Enhancing Large Language Model Reasoning for Optimization Problems: Methods, Evaluation, and Application to Urban Planning

by

Zahra Sarayloo

Supervisor(s):
David C. Del Rey Fernández
Leia Minaker
Stephen L. Smith

A thesis
presented to the University of Waterloo
in fulfillment of the
research paper requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2026

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis investigates the use of large language models (LLMs) for solving mathematical and optimization problems expressed in natural language, with a particular focus on applications in urban planning. While recent reasoning methods such as Chain-of-Thought (CoT), Tree-of-Thought (ToT), and Program-of-Thought (PoT) have improved multi-step reasoning, these approaches remain limited in their ability to enforce constraints, ensure solution validity, and handle structured optimization tasks.

To address these challenges, this work proposes a unified reasoning framework that integrates structured reasoning, execution-based computation, and verification mechanisms. The framework combines multiple reasoning strategies, including CoT and PoT, with external validation through a verifier-in-the-loop design, where constraint satisfaction and solution consistency are explicitly checked. In addition, a reasoning vector engineering approach is introduced to analyze and influence internal model representations, enabling improved control over the reasoning process.

The proposed methods are evaluated on two benchmark datasets, NL4OPT and LiveMathBench, which include optimization and mathematical reasoning problems in natural language form. Furthermore, the framework is applied to a real-world case study of high school site selection in the Waterloo Region, formulated as a constrained spatial optimization problem based on planning criteria such as accessibility, zoning, and land availability.

The results demonstrate that combining structured reasoning with verification significantly improves the reliability and feasibility of solutions compared to baseline methods. The proposed approach provides a practical pathway for bridging natural language understanding and formal optimization, enabling the use of LLMs in complex decision-making tasks.

Acknowledgements

I would like to sincerely thank my supervisors, Prof. David C. Del Rey Fernández, Prof. Leia Minaker, and Prof. Stephen L. Smith, for their guidance, support, and valuable insights throughout this research. I am also grateful to the Waterloo Region District School Board for providing valuable information and resources related to urban planning and high school site selection. My heartfelt thanks go to my family, my spouse, and my dear friends for their constant support and encouragement. I would also like to acknowledge the members of the Computational Mathematics (CM) lab and all those who supported me during this work. Their support has been truly invaluable.

Dedication

This work is dedicated to my beloved parents.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Chapter Overview

Urban planning is a complex decision-making process that requires balancing economic, social, and environmental objectives under multiple constraints. Many real-world planning problems, such as facility location, land-use allocation, and infrastructure development, can be formulated as optimization problems [3]. However, these problems are rarely presented in a formal mathematical structure[4]. Instead, they are typically expressed in natural language through policy documents, regulations, and stakeholder requirements. This gap between natural language descriptions and formal optimization models presents a significant challenge. While optimization techniques require structured inputs such as variables, constraints, and objective functions, planners must manually translate textual information into mathematical formulations. This process is time-consuming, error-prone, and difficult to scale.

Recent advances in Large Language Models (LLMs) offer a promising direction to bridge this gap. LLMs have demonstrated strong capabilities in understanding and generating human language, as well as performing reasoning tasks. However, despite these capabilities, their application to optimization and mathematical problem solving remains problematic due to issues such as weak constraint handling, hallucination, and lack of verification.

This thesis is motivated by the need to develop a framework that enables LLMs to transform natural language problems into reliable optimization solutions. In particular, this work investigates how structured reasoning, external verification, and internal representation steering can improve the performance of LLMs in optimization and decision-making tasks, with a specific application to urban planning. In the rest of this chapter, we

introduce the fundamental context of the thesis. It begins by reviewing the development of LLMs and their reasoning capabilities, followed by a discussion of challenges in urban planning and the role of LLMs in decision-making processes. The chapter then examines the limitations of current LLM-based approaches for optimization and mathematical reasoning and formally defines the problem addressed in this work. Finally, the proposed approach is outlined and the overall structure of the thesis is presented to guide the reader through the subsequent chapters.

1.2 Large Language Models

LLMs have fundamentally transformed the field of Artificial Intelligence (AI), evolving from text generation systems into models capable of reasoning, planning, and decision-making [5]. This transition reflects a broader shift from generative AI systems to agentic AI systems, where models are expected to perform multi-step reasoning and interact with external tools [6, 7]. Recent developments in LLM reasoning include structured prompting techniques such as Chain-of-Thought (CoT), Tree-of-Thought (ToT), and Program-of-Thought (PoT), which guide the model to generate intermediate reasoning steps [7]. These approaches aim to improve reasoning by decomposing complex problems into simpler subproblems. Despite these advances, LLMs still face challenges in logical consistency and reasoning alignment [8, 9]. Mathematical reasoning, in particular, remains one of the most difficult tasks for these models, requiring precise logical structure and correctness guarantees [10]. Recent work has also highlighted the importance of combining reasoning generation with verification mechanisms and structured feedback to improve reliability [11].

In addition to output-level improvements, recent research has explored reasoning from the perspective of internal representations. Representation engineering studies how hidden activations encode reasoning patterns and how they can be manipulated to guide model behavior [12]. This has led to the development of reasoning vectors, which capture directions in latent space associated with correct reasoning. These techniques have been extended to multilingual reasoning and robustness analysis [13, 14], and further studies suggest that reasoning improvements correspond to localized transformations in representation space rather than global changes [15].

1.3 Urban Planning Challenges

Urban planning is inherently a multi-objective optimization problem involving diverse and often conflicting criteria such as accessibility, sustainability, cost, and equity [16]. Modern cities introduce additional complexity through increasing population density, infrastructure demands, and regulatory constraints [17]. Planning problems are typically weakly structured, meaning that they involve heterogeneous data sources and multiple stakeholders [3]. Decisions are often based on textual descriptions rather than structured data, making it difficult to directly apply traditional optimization techniques. Furthermore, urban planning requires integrating spatial reasoning, policy constraints, and long-term forecasting. This complexity makes it challenging to design automated systems that can reliably support decision-making. Traditional approaches rely heavily on expert knowledge and manual modeling, which limits scalability and reproducibility.

1.4 LLMs in Urban Planning and Decision-Making

LLMs have recently been explored as tools for supporting urban planning and decision-making processes. Studies have shown that LLMs can assist in tasks such as policy interpretation, scenario analysis, and stakeholder simulation [18, 19]. Multi-agent LLM systems have been proposed to simulate interactions between stakeholders and generate planning strategies.

Recent benchmark studies demonstrate that LLMs can perform a wide range of urban planning tasks, including documentation, analysis, and coding, although they still struggle with spatial reasoning and high-level decision-making [20]. Other work has focused on prompt engineering frameworks to improve LLM performance in planning applications, emphasizing structured prompt design for complex decision environments [3].

LLM-based planning agents have also been developed to evaluate urban development proposals, showing potential for automated planning support while highlighting limitations such as lack of spatial awareness and over-conservative decision behavior [21]. Hybrid systems such as City-LEO demonstrate how LLMs can be integrated with optimization frameworks to translate natural language queries into mathematical models [22]. More broadly, LLMs are increasingly viewed as general-purpose intelligence systems capable of reasoning, planning, and decision-making across domains [23]. However, their performance in complex decision-making tasks remains limited due to challenges in constraint handling, long-term planning, and causal reasoning [24].

1.5 LLMs for Optimization and Mathematical Reasoning

Optimization and mathematical reasoning represent critical challenges for LLMs. Unlike natural language tasks, these problems require strict adherence to logical and numerical constraints. Recent studies highlight that LLMs often fail to maintain consistency and accuracy in mathematical reasoning tasks [25, 26].

Efforts to improve LLM performance in optimization include frameworks that decompose problems into structured sub-tasks and integrate external solvers. The OR-LLM-Agent framework demonstrates how LLMs can automate the modeling and solving of operations research problems through structured reasoning pipelines [27]. Similarly, neuro-symbolic approaches combine LLM reasoning with deterministic logic engines to improve reliability in optimization tasks [28].

Despite these advances, challenges remain in translating natural language into formal optimization models and ensuring solution correctness. Errors in problem formulation can lead to infeasible or suboptimal solutions, highlighting the need for improved reasoning and verification mechanisms.

1.6 Challenges in LLM-Based Optimization

Although LLMs provide a flexible interface for reasoning over textual data, they exhibit several fundamental limitations when applied to optimization problems. One major limitation is their inability to inherently enforce constraints, which can lead to solutions that violate problem requirements. In addition, their mathematical reasoning capabilities remain limited, particularly for multi-step problems that require precise logical consistency [4]. Another challenge is the presence of hallucinations, where the model may generate incorrect or unsupported conclusions with high confidence [8]. Furthermore, LLMs do not include built-in verification mechanisms, making it difficult to guarantee the correctness and feasibility of their outputs. These limitations indicate that LLMs alone are insufficient for solving optimization problems reliably. Instead, they must be combined with structured reasoning methods and external verification mechanisms.

1.7 Problem Statement

This thesis investigates the use of LLMs for solving optimization and mathematical problems that are expressed in natural language. Many real-world decision-making problems, particularly in domains such as urban planning, can be formulated as constrained optimization problems. However, these problems are typically described in textual form, requiring a transformation from natural language into structured mathematical representations before they can be solved.

The primary objective of this work is to improve the capability of LLMs to reliably solve such problems. Rather than directly applying LLMs to domain-specific applications, this study first focuses on developing stable and effective reasoning mechanisms for solving complex optimization and mathematical tasks. Urban planning challenges serve as a motivating application domain, as many planning problems naturally involve multi-constraint optimization under uncertainty. In particular, this research considers the problem of high school site selection in the Waterloo Region as a representative case study. This problem involves selecting suitable locations based on multiple criteria such as accessibility, land availability, regulatory constraints, and environmental considerations. To enable systematic evaluation, the problem is defined and simplified using criteria derived from planning documents provided by the Waterloo Region District School Board [29], while synthetic data is used to construct candidate solutions and test scenarios.

To evaluate the effectiveness of the proposed methods, this work employs two benchmark datasets. The first dataset, NL4OPT [4, 30], consists of optimization problems described in natural language along with their corresponding structured formulations and solutions. The second dataset, LiveMathBench [31], contains a diverse set of mathematical problems across multiple topics, providing a testbed for evaluating reasoning capabilities in symbolic and numerical tasks.

The central research question addressed in this thesis is:

> How can structured reasoning methods, external verification, and representation-level interventions improve the ability of LLMs to solve optimization and mathematical problems derived from natural language descriptions?

1.8 Proposed Approach

To address these challenges, this thesis proposes a unified framework for enhancing LLM reasoning for optimization and mathematical problem solving. The approach is developed

progressively, beginning with baseline reasoning strategies and extending toward more advanced architectures that improve reliability and performance.

The study begins by examining structured reasoning methods, including CoT and PoT, as foundational approaches for guiding LLM reasoning. Initial experiments investigate the effectiveness of these methods in solving optimization and mathematical problems. Building on these baselines, the reasoning architectures are refined to improve solution quality, robustness, and interpretability. In particular, PoT-based approaches are extended to better integrate computation and reasoning, allowing the model to generate executable solutions that can be validated externally.

A key component of the proposed framework is the integration of a verifier-in-the-loop mechanism. In this setting, candidate solutions generated by the LLM are evaluated using external verification processes that assess constraint satisfaction and solution quality. Feedback from the verifier is incorporated into subsequent reasoning steps, enabling iterative refinement and reducing the likelihood of infeasible or incorrect outputs.

In addition to output-level improvements, this work investigates representation-level reasoning enhancement through reasoning vector engineering. This method analyzes hidden-state activations associated with correct and incorrect reasoning trajectories and constructs vectors that capture meaningful directions in the model's representation space. These vectors are then used to steer the model during inference, with experiments conducted on a LLaMA model with 3 billion parameters to evaluate the impact of this approach on mathematical reasoning performance.

The proposed framework is evaluated across both benchmark datasets and the urban planning case study. In the case study, the developed methods are applied to the high school site selection problem, demonstrating how natural language descriptions of planning criteria can be transformed into structured decision-making processes. This application highlights the broader goal of the research, which is to enable LLMs to serve as reliable tools for solving real-world optimization problems that originate from textual descriptions.

1.9 Thesis Structure

The remainder of this thesis is organized as follows.

Chapter 2 provides a review of existing reasoning methods in LLMs, with a focus on structured approaches such as CoT, ToT, and PoT. Chapter 3 introduces the proposed framework for enhancing LLM reasoning in optimization and mathematical tasks, including the integration of a verifier-in-the-loop architecture and the development of reasoning

vector techniques. Chapter 4 presents the experimental evaluation of the proposed methods on benchmark datasets, examining their performance, robustness, and limitations. Chapter 5 applies the developed framework to a real-world urban planning case study on high school site selection, illustrating its practical applicability. Finally, chapter 6 concludes the thesis and discusses potential directions for future research.

Chapter 2

Reasoning Frameworks in Large Language Models

2.1 Motivation and Chapter Overview

The development of LLMs has significantly advanced the ability of AI systems to process and generate natural language. Beyond standard language tasks, recent work has explored their use for reasoning in domains that require multi-step inference, structured decision-making, and mathematical formulation. Despite these advances, LLMs remain limited in solving structured problems reliably, particularly those involving optimization and formal constraints[32]. These limitations stem from challenges such as error propagation in sequential reasoning [15], lack of constraint enforcement, and the absence of verification mechanisms[33].

To address these issues, a range of reasoning frameworks has been proposed to guide the inference process of LLMs [12]. These approaches differ in how reasoning is structured, how the solution space is explored, and how correctness is ensured. Some methods focus on generating intermediate reasoning steps, while others incorporate stochastic sampling, search-based exploration, executable representations, or external verification [12].

This chapter provides a structured review of these frameworks. Section 2.2 introduces Chain-of-Thought (CoT) reasoning, which improves multi-step reasoning by generating explicit intermediate steps. Section 2.3 presents self-consistency, which enhances robustness by aggregating multiple reasoning trajectories. Section 2.4 describes Tree-of-Thought (ToT) reasoning, which formulates reasoning as a search process over multiple candidate

paths. Section 2.5 introduces Program-of-Thought (PoT), where reasoning is expressed as executable programs to improve computational accuracy.

The discussion then focuses on reliability. Section 2.6 reviews verifier-based frameworks that evaluate reasoning steps and enforce logical consistency. Section 2.7 examines execution-based and trace-grounded reasoning, where reasoning is tied to executable processes. Finally, section 2.8 compares these approaches, and Section 2.9 summarizes the key insights.

2.2 Chain-of-Thought Reasoning

CoT reasoning is a prompting-based framework that improves the ability of LLMs to solve complex reasoning tasks by explicitly generating intermediate reasoning steps. This approach was introduced by Wei et al. [34], who showed that providing examples of step-by-step reasoning in the prompt enables large models to perform significantly better on arithmetic, commonsense, and symbolic reasoning tasks. The key idea is to augment standard input-output examples with natural language explanations that reflect the reasoning process leading to the final answer.

Let $x \in \mathcal{X}$ denote an input problem, such as a mathematical word problem, and let $y \in \mathcal{Y}$ denote the corresponding output. In standard prompting, the model directly generates the output as $y \sim p_\theta(y \mid x)$, where p_θ represents the conditional probability distribution defined by the model with parameters θ. In contrast, CoT introduces an intermediate sequence of reasoning steps $z = (z_1, z_2, \ldots, z_T)$, where each z_t corresponds to a natural language statement describing a partial step in the reasoning process, and T denotes the total number of steps [34]. The generation process can then be expressed as:

$$z_t \sim p_\theta(z_t \mid x, z_{<t}), \quad t = 1, \ldots, T, \quad \text{and} \quad y \sim p_\theta(y \mid x, z), \tag{2.1}$$

where $z_{<t} = (z_1, \ldots, z_{t-1})$ represents the previously generated reasoning steps [34].

In practice, CoT is implemented through few-shot prompting, where the model is provided with a set of examples of the form $\langle x^{(i)}, z^{(i)}, y^{(i)} \rangle$. Each example includes an input problem $x^{(i)}$, a corresponding chain of reasoning steps $z^{(i)}$ [34], and a final answer $y^{(i)}$. At inference time, given a new input x^*, the model generates a reasoning sequence z^* followed by the final output y^*. This mechanism leverages in-context learning, allowing the model to imitate the structure of reasoning demonstrated in the prompt without requiring additional training [34]. As illustrated in the original work, the inclusion of reasoning

steps enables the model to correctly solve problems that it fails to answer under standard prompting [34].

The effectiveness of CoT stems from its ability to decompose complex problems into a sequence of simpler steps [35]. Each intermediate step incrementally transforms the input into a more structured representation, reducing the difficulty of the final prediction. In addition, generating intermediate tokens effectively increases the computational budget allocated to the problem, allowing the model to perform more extensive reasoning before producing the final answer [36]. The resulting reasoning sequence also provides an interpretable trace of the model's decision-making process, which can be useful for analysis and debugging [36].

Despite these advantages, CoT reasoning has several limitations. The reasoning process is sequential and greedy, meaning that errors in early steps can propagate through the entire chain and lead to incorrect final answers. Furthermore, there is no explicit mechanism to verify the correctness of intermediate steps [35], and the model may generate reasoning that appears plausible but is logically or mathematically incorrect. The original study also shows that the benefits of CoT emerge primarily in sufficiently large models, while smaller models often produce inconsistent or invalid reasoning chains [36]. These limitations highlight the need for additional mechanisms, such as verification and structured control, to ensure reliable reasoning in complex tasks.

2.3 Self-Consistency in Reasoning

Self-Consistency (SC) is a decoding strategy designed to improve the reliability of CoT reasoning in LLMs. It was introduced by Wang et al. [37] as an extension of CoT that replaces greedy decoding with a sampling-based aggregation mechanism. The core idea is that complex reasoning problems often admit multiple valid reasoning paths that converge to the same correct answer. By leveraging this diversity, SC aims to identify the most reliable solution through agreement among independently generated reasoning trajectories.

Let $x \in \mathcal{X}$ denote the input problem and let $y \in \mathcal{Y}$ denote the final answer, consistent with the notation introduced in Section 2.2. In the CoT framework, a single reasoning sequence $z = (z_1, \ldots, z_T)$ is generated, followed by the output y. In contrast, self-consistency introduces multiple reasoning trajectories by sampling from the model's conditional distribution. Specifically, the model generates M independent reasoning paths [37]:

$$z^{(i)} \sim p_\theta(z \mid x), \quad i = 1, \ldots, M, \tag{2.2}$$

where each $z^{(i)}$ represents a full chain of reasoning steps. Each reasoning path produces a corresponding answer [37]:

$$y^{(i)} \sim p_\theta(y \mid x, z^{(i)}). \tag{2.3}$$

The final prediction is then obtained by aggregating these candidate answers through a marginalization process over the reasoning paths. In practice, this is implemented as a majority vote [37]:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{M} \mathbb{I}(y^{(i)} = y), \tag{2.4}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function, which takes the value 1 if the argument is true and 0 otherwise [37].

This procedure can be interpreted as approximating the marginal probability of an answer by integrating over latent reasoning paths. Each sampled reasoning trajectory acts as a latent variable that contributes to the final decision. This "sample-and-marginalize" strategy replaces the deterministic greedy decoding used in standard CoT with a stochastic exploration of the reasoning space.

The effectiveness of self-consistency arises from the observation that correct reasoning paths, even if diverse in structure, tend to converge to the same final answer, whereas incorrect reasoning paths are more likely to produce inconsistent outputs. By aggregating across multiple samples, the method reduces the impact of individual reasoning errors and mitigates the risk of selecting a suboptimal reasoning trajectory. This behavior is analogous to a self-ensemble mechanism, where multiple reasoning attempts from a single model are combined to improve robustness [37].

In practice, diversity in reasoning paths is achieved through stochastic decoding strategies such as temperature sampling, top-k sampling, or nucleus sampling, which allow the model to explore different plausible reasoning trajectories. Increasing the number of sampled paths generally improves performance, as it provides a richer set of candidate solutions and increases the likelihood of identifying the correct answer. However, this comes at the cost of increased computational complexity, as multiple forward passes through the model are required [37].

Despite its effectiveness, SC does not explicitly verify the correctness of reasoning steps. The aggregation process operates only on the final answers, meaning that incorrect reasoning paths can still influence the result if they produce the same output. Additionally, the method assumes that the correct answer is consistent across reasoning paths, which may not hold in tasks with ambiguous or open-ended outputs. Nevertheless, SC provides a

simple and effective mechanism for improving CoT reasoning without requiring additional training, supervision, or external modules [37].

2.4 Tree-of-Thought Reasoning

ToT is a generalization of CoT that formulates reasoning as a structured search process over multiple intermediate reasoning paths [1]. Unlike CoT, which generates a single reasoning trajectory in a left-to-right manner, ToT explicitly explores and evaluates multiple candidate reasoning paths, enabling deliberate decision-making through planning, lookahead, and backtracking.

Figure 2.1 illustrates the difference between standard prompting, CoT, SC, and the ToT framework. Each box represents a reasoning step, and ToT enables structured exploration over multiple reasoning paths [1].

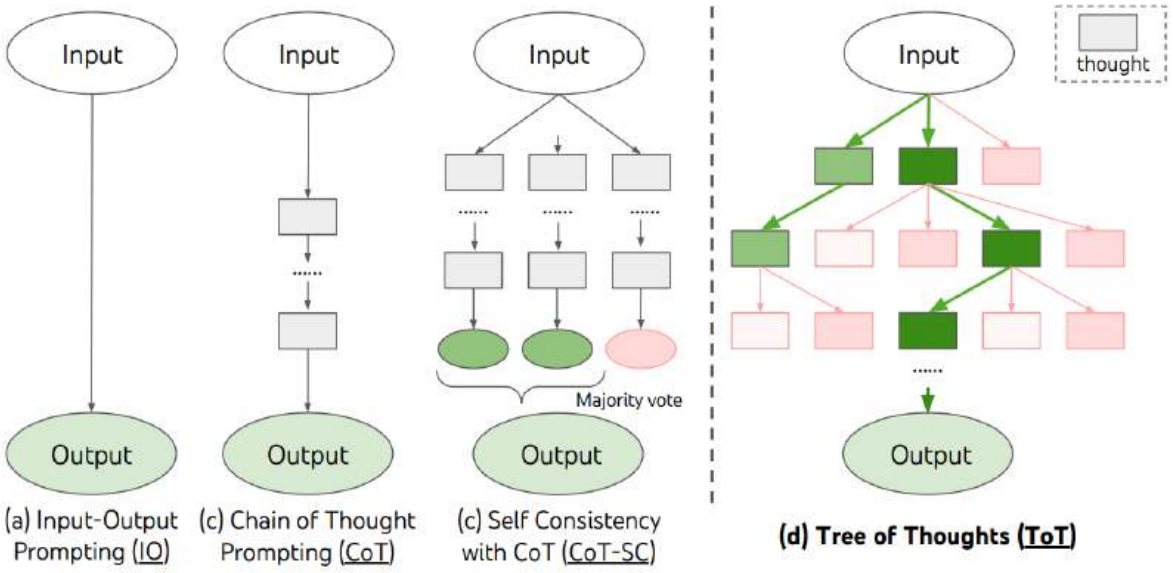

Figure 2.1: Comparison of reasoning frameworks, including input-output prompting, CoT, SC, and ToT (figure reproduced from [1]).

The key idea is to represent reasoning as a tree of *thoughts*, where each thought corresponds to a coherent intermediate reasoning step. Formally, given an input $x \in \mathcal{X}$, a

reasoning state at step t is defined as [1]

$$s_t = (x, z_{1:t}), \tag{2.5}$$

where $z_{1:t} = (z_1, \ldots, z_t)$ denotes the sequence of thoughts generated up to step t [1]. Each node in the tree corresponds to such a state, representing a partial solution to the problem.

At each step, the language model generates multiple candidate thoughts conditioned on the current state [1]:

$$z^{(i)} \sim p_\theta(\cdot \mid s_t), \quad i = 1, \ldots, k, \tag{2.6}$$

where k denotes the number of candidate thoughts. Each candidate extends the current state, forming a new set of expanded states:

$$s_{t+1}^{(i)} = (x, z_{1:t} \oplus z^{(i)}), \tag{2.7}$$

where $\oplus$ denotes sequence concatenation [1].

To guide the search process, a scoring function $H(s)$ evaluates the quality of each state based on its potential to reach a correct solution. This evaluation can be performed using heuristic rules or by prompting the language model itself to assess intermediate reasoning states [1]. The search procedure then selects a subset of promising states according to a predefined strategy, such as beam search [1].

A common instantiation of ToT is the beam search variant, summarized in Algorithm 1 [1]. Starting from the initial state $S_0 = \{(x, \emptyset)\}$, the algorithm iteratively expands candidate states and retains the top-b states at each step based on their scores. Here, b denotes the beam width, which controls the trade-off between exploration and computational cost.

Algorithm 1 Tree-of-Thought (beam search)

Require: Input x, beam width b, number of proposals k, depth T

$S_0 \leftarrow \{(x, \emptyset)\}$
for $t = 1$ to T **do**
    $C \leftarrow \emptyset$
    for all $s \in S_{t-1}$ **do**
        Sample k thoughts: $\{z^{(1)}, \ldots, z^{(k)}\} \sim p_\theta(\cdot \mid s)$
        for $i = 1$ to k **do**
            $C \leftarrow C \cup \{(x, z_{1:t-1} \oplus z^{(i)})\}$
        end for
    end for
    Score each $u \in C$ using $H(u)$
    $S_t \leftarrow$ Top-b states in C ranked by H
end for
return $\hat{y} = \text{Decode}(S_T)$

In this formulation, k controls the branching factor (number of candidate thoughts per state), b controls the number of states retained at each level, and T denotes the maximum depth of the reasoning tree. The final prediction $\hat{y}$ is obtained by decoding the best state in the final set S_T.

Conceptually, ToT introduces two key capabilities absent in standard CoT. First, it enables *local exploration* by generating multiple candidate thoughts at each step instead of committing to a single trajectory [1]. Second, it enables *global planning* through the use of search algorithms, which allow the model to evaluate intermediate states, perform lookahead, and backtrack when necessary. This aligns with classical views of problem solving as search over a combinatorial space of partial solutions.

The effectiveness of ToT stems from its ability to combine generation and evaluation within a unified framework. The language model serves both as a generator of candidate thoughts and as a heuristic evaluator of intermediate states, enabling flexible and task-adaptive reasoning strategies [1]. As demonstrated in [1], this approach significantly improves performance on tasks that require structured reasoning and planning, such as mathematical problem solving and combinatorial search.

However, these benefits come at the cost of increased computational complexity. Compared to CoT, ToT requires multiple forward passes at each reasoning step and depends on the quality of the scoring function H. Poor evaluation heuristics may lead to suboptimal pruning decisions, limiting the effectiveness of the search. Despite these challenges, ToT

provides a principled framework for extending LLM reasoning from sequential generation to structured decision-making processes [1].

2.5 Program-of-Thought Reasoning

PoT reasoning introduces a fundamentally different paradigm for improving reasoning in LLMs by explicitly separating reasoning from computation [2]. While CoT relies on natural language to perform both reasoning and intermediate calculations, PoT delegates computational steps to an external execution environment, allowing the language model to focus on structuring the solution process.

Figure 2.2 illustrates this paradigm. In contrast to CoT, where the model generates textual reasoning and performs arithmetic implicitly, PoT generates a structured program that encodes the reasoning process [2]. This program is then executed by an external interpreter to produce the final result [2].

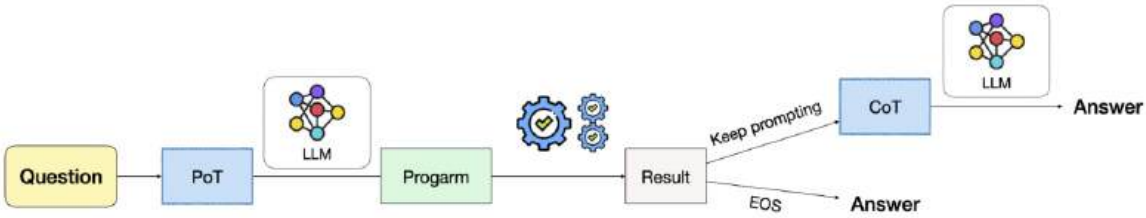

Figure 2.2: PoT combined with CoT for multi-stage reasoning (figure reproduced from [2]).

Formally, let $x \in \mathcal{X}$ denote the input problem and let $y \in \mathcal{Y}$ denote the final output. Instead of generating a reasoning sequence z purely in natural language, PoT produces a program π that represents the reasoning process [2]:

$$\pi \sim p_\theta(\pi \mid x), \tag{2.8}$$

where π consists of structured instructions written in a programming language, typically Python. The final answer is obtained by executing this program using an external interpreter R [2]:

$$y = R(\pi), \tag{2.9}$$

where R denotes a deterministic execution function that evaluates the program and returns the computed result.

The key mechanism of PoT lies in its ability to represent reasoning steps as executable operations. Instead of expressing intermediate computations in natural language, the model generates semantically meaningful variables and structured operations that reflect the underlying problem. For example, variables such as `interest_rate` or `total_cost` are used to encode domain-specific concepts [2], while symbolic libraries (e.g., for solving equations) enable precise computation. This structured representation reduces ambiguity and improves the correctness of multi-step calculations.

As discussed in [35], one of the primary limitations of CoT is that LLMs are required to perform both reasoning and computation within the same generation process, which often leads to arithmetic errors, especially for long or complex calculations. PoT addresses this limitation by offloading computation to a reliable external system, thereby improving numerical accuracy and enabling the handling of more complex mathematical expressions, including iterative processes and symbolic equations [2].

In practice, PoT can be implemented using few-shot prompting, where the model is provided with examples of input problems paired with corresponding programs. During inference, the model generates a program for a new input, which is then executed to obtain the answer [2]. In some cases, PoT can also be combined with CoT, where the execution result serves as an intermediate input for further reasoning, as illustrated in Figure 2.2. This hybrid approach is particularly useful for tasks that require both precise computation and high-level reasoning [2].

The effectiveness of PoT arises from its ability to disentangle two fundamentally different aspects of reasoning. The language model is responsible for understanding the problem and generating a structured solution, while the execution engine ensures correctness in computation. This division of responsibility leads to significant improvements in performance on numerical reasoning tasks, with empirical studies reporting substantial gains over CoT across multiple benchmarks [2].

Despite these advantages, PoT introduces additional practical considerations. The approach requires access to a secure and reliable execution environment, as executing generated code may pose safety risks if not properly controlled. Furthermore, PoT is primarily designed for tasks involving symbolic or numerical reasoning, and its benefits may be limited in domains where reasoning cannot be easily expressed as executable programs [2]. Nevertheless, PoT represents an important step toward integrating LLMs with external tools, enabling more robust and accurate reasoning systems.

2.6 Verifier-Based Reasoning Frameworks

While structured reasoning methods improve the coherence of model outputs, they do not inherently guarantee correctness. Errors introduced at intermediate steps may still propagate through the reasoning process, leading to incorrect final conclusions. This limitation has motivated the development of verifier-based reasoning frameworks, which explicitly evaluate reasoning steps and provide corrective feedback to improve reliability [38].

A general step-level verification framework is introduced in [38], where each reasoning step is assessed according to three core criteria: relevance, mathematical accuracy, and logical consistency. These criteria are applied independently to each step, enabling early detection of errors within a reasoning chain. By assigning scores to intermediate steps and discouraging low-quality reasoning, the framework guides the generation process toward more coherent and accurate solutions.

Beyond heuristic verification, recent work has explored the integration of symbolic reasoning into the verification process. The VERICOT framework [35] provides a representative example of this approach. It combines neural generation with symbolic validation by translating natural language reasoning steps into formal logical representations and checking their consistency using constraint solvers.

Algorithm 2 outlines the core procedure of the VERICOT framework. The method maintains a growing set of logical facts and incrementally verifies each reasoning step. For a given step, the framework first converts the natural language statement into a formal logical expression. This expression is then evaluated against previously established knowledge. If the new step contradicts existing facts, it is flagged as inconsistent. If it is not logically entailed, the system attempts to generate additional supporting premises. This iterative verification process ensures that each step is both locally valid and globally consistent with the reasoning chain.

This class of methods improves trustworthiness by enforcing logical consistency at each step of the reasoning process. However, it introduces additional computational overhead and depends on accurate translation from natural language into formal logic. In practice, errors in this translation stage can limit the effectiveness of symbolic verification, particularly for complex or ambiguous reasoning tasks [35].

Algorithm 2 VERICOT Reasoning Verification Framework [35]

```
Initialize knowledge set F_0 = ∅
for each reasoning step C_i do
    Convert C_i to logical form F_i
    if F_i contradicts previous knowledge then
        Mark as contradiction
    else if F_i is not entailed then
        Generate supporting premises
    end if
end for
```

2.7 Execution-Based and Trace-Grounded Reasoning

An alternative direction focuses on grounding reasoning in executable processes [39]. Instead of relying solely on generated explanations, these approaches derive reasoning steps directly from execution traces, ensuring correctness by construction. In this paradigm, reasoning is tied to the actual behavior of a system, such as program execution. Execution traces capture dynamic information including variable states, control flow decisions, and state transitions. By translating these traces into natural language explanations, each reasoning step becomes verifiable against the underlying execution process.

This approach addresses a critical limitation of standard reasoning methods, namely hallucinated intermediate steps. Since explanations are derived from concrete execution data, they reflect true system behavior rather than plausible but incorrect reasoning. As a result, the generated reasoning chains are both interpretable and reliable [39]. Despite these advantages, execution-based reasoning is inherently domain-dependent. It requires access to executable environments and is most effective in settings such as code reasoning or structured problem solving. Consequently, its applicability to general reasoning tasks remains limited.

2.8 Discussion and Comparison

The reasoning frameworks reviewed in this chapter can be grouped into three main categories: linear reasoning methods such as CoT [34], search-based approaches such as ToT [1], and execution-based methods including PoT [2] and trace-grounded reasoning [39]. Verifier-based approaches, such as step-level verification [38] and neuro-symbolic frame-

works like VERICOT [35], operate across these categories by introducing validation mechanisms at different stages of the reasoning process. This categorization highlights that different frameworks address complementary aspects of reasoning, including structure, exploration, computation, and correctness.

Each framework reflects a different design trade-off between interpretability, robustness, and reliability. CoT improves reasoning performance by decomposing complex problems into intermediate natural language steps, providing interpretability and improved task performance [34]. However, its sequential and greedy nature makes it vulnerable to error propagation, as early mistakes can influence all subsequent steps [36]. Self-consistency partially addresses this limitation by sampling multiple reasoning trajectories and aggregating their outputs, thereby improving robustness through redundancy [37]. Nevertheless, this approach operates at the level of final answers and does not enforce correctness within individual reasoning paths.

Search-based reasoning methods extend this paradigm by explicitly exploring multiple candidate solutions. ToT formulates reasoning as a structured search process over intermediate states, enabling lookahead, branching, and backtracking [1]. This allows the model to recover from intermediate errors and consider alternative reasoning paths, leading to improved performance on complex tasks. However, these benefits come at the cost of increased computational complexity and a strong dependence on the quality of the evaluation function used to guide the search.

Execution-based methods take a different approach by separating reasoning from computation. PoT improves numerical accuracy by generating executable programs and delegating computation to external interpreters [2]. This reduces arithmetic errors and enables precise handling of symbolic and numerical operations. Similarly, trace-grounded reasoning strengthens reliability by grounding intermediate steps in observable execution traces [39]. By linking reasoning to actual system behavior, these approaches reduce hallucinated steps and provide stronger correctness guarantees. Their applicability, however, is constrained to domains where computation can be explicitly represented and executed.

Verifier-based frameworks complement these methods by introducing explicit validation mechanisms. Step-level verification evaluates intermediate reasoning steps based on criteria such as logical consistency and mathematical correctness [38], while neuro-symbolic approaches such as VERICOT enforce global consistency through formal logical representations and constraint checking [35]. These methods improve reliability across different reasoning paradigms, but introduce additional computational overhead and depend on accurate translation between natural language and formal representations.

Overall, these approaches demonstrate that improvements in LLM reasoning arise from

multiple complementary mechanisms, including structured decomposition, search-based exploration, external computation, and explicit verification. While each framework addresses specific limitations, none fully resolves the challenges of reliable reasoning on its own, highlighting the need for integrated approaches that combine these capabilities.

2.9 Summary

This chapter reviewed key reasoning frameworks for LLMs, including CoT, self-consistency, ToT, PoT, and verifier-based methods. Each approach addresses specific limitations of reasoning, such as error propagation, lack of exploration, or computational inaccuracies. The analysis demonstrates that improving reasoning requires combining multiple components. Structured reasoning improves interpretability, execution-based methods enhance correctness, and verification mechanisms ensure logical consistency. However, none of these approaches alone fully resolves the challenges of reliability and constraint satisfaction. These insights motivate the development of a more comprehensive framework that integrates reasoning, verification, and representation-level control, which will be introduced in the next chapter.

Chapter 3

Proposed Methods and Architecture

3.1 Motivation and Chapter Overview

This chapter establishes the methodological framework of the thesis by defining a set of reasoning architectures that transform natural language problems into structured, verifiable, and computationally grounded solutions. The objective is not to propose a single method, but to construct a design space of reasoning strategies that can be systematically analyzed and compared in the subsequent empirical evaluation.

The need for this framework arises from the observation that reasoning in LLMs is inherently multi-faceted. Language-based methods such as CoT and CoT+SC provide interpretable reasoning traces, but lack mechanisms to enforce feasibility or correctness. Execution-based approaches, including PoT, improve numerical reliability by delegating computation to external programs, yet remain sensitive to errors in code generation. Optimization-based methods offer exact solutions when a correct mathematical formulation is obtained, but depend critically on accurate extraction of variables, constraints, and objectives from natural language. These limitations motivate a structured perspective in which reasoning is decomposed into complementary components rather than treated as a single process.

To make this structure explicit, the chapter is organized around the main components of the proposed architecture. Section 3.2 introduces the overall design and explains how different reasoning pathways interact within a unified framework. Sections on language-based, execution-based, and optimization-based reasoning then examine each component in detail, focusing on how problems are represented, processed, and solved within each

paradigm. The verification layer is integrated across these sections, reflecting its role in validating intermediate outputs and ensuring consistency with problem constraints. This organization allows each section to contribute a specific aspect of the reasoning pipeline, while maintaining a coherent view of the overall system.

In addition to these output-level methods, the chapter includes a representation-level analysis based on reasoning vector engineering. This component, developed using LLaMA-3B, investigates whether internal hidden-state directions encode signals associated with correct reasoning and whether these signals can be manipulated during inference. By combining architectural design with representation-level analysis, the chapter provides two complementary perspectives: one that structures reasoning externally through prompts, execution, and verification, and another that probes how reasoning is internally represented within the model.

3.2 Overview of the Proposed Architecture

The proposed framework is illustrated in Figure 3.1. The architecture begins with a natural language problem x and processes it through multiple reasoning pathways. These pathways are not designed as a single fixed pipeline; instead, they define a structured design space from which different reasoning architectures can be instantiated and evaluated.

The architecture is organized into three primary reasoning pathways, followed by a verification and integration layer. All method names introduced in this section (e.g., strong PoT, LP-Semantic) correspond to architectures proposed and implemented in this thesis, rather than existing methods from prior work.

The first pathway is language-based reasoning, in which the model generates intermediate reasoning steps and candidate answers directly in natural language. This includes CoT and CoT with SC (CoT+SC), both of which produce interpretable reasoning traces but do not enforce formal correctness.

The second pathway is execution-based reasoning. In this setting, the model generates executable Python code, and the final answer is obtained by running the generated program. This pathway includes the standard PoT approach [2], as well as a stronger variant, referred to in this thesis as strong PoT. strong PoT extends the baseline PoT framework by incorporating structured prompting, execution feedback, and iterative repair of generated programs. These enhancements are designed to improve robustness when the initial code generation is incomplete or incorrect.

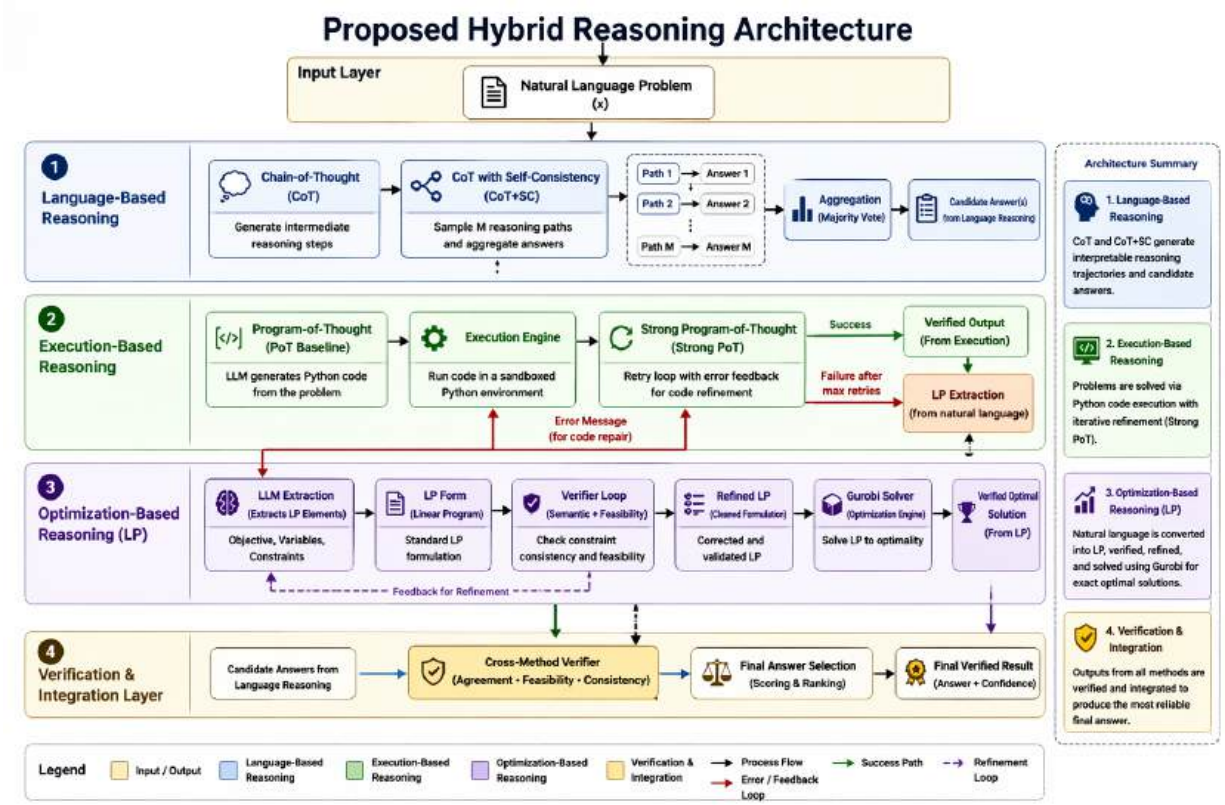

Figure 3.1: Proposed hybrid reasoning architecture (designed in this thesis).

The third pathway is optimization-based reasoning. Here, the model transforms the natural language problem into a formal optimization problem, specifically a linear program (LP), where LP stands for Linear Programming. In this formulation, decision variables, constraints, and an objective function are extracted from the input and passed to a solver. The resulting optimization problem is solved using Gurobi, a state-of-the-art commercial optimization solver widely used for linear and mixed-integer programming.

Several variants of optimization-based reasoning are proposed in this work, including LP-Gurobi, LP-Semantic, LP-VerifierLoop, LP-Semantic+Verifier, LP-SelfCheck, and LP-SelfCheck+Patch. These methods differ in how the LP formulation is extracted, validated, and refined prior to solving. Unlike prior approaches, these variants are designed to systematically study the impact of semantic structure, verification, and iterative correction on optimization reliability.

The final component of the architecture is the verification and integration layer. This layer evaluates candidate outputs produced by different reasoning pathways and checks whether they are feasible, consistent, and aligned with the problem requirements. In the optimization setting, this step is particularly important because small errors in extracted variables or constraints can lead to incorrect solutions, even when the solver itself produces an exact optimum. By incorporating verification, the architecture ensures that reasoning outputs are not only plausible but also formally valid.

3.3 Language-Based Reasoning Architectures

Language-based reasoning methods operate directly in the output token space and do not rely on external execution engines or optimization solvers. Their primary advantage is interpretability, as the generated reasoning trace can be inspected and analyzed by a human reader.

The basic language-based architecture is CoT. As previously discussed, given an input problem x, the model generates a sequence of intermediate reasoning steps

$$z = (z_1, z_2, \ldots, z_T), \tag{3.1}$$

followed by a final answer y. The process can be written as

$$z \sim p_\theta(z \mid x), \qquad y \sim p_\theta(y \mid x, z), \tag{3.2}$$

where p_θ denotes the conditional distribution of the model. CoT is particularly useful for problems that require stepwise symbolic or mathematical reasoning. However, since the reasoning chain is generated sequentially, errors introduced in early steps may propagate and affect the final prediction.

CoT with SC (CoT+SC) extends this approach by generating multiple reasoning paths instead of a single trajectory. If M reasoning paths are sampled, each produces a candidate answer $y^{(i)}$. The final prediction is obtained through aggregation [37]:

$$\hat{y} = \arg\max_{y} \sum_{i=1}^{M} \mathbf{1}\{y^{(i)} = y\}. \tag{3.3}$$

This aggregation improves robustness by reducing dependence on any single reasoning trajectory and favoring answers that consistently appear across independent samples. The method is particularly effective when multiple plausible reasoning paths exist. Nevertheless, CoT+SC does not explicitly verify whether the generated answers satisfy underlying constraints [37].

3.4 Execution-Based Reasoning Architectures

Execution-based reasoning converts the problem into executable code. This approach separates reasoning from computation: the model is responsible for generating the program, while the execution engine performs the numerical computation.

The baseline execution architecture is PoT. As shown in Figure 3.2, the input problem is passed to the model, the model generates Python code, and the code is executed to obtain a numeric answer.

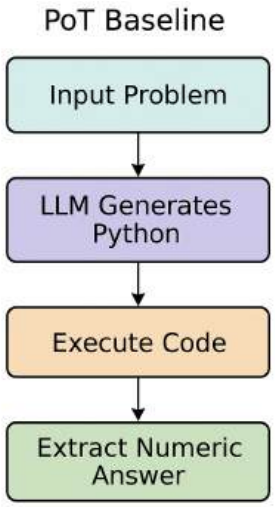

Figure 3.2: PoT baseline architecture.

As previously discussed, the model generates a program π conditioned on the input x [40]:

$$\pi \sim p_\theta(\pi \mid x). \tag{3.4}$$

The answer is then produced by an execution engine $\mathcal{E}$ [40]:

$$y = \mathcal{E}(\pi). \tag{3.5}$$

PoT is effective for numerical and algebraic problems because arithmetic is delegated to Python rather than performed implicitly by the model. Its main limitation is that the generated code may be syntactically invalid, incomplete, or logically inconsistent with the problem [40].

To address this limitation, strong PoT adds a structured code-generation prompt, an execution retry loop, and error-based repair. The architecture is shown in Figure 3.3. At iteration k, the model generates a program π_k. If execution fails, the error message e_k is appended to the prompt and used to generate a revised program:

$$\pi_{k+1} = \mathcal{M}(x, e_k), \tag{3.6}$$

where $\mathcal{M}$ denotes the language model used for code generation. If the program executes successfully, the output is checked by a feasibility verifier. If repeated code repair fails, the method can fall back to LP extraction and solver-based optimization. This makes strong PoT a hybrid method: it is primarily execution-based, but it can use optimization-based recovery when program execution is unsuccessful.

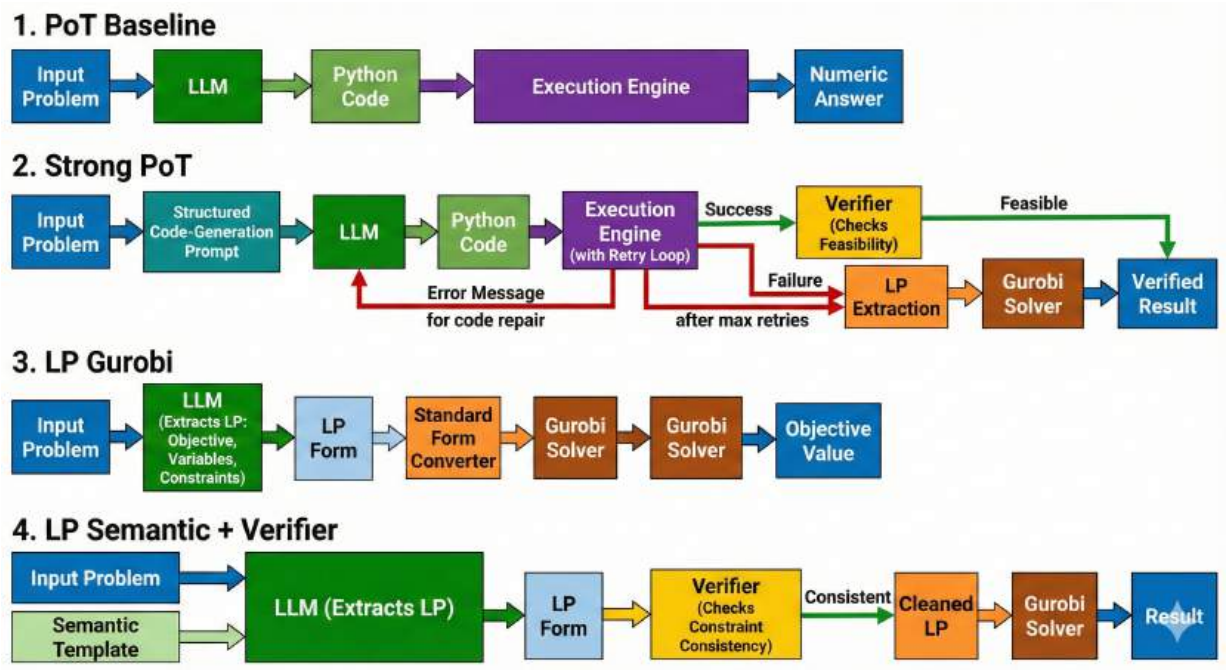

Figure 3.3: Execution and LP-based architectures.

3.5 Optimization-Based Reasoning Architectures

Optimization-based reasoning aims to convert a natural language problem into a formal linear programming formulation. The model extracts decision variables, constraints, and

the objective function. The extracted formulation is denoted by

$$F = (\mathcal{V}, \mathcal{C}, \mathcal{O}), \tag{3.7}$$

where $\mathcal{V}$ is the set of decision variables, $\mathcal{C}$ is the set of constraints, and $\mathcal{O}$ is the objective function.

Once the formulation is obtained, it is converted into a standard linear programming (LP) form and solved using Gurobi. A general LP can be written as

$$\begin{aligned} \min_{x} \quad & \mathcal{O}(x) \\ \text{s.t.} \quad & Ax \leq b, \\ & A_{\text{eq}} x = b_{\text{eq}}, \\ & x \in \mathbb{R}^n, \end{aligned} \tag{3.8}$$

where $\mathcal{O}(x)$ is the objective function, and the constraints define the feasible set. The solution is then given by

$$y^\star = \arg\min_{x \in \mathcal{C}} \mathcal{O}(x), \tag{3.9}$$

where $\mathcal{C} = \{x \mid Ax \leq b,\ A_{\text{eq}} x = b_{\text{eq}}\}$ denotes the feasible region defined by the constraints.

The advantage of this approach is that the solver can produce exact solutions when the formulation is correct. The main difficulty is that the LLM may extract incorrect variables, omit constraints, reverse inequalities, or misinterpret the objective, which directly affects the definition of the feasible set and can lead to invalid solutions. The simplest optimization-based method is LP-Gurobi. In this architecture, the model directly extracts the LP formulation from the input problem, converts it to solver-compatible form, and passes it to Gurobi. This method is efficient and strong when the LP extraction is correct, but it has no correction mechanism.

LP-Semantic improves the extraction stage by adding a structured semantic template. The template guides the model to identify the objective, variables, constraints, bounds, and variable types more systematically. This improves formulation quality compared with direct extraction, but the method can still fail if the extracted formulation is inconsistent.

LP-VerifierLoop adds a verifier after LP extraction. The verifier checks whether the extracted formulation is consistent with the natural language problem and whether the constraints are structurally valid. When an issue is detected, the formulation is refined:

$$F_{t+1} = \mathcal{R}(F_t, \phi_t), \tag{3.10}$$

where F_t is the formulation at iteration t, ϕ_t is the verifier feedback, and $\mathcal{R}$ is the refinement operator. This method is designed to reduce formulation errors before solving.

LP-Semantic+Verifier combines semantic extraction with verifier-based correction. It uses the semantic template to improve the initial LP formulation and then applies a verifier to check constraint consistency. As shown in Figure 3.3, this method cleans the LP before passing it to Gurobi.

LP-SelfCheck is another validation-based variant. Instead of relying only on an external verifier, it asks the model to review its own extracted formulation and identify possible inconsistencies. LP-SelfCheck+Patch extends this process by applying an explicit correction step before solving. These two variants test whether model-based self-review can improve formulation reliability, although they may still be less stable than verifier-guided correction.

3.6 Representation Engineering and Reasoning Vectors

The methods described above operate at the output level: they modify prompts, generate programs, extract mathematical formulations, or verify candidate answers. In contrast, this section investigates reasoning from a representation-level perspective, where control is applied directly to the internal states of the model rather than its outputs.

This approach, referred to in this thesis as reasoning vector engineering (RVE), studies whether directions in the hidden representation space of a language model encode meaningful reasoning behavior and can be used to influence inference. Instead of guiding reasoning through explicit intermediate steps or external tools, RVE operates by identifying and manipulating latent features associated with correct and incorrect reasoning.

RVE complements the previously introduced reasoning architectures by providing an internal view of reasoning. While output-level methods structure or verify the reasoning process externally, RVE examines whether reasoning signals are already embedded within the model's hidden states and whether they can be amplified or suppressed during inference.

In this part of the study, LLaMA-3B is used as an analysis model, while the other reasoning architectures are evaluated using GPT-based APIs. The objective is not to replace the external reasoning pipeline, but to investigate whether reasoning behavior can be influenced through controlled modifications of internal representations during inference.

The LLaMA-3B architecture used in this analysis is shown in Figure 3.4. The model consists of tokenization, an embedding layer, a stack of transformer layers, and a final output layer. In this architecture, hidden states are produced at each transformer layer and can be inspected during inference. Hidden-state capture is performed using forward hooks, as shown in Figure 3.5. Let $h_\ell(x)$ denote the hidden representation of input x at layer ℓ. During inference, hidden states are recorded for examples that produce correct and incorrect answer [12].

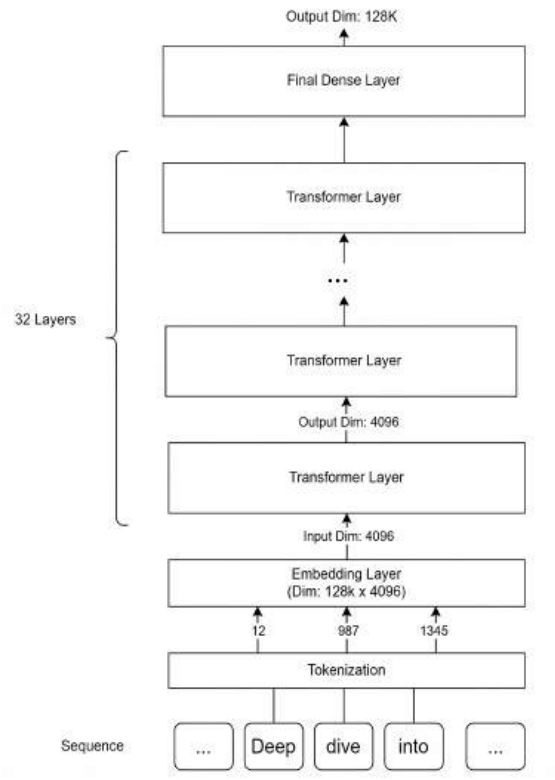

Figure 3.4: LLaMA-3B architecture.

Let $\mathcal{H}_\ell^+$ be the set of hidden states at layer ℓ associated with correct reasoning outputs, and let $\mathcal{H}_\ell^-$ be the corresponding set associated with incorrect outputs. The reasoning vector at layer ℓ is defined as

$$v_\ell = \mathbb{E}_{h \in \mathcal{H}_\ell^+}[h] - \mathbb{E}_{h \in \mathcal{H}_\ell^-}[h]. \tag{3.11}$$

where $\mathbb{E}$ denotes the empirical mean over the corresponding set of hidden represen-

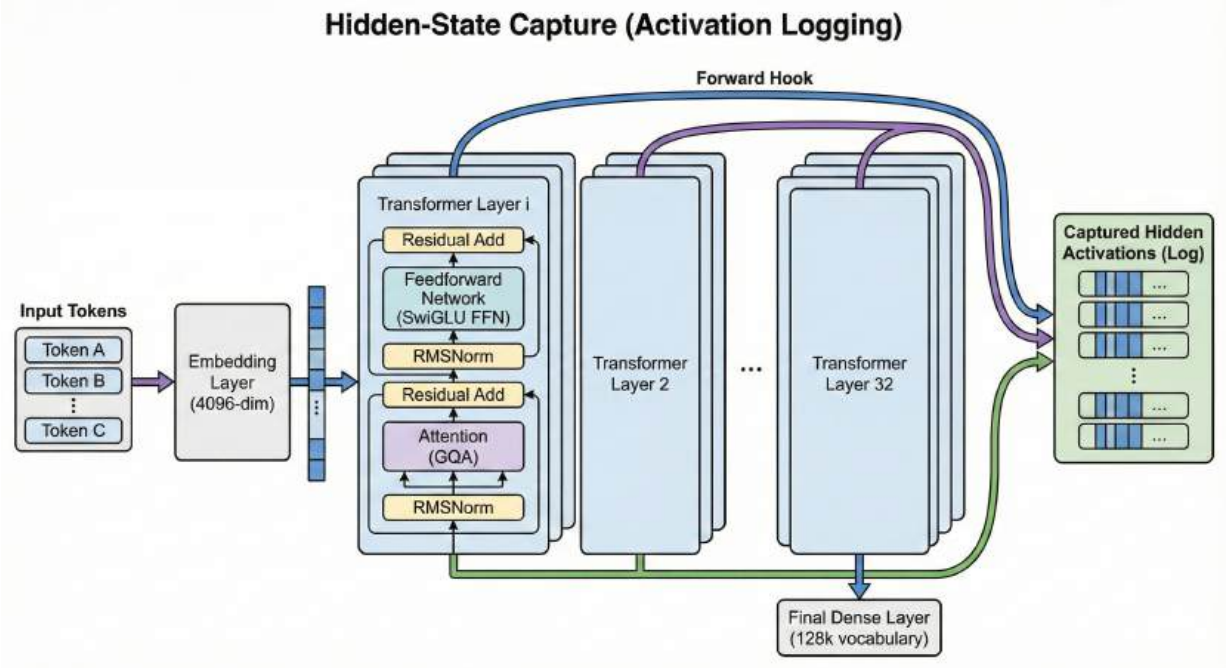

Figure 3.5: Hidden-state capture for reasoning vector analysis.

tations. This vector represents a direction in hidden-state space associated with correct reasoning behavior. During inference, the hidden state can be modified by adding a scaled version of this vecto[12]r:

$$h'_\ell = h_\ell + \alpha v_\ell, \tag{3.12}$$

where α is the steering strength. Positive or negative values of α change the hidden representation in different directions. In this thesis, this method is applied to selected mathematical reasoning problems to investigate whether hidden-state steering can influence the model's reasoning trajectory [12]. .

3.7 Summary

This chapter introduced the proposed reasoning architecture and the set of methods evaluated in this thesis. Table 3.1 summarizes the naming convention used throughout the remainder of the thesis and groups the methods according to their primary reasoning mechanism.

Table 3.1: Summary of reasoning architectures evaluated in this thesis.

Method	**Abbreviation**	**Reasoning Category**
Chain-of-Thought	CoT	Language-based reasoning
CoT with SC	CoT+SC	Language-based reasoning with answer aggregation
Tree-of-Thought	ToT	Search-based language reasoning
Program-of-Thought Baseline	PoT	Execution-based reasoning
Strong Program-of-Thought	strong PoT	Execution-based reasoning with repair and verification
Linear Programming with Gurobi	LP-Gurobi	Solver-based optimization reasoning
Semantic LP Extraction	LP-Semantic	Structured LP extraction
LP Verifier Loop	LP-VerifierLoop	LP extraction with iterative verifier feedback
Semantic LP with Verifier	LP-Semantic+Verifier	Semantic LP extraction with constraint verification
LP Self-Check	LP-SelfCheck	LP extraction with model-based validation
LP Self-Check with Patch	LP-SelfCheck+Patch	LP extraction with validation and repair
Reasoning Vector Engineering	RVE	Representation-level reasoning analysis

Language-based methods, including CoT, CoT+SC, and ToT, operate directly in natural language and provide interpretable reasoning trajectories. Execution-based methods, namely PoT and strong PoT, improve numerical reliability by generating and executing Python programs, with strong PoT further incorporating structured prompting, repair, and verification mechanisms. Optimization-based methods convert natural language problems into formal linear programs and solve them using Gurobi, with different variants introducing semantic extraction, verifier feedback, and self-correction strategies. In contrast, the representation-level approach, RVE, does not modify the external reasoning pipeline; instead, it investigates whether hidden-state directions in LLaMA-3B can influence reasoning behavior during inference.

Overall, the framework organizes reasoning into language-based, execution-based, optimization-

based, verification-based, and representation-level components. Rather than proposing a single universal method, this chapter defines a structured design space of reasoning architectures that can be systematically analyzed and compared.

These methods establish the methodological foundation for the empirical evaluation presented in the next chapters, where their performance is assessed on NL4OPT, LiveMathBench, and the UrbanMind-AI case study.

Chapter 4

Experimental Results

4.1 Motivation and Chapter Overview

This chapter presents a comprehensive empirical evaluation of the reasoning architectures introduced in Chapter 3. The objective is to analyze how different reasoning paradigms, language-based, execution-based, and optimization-based, perform across heterogeneous problem domains, and to identify the conditions under which each approach is effective.

The evaluation is conducted along three complementary dimensions. The first examines mathematical reasoning tasks to assess structured prompting and sampling strategies. The second focuses on optimization problems expressed in natural language, emphasizing program synthesis and solver integration. The third investigates system-level and representation-level mechanisms, including hybrid routing strategies and reasoning vector steering, with the aim of understanding controllability and robustness. The results are organized to reflect this progression, moving from task-level performance to architectural comparisons and finally to system-level insights. All code and implementation results are publicly available at GitHub repository [41].

4.2 Experimental Setup

4.2.1 Datasets

The empirical evaluation is conducted on two benchmark datasets that capture complementary aspects of reasoning: mathematical problem solving and natural language op-

timization. The first dataset, LiveMathBench [31], is a collection of competition-level mathematical problems designed to evaluate multi-step reasoning capabilities of LLMs. It consists of several subsets, including the American Mathematics Competitions (AMC), the Chinese College Entrance Examination (CCEE), the Chinese National Mathematical Olympiad (CNMO), the William Lowell Putnam Mathematical Competition (WLPMC), and a designated *hard* subset. Each subset reflects a different level of difficulty and reasoning style, ranging from structured algebraic manipulation to abstract problem solving.

Each instance in LiveMathBench is defined by a tuple $(q, a^\star)$, where q denotes the problem statement in natural language and $a^\star$ denotes the ground-truth answer. The evaluation requires the model to produce a final answer that matches $a^\star$, either numerically or symbolically. An example from the dataset is shown below:

> *Let an infinite geometric sequence* $\{a_n\}$ *have a common ratio* q *with* $0 < |q| < 1$. *If the sum of all terms equals the sum of the squares of all terms, determine the range of values for* a_2.

The second dataset, NL4OPT (Natural Language for Optimization) [4, 30], focuses on structured reasoning tasks where optimization problems are described in natural language. Each instance requires extracting decision variables, constraints, and objective functions, followed by solving the resulting mathematical program. Formally, each NL4OPT instance can be represented as a tuple $(q, z^\star)$, where q is a textual description of a LP problem and $z^\star$ is the optimal objective value. The dataset contains 245 problems, of which 231 have valid numeric optimal solutions after filtering infeasible or non-numeric cases, as implemented in the experimental pipeline. A representative example is given below:

> *A bakery produces bagels and croissants. Bagels require 2 hours of oven time and 0.25 hours of labor, while croissants require 1 hour of oven time and 2 hours of labor. The bakery has 70 oven hours and 32 labor hours available. If the profit per batch is \$20 for bagels and \$40 for croissants, what is the maximum achievable profit?*

This problem is translated into a linear program of the form

$$\max_{x,y} \; 20x + 40y \quad \text{subject to} \quad \begin{cases} 2x + y \leq 70, \\ 0.25x + 2y \leq 32, \\ x, y \geq 0, \end{cases} \tag{4.1}$$

and the optimal solution $z^\star = 1060$.

Together, these datasets enable a systematic comparison between unstructured reasoning (LiveMathBench) and structured optimization reasoning (NL4OPT).

4.2.2 Methods

All reasoning architectures introduced in Chapter 3 are evaluated within a unified experimental framework. These methods differ in how they generate, verify, or execute reasoning. Language-based methods, including CoT and Cot with CoT+SC, generate intermediate reasoning steps in natural language. In CoT+SC, multiple reasoning trajectories are sampled and aggregated through majority voting, improving robustness by reducing variance in individual reasoning paths.

Execution-based methods, namely PoT and strong PoT, generate executable Python programs that compute the final answer. The generated code is extracted and executed in a sandboxed environment. Strong PoT extends this approach by incorporating structured prompts, retry mechanisms, and error handling during execution, leading to improved numerical stability.

Optimization-based methods convert natural language descriptions into formal LP representations. The extracted models follow a standard formulation consisting of an objective function, a set of linear constraints, and variable bounds, as implemented in the experiment pipeline. These formulations are solved using the Gurobi optimizer. Variants differ in the use of semantic templates, verifier loops, and self-checking mechanisms that iteratively refine the extracted program.

In addition, representation-level methods based on RVE are evaluated. These methods do not modify the external reasoning process but instead manipulate hidden-state representations during inference to influence reasoning behavior.

4.2.3 Evaluation Metrics

The primary evaluation metric is accuracy, which measures the proportion of correctly solved instances. Let $\mathcal{D} = \{(x_i, y_i^\star)\}_{i=1}^N$ denote a dataset of size N, where x_i is the input problem and $y_i^\star$ is the ground-truth answer. Let $\hat{y}_i$ denote the model prediction. Accuracy is defined as

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\hat{y}_i = y_i^\star), \tag{4.2}$$

where $\mathbf{1}(\cdot)$ is the indicator function.

For numerical tasks, exact equality is replaced by a tolerance-based criterion. A prediction is considered correct if the absolute error satisfies

$$|\hat{y}_i - y_i^\star| \leq \epsilon, \tag{4.3}$$

where $\epsilon = 10^{-3}$ (This value was determined empirically.) is the predefined tolerance threshold used in the experiments.

In addition to accuracy, two error-based metrics are reported. The absolute error is defined as

$$\text{AbsError}_i = |\hat{y}_i - y_i^\star|, \tag{4.4}$$

and the relative error is defined as

$$\text{RelError}_i = \frac{|\hat{y}_i - y_i^\star|}{|y_i^\star| + \delta}, \tag{4.5}$$

where $\delta = 10^{-9}$ is a small constant introduced to avoid division by zero.

Computational efficiency is measured using latency. For each instance i, latency is defined as

$$\text{Latency}_i = t_i^{\text{end}} - t_i^{\text{start}}, \tag{4.6}$$

where t_i^{start} and t_i^{end} denote the timestamps before and after inference, respectively. Aggregate statistics such as mean and median latency are reported across the dataset.

4.2.4 Implementation Details

All experiments are implemented in a unified Python-based framework that integrates data loading, model inference, execution, and evaluation. Language-based and execution-based methods are implemented using the OpenAI API through a custom wrapper class, which interfaces with the `gpt-5.1` model. The API is accessed via the `OpenAI` client, with deterministic decoding (`temperature` $= 0.0$) to ensure reproducibility across runs.

For execution-based methods, generated Python code is extracted from model outputs and executed in a controlled environment. The execution pipeline includes parsing, runtime evaluation, and exception handling. If execution fails or produces invalid output, fallback mechanisms are applied, particularly in the Strong PoT variant.

Optimization-based methods rely on converting natural language descriptions into structured linear programs. The internal representation includes decision variables, linear constraints, and objective coefficients, as defined in the experimental schema. These programs

are solved using the Gurobi optimizer, which guarantees optimal solutions for linear programs under standard assumptions.

Experiments are conducted on university computing servers equipped with GPU and CPU resources. Local inference for open-source models, including LLaMA-3.2-3B and Mistral-7B, is performed using standard deep learning frameworks. The combination of API-based and local inference enables a comprehensive evaluation across different model families. All experimental results, including intermediate outputs, latency measurements, and evaluation metrics, are stored in structured CSV files and processed using dedicated analysis scripts. The complete codebase, including experiment configurations and data processing pipelines, is publicly available in the accompanying repository [41].

4.3 Results on NL4OPT

This section presents the evaluation of the proposed reasoning architectures on the NL4OPT benchmark. Each NL4OPT instance describes a linear optimization problem in natural language, paired with its ground-truth optimal objective value z^*. For a given method, the predicted objective value is denoted by $\hat{z}$, and a prediction is considered correct when $\hat{z}$ matches z^* within a predefined tolerance. The goal of this evaluation is to assess not only the accuracy of different reasoning strategies, but also their computational efficiency and robustness under varying problem characteristics. All figures in this section are generated from our implementation results.

4.3.1 Overall Performance

A central question in optimization-oriented reasoning is whether structured reasoning mechanisms can outperform both direct code generation and classical solver-based pipelines. Figure 4.1 provides a comparative view of all evaluated methods in terms of objective accuracy.

The results indicate that strong PoT achieves the highest accuracy among all evaluated approaches, reaching approximately 80%. This improvement over the PoT baseline (approximately 64%) highlights the importance of incorporating structured reasoning, execution feedback, and repair mechanisms. While the PoT paradigm enables executable reasoning, it remains sensitive to errors in code generation. Strong PoT mitigates this limitation by iteratively refining the generated program, thereby increasing the likelihood of producing a semantically correct formulation.

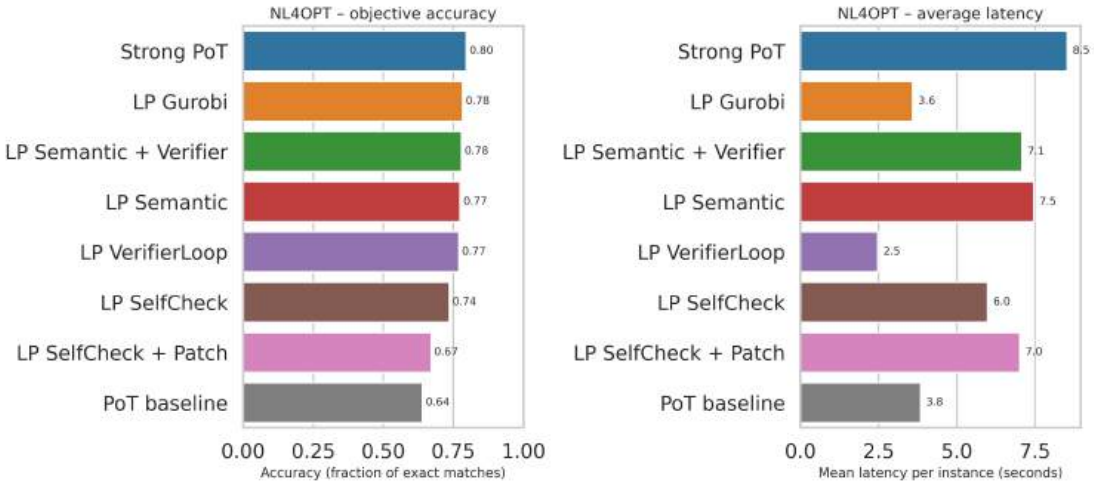

Figure 4.1: NL4OPT accuracy across methods.

Interestingly, LP-based methods such as LP-Gurobi and LP-Semantic variants achieve comparable performance, with accuracies clustered around 77%–78%. This suggests that once a correct mathematical formulation is extracted, deterministic solvers provide reliable optimization performance. However, the limited variation across LP-based methods indicates that improvements in solver design alone are insufficient; the primary challenge lies in accurately translating natural language into formal optimization constraints. This observation reinforces the role of reasoning-based approaches, which explicitly address this translation step.

4.3.2 Accuracy–Latency Trade-off

While accuracy is a primary metric, practical deployment also requires understanding the computational cost of each reasoning strategy. Figure 4.2 examines the relationship between accuracy and median latency across methods.

A clear trade-off emerges. Strong PoT achieves the highest accuracy (82.5%), but at the cost of increased latency (approximately 7.97 seconds per instance). This reflects the iterative nature of the method, which involves multiple reasoning and repair steps before arriving at a final solution. In contrast, LP-Gurobi achieves nearly comparable accuracy (81.2%) with significantly lower latency (approximately 3.18 seconds), as it relies on a direct formulation-to-solver pipeline.

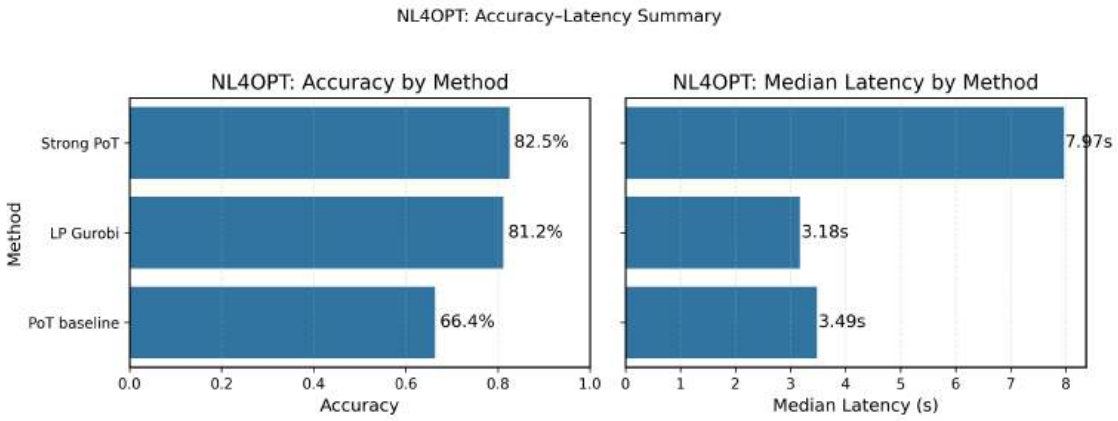

Figure 4.2: Accuracy–latency trade-off on NL4OPT.

The PoT baseline occupies an intermediate position, with moderate latency (3.49 seconds) but lower accuracy. This suggests that execution alone is not sufficient to guarantee correctness; rather, the effectiveness of execution depends on the quality of the generated program. Overall, these results highlight that accuracy gains in reasoning-based methods often come at the expense of computational efficiency, and selecting a method requires balancing these competing objectives.

4.3.3 Scaling Behavior

Beyond aggregate performance, it is important to understand how reasoning methods behave as the numerical magnitude of optimization problems increases. In this context, we do not refer to scaling in the conventional machine learning sense (e.g., increasing dataset size or model capacity), but rather to the magnitude of the objective value induced by the problem formulation.

Figure 4.3 analyzes accuracy as a function of the absolute value of the optimal objective, $|z^*|$. This quantity is used as a proxy for problem magnitude, as larger objective values typically arise from problems with larger coefficients, wider feasible regions, or greater accumulation of numerical terms.

Strong PoT consistently outperforms the PoT baseline across most magnitude ranges, with particularly large improvements observed in the 10–100 interval, where accuracy increases from 69.7% to 93.9%. Similar gains are observed in the 100–1k and 1k–10k ranges, indicating that structured reasoning and repair are especially beneficial for problems with

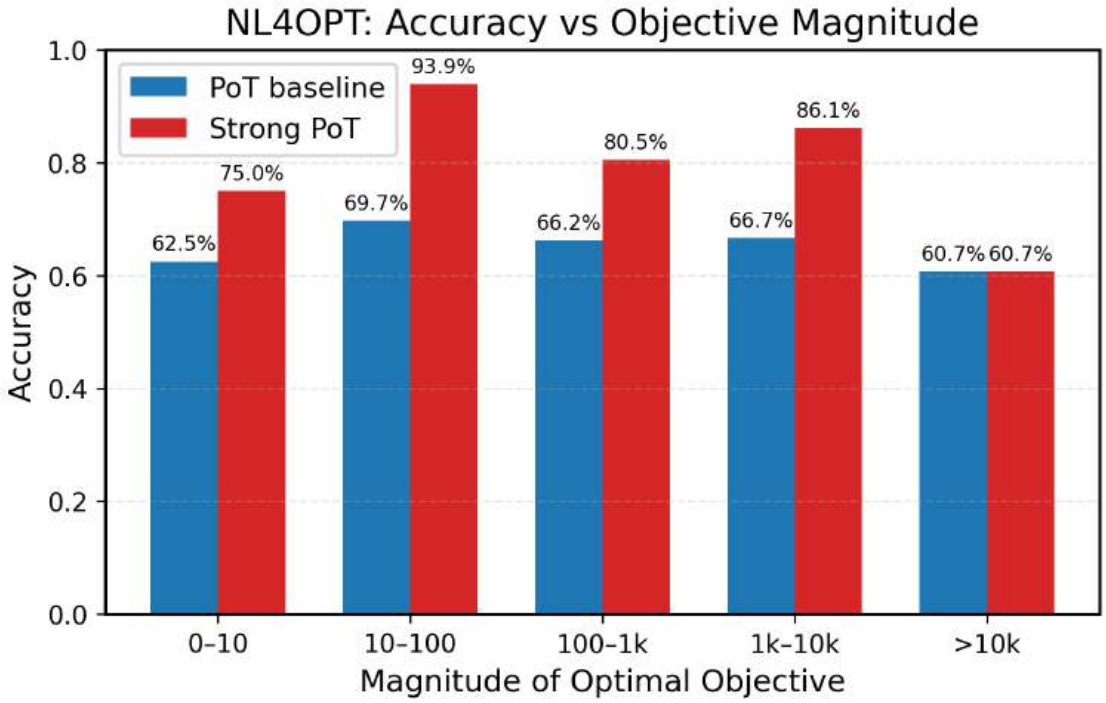

Figure 4.3: Accuracy as a function of objective magnitude $|z^*|$.

moderate numerical complexity.

However, this advantage diminishes at larger magnitudes. In the $> 10k$ range, both methods converge to the same accuracy (60.7%), suggesting that additional factors limit performance. These may include numerical instability, accumulation of modeling errors, or increased difficulty in identifying active constraints. This trend indicates that while Strong PoT improves robustness, it does not fully resolve challenges associated with problems involving large numerical scales.

4.3.4 Error Analysis

To better understand the limitations of current methods, we analyze the dominant sources of error in NL4OPT. The results reveal that failures are primarily attributable to three factors: incorrect problem formulation, numerical scaling issues, and incomplete constraint representation.

For LP-based methods, errors arise almost exclusively during the formulation stage. Since the solver itself is deterministic, any mistake in defining variables, constraints, or objective functions directly leads to incorrect solutions. This explains why improvements in solver pipelines yield diminishing returns when formulation quality remains unchanged.

For PoT-based methods, the failure mode is different. Generated programs are often syntactically valid but semantically incorrect. In such cases, the code executes successfully but solves a problem that differs from the intended one. Strong PoT reduces this issue by introducing execution-based feedback and repair; however, it cannot guarantee semantic correctness in all cases. The persistence of errors suggests that execution alone is insufficient without deeper alignment between natural language understanding and formal representation.

Overall, these findings highlight that optimization reasoning is fundamentally a translation problem. Success depends not only on solving the optimization problem, but on correctly interpreting and encoding it. While strong PoT improves this process, the remaining errors indicate that further progress requires tighter integration between language understanding, symbolic reasoning, and formal verification.

4.4 Results on LiveMathBench

This section evaluates reasoning performance on the LiveMathBench benchmark, which focuses on mathematical problem solving across multiple difficulty levels and competition-style datasets. Unlike NL4OPT, where the output is a numerical objective value derived from an optimization problem, LiveMathBench requires generating a correct final answer through multi-step mathematical reasoning.

For each instance, a prediction is considered correct if the generated answer exactly matches the ground-truth solution. Let $\hat{y}$ denote the predicted answer and y^* the ground truth. Accuracy is therefore defined as the fraction of instances for which $\hat{y} = y^*$. The evaluation focuses on comparing reasoning strategies such as CoT and CoT with SC, as well as analyzing performance across datasets of varying difficulty.

4.4.1 Overall Performance

A key question in mathematical reasoning is whether structured reasoning and sampling-based strategies improve reliability over standard CoT prompting. Figure 4.4 summarizes the overall accuracy across models and reasoning strategies.

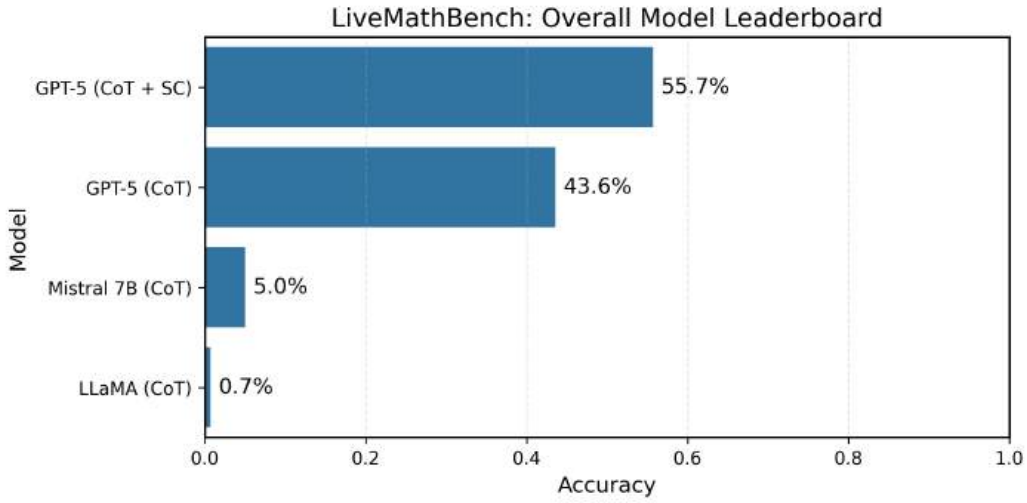

Figure 4.4: Overall accuracy on LiveMathBench across models and reasoning strategies.

The results show a substantial gap between large and smaller models. GPT-5 with CoT achieves 43.6% accuracy, while incorporating SC improves performance significantly to 55.7%. This demonstrates that sampling multiple reasoning trajectories and selecting the most consistent answer reduces variance and improves correctness.

In contrast, smaller models such as Mistral 7B and LLaMA perform poorly, achieving 5.0% and 0.7% accuracy, respectively. This indicates that mathematical reasoning on this benchmark requires not only structured prompting but also strong underlying reasoning capabilities. The large performance gap suggests that scaling model capacity plays a critical role in solving complex mathematical problems.

4.4.2 Per-Split Analysis

While overall accuracy provides a high-level comparison, it does not reveal how reasoning performance varies across problem types. LiveMathBench includes multiple subsets, each corresponding to a different mathematical domain or difficulty level. Figure 4.5 presents the per-split accuracy for GPT-5 under CoT and CoT with SC.

The results reveal that performance varies significantly across datasets. On the AMC subset, which consists of relatively structured competition-style problems, CoT already achieves a strong accuracy of approximately 0.67, which further improves to 0.74 with SC.

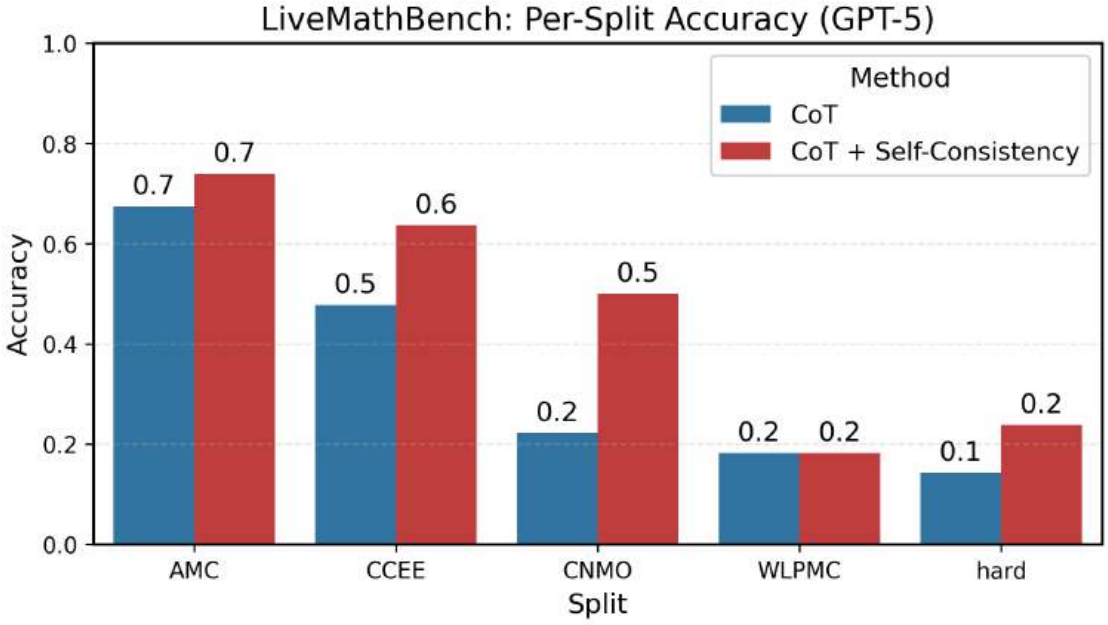

Figure 4.5: Per-split accuracy on LiveMathBench for GPT-5 under CoT and CoT + SC.

Similar improvements are observed on the CCEE subset, where accuracy increases from approximately 0.48 to 0.64.

More challenging datasets exhibit a larger gap between reasoning strategies. On the CNMO subset, accuracy improves from 0.22 to 0.50 when using SC, indicating that difficult problems benefit more from multiple reasoning attempts. However, performance remains low on the WLPMC and *hard* subsets, where both methods struggle, with accuracies less than or equal to 0.2. This suggests that these subsets require deeper reasoning or more advanced symbolic capabilities beyond current prompting strategies.

Overall, SC consistently improves performance across most subsets, but its impact is more pronounced in moderately difficult problems than in extremely challenging ones.

4.4.3 Analysis of Reasoning Behavior

To further interpret these results, we examine execution traces and prediction logs to understand how different reasoning strategies behave in practice. In the context of CoT prompting typically generates a single reasoning trajectory. While this trajectory often

captures the correct high-level structure of the solution, it remains vulnerable to small logical or arithmetic errors. Such local errors propagate through subsequent steps, ultimately leading to incorrect final answers despite otherwise coherent reasoning.

In contrast, SC improves robustness by sampling multiple independent reasoning paths and selecting the most frequent answer. This process reduces the impact of individual faulty trajectories and increases the likelihood of recovering the correct solution when at least some reasoning paths are valid. However, performance on more challenging subsets reveals a different failure mode. Errors are frequently caused by incorrect problem interpretation or incomplete reasoning rather than simple arithmetic mistakes. In these cases, multiple sampled trajectories tend to exhibit similar flawed reasoning patterns, preventing convergence to the correct answer. This behavior is reflected in the persistently low accuracy observed on the WLPMC and *hard* subsets.

Overall, SC acts primarily as a variance-reduction mechanism rather than a fundamental improvement in reasoning capability. It enhances reliability when correct reasoning paths are present, but does not address deeper limitations in problem understanding or reasoning depth.

4.4.4 Discussion

The LiveMathBench results highlight a different challenge compared to NL4OPT. In optimization tasks, errors are often due to incorrect problem formulation, whereas in mathematical reasoning tasks, errors arise from incomplete or inconsistent reasoning processes. The effectiveness of SC demonstrates that reasoning reliability can be improved through sampling and aggregation. However, the persistent gap on harder problems indicates that further improvements require stronger reasoning representations, better decomposition strategies, or integration with symbolic solvers. Overall, these results emphasize that structured reasoning alone is not sufficient; both model capacity and reasoning strategy play critical roles in achieving robust mathematical problem solving.

4.5 Results on Reasoning Vector Engineering

This section investigates whether reasoning vectors provide a meaningful and controllable representation of reasoning behavior in language models. We analyze both the structure of these representations across layers and the effect of manipulating them through steering.

4.5.1 Layer-wise Structure of Reasoning Representations

Figure 4.6 examines how reasoning-related information is distributed across transformer layers. To quantify the distinction between correct and incorrect reasoning, two metrics are used: the separation gap and the normalized separation ratio.

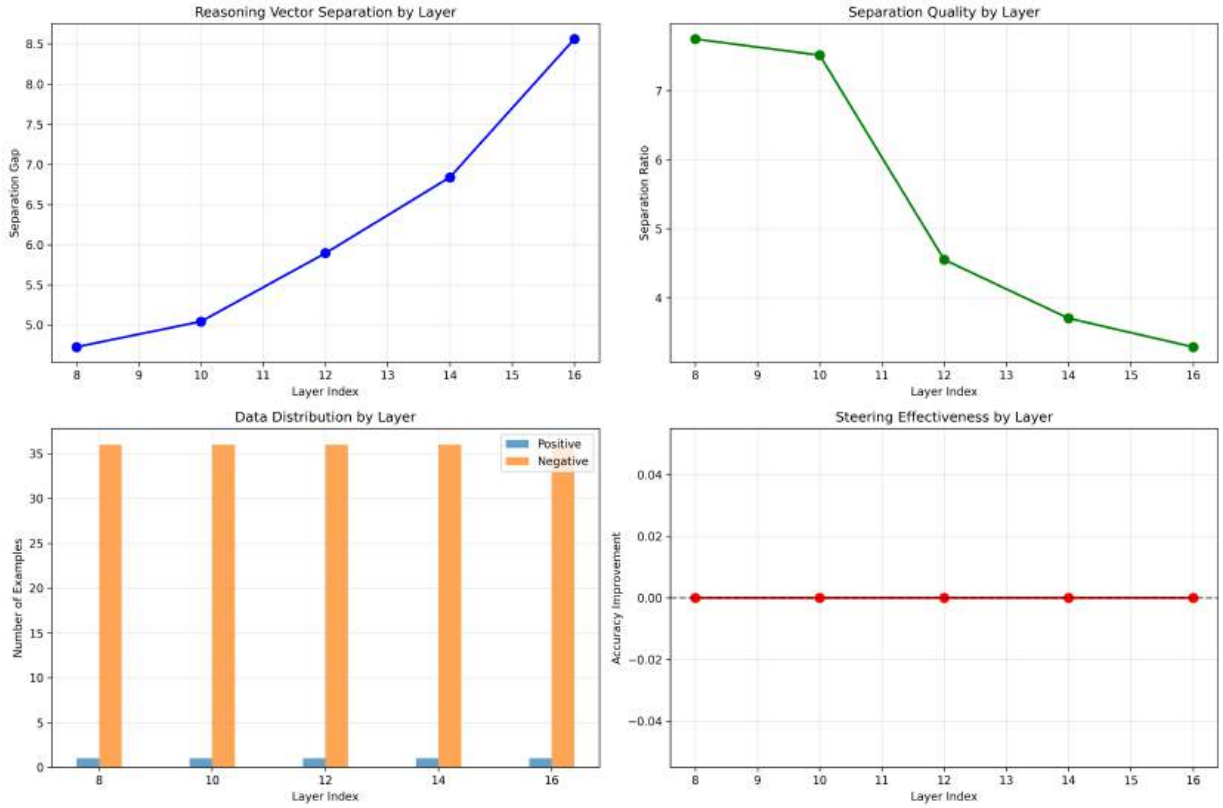

Figure 4.6: Layer-wise analysis of reasoning representations. Top-left: separation gap between correct and incorrect reasoning traces. Top-right: normalized separation ratio. Bottom-left: distribution of positive (correct) and negative (incorrect) examples. Bottom-right: steering effectiveness measured as accuracy improvement.

Let $\mathcal{H}_\ell^+$ and $\mathcal{H}_\ell^-$ denote the sets of hidden representations at layer ℓ corresponding to correct and incorrect reasoning traces. The mean representations are defined as [12]

$$\mu_\ell^+ = \mathbb{E}_{h \in \mathcal{H}_\ell^+}[h], \qquad \mu_\ell^- = \mathbb{E}_{h \in \mathcal{H}_\ell^-}[h]. \tag{4.7}$$

The separation gap is defined as [12]

$$\Delta_\ell = \|\mu_\ell^+ - \mu_\ell^-\|_2, \tag{4.8}$$

which measures the absolute distance between correct and incorrect reasoning representations. A larger value indicates stronger separability.

To account for intra-class variability, the normalized separation ratio is defined as [12]

$$R_\ell = \frac{\|\mu_\ell^+ - \mu_\ell^-\|_2}{\sigma_\ell^+ + \sigma_\ell^-}, \tag{4.9}$$

where σ_ℓ^+ and σ_ℓ^- denote the average dispersion within each group. This ratio captures separation relative to variability.

The separation gap increases consistently with depth, growing from approximately 4.7 at layer 8 to over 8.5 at layer 16. This indicates that deeper layers encode more discriminative signals related to reasoning correctness. In contrast, the normalized separation ratio decreases with depth, suggesting that deeper layers exhibit greater variability despite stronger absolute separation. Earlier layers produce more stable but less expressive representations, while deeper layers capture more specialized reasoning features.

The distribution of positive and negative examples across layers (Figure 4.6, bottom-left) confirms that these trends are not driven by data imbalance. Both classes remain well represented, indicating that the observed separation reflects genuine representational differences rather than sampling artifacts.

Despite the clear separation in representation space, the measured steering effectiveness (bottom-right) remains close to zero across all layers. This shows that the presence of a well-defined reasoning direction does not necessarily imply that it can be used to influence model outputs during inference.

These results highlight a key limitation of representation-level control: reasoning signals can be encoded internally without being directly accessible for manipulation. While deeper layers contain more discriminative information, this structure alone is insufficient for reliable steering.

4.5.2 Effect of Reasoning Vector Steering

The effect of reasoning vector steering is shown in Figure 4.7, where α denotes the scaling factor applied along the reasoning direction. The baseline accuracy at $\alpha = 0$ is approximately 4.8%, and performance remains unchanged for a broad range of moderate values

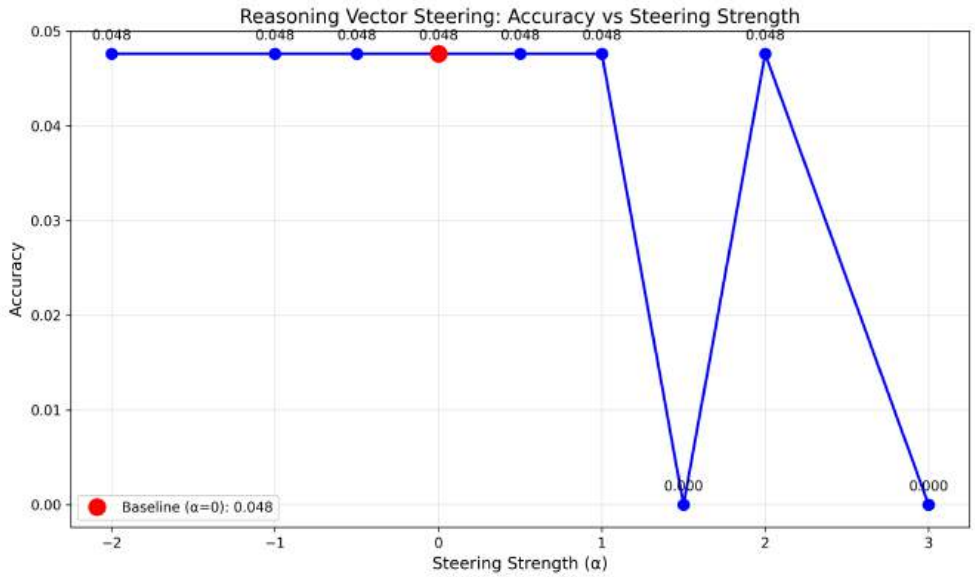

Figure 4.7: Accuracy as a function of steering strength α for reasoning vector intervention. The baseline corresponds to $\alpha = 0$. All results are generated from our implementation.

($\alpha \in [-2.0, 1.0]$). This indicates that small perturbations along the reasoning vector do not alter the model's predictions.

When α exceeds this range, the model becomes unstable. Accuracy drops sharply to 0% at $\alpha = 1.5$ and $\alpha = 3.0$, showing that strong interventions disrupt the internal computation rather than improving reasoning. Importantly, no value of α leads to an improvement over the baseline.

These results demonstrate that although a meaningful reasoning direction can be identified, linear movement along this direction is not sufficient to correct errors. Instead, model behavior is preserved within a narrow stability region and deteriorates outside it.

4.5.3 Adaptive Steering and Robustness

To mitigate the sensitivity observed with fixed steering, an adaptive strategy is applied in which the steering strength α is selected based on an estimate of problem difficulty. In this setting, difficulty is approximated using observable properties of the input, such as the number of variables, constraints, or the length and structural complexity of the problem description. These features provide a coarse proxy for how challenging the reasoning

process is expected to be.

Given a normalized difficulty score $d \in [0, 1]$, the steering parameter is chosen through a linear mapping [12]

$$\alpha(d) = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \cdot d, \tag{4.10}$$

where $\alpha_{\min} = 0.24$ and $\alpha_{\max} = 0.42$ define the empirically identified stable region of the steering curve (Figure 4.7). This ensures that steering remains within a range where model behavior is not disrupted, while still allowing moderate variation based on problem complexity.

The resulting accuracy reaches 93.8% (15 out of 16 examples), matching the baseline performance. This indicates that adaptive steering preserves model behavior but does not introduce measurable improvements. Although the dependence of α on problem difficulty captures some structural variation across instances, the results suggest that this information is insufficient to correct reasoning errors through linear intervention alone.

4.5.4 Discussion

The results highlight a clear distinction between representation and control. Reasoning vectors successfully identify a direction that separates correct and incorrect reasoning, particularly in deeper layers. However, manipulating this direction does not improve performance.

This gap suggests that reasoning errors cannot be attributed to a simple misalignment along a single linear direction. Instead, they arise from more complex, nonlinear interactions within the model. As a result, reasoning vector engineering provides valuable insight into internal representations, but it does not constitute an effective mechanism for directly improving reasoning accuracy. These findings emphasize the need for more expressive and structured approaches to reasoning control, beyond linear interventions in representation space.

4.6 Discussion and Insights

This chapter presented empirical results across multiple benchmarks, including NL4OPT, LiveMathBench, and controlled experiments on reasoning vector engineering. In this section, we synthesize these findings to identify broader patterns in reasoning behavior and to understand the strengths and limitations of different reasoning paradigms.

4.6.1 The PoT Gap Across Problem Domains

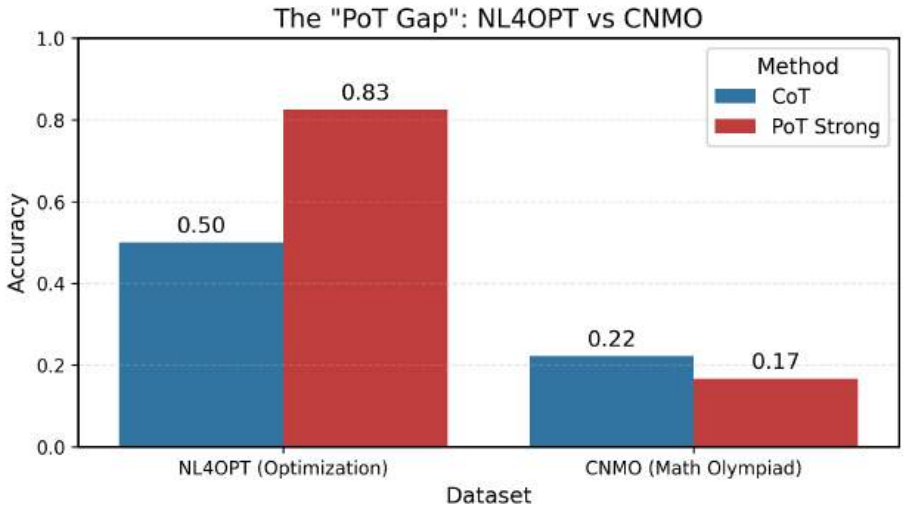

Figure 4.8: Performance comparison between CoT and strong PoT across optimization (NL4OPT) and olympiad-level mathematics (CNMO). The results illustrate a domain-dependent performance gap.

Figure 4.8 highlights a striking contrast in the behavior of reasoning methods across domains. On NL4OPT, which involves structured optimization problems, Strong PoT significantly outperforms CoT, improving accuracy from 0.50 to 0.83. This gain reflects the advantage of programmatic reasoning, where intermediate computations can be explicitly executed and verified.

In contrast, the same method underperforms on CNMO, a benchmark of olympiad-level mathematics. Here, accuracy decreases from 0.22 under CoT to 0.17 with strong PoT. This reversal indicates that the benefits of structured computation do not transfer uniformly across tasks. Optimization problems tend to follow well-defined procedural patterns, whereas olympiad problems require flexible symbolic reasoning, abstraction, and creative problem decomposition.

This phenomenon, which we refer to as the *PoT gap*, demonstrates that improvements in one domain can correspond to degradation in another. It suggests that reasoning strategies must be aligned with the underlying structure of the task rather than applied uniformly.

4.6.2 Method Suitability Across Domains

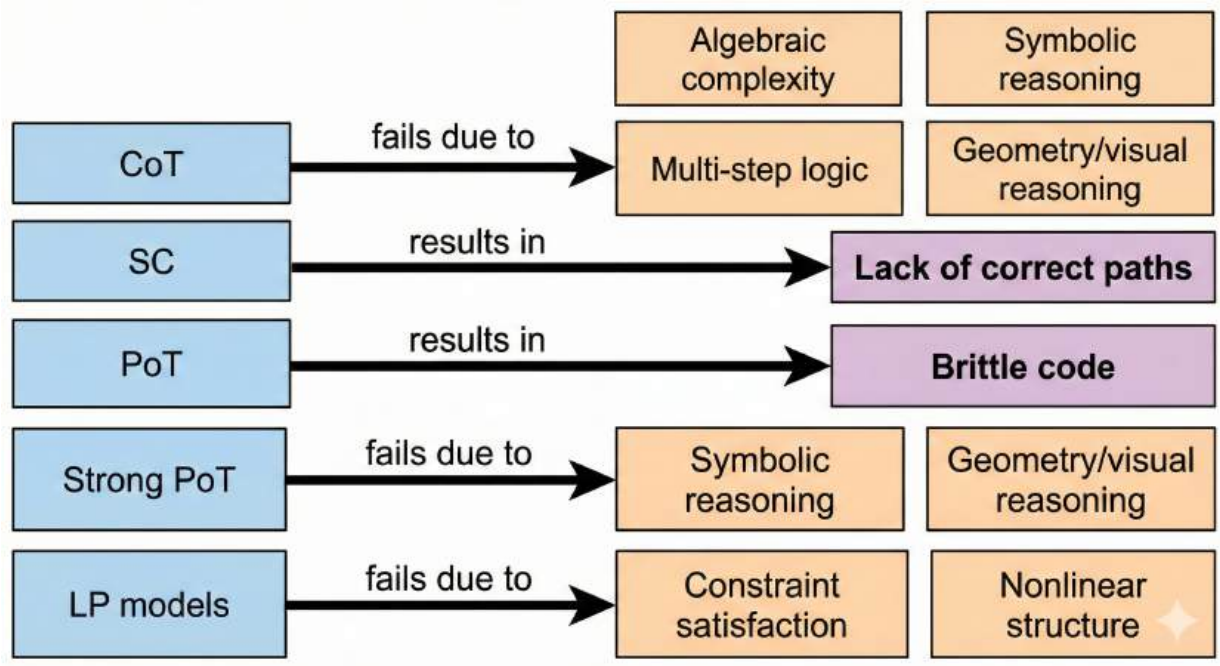

Figure 4.9: Suitability scores of reasoning methods across problem domains. Scores are normalized to a 0–100 scale to facilitate comparison across tasks.

The broader pattern of domain dependence is further illustrated in Figure 4.9, which summarizes the suitability of different reasoning methods across three categories: formal optimization, standard mathematical reasoning, and olympiad-level reasoning.

PoT methods show the highest suitability for formal optimization tasks, where explicit computation and constraint handling are essential. Their performance decreases for standard mathematical reasoning and declines further for olympiad problems, where reasoning becomes less procedural and more conceptual.

Conversely, CoT-based methods, particularly when combined with SC, demonstrate more balanced performance across domains. While they do not achieve the highest scores in optimization, they remain relatively robust in standard and olympiad-level reasoning. This suggests that natural language reasoning provides greater flexibility when problem structure is less rigid.

LP-based methods exhibit a complementary pattern. They achieve high suitability in

optimization tasks but perform poorly in mathematical reasoning settings, highlighting their reliance on structured, well-defined problem formulations.

Overall, these results indicate that no single reasoning paradigm dominates across all domains. Instead, each method occupies a specific region of the problem space, defined by the degree of structure and formality required.

4.6.3 Failure Modes of Reasoning Strategies

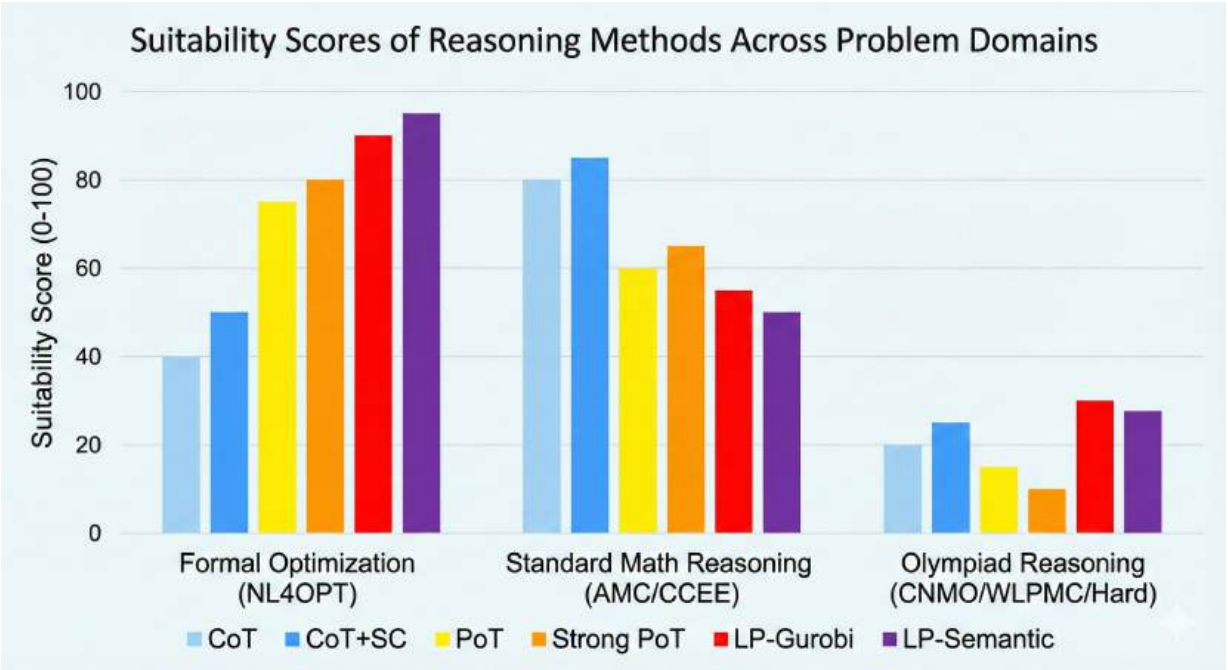

Figure 4.10: Illustration of common failure modes across reasoning strategies. Each method fails for distinct reasons depending on the complexity and structure of the problem.

To better understand the limitations of each approach, Figure 4.10 summarizes the dominant failure modes observed in our experiments. CoT methods often struggle with multi-step logical dependencies and complex algebraic manipulations, where small errors propagate through the reasoning chain. Although SC reduces variance by aggregating multiple reasoning paths, it cannot compensate when all generated paths are incorrect.

PoT methods introduce a different class of failure. While they excel in structured settings, they rely on the correctness of generated programs. Errors in code generation, such

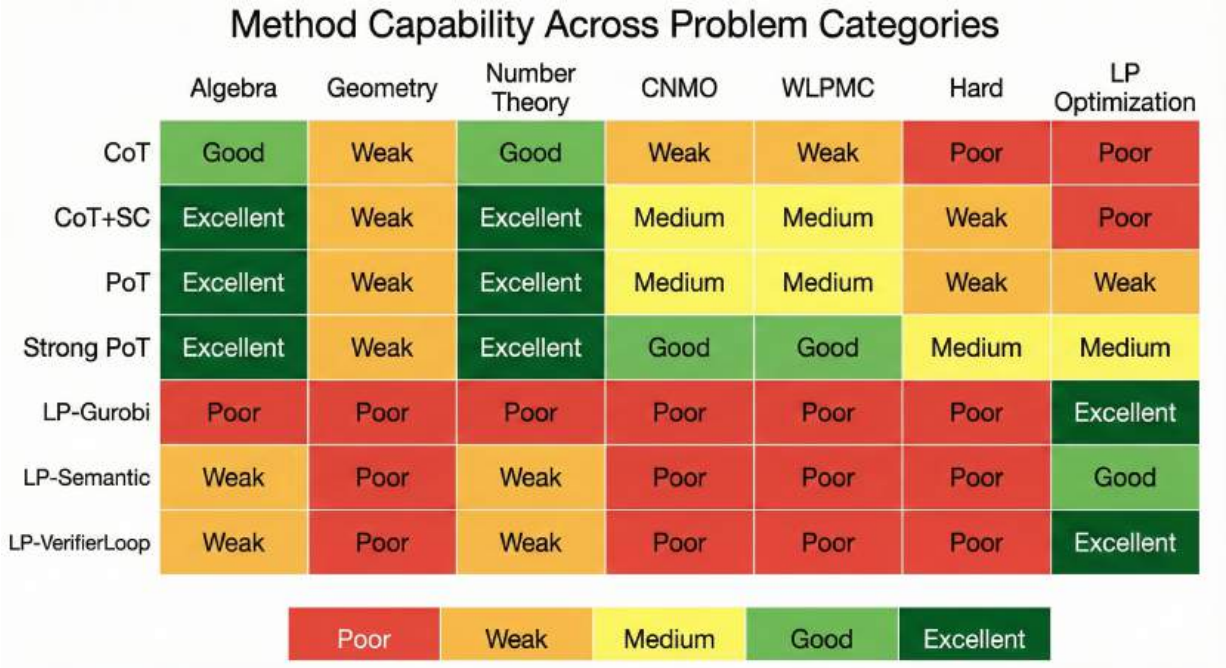

Figure 4.11: Capability matrix of reasoning methods across problem categories. Performance is categorized into five levels: Poor, Weak, Medium, Good, and Excellent.

as incorrect variable definitions or logical inconsistencies, can lead to brittle execution and incorrect results. This brittleness becomes more pronounced in tasks requiring symbolic or geometric reasoning, where formal program representations are harder to construct.

LP-based methods fail primarily due to their dependence on precise problem formulation. When constraints are incomplete, nonlinear, or implicitly defined, these methods cannot produce valid solutions. This limitation is particularly evident in olympiad problems, where constraints are often implicit and reasoning must go beyond formal optimization.

These observations suggest that the limitations of each method are closely tied to their underlying assumptions about problem structure. Methods that rely on explicit structure fail when such structure is absent, while flexible reasoning methods struggle with precision and consistency.

4.6.4 Capability Landscape Across Problem Categories

Figure 4.11 provides a consolidated view of method capabilities across different problem categories. The matrix reveals a clear specialization of reasoning methods. CoT-based approaches perform well in algebra and number theory but struggle with geometry and optimization. Program-based methods extend these capabilities to more structured domains but remain limited in highly abstract settings.

LP-based methods achieve strong performance in optimization tasks but fail to generalize beyond them. Hybrid approaches, such as strong PoT and verifier-based methods, partially bridge these gaps but do not fully resolve the trade-offs between flexibility and structure.

The overall pattern suggests that reasoning in large language models is not governed by a single unified mechanism. Instead, it emerges from the interaction between multiple reasoning modes, each suited to a different class of problems.

4.6.5 Implications for Reasoning Systems

The findings of this chapter point toward several important implications for the design of reasoning systems. First, the effectiveness of a reasoning strategy depends strongly on the structure of the problem domain. Methods that explicitly encode structure, such as PoT and LP-based approaches, perform well when such structure is available but degrade when it is not.

Second, improvements in reasoning accuracy cannot be achieved solely by scaling a single paradigm. The observed PoT gap demonstrates that gains in one domain may come at the cost of performance in another. This highlights the need for adaptive or hybrid systems that can dynamically select or combine reasoning strategies based on task characteristics.

Third, the results on reasoning vector engineering suggest that internal representations capture meaningful reasoning signals, but these signals are not easily controllable through simple interventions. This reinforces the idea that reasoning behavior is governed by complex, nonlinear dynamics rather than a single interpretable direction in representation space.

Taken together, these insights suggest that future progress in reasoning with large language models will require integrating multiple reasoning paradigms and developing mechanisms for selecting the appropriate strategy for each task. Rather than seeking a universal

reasoning method, it may be more effective to design systems that adapt their reasoning approach to the structure and demands of the problem.

4.7 Summary

This chapter presented a comprehensive empirical evaluation of reasoning architectures across diverse problem domains, including mathematical reasoning, natural language optimization, and representation-level analysis. The results demonstrate that the effectiveness of reasoning strategies is highly dependent on the structure and requirements of the task.

On NL4OPT, execution-based and optimization-aware methods, particularly strong PoT, achieve strong performance by leveraging structured computation and solver integration. In contrast, results on LiveMathBench highlight the importance of model capacity and sampling-based strategies, where SC improves robustness but does not fully address limitations in deep reasoning.

The analysis of reasoning vector engineering reveals that meaningful reasoning signals emerge within model representations, especially in deeper layers. However, these signals are not easily exploitable for improving performance through direct intervention, indicating a gap between representation and controllability.

Across all experiments, a consistent pattern emerges: no single reasoning paradigm performs optimally across all domains. Instead, reasoning behavior reflects a combination of structured computation, flexible language-based reasoning, and implicit internal representations. These findings suggest that future reasoning systems should move toward hybrid and adaptive approaches that align reasoning strategies with the underlying structure of each task.

Overall, the results of this chapter provide both empirical evidence and conceptual insight into the strengths and limitations of current LLM-based reasoning methods, establishing a foundation for the application-driven framework developed in the next chapter.

Chapter 5

Case Study: Urban School Site Selection Using LLM-Based Reasoning

5.1 Motivation and Chapter Overview

This chapter presents a real-world case study that demonstrates how LLMs reasoning can be applied to a structured urban planning problem. In particular, we consider the task of selecting an optimal location for a new high school in the Waterloo Region. This problem is formulated as a multi-criteria optimization task and is addressed using LLM-based reasoning methods, with a primary focus on CoT reasoning.

The proposed framework integrates domain-specific planning criteria derived from the Waterloo District School Board (WDSB), formalizes them into a quantitative scoring model, and combines this model with interpretable reasoning generated by LLMs. In addition, the full pipeline is implemented as a functional system, *UrbanMind-AI*, and made publicly available.[1] The repository provides the complete implementation, including data processing, reasoning modules, and visualization tools.

[1] https://github.com/zsarayloo/UrbanMind-AI

5.2 Problem Formulation

The selection of a suitable site for a new high school is inherently a multi-criteria decision problem involving regulatory, spatial, and socio-economic considerations. The objective is to identify a location that satisfies mandatory constraints while maximizing accessibility, safety, and long-term utility.

We formalize this task as a weighted optimization problem. For a candidate site s, the overall score is defined as

$$S(s) = \sum_{i=1}^{n} w_i \, S_i(s), \tag{5.1}$$

where $S_i(s)$ denotes the normalized score of site s under criterion i, and w_i represents the importance weight assigned to that criterion.

The criteria are derived from WDSB planning guidelines and Ontario Regulation 20/98, and include area compliance, zoning suitability, population accessibility, transportation access, environmental safety, infrastructure readiness, and proximity to community amenities. Each component score $S_i(s)$ is normalized to the interval $[0, 1]$, ensuring comparability across heterogeneous data sources [42].

This formulation enables a direct comparison between candidate sites and provides a quantitative baseline for evaluating LLM-based reasoning.

5.3 LLM-Based Reasoning Framework

Rather than relying solely on numerical optimization, we employ LLM-based reasoning to emulate expert decision-making. In particular, CoT reasoning is used to guide the model through a structured evaluation process.

Given a set of candidate sites, the LLM generates a step-by-step analysis that first applies hard constraints, such as zoning and environmental restrictions, and then evaluates remaining candidates based on soft criteria. The reasoning process explicitly considers each component of the scoring function, allowing the model to produce both qualitative explanations and quantitative scores.

An example of this reasoning behavior is shown in the generated output, where infeasible sites are first eliminated based on zoning and environmental constraints, and the remaining sites are compared in terms of accessibility, population coverage, and infrastructure. The final recommendation is obtained by aggregating these considerations into an

overall score. This approach provides interpretability by exposing intermediate reasoning steps, while maintaining alignment with the underlying optimization formulation.

5.4 System Implementation: UrbanMind-AI

The proposed framework is implemented as an interactive system, *UrbanMind-AI*, which integrates data processing, LLM reasoning, and visualization into a unified platform. Figure 5.1 presents the user interface of the system. The interface allows the user to select a reasoning method, provide problem requirements, and obtain optimized candidate locations for high school site selection. The results are displayed through an interactive map along with detailed scoring and reasoning outputs, enabling direct comparison between candidate sites.

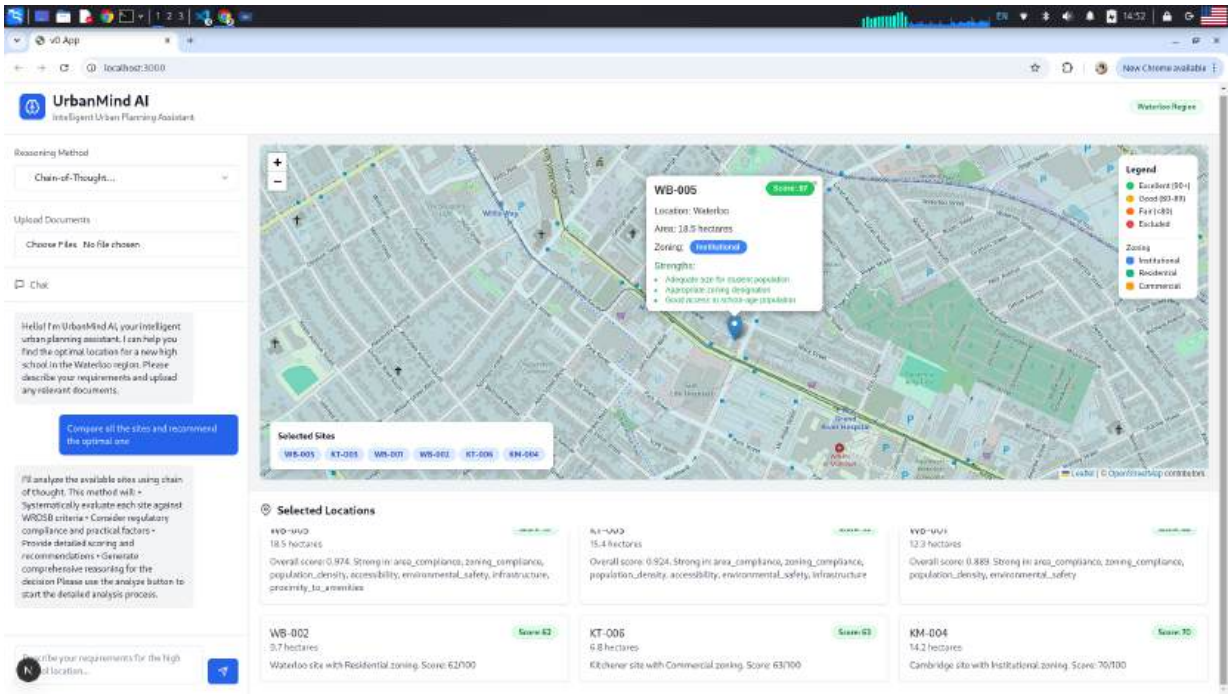

Figure 5.1: UrbanMind-AI interface for interactive site selection. The system combines map-based visualization, reasoning outputs, and candidate comparison.

The system architecture consists of a backend implemented in Python using FastAPI, which handles reasoning and scoring, and a frontend built with React and Next.js for inter-

active visualization. The platform allows users to explore candidate sites, run reasoning-based evaluations, and compare results across different methods.

The implementation supports multiple reasoning strategies, including CoT, ToT , and hierarchical reasoning approaches, enabling comparative analysis of different LLM reasoning paradigms.

5.5 Empirical Results

5.5.1 Spatial Distribution of Candidate Sites

The evaluation considers a set of candidate sites across the Waterloo Region, including locations in Waterloo, Kitchener, and Cambridge. Each site is characterized by its geographic coordinates and a set of planning-related attributes, including area, zoning type, population accessibility, transportation access, environmental constraints, infrastructure readiness, and proximity to amenities. These attributes are derived from publicly available planning data and reflect criteria commonly used in high school site selection by the Waterloo Region District School Board.

To evaluate candidate locations, a weighted scoring function $S(s)$ is defined for each site s:

$$S(s) = \sum_{i=1}^{K} w_i \, S_i(s), \tag{5.2}$$

where $S_i(s)$ denotes the normalized score of site s with respect to criterion i, and w_i represents the corresponding weight. All criterion scores are normalized to the interval $[0, 1]$ to ensure comparability across heterogeneous attributes.

The relative importance of each criterion is determined through predefined weights, summarized in Table 5.1. These weights reflect planning priorities, with greater emphasis placed on population accessibility and zoning feasibility, as these factors directly influence the long-term viability of a school location.

The results presented in this section are generated using the CoT-based reasoning pipeline, where the model evaluates each site by reasoning over these criteria and producing structured assessments that are aggregated into the final score.

Figure 5.2 illustrates the spatial distribution of candidate sites along with their corresponding normalized scores. Higher-scoring locations are primarily concentrated in Cambridge and parts of Waterloo, where favorable zoning conditions and strong population

Table 5.1: Evaluation criteria and relative weights used in the scoring function.

Criterion	Weight
Population accessibility	0.20
Zoning compliance	0.20
Area suitability	0.15
Transportation access	0.15
Environmental constraints	0.10
Infrastructure readiness	0.10
Proximity to amenities	0.10

accessibility contribute to overall performance. In contrast, several sites in Kitchener exhibit lower scores, often due to environmental constraints, limited infrastructure capacity, or less suitable zoning designations.

5.5.2 Baseline Ranking of Candidate Sites

Figure 5.3 presents the ranking of all candidate sites using the weighted scoring function $S(s)$ introduced earlier. This baseline evaluation is purely optimization-based and does not involve LLM reasoning. Instead, it aggregates normalized criterion scores using predefined weights.

The highest-scoring site is CM-009 with a score of 0.942, followed by WB-001 with a score of 0.901. These sites satisfy all major constraints and achieve strong performance across key criteria such as zoning feasibility and population accessibility. The inclusion of all candidate sites in this figure provides a global view of the solution space and highlights the relative differences in suitability across locations.

The clear separation between top-performing and lower-performing sites indicates that the scoring framework provides sufficient discriminative power for decision-making. However, this ranking is based solely on fixed weights and does not incorporate higher-level contextual reasoning.

5.5.3 CoT Evaluation

Figure 5.4 shows the results obtained using CoT-based reasoning. In contrast to the baseline ranking, this evaluation is not performed on the full set of candidates. Instead, a subset

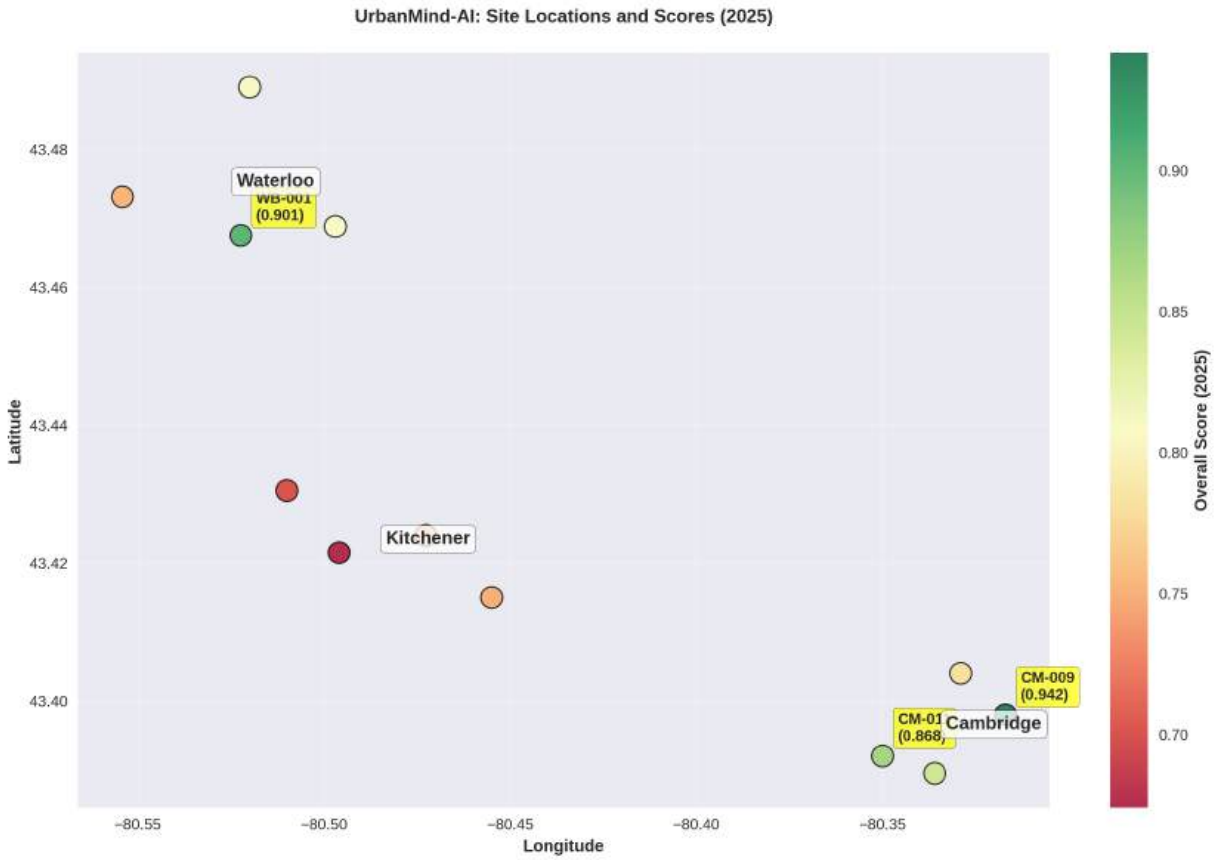

Figure 5.2: Geographical distribution of candidate sites across Waterloo, Kitchener, and Cambridge, with color indicating normalized scores.

of high-ranking sites from Figure 5.3 is selected and re-evaluated using the CoT reasoning pipeline. This two-stage design reduces computational cost while focusing reasoning on the most promising candidates.

Within this subset, the CoT method identifies WB-005 as the optimal site, with an overall score of approximately 0.974, outperforming the top-ranked baseline candidates. This shift in ranking reflects the influence of structured reasoning, which reinterprets the evaluation criteria beyond fixed weights.

The CoT-based evaluation places greater emphasis on factors such as accessibility, population coverage, and spatial distribution relative to existing schools. The reasoning traces indicate that WB-005 achieves strong balance across these dimensions, particularly

Figure 5.3: Normalized scores of all candidate sites based on the weighted optimization model.

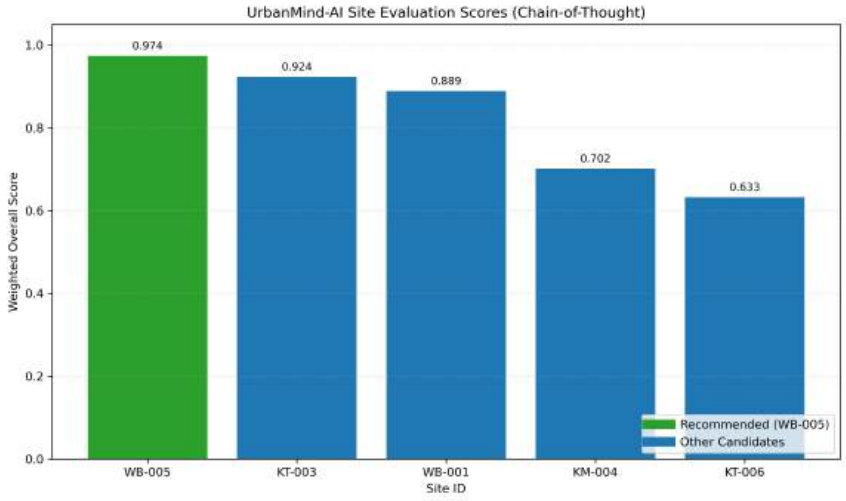

Figure 5.4: Site evaluation scores for a selected subset of top candidate sites using CoT reasoning.

in transportation access and coverage of underserved populations.

This comparison highlights a key distinction between the two approaches. The baseline model provides a consistent global ranking based on predefined weights, while the CoT-based evaluation refines this ranking by incorporating contextual and spatial reasoning. As a result, LLM-based reasoning can adjust decisions in cases where multiple candidates have similar quantitative scores but differ in qualitative characteristics.

5.5.4 Criterion-Level Analysis

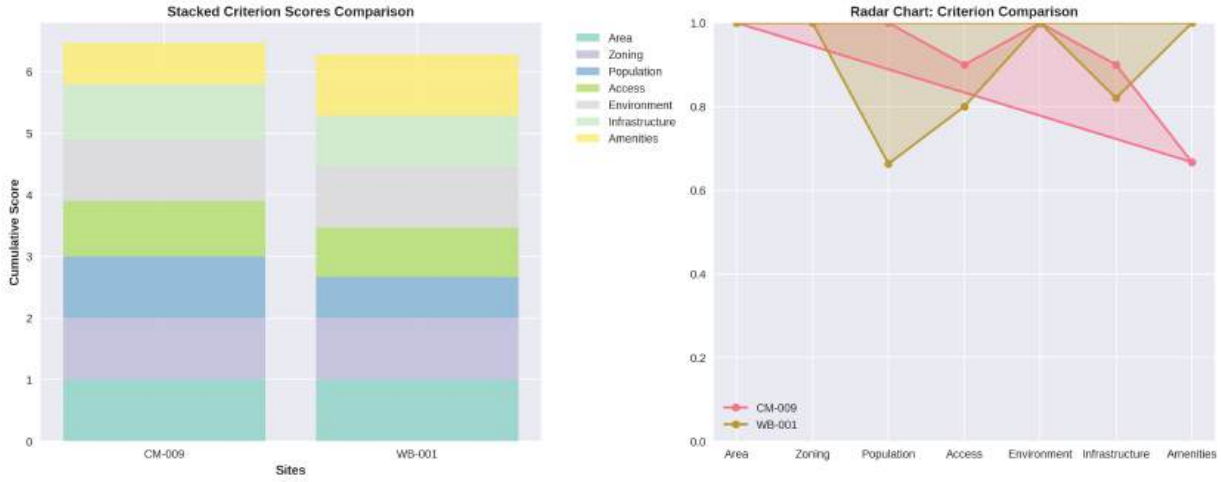

Figure 5.5: Stacked criterion scores and radar chart comparing performance across evaluation criteria.

Figure 5.5 provides a detailed breakdown of performance across individual criteria. The stacked bar chart shows cumulative contributions to the overall score, while the radar chart highlights relative strengths across dimensions.

The analysis reveals that CM-009 achieves high scores in population coverage and environmental safety, whereas WB-001 provides more balanced performance across all criteria. However, WB-005, identified by the CoT method, achieves consistently strong performance across all dimensions, particularly in population accessibility and infrastructure.

This comparison highlights the importance of balanced performance in multi-criteria decision problems.

5.6 Discussion and Insights

The results of this case study highlight the effectiveness of combining structured optimization with LLM-based reasoning for complex decision-making tasks in urban planning.

The weighted optimization model provides a reliable quantitative baseline by aggregating multiple planning criteria into a unified score. This formulation captures essential regulatory and spatial constraints, enabling a consistent comparison across candidate sites. The resulting ranking successfully identifies high-quality locations, demonstrating that multi-criteria optimization remains a strong foundation for infrastructure planning problems.

At the same time, the integration of CoT reasoning introduces an additional layer of analysis that refines these outcomes. The difference between the baseline ranking, which selects CM-009, and the CoT-based evaluation, which identifies WB-005 as the optimal site, reflects the ability of the reasoning process to incorporate contextual and relational factors. In particular, aspects such as spatial distribution, accessibility patterns, and proximity to existing schools are considered more holistically within the reasoning framework, leading to a more balanced evaluation.

A notable advantage of this approach lies in its interpretability. The reasoning process produces explicit, step-by-step justifications for each decision, making the evaluation transparent and traceable. This is particularly valuable in urban planning contexts, where decisions must be communicated clearly to stakeholders and supported by understandable evidence rather than opaque numerical outputs.

These observations suggest that optimization and reasoning play complementary roles. The optimization model ensures consistency and quantitative rigor, while the LLM-based reasoning enriches the decision process by incorporating contextual understanding and explanatory depth. The resulting hybrid framework provides both analytical reliability and interpretability, making it well-suited for real-world planning applications.

5.7 Summary

In this chapter, we formulated the problem of high school site selection in the Waterloo Region as a multi-criteria optimization problem and demonstrated how LLM-based reasoning can be applied to solve it. The UrbanMind-AI system provides a practical implementation of this framework, integrating data-driven scoring with interpretable reasoning.

The results show that while the optimization model identifies strong candidate sites, LLM-based reasoning can refine these decisions and provide additional insights. This case study illustrates the potential of combining optimization and reasoning for real-world decision-making tasks in urban planning.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis investigated the use of large language models for solving mathematical and optimization problems expressed in natural language, with a particular focus on structured decision-making in urban planning. The central challenge addressed in this work is the transformation of unstructured textual descriptions into reliable and verifiable computational solutions.

The results demonstrate that reasoning in LLMs cannot be understood as a single unified process. Language-based reasoning methods provide interpretability and flexibility, but they are prone to error propagation and lack formal guarantees. Execution-based approaches improve numerical reliability by separating reasoning from computation, yet they depend on the correctness of generated programs. Optimization-based methods offer exact solutions when the problem formulation is accurate, although their effectiveness is limited by the difficulty of extracting precise mathematical representations from natural language.

The integration of these approaches within a unified framework leads to a more reliable reasoning pipeline. In particular, the incorporation of verifier-in-the-loop mechanisms improves solution feasibility by explicitly checking constraint satisfaction and consistency. Empirical results on NL4OPT show that structured reasoning combined with verification significantly improves accuracy, while experiments on LiveMathBench highlight the importance of reasoning robustness and model capacity in symbolic problem solving.

The analysis of reasoning vector engineering provides a complementary perspective. Hidden-state representations capture meaningful distinctions between correct and incorrect

reasoning trajectories, especially in deeper layers of the model. However, the inability to translate these representations into effective control mechanisms suggests that reasoning behavior is governed by complex interactions rather than simple linear structures.

The urban planning case study further demonstrates the practical relevance of the proposed framework. The UrbanMind-AI system shows how natural language descriptions of planning criteria can be translated into structured decision processes, combining quantitative optimization with qualitative reasoning. The observed differences between optimization-based rankings and reasoning-based evaluations highlight the importance of integrating multiple perspectives in real-world decision-making tasks.

Taken together, these findings indicate that reliable reasoning requires the combination of structured computation, flexible language understanding, and explicit verification. The proposed framework provides a step toward bridging natural language and formal optimization, enabling large language models to operate in more rigorous and application-driven settings.

6.2 Future Work

The results of this thesis open several directions for further research, both at the methodological level and in practical applications. A key limitation observed throughout the experiments is the difficulty of translating natural language into precise formal representations. Improving this translation process requires deeper integration between language models and symbolic reasoning systems. Future work may explore hybrid neuro-symbolic architectures that combine LLMs with constraint solvers, logic engines, or program synthesis frameworks in a more tightly coupled manner.

Another important direction concerns adaptive reasoning systems. The empirical results show that different reasoning strategies perform well under different problem structures. Developing mechanisms that automatically select or combine reasoning paradigms based on task characteristics could significantly improve performance and efficiency. This may involve meta-reasoning models or controller architectures that dynamically route problems to appropriate reasoning pipelines.

The findings on reasoning vector engineering suggest that internal representations contain useful information but are not easily controllable through simple interventions. Future research may investigate nonlinear or structured approaches to representation manipulation, as well as training-time methods that align internal representations with desired reasoning behaviors.

Scalability and robustness remain open challenges, particularly for large-scale optimization problems. Extending the proposed framework to handle more complex, nonlinear, or stochastic optimization settings would improve its applicability to real-world scenarios. This includes incorporating uncertainty modeling, probabilistic reasoning, and multi-objective optimization within the reasoning pipeline.

From an application perspective, the integration of LLM-based reasoning into decision-support systems presents significant opportunities. In urban planning, this includes expanding the system to incorporate real geospatial data, policy constraints, and stakeholder interactions. More broadly, the framework can be applied to domains such as supply chain optimization, energy systems, and infrastructure planning, where decisions are often described in natural language but require formal analysis.

Finally, improving the interpretability and trustworthiness of reasoning systems is essential for deployment in high-stakes settings. Future work may focus on developing more transparent verification mechanisms, better explanation generation, and human-in-the-loop systems that allow domain experts to interact with and validate model outputs. Advancing these directions will contribute to the development of reasoning systems that are not only more accurate, but also more reliable, interpretable, and aligned with real-world decision-making processes.

References

[1] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," in *Advances in Neural Information Processing Systems*, 2023.

[2] W. Chen, S. Lee, X. Wang, and W. Zhang, "Program-of-thoughts prompting: Disentangling computation from reasoning for numerical tasks," in *EMNLP 2023*, 2023. Introduces PoT framework; separates reasoning and execution.

[3] K. Liu, T. Yigitcanlar, W. Browne, and Y. Fu, "Prompts for planning-ai integration: Llm prompt design for supporting sustainable urban development," *Journal of Open Innovation: Technology, Market, and Complexity*, vol. 11, p. 100666, 2025.

[4] Z. Tang, C. Huang, X. Zheng, S. Hu, Z. Wang, D. Ge, and B. Wang, "Orlm: Training large language models for optimization modeling," *arXiv preprint arXiv:2405.17743*, 2024.

[5] A. Patil and A. Jadon, "Advancing Reasoning in Large Language Models: Promising Methods and Approaches," *arXiv preprint arXiv:2502.03671*, 2025.

[6] J. Schneider, "Generative to Agentic AI: Survey, Conceptualization, and Challenges," *arXiv preprint arXiv:2504.18875*, 2025.

[7] M. A. Ferrag, N. Tihanyi, and M. Debbah, "From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review," *arXiv preprint arXiv:2504.19678*, 2025.

[8] F. Cheng, H. Li, F. Liu, R. v. Rooij, K. Zhang, and Z. Lin, "Empowering LLMs with Logical Reasoning: A Comprehensive Survey," *arXiv preprint arXiv:2502.15652*, 2025.

[9] H. Liu, Z. Fu, M. Ding, R. Ning, C. Zhang, X. Liu, and Y. Zhang, "Logical Reasoning in Large Language Models: A Survey," *arXiv preprint arXiv:2502.09100*, 2025.

[10] P.-Y. Wang, T.-S. Liu, C. Wang, Z. Li, Y. Wang, S. Yan, C. Jia, X.-H. Liu, X. Chen, J. Xu, *et al.*, "A survey on large language models for mathematical reasoning," *ACM Computing Surveys*, vol. 58, no. 8, pp. 1–35, 2026.

[11] Z. Ke, F. Jiao, Y. Ming, X.-P. Nguyen, A. Xu, D. X. Long, M. Li, C. Qin, P. Wang, S. Savarese, C. Xiong, and S. Joty, "A Survey of Frontiers in LLM Reasoning: Inference Scaling, Learning to Reason, and Agentic Systems," *arXiv preprint arXiv:2504.09037*, 2025.

[12] B. Højer, O. Jarvis, and S. Heinrich, "Improving reasoning performance in large language models via representation engineering," *arXiv preprint arXiv:2504.19483*, 2025.

[13] Q. Li, X. Feng, Y. Ma, Z. Ye, R. Chen, X. Feng, and B. Qin, "Unlocking multilingual reasoning capability of llms and lvlms through representation engineering," *arXiv preprint arXiv:2511.23231*, 2025.

[14] J. Pan, "Conversational context classification: A representation engineering approach," *arXiv preprint arXiv:2601.12286*, 2026.

[15] W. Y. Chan, S. Chen, H. Jing, K. H. Lau, E. C.-C. Li, Z. Wang, H. Li, and Y. Song, "Do reasoning models enhance embedding models?," *arXiv preprint arXiv:2601.21192*, 2026.

[16] Z. Zhou, Y. Lin, and Y. Li, "Large language model empowered participatory urban planning," *arXiv preprint arXiv:2402.01698*, 2024.

[17] Y. Zheng, F. Xu, Y. Lin, P. Santi, C. Ratti, Q. R. Wang, and Y. Li, "Urban planning in the era of large language models," *Nature Computational Science*, 2025.

[18] Z. Zhou, Y. Lin, D. Jin, and Y. Li, "Large Language Model for Participatory Urban Planning," *arXiv preprint arXiv:2402.17161*, 2024.

[19] Y. Jiang, Q. Chao, Y. Chen, X. Li, S. Liu, and G. Cong, "UrbanLLM: Autonomous Urban Activity Planning and Management with Large Language Models," *arXiv preprint arXiv:2406.12360*, 2024.

[20] X. Zhao, H. Huang, T. Yang, Y. Lu, L. Zhang, R. Wang, Z. Liu, T. Zhong, and T. Liu, "Urban planning in the age of large language models: Assessing openai o1's performance and capabilities across 556 tasks," *Computers, Environment and Urban Systems*, vol. 121, p. 102332, 2025.

[21] E. R. Carranza, S.-Y. Huang, and G. Li, "Llm planning agents: Exploring the potential and challenges of large language model agents in urban design and planning," in *Architectural Informatics, Proceedings of the 30th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2025*, vol. 1, pp. 223–232, 2025.

[22] Z. Jiao, M. Sha, H. Zhang, X. Jiang, and W. Qi, "City-leo: Toward transparent city management using llm with end-to-end optimization," *arXiv preprint arXiv:2406.10958v2*, 2024.

[23] F. Zhang and B. Wu, "Large language models as general purpose intelligence systems for reasoning, planning and decision making," *American Journal of Artificial Intelligence and Neural Networks*, vol. 6, no. 4, pp. 45–72, 2025.

[24] C. Xie and D. Zou, "A human-like reasoning framework for multi-phases planning task with large language models," *arXiv preprint arXiv:2405.18208v1*, 2024.

[25] A. Forootani, "A survey on mathematical reasoning and optimization with large language models," *arXiv preprint arXiv:2503.17726v1*, 2025.

[26] M. J. Abdel-Rahman, Y. Alslman, D. Refai, A. Saleh, M. A. Abu Loha, and M. Y. Hamed, "Teaching llms to think mathematically: A critical study of decision-making via optimization," *arXiv preprint arXiv:2508.18091v1*, 2025.

[27] B. Zhang, P. Luo, G. Yang, B.-H. Soong, and C. Yuen, "Or-llm-agent: Automating modeling and solving of operations research optimization problems with reasoning llm," *arXiv preprint arXiv:2503.10009v3*, 2025.

[28] J. Chu-Carroll, A. Beck, G. Burnham, D. O. Melville, D. Nachman, A. E. Özcan, and D. Ferrucci, "Beyond llms: Advancing the landscape of complex reasoning," *arXiv preprint arXiv:2402.08064v1*, 2024.

[29] Waterloo Region District School Board, "Long-term accommodation plan 2020–2030 (draft)," April 2021. Draft report.

[30] R. Ramamonjison, T. Yu, R. Li, H. Li, G. Carenini, B. Ghaddar, S. He, M. Mostajabdaveh, A. Banitalebi-Dehkordi, Z. Zhou, *et al.*, "Nl4opt competition: Formulating optimization problems based on their natural language descriptions," in *NeurIPS 2022 Competition Track*, pp. 189–203, PMLR, 2023.

[31] J. Liu, H. Liu, L. Xiao, Z. Wang, K. Liu, S. Gao, W. Zhang, S. Zhang, and K. Chen, "Are your llms capable of stable reasoning?," *arXiv preprint arXiv:2412.13147*, 2024.

[32] M. Li, X. Zhou, and H. Cheng, "Verifier feedback improves llm-based optimization reasoning," *arXiv preprint arXiv:2504.07640*, 2025. Empirical evaluation of solver feedback for optimization.

[33] D. Yang, Y. Zhao, H. Chen, and G. Li, "Verifier-in-the-loop reasoning: Enhancing reliability of llm outputs through programmatic checking," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE 2024)*, 2024. Introduces verifier-guided iterative correction.

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, 2022.

[35] Y. Feng, N. Weir, K. Bostrom, S. Bayless, D. Cassel, S. Chaudhary, B. Kiesl-Reiter, and H. Rangwala, "Vericot: Neuro-symbolic chain-of-thought validation via logical consistency checks," *University of Pennsylvania and Amazon Web Services*, 2025.

[36] M. Liu, R. Zhao, B. Tang, and Z. Li, "Beyond chain-of-thought: A comprehensive survey of reasoning methods in large language models," in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025. Survey covering CoT, ToT, PoT, and verifier methods.

[37] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[38] R. Vacareanu, A. Pratik, E. Spiliopoulou, Z. Qi, G. Paolini, N. A. John, J. Ma, Y. Benajiba, and M. Ballesteros, "General purpose verification for chain of thought prompting," *AWS AI Labs*, 2024.

[39] S. Thakur, V. Saxena, R. Kulkarni, S. Singh, P. Selvam, H. Patel, and H. Kanayama, "Generating verifiable cot from execution-traces," *IBM Research*, 2025.

[40] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2022.

[41] Z. Sarayloo, "Llm reasoning benchmarks: Evaluating mathematical and optimization problem solving." https://github.com/zsarayloo/LLM-Reasoning-Benchmarks, 2025.

[42] M. Moussa, Y. Mostafa, and A. A. Elwafa, "School site selection process." Procedia Environmental Sciences, 2017. International Conference - Green Urbanism, GU 2016.