yes	Yes
Real deployment	No	Yes	Yes

quality.

The LLM^3 study (Chapter 4) introduces geometric uncertainty while maintaining a known symbolic layout. The system knows which objects exist in the environment and understands their relationships, but must discover through motion planning whether specific continuous action parameters are geometrically feasible. This requires reasoning about physical constraints such as collision avoidance, reachability limits, and kinematic constraints that are not directly observable from the symbolic state representation. The introduction of physical embodiment and continuous geometric reasoning represents a significant step beyond the discrete, fully-known chess domain.

The EPoG study (Chapter 5) adds the most challenging form of uncertainty: unknown object locations and dynamic environmental changes. The robot initially does not know where task-relevant objects are located and must actively explore to gather information. Furthermore, the environment may change during task execution as objects are moved or new obstacles are discovered. This requires sophisticated belief management, continuous adaptation based on observations, and integration of exploration with task execution. The combination of location uncertainty with geometric constraints and long action horizons makes this the most complex and realistic setting among the three studies.

This progression systematically adds complexity while building on insights from previous studies, isolating different aspects of reasoning and demonstrating how language-based reasoning scales across increasingly challenging domains.

6.1.2 Evolution of Reasoning Types

Each study emphasizes a different type of reasoning that is appropriate for its domain characteristics, demonstrating the flexibility and breadth of language-based reasoning approaches.

The chess study demonstrates that explicit reasoning explanations combining strategic and tactical components enhance decision-making quality. Strategic explanations articulate long-term positional principles such as piece activity, pawn structure, and king safety, while tactical explanations provide concrete move sequences and their consequences. This dual-mode approach mirrors how human expert chess players think, combining intuitive pattern recognition with deliberate calculation. The key insight is that making reasoning explicit through language improves performance by transforming implicit heuristics into verifiable logical steps.

The LLM^3 study shows that failure reasoning enables iterative improvement through understanding why actions fail. By categorizing motion planning failures into semantically meaningful types—specifically collision failures where goal configurations interfere with obstacles, and unreachability failures where goal configurations lie beyond the robot's workspace—the system can reason about which constraints were violated and how to adjust action parameters accordingly. This categorized feedback is substantially more informative than binary success/failure signals, allowing the LLM to generate targeted refinements that address specific geometric problems.

The EPoG study establishes that situated reasoning handles unexpected situations through dynamic adaptation. When the robot's predictions about object locations prove incorrect or when execution encounters obstacles, the system must reason about the current situation and generate appropriate responses. This includes updating beliefs based on observations, regenerating plans that account for newly discovered information, and handling exceptions such as blocking objects or inaccessible containers. The key capability is reasoning about the specific situation at hand rather than following predetermined scripts, enabling robust operation in unpredictable environments.

Each study builds on insights from the previous ones, demonstrating progressively sophisticated reasoning capabilities. The explicit reasoning developed for chess provides a foundation for understanding how language can make decision processes transparent and improvable. The failure reasoning developed for task and motion planning extends these ideas to iterative refinement based on execution feedback. The situated reasoning developed for exploration and manipulation integrates these concepts with belief management and dynamic replanning. Together, these studies chart a path from controlled reasoning in known environments to adaptive reasoning in partially known, dynamic settings.

6.2 Insights

Despite their different domains and challenges, the three studies reveal several common themes that provide general insights about how language models can support reasoning in complex systems.

6.2.1 Language as a Reasoning Medium

All three studies leverage language as a fundamental medium for reasoning, though in different ways appropriate to each domain's requirements. This consistent finding across diverse settings suggests that language plays a genuine functional role in reasoning rather than merely serving as a post-hoc description of decisions made through other mechanisms.

In the chess domain, strategic and tactical explanations guide move selection by articulating the reasoning behind each candidate move. Strategic explanations describe high-level positional principles such as "White increases central control" or "Black's king becomes vulnerable," while tactical explanations provide concrete sequences such as "After Knight takes pawn, Queen captures knight, leading to material advantage." This language makes implicit reasoning explicit and improves performance by enabling systematic evaluation of competing considerations. The 39% relative improvement observed when models receive these explanations demonstrates that linguistic articulation genuinely enhances decision quality.

In the task and motion planning domain, motion failure descriptions enable understanding of geometric constraints that would otherwise be opaque. When a motion planner reports that an action failed, a simple binary signal provides insufficient information for improvement. However, descriptions such as "Placement at location [x,y] causes collision with object on table" or "Target location [x,y] is unreachable given robot's workspace limits" provide actionable information about which constraints were violated. This textual feedback is more interpretable than numerical signals or geometric coordinates alone, allowing the LLM to reason about how to modify action parameters to satisfy the violated constraints.

In the exploration and manipulation domain, language provides actionable

information for both prediction and exception handling. Location estimates expressed as "cup is likely on coffee table in living room" or "bread is typically in kitchen cabinet" leverage commonsense knowledge about household organization. Exception descriptions such as "object X blocks access to object Y" or "container Z must be opened before retrieving object W" enable the system to understand and resolve obstacles. The linguistic representation provides a natural interface between perception (observing the environment), reasoning (understanding what the observations mean), and action (deciding how to respond).

The key insight unifying these uses of language is that it serves as an interface between multiple aspects of intelligent behavior. Language bridges abstract goals and concrete actions by enabling high-level objectives to be progressively refined into specific executable steps through explicit reasoning. It connects implicit knowledge stored in pre-trained models with explicit reasoning required for decision-making by forcing the articulation of relevant considerations. It provides a shared representation for communication between system components such as perception modules, planning algorithms, and execution systems, enabling modular architectures where components can operate independently while maintaining coherent overall behavior.

6.2.2 Domain-Independent Heuristics

All three frameworks demonstrate the utility of LLMs as providers of domain-independent heuristics that reduce search space, improve sample efficiency, and generalize across tasks without requiring manual engineering of domain-specific rules.

In the chess domain, strategic principles such as "control the center," "ensure king safety," and "improve piece activity" guide move evaluation without

requiring exhaustive search through all possible continuations. These principles, encoded in the language annotations and learned by the model during training, provide soft heuristics that focus attention on promising moves. Rather than considering all legal moves equally, the model prioritizes moves that align with established strategic principles while using tactical calculations to verify concrete consequences. This dramatically reduces the effective search space compared to uninformed approaches.

In the task and motion planning domain, action parameter selection uses implicit spatial intuition derived from the LLM's pre-training on text describing physical environments and object interactions. When asked to select a placement location for an object, the LLM draws on patterns learned from descriptions of household organization, typical object arrangements, and spatial relationships. This produces informed initial guesses that are far more likely to be geometrically feasible than random sampling. The experimental results demonstrate that LLM-based parameter sampling reduces motion planning queries by 90% compared to random sampling (from 663 calls to 70 calls on average), with each motion planning query representing expensive geometric computation.

In the exploration and manipulation domain, object location predictions prioritize exploration targets based on commonsense knowledge about household organization. Rather than systematically searching all rooms and all containers, the robot uses LLM predictions to focus on likely locations: "Cups are typically in the kitchen, often on countertops or in cabinets near the sink." These predictions are not always correct, but they provide much better guidance than hand-crafted rules that would need to enumerate all possible object-location associations. The 40% reduction in exploration compared to exhaustive search demonstrates that these heuristics effectively focus effort on

high-probability locations.

The key insight is that LLM commonsense knowledge, acquired through pre-training on diverse text corpora, provides soft heuristics that guide reasoning without the brittleness of hard-coded rules. These heuristics reduce search space by focusing attention on promising options, improve sample efficiency by avoiding obviously poor choices, and generalize across domains because they draw on broad patterns rather than task-specific engineering. While these heuristics are not perfect and require verification through classical methods (motion planning, perception, execution), they dramatically improve the efficiency of search and reasoning processes.

The domain independence of these heuristics is particularly valuable because it reduces the engineering effort required to deploy systems in new environments. Rather than manually encoding rules like "cups are usually in kitchens" or "place objects near the center of surfaces," we can leverage knowledge already present in pre-trained models. This democratizes access to sophisticated reasoning capabilities and enables rapid adaptation to new tasks.

6.3 Summary of Contributions

This dissertation systematically investigated and enhanced reasoning capabilities of Large Language Models across three progressively complex domains: constrained game playing (chess), task and motion planning (robotic manipulation), and sequential manipulation in partially known environments (exploration-based planning). Through this progression, we have made contributions to the understanding of how language models can support reasoning in complex autonomous systems.

Our contributions demonstrate the practical effectiveness of our approaches

through comprehensive experiments in simulation and validation on physical robots.

The quantitative results from our three studies provide strong evidence for the benefits of language-based reasoning. In chess, our fine-tuned models achieve 95.2% accuracy on positions with both strategic and tactical annotations, representing a 24.2% error rate reduction compared to the best commercial LLM (o1-preview at 78.6%). This dramatic improvement demonstrates that domain-specific training with expert language annotations can overcome the advantages of scale and general capabilities present in much larger frontier models. In task and motion planning, LLM^3 achieves 91.3% success rate across diverse manipulation scenarios while requiring 36.1% fewer motion planning calls compared to methods without feedback, demonstrating both effectiveness and efficiency. In exploration and manipulation, EPoG achieves 91.3% success rate while exploring 40% fewer nodes and traveling 36.2% less distance compared to exhaustive exploration, showing that informed exploration guided by LLM predictions dramatically improves efficiency without sacrificing reliability.

Our real-world validation through physical robot experiments demonstrates that these approaches transfer effectively from simulation to physical systems despite added challenges of perception uncertainty, execution noise, and environmental variability. The LLM^3 framework was successfully deployed on a Franka Research 3 manipulator, where it handled complex scenarios including unknown object locations that required belief updating when predictions proved incorrect, and cascading exceptions involving closed containers with blocking objects requiring multi-step corrective actions. The EPoG framework was similarly validated on a mobile manipulator, demonstrating robust performance on tasks requiring integrated exploration and manipulation with

dynamic replanning based on observations. These real-world demonstrations confirm that the benefits observed in controlled simulation studies persist in realistic deployment scenarios, validating the practical applicability of our approaches.

6.4 Limitations

While this dissertation demonstrates significant progress in language-based reasoning for autonomous systems, our work also reveals several important limitations that suggest directions for future research.

One limitation stems from context window constraints inherent in transformer-based architectures. All current language models have finite context windows that limit the amount of information they can consider simultaneously. This constrains the length of reasoning traces that can be maintained across multiple iterations, as each iteration adds feedback that consumes context space. It limits the size of scene graphs that can be processed, as large environments with hundreds of objects may exceed the model's capacity. It restricts the number of past iterations that can inform current decisions, potentially causing the system to repeat mistakes when earlier lessons fall outside the context window. While context windows have grown substantially (from 2K tokens in early GPT models to 200K+ in recent systems), they remain finite, and complex real-world tasks can still exceed these limits. Future work might address this through hierarchical memory systems that selectively retain important information while summarizing or discarding less relevant details.

Another limitation is the difficulty of spatial and geometric reasoning in language models trained primarily on text. LLMs struggle with precise three-dimensional spatial relationships that require accurate geometric computation, such as determining exact collision geometries or computing optimal grasp

points. They have difficulty verifying geometric feasibility without external tools, unable to reliably determine whether a proposed object placement will actually fit in available space or whether a robot can reach a specified location. They cannot perform accurate physics-based prediction of dynamic events such as object trajectories or stability under manipulation. These limitations stem from the fundamental mismatch between linguistic representations (which are inherently discrete and sequential) and spatial relationships (which are continuous and multi-dimensional). This necessitates integration with specialized geometric reasoning modules such as motion planners, collision checkers, and physics simulators, as demonstrated throughout our work. Future research might explore multi-modal architectures that more tightly integrate visual and geometric representations with linguistic reasoning.

A third limitation is maintaining long-horizon coherence as planning horizons increase. When tasks require sequences of 20+ actions spanning multiple subgoals, several problems emerge. Action sequences may become repetitive, with the model suggesting the same action multiple times or cycling through previously attempted approaches. Actions may become contradictory, violating earlier constraints or undoing previously achieved subgoals. Subgoal tracking deteriorates as the model loses sight of which intermediate objectives have been achieved and which remain. Error accumulation affects downstream decisions as small mistakes in early actions compound to create larger problems later. These issues are particularly pronounced in EPoG where long-horizon household tasks require coordinated manipulation of multiple objects across multiple rooms. While our hierarchical architectures partially address these issues through structured planning at multiple levels, maintaining perfect coherence over very long horizons remains challenging. Future work might explore explicit subgoal tracking mechanisms or periodic re-evaluation of overall

progress.

6.5 Closing Remarks

This dissertation explored how Large Language Models can reason effectively in complex, real-world scenarios through three complementary studies spanning controlled game domains to uncertain physical environments. The progression from chess's structured simplicity to manipulation planning's geometric complexity to exploration's environmental uncertainty has demonstrated several key principles.

Language explanations genuinely enhance reasoning quality, not merely providing post-hoc descriptions but actively improving decision-making through explicit articulation of reasoning steps. Structured feedback enables iterative improvement, where categorized information about failures and observations guides progressive refinement toward successful solutions. Hybrid architectures that thoughtfully integrate language models with classical methods leverage complementary strengths, achieving results superior to either approach alone. Domain-independent heuristics from pre-trained models reduce engineering effort while providing valuable guidance that generalizes across tasks. Practical efficiency in terms of computational cost and physical effort is achievable alongside high accuracy through informed search and optimization.

While significant challenges remain—particularly in geometric reasoning, long-horizon coherence, and computational efficiency—the results presented here suggest that LLMs can play a central role in advancing autonomous systems' reasoning capabilities. The frameworks and insights provide concrete demonstrations of how language-based reasoning can be effectively integrated into complete systems that perceive, reason, plan, and act in complex environments.

As Large Language Models continue to evolve, becoming more capable, efficient, and reliable, their integration with classical planning methods, perception systems, and physical robots promises to enable increasingly sophisticated autonomous behavior. The careful attention to integration strategies, structured representations, and iterative refinement demonstrated in this work will remain essential as we push toward more ambitious applications.

The journey from chess board to household robot represents not merely increasing task complexity, but a fundamental progression in how artificial systems can reason about the world, adapt to uncertainty, and accomplish meaningful goals. Chess taught us that language explanations combining strategic and tactical reasoning mirror human expert thinking and dramatically improve decision quality. Task and motion planning showed us that categorized failure feedback enables efficient iterative refinement across symbolic and geometric domains. Exploration-based manipulation demonstrated that situated reasoning with dynamic belief updates enables robust operation despite fundamental uncertainty about the world.

Together, these studies chart a path forward for autonomous systems that genuinely reason through language. By making reasoning explicit, structuring feedback appropriately, and thoughtfully integrating diverse capabilities, we can build systems that exhibit increasingly sophisticated intelligence. The principles established here—explicit reasoning, iterative refinement, hybrid architectures, domain-independent heuristics, and practical efficiency—provide a foundation for continued progress toward autonomous systems that can robustly handle the complexity and uncertainty of real-world environments.

As we continue to push the boundaries of what AI systems can understand and accomplish, these principles will remain essential guides. The future of autonomous systems lies not in replacing human reasoning or traditional

algorithms, but in thoughtfully combining the unique strengths of language models, classical methods, and structured knowledge to create systems that are more capable than any single approach alone. This dissertation provides both theoretical insights and practical frameworks for pursuing this vision, contributing to the ongoing effort to build truly intelligent systems that can reason effectively in the face of uncertainty and achieve meaningful goals in the real world.

"The future belongs to those who can reason effectively in the face of uncertainty."

Bibliography

Iro Armeni et al. 3d scene graph: A structure for unified semantics, 3d space, and camera. In *IEEE International Conference on Computer Vision*, 2019.

Tom Brown et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Sébastien Bubeck et al. Sparks of artificial general intelligence: Early experiments with gpt-4. 2023.

Tianyu Chen, Yasi Zhang, Zhi Zhang, Peiyu Yu, Shu Wang, Zhendong Wang, Kevin Lin, Xiaofei Wang, Zhengyuan Yang, Linjie Li, et al. Edival-agent: An object-centric framework for automated, fine-grained evaluation of multi-turn editing. *arXiv preprint arXiv:2509.13399*, 2025.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. *arXiv:2306.06531*, 2023.

Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems (RSS)*, 2016.

Matt Deitke et al. Procthor: Large-scale embodied ai using procedural generation. In *Advances in Neural Information Processing Systems*, 2022.

Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

Abhimanyu Dubey et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Nouha Dziri et al. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 2024.

Evelina Fedorenko et al. Language is primarily a tool for communication rather than thought. *Nature*, 630:575–586, 2024.

Xiaofeng Gao, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. Vrkitchen: an interactive 3d virtual environment for task-oriented learning. *arXiv preprint arXiv:1903.05757*, 2019.

Xiaofeng Gao, Ran Gong, Yizhou Zhao, Shu Wang, Tianmin Shu, and Song-Chun Zhu. Joint mind modeling for explanation generation in complex human-robot collaborative tasks. In *2020 29th IEEE international conference on robot and human interactive communication (RO-MAN)*, pages 1119–1126. IEEE, 2020.

Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S Sukhatme. Dialfred: Dialogue-enabled agents for embodied instruction following. *IEEE Robotics and Automation Letters*, 7(4):10049–10056, 2022.

Alan Garnham and Jane Oakhill. *Thinking and reasoning*. Basil Blackwell, 1994.

Caelan Garrett et al. Integrated task and motion planning. 2021.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. 2020a.

Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online replanning in belief space for partially observable task and motion problems. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020b.

Ran Gong, Xiaofeng Gao, Qiaozi Gao, Suhaila Shakiah, Govind Thattai, and Gaurav S Sukhatme. Lemma: Learning language-conditioned multi-robot manipulation. *IEEE Robotics and Automation Letters*, 8(10):6835–6842, 2023.

Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. Interpret: Interactive predicate learning from language feedback for generalizable task planning. *arXiv preprint arXiv:2405.19758*, 2024.

Muzhi Han et al. Reconstructing interactive 3d scenes by panoptic mapping and cad model alignments. In *IEEE International Conference on Robotics and Automation*, 2021.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022a.

Wenlong Huang et al. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *International Conference on Machine Learning*, 2022b.

Wenlong Huang et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, 2023.

Han Jiang, Xiaoyuan Yi, Zhihua Wei, Shu Wang, and Xing Xie. Raising the bar: Investigating the values of large language models via generative evolving testing. *arXiv:2406.14230*, 2024a.

Hanxiao Jiang et al. Roboexp: Action-conditioned scene graph via interactive exploration for robotic manipulation. 2024b.

Ziyuan Jiao et al. Efficient task planning for mobile manipulation: a virtual kinematic chain perspective. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

Ziyuan Jiao et al. Sequential manipulation planning on scene graph. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

Takeshi Kojima et al. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

Deqian Kong, Minglu Zhao, Dehong Xu, Bo Pang, Shu Wang, Edouardo Honig, Zhangzhang Si, Chuan Li, Jianwen Xie, Sirui Xie, et al. Scalable language models with posterior inference of latent thought vectors. *arXiv e-prints*, pages arXiv–2502, 2025.

Puhao Li, Tengyu Liu, Yuyang Li, Muzhi Han, Haoran Geng, Shu Wang, Yixin Zhu, Song-Chun Zhu, and Siyuan Huang. Ag2manip: Learning novel

manipulation skills with agent-agnostic visual and action representations. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 573–580. IEEE, 2024.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Xiao Liang, Zhong-Zhi Li, Yeyun Gong, Yang Wang, Hengyuan Zhang, Yelong Shen, Ying Nian Wu, and Weizhu Chen. Sws: Self-aware weakness-driven problem synthesis in reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.08989*, 2025.

Ziyong Lin, Haoyi Wu, Shu Wang, Kewei Tu, Zilong Zheng, and Zixia Jia. Look both ways and no sink: Converting llms into text encoders without training. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22839–22853, 2025.

Suvir Mirchandani, Fei Xia, Pete Florence, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, Andy Zeng, et al. Large language models as general pattern machines. In *Conference on Robot Learning (CoRL)*, 2023.

Iman Mirzadeh et al. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Zhixiong Nan, Tianmin Shu, Ran Gong, Shu Wang, Ping Wei, Song-Chun Zhu,

and Nanning Zheng. Learning to infer human attention in daily activities. *Pattern Recognition*, 103:107314, 2020.

Lixing Niu, Jiapeng Li, Xingping Yu, Shu Wang, Ruining Feng, Bo Wu, Ping Wei, Yisen Wang, and Lifeng Fan. Rˆ 3-vqa:" read the room" by video social reasoning. *arXiv preprint arXiv:2505.04147*, 2025.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

Antoni Rosinol et al. Kimera: From slam to spatial perception with 3d dynamic scene graphs. *International Journal of Robotics Research*, 2020.

Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, and Tim Genewein. Grandmaster-level chess without search. *arXiv preprint arXiv:2402.04494*, 2024.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, Brian Ichter, and Sergey Levine. Navigation with large language models: Semantic guesswork as a heuristic for planning. In *Conference on Robot Learning (CoRL)*, pages 2683–2699. PMLR, 2023.

David Silver et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 2017.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *IEEE International Conference on Computer Vision (ICCV)*, 2023.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. *arXiv preprint arXiv:2410.03439*, 2024a.

Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. Llmˆ3: Large language model-based task and motion planning with motion failure reasoning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024b.

Shu Wang, Lei Ji, Renxi Wang, Wenxiao Zhao, Haokun Liu, Yifan Hou, and

Ying Nian Wu. Explore the reasoning capability of llms in the chess testbed. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 611–622, 2025.

Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Active model learning and diverse action sampling for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022b.

Fangzhi Xu et al. Are large language models really good logical reasoners? *arXiv preprint arXiv:2306.09841*, 2023.

Yiming Xu et al. A framework to co-optimize robot exploration and task planning in unknown environments. *IEEE Robotics and Automation Letters*, 2022.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*, 2024.

Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. Isr-llm: Iterative self-refined large language model for long-horizon sequential task

planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2081–2088. IEEE, 2024.