Uncertainty-Aware Test-Time Search for Optimization Problem Solving

Linlin Yu[1]*, Xujiang Zhao[2]†, Dong Li[3], Yanchi Liu[2], Wei Cheng[2], Zhengzhang Chen[2], Chen Zhao[3], Feng Chen[4], Haifeng Chen[2]

[1]Augusta University, [2]NEC Labs America,
[3]Baylor University, [4]The University of Texas at Dallas
linyu@augusta.edu, {xuzhao, yanchi, weicheng, zchen, haifeng}@nec-labs.com
{dong_li1, chen_zhao}@baylor.edu, feng.chen@utdallas.edu

Abstract

Automatically solving optimization problems from natural language descriptions with both efficiency and reliability is highly desirable but remains challenging. Language model hallucinations and the limited availability of labeled datasets often result in misaligned formulations, code errors, and feasibility failures. We propose **UMCTS**, an Uncertainty-aware Monte Carlo Tree Search framework that combines the language understanding capability of large language models with the reliability of well-established solvers. UMCTS structures the solution process into four stages: global instruction, assumptions, mathematical formulation, and solver code generation. It employs Monte Carlo Tree Search with semantic-equivalence pruning, prior-guided exploration, and solver-based feasibility checks. An LLM judge provides numerical reward signals, qualitative error information, and uncertainty estimates. These signals are backpropagated to guide the search and flag unreliable outputs. Across six public benchmarks, UMCTS achieves state-of-the-art solution accuracy and improves efficiency by reducing token usage.

1 Introduction

An optimization problem aims to identify the best possible decision by maximizing or minimizing a numerical objective while satisfying a set of constraints. Optimization problems are widespread in domains such as planning, logistics, finance, and engineering (Belegundu and Chandrupatla, 2019; Chen et al., 2021; Delgado et al., 2022; Li et al., 2023), where they play a critical role in decision-making. Given a natural language description of an optimization problem, the *modeling* step is to translate the description into a formal mathematical model. This model defines the decision variables, the objective function, and the constraints. Once the formulation is constructed, the problem is solved using mathematical algorithms, typically through code tailored to an optimization solver (*implementation*), which produces the final solution.

Solving optimization problems requires expertise across multiple domains, making automation highly desirable yet challenging. The modeling step demands domain-specific knowledge (e.g., in finance or operations) and a deep understanding of mathematical modeling. The implementation step requires programming proficiency and familiarity with solver APIs. Notably, over 82% of Gurobi users hold an advanced degree (Gurobi Optimization, 2024), highlighting the steep expertise barrier. As a result, automating the entire pipeline from natural language description to solver-executable code has become an important and technically demanding research goal. Recent advances in large language models (LLMs) have sparked significant interest in using them to automate optimization problem-solving. Compound AI systems aim to integrate the language understanding capabilities of LLMs with the computational reliability of classical optimization solvers. Designing and verifying compound systems that connect these two techniques to do an end-to-end solution is an active area of research (Zaharia et al., 2024; Pan et al., 2023).

Current approaches to bridging LLM with optimization solvers fall into three main categories. *Prompt-based* methods directly prompt LLMs to generate solver-ready code (Xiao et al., 2023; AhmadiTeshnizi et al., 2023). However, this approach often results in misaligned formulations, syntactic errors, and infeasible models due to ambiguity in problem descriptions, hallucinations in generation, and the inherent uncertainty of LLM outputs. *Learning-based* methods fine-tune a pretrained LLMs using domain-specific datasets (Ma et al., 2024; Jiang et al., 2024; Yang et al., 2025; Lu et al., 2025). This method is limited by the scarcity and

*Work done during an internship at NEC Labs America.
†Corresponding author.

low quality of structured datasets in the optimization domain. For example, at least 26% of problems in the widely used IndustryOR (Tang et al., 2024) dataset are ill-defined or incomplete, limiting the reliability of fine-tuned models. Generalization also remains a concern, as models trained on narrow domains often fail to adapt to unseen problem types. *Search-based* methods (Astorga et al., 2024) apply test-time scaling techniques (Zhang et al., 2025) to enhance LLM performance during inference through search or ensemble strategies. These methods reduce reliance on high-quality training data and improve generalization by leveraging the inherent capabilities of LLMs. However, their efficiency and reasoning limitations remain practical challenges.

Motivated by test-time scaling methods that operate without a training corpus, and with the goal of improving efficiency and reliability, we propose an Uncertainty-aware Monte-Carlo Tree Search framework. UMCTS structures the formulation process into a four-stage pipeline: global instruction, assumptions, mathematical formulation, and solver code generation. Our main contributions are threefold:

- UMCTS is training-free and generalizable, using the inherent reasoning capabilities of LLMs to handle diverse optimization problems without reliance on large labeled datasets.
- UMCTS backpropagates uncertainty-aware value estimates through MCTS to guide the search, using epistemic uncertainty to prioritize high-confidence branches and prune low-confidence ones, which improves search efficiency and solution reliability.
- Extensive experiments on six standard benchmarks show that UMCTS consistently outperforms prompt-based, learning-based, and search-based baselines while reducing token usage.

2 Related Work

Optimization Problem Solving with LLMs. *Prompt-based* systems explicitly steer an LLM to produce a mathematical formulation, generate solver code, and run it. Chain-of-Experts (Xiao et al., 2023), OptiMUS (AhmadiTeshnizi et al., 2023), OptimAI (Thind et al., 2025), adopt a multi-agent setting where each agent is assigned a specific role (e.g., problem understanding, formulation, coding, and verification). These methods rely on carefully tuned agent roles and prompt templates, limiting their generalization ability on unfamiliar optimization tasks. *Learning-based* methods including LLMOPT (Jiang et al., 2024) and OptMATH (Lu et al., 2025) collect training data and fine-tune pretrained models to create task-specific language models. While they can generate more accurate mathematical formulations, they are computationally expensive.

Our UMCTS framework belongs to the *search-based* category built on test-time scaling, which allocates extra computation at inference to draw more reasoning ability from a fixed language model (Zhang et al., 2025). AutoFormulation (Astorga et al., 2024) decomposes the formulation into sets, variables, parameters, objective, and constraints, builds a search tree over these components, and sequentially generates each part conditioned on the previously generated ones. At each step, it samples multiple candidates per component. Whether the solver can generate feasible numerical values based on the complete component or not is set as the numerical reward to guide the search, which substantially increases token consumption.

Uncertainty Quantification in MCTS. MCTS balances exploration and exploitation using upper-confidence rules that rely on visit counts and rollout returns, but these capture only statistical uncertainty from finite sampling at a node. Moerland et al. (Moerland et al., 2020) argue that a second source of uncertainty arises from limited knowledge and show that ignoring it can bias search toward locally well-explored yet globally uncertain branches. EMCTS (Oren et al., 2022) extends MCTS with explicit epistemic signals, propagating uncertainty from learned value and reward models alongside returns, which improves search efficiency when combined with AlphaZero-style models.

When using MCTS for optimization problem solving with a vast search space and possible noisy generation from LLM, we argue that quantifying and leveraging uncertainty is important for both reliability and efficiency.

LLM as Judge. Automatically evaluating the quality of LLM-generated content is a challenging task, especially for complex outputs like mathematical formulations or programming code, where diverse outputs share the correctness. LLM-as-a-Judge (Zheng et al., 2023) uses another advanced LLM as evaluators, employing Grading (question-answering for single answer score) or Pairwise Evaluation (ranking multiple outputs) (Doddapaneni et al., 2024). This approach leverages the LLM's understanding of language, but concerns

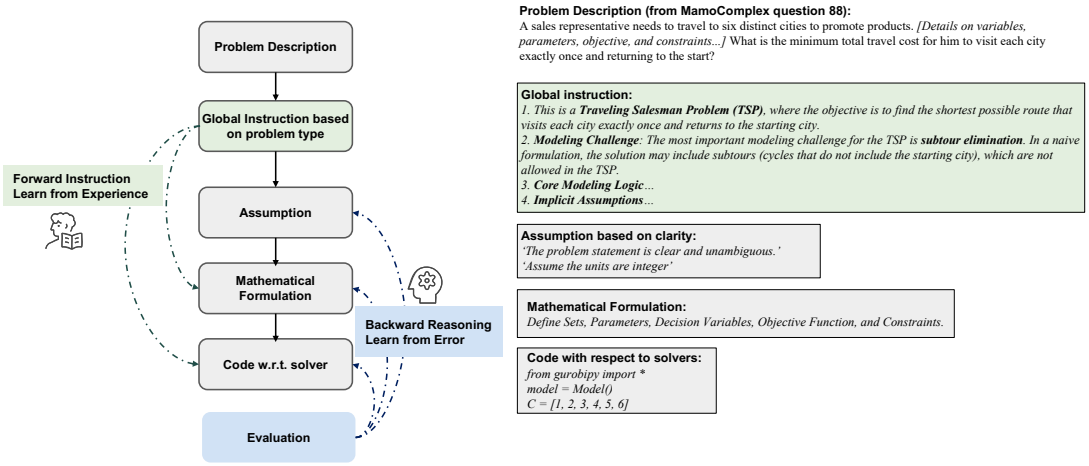

Figure 1: Tree Structure of the Formulation Process. The formulation process can be naturally represented as a tree structure, where each node corresponds to one step. The root node represents the initial natural language problem description, and leaf nodes correspond to executable code snippets. The left panel shows sequential stages, from Problem Description to Evaluation, with a forward "learn from experience" path and a backward "learn from error" path. The right panel gives a Traveling Salesman Problem example that lists modeling guidance (e.g., subtour elimination), clarifies assumptions, sketches the formulation, and shows a code stub.

remain about reliability and bias (Yamauchi et al., 2025). In our work, we not only adopt LLM as a judge to evaluate the quality and feasibility of generated mathematical formulation from the perspectives of completeness and consistency, but also incorporate the solver execution results as another judge to improve the reliability of evaluation.

3 Methodology

Optimization problems aim to find the best outcome under given constraints by minimizing or maximizing an objective function over decision variables. A standard formulation is given by:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \le 0, \quad i = 1,\dots,m, \\ & h_j(\mathbf{x}) = 0, \quad j = 1,\dots,p, \\ & x_k \in \mathbb{Z}, \quad k \in \mathcal{I} \subseteq \{1,\dots,n\} \end{aligned}$$

Translating the natural language description to the above mathematical formulation typically involves extracting five key elements: sets, variables, parameters, objectives, and constraints (Jiang et al., 2024; Astorga et al., 2024). We propose UMCTS, a test-time search framework that couples a language model with classical solvers to automatically generate and execute solver-ready code for optimization problems.

3.1 Tree Structure

UMCTS structures the formulation process into a four-stage pipeline: global instruction, assumptions, mathematical formulation, and solver code generation, which is different from the previous search-based models (Astorga et al., 2024). The root node represents the initial natural language problem description, which provides the overall context and requirements for the optimization task. Each subsequent level of the tree corresponds to one of the four stages, with leaf nodes representing complete solver-executable code snippets. The tree structure allows for systematic exploration of different formulation and implementation options, enabling the model to consider multiple interpretations and solution strategies. Figure 1 illustrates the tree structure and the four-stage pipeline.

Global Instruction. The first stage generates a global instruction that provides a high-level plan based on the problem type. Given a new optimization problem, a human typically identifies its type (e.g., traveling salesman or maximum flow) and recalls relevant solution patterns. For example, the traveling salesman problem requires subtour elimination, while maximum flow problems often invoke the Ford-Fulkerson or Edmonds-Karp algorithms. This planning step highlights essential modeling elements and provides a global view that reduces forgetting in long contexts and distribution bias within the training data. We experimentally observe that current LLMs (e.g., the GPT family)

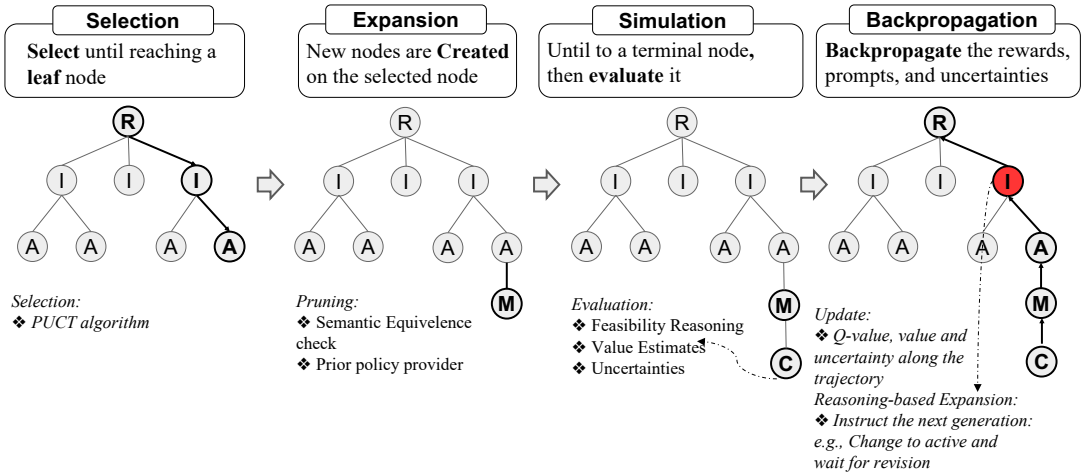

Figure 2: Uncertainty-aware MCTS Framework for Optimization Problem Solving. The pipeline follows four phases: Selection, Expansion, Simulation, and Backpropagation and over a tree of reasoning states (R), instructions (I), assumptions (A), formulations (M), and code (C). Selection uses a PUCT policy to reach a leaf; Expansion adds children and prunes by semantic equivalence and a prior policy. Simulation evaluates a terminal node with feasibility checks, value estimates, and uncertainty metrics. Backpropagation updates Q-values, values, and uncertainties along the trajectory, and uses the feedback to guide the next expansion step.

can identify the key steps for solving most optimization problems. This is intuitive because many optimization problems follow standard paradigms, and abundant unstructured information about them is available on the Internet and likely present in LLM pretraining data. We therefore leverage the model's stored knowledge and reasoning to generate effective global instructions that guide subsequent problem-solving steps.

Assumption. The second stage focuses on clarifying any assumptions or simplifications needed to make the problem feasible. The current datasets often contain incomplete or ambiguous descriptions, which are also common scenarios in practice. In addition, Real-world optimization problems often involve complexities and uncertainties that require assumptions to define a solvable model. For example, one might assume linear relationships between variables, ignore certain constraints, or consider average-case scenarios. The second stage is to identify and state any assumptions required to fill in missing details or resolve ambiguities. Clearly stating these assumptions helps ensure that the mathematical formulation accurately reflects the intended problem while remaining computationally feasible. This stage is crucial for setting the boundaries of the model and guiding the formulation process. Notably, this stage also helps increase the diversity of the generated formulations by allowing the model to explore different plausible interpretations of the problem, such as the integer problem or continuous problem. By varying the assumptions, the model can produce a range of formulations that capture different aspects of the problem space, which is beneficial for robust optimization.

Mathematical Formulation. The third stage generates the core mathematical formulation, specifying sets, variables, parameters, objectives, and constraints. This stage translates the problem description and assumptions into a mathematical model. Different from (Jiang et al., 2024; Astorga et al., 2024) where the language model is prompted to generate each component sequentially conditioned on previously produced parts, our approach generates the entire formulation in a single step. Experiments show that generating the formulation in a single step better captures interdependence among components and reduces error accumulation compared to sequential generation.

Solver Code Generation. The final stage converts the mathematical formulation into executable code for a chosen optimization solver. The code must define the model objects, set solver options, and call the optimizer to return a solution. The generator is equipped with a self-reflection mechanism that iteratively refines the code based on runtime feedback until a syntactically and semantically valid implementation is obtained. In other words, the generated code will be at least syntactically correct, as checked by the solver execution return. Because no single solver covers all problem classes, for example, Gurobi is strong on Linear Programming,

Mixed Integer Linear Programming, Quadratic Programming, and Mixed Integer Quadratic Programming, but does not support general nonlinear programs, while (Mixed Integer) Non Linear workloads are typically handled via frameworks such as SCIP (often with an NLP engine like IPOPT). We add the randomness during this step to generate code with respect to different solvers.

3.2 MCTS for Optimization Formulation

MCTS maintains a tree structure by tracking, for each node s_k at level $k \in [1, 4]$, its parent node $\texttt{parent}(s)$, children nodes $\texttt{child}(s)$, the current state s, the estimated value $\hat{V}(s)$, and the epistemic uncertainty $\text{Var}(\hat{V}(s))$. We suppose a deterministic transition and do not express the action explicitly, but related to the state transition, i.e., $a := s_{k-1} \to s_k$. For each transition (s, a), the algorithm also maintains the mean reward $\hat{R}(s, a)$, its variance $\text{Var}(\hat{R}(s, a))$, and the action probability $\pi(a|s)$. As in standard MCTS, we additionally track the visit count $N(s, a)$ and the Q-value $Q(s, a)$ for each state-action pair. Figure 2 illustrates the overall UMCTS framework.

Selection. At each step, we select one of the current node's children by balancing exploitation (high-reward nodes) and exploration (less-visited nodes). We adopt the Predictor Upper Confidence Bound for Trees (PUCT) (Rosin, 2011) algorithm. The selection phase terminates upon reaching a leaf node. The action selection at state s_k is given by:

$$a^* = \arg\max_a \Big[Q^*(s_k, a) + \pi(a \mid s_k)\, C_{\text{PUCT}} \frac{\sum_{a'} N(s_k, a')}{1 + N(s_k, a)} \Big] \tag{1}$$

Here, $Q^*(s_k, a)$ is the current Q-value estimate for taking a at s_k (e.g., the weighted mean return over all trajectories include (s_k, a)), $\pi(a \mid s_k)$ is a prior policy that biases search toward promising actions, $N(s_k, a)$ is the visit count of edge (s_k, a), $\sum_{a'} N(s_k, a')$ is the total visits of the parent, and C_{PUCT} controls the exploration strength. The first term favors actions with high estimated quality, while the second term increases the score of rarely visited actions in proportion to their prior, yielding a principled trade-off between trying new branches and exploiting known good ones.

Expansion. Each node in the tree represents a component generated by the language model, conditioned on its associated prompt and the sequence of previously generated components (i.e., parent nodes). To do effective and efficient expansion, we propose a two-step procedure: components generation and components clustering.

Step 1: Components Generation. At level $k \in \{1, 2, 3, 4\}$, given the context s_0, the recursive parent node list $[s_1, \cdots, s_{k-1}]$, and prompt ρ_k, with an LLM, we can sample N_s responses, i.e.,

$$m^1, m^2, \cdots, m^{N_s} \leftarrow \texttt{LLM}(s_0, \cdots, s_{k-1}, \rho_k) \tag{2}$$

For each component m with L tokens, represented as $m = (m_1, m_2, \ldots, m_L)$, one can get the token-level probability $p(m_l)$, and then calculate the entropy $\mathbb{H}(m_l) = -\sum_{\mathcal{V}} p(m_l) \log p(m_l)$. Then the entropy for the whole response is,

$$\begin{aligned} \mathbb{H}(m) &= \frac{1}{L} \sum_l \mathbb{H}(m_l) \\ &= -\frac{1}{L} \sum_l \sum_{\mathcal{V}} p(m_l) \log p(m_l) \end{aligned} \tag{3}$$

Step 2: Components Clustering. We then cluster the components based on the semantic equivalence. Specifically, we can derive clusters $\mathcal{C}^1, \mathcal{C}^2, \ldots$. For each cluster $\mathcal{C} = (m^1, \cdots, m^{|\mathcal{C}|})$, we will derive,

1. Random sample one element from the cluster as the *candidate* s_k;

2. The policy probability is:

$$\pi(a_{s_{k-1} \to s_k} | s_{k-1}) = \frac{|\mathcal{C}|}{N_s} \tag{4}$$

3. The mean reward function is:

$$\hat{R}(s_{k-1}, a_{s_{k-1} \to s_k}) = -\frac{1}{|\mathcal{C}|} \sum_c \mathbb{H}(m^c) \tag{5}$$

4. The epistemic uncertainty of reward (variance) is:

$$\text{Var}\left(\hat{R}(s_{k-1}, a_{s_{k-1} \to s_k})\right) \tag{6}$$

Steps 1 and 2 together produce a diverse yet manageable set of candidate child nodes for expansion. The clustering step prunes semantically equivalent candidates, reducing redundancy and focusing the search on distinct formulations. The prior policy $\pi(a|s_{k-1})$ and reward statistics $\hat{R}(s_{k-1}, a)$, $\text{Var}(\hat{R}(s_{k-1}, a))$ provide valuable signals for guiding future selection and expansion decisions.

Value Estimate. Instead of rolling out to a terminal state at each leaf node during the tree construction as classic MCTS, we estimate the value and variance of the value of a terminal node $\hat{V}(s_4)$, which represents the quality of the final solution. An LLM-based evaluator is prompted to assess the quality of the generated formulation and the value of this simulation trajectory is evaluated from two complementary perspectives as illustrated in Figure 3:

(i) Consistency between the Mathematical formulation and the original problem description: binary indicator $\mathbb{I}_{\text{consistency}}$ associated with error explanation $e_{\text{consistency}}$ if not met. The consistency check will mainly focus on the component consistency, such as the hallucination of parameters or the omission of constraints.

(ii) Feasibility check with the solver execution result: binary indicator $\mathbb{I}_{\text{feasibility}}$ associated with error explanation $e_{\text{feasibility}}$ if not met. Without the ground truth value, there is currently no reliable way to verify the global optimality of a solution, but a feasibility check is a necessary condition for optimality and a basic requirement for any practical solution. The feasibility check will check whether the solution derived from the Solver satisfies the constraints in the problem description.

Suggested by (Yamauchi et al., 2025) for LLM-as-a-Judge, sampling-based scoring with mean aggregation outperforms scoring with greedy decoding and single evaluation. We therefore sample N_e evaluations from the LLM and aggregate the binary indicators by majority vote. The overall value of the final node s_T^i is defined as a weighted sum of the two binary indicators:

$$\hat{V}(s_T^i) := \text{Mean}\left(\lambda_1 \mathbb{I}^p_{\text{feasibility}} + \lambda_2 \mathbb{I}^p_{\text{consistency}}\right)$$
$$\text{Var}\left(\hat{V}(s_T^i)\right) := \text{Var}\left(\lambda_1 \mathbb{I}^p_{\text{feasibility}} + \lambda_2 \mathbb{I}^p_{\text{consistency}}\right)$$

The variance term captures epistemic uncertainty in the value estimate. The value estimate aggregates evidence of numerical solvability and factual correctness from both solver outputs and language-based checks, and it incorporates the associated uncertainty.

Backpropagation with uncertainty. The backpropagation phase updates the statistics of all nodes and edges along the trajectory from the terminal node s_T back to the root s_0, as well as the error information. This includes updating visit counts, Q-values, value estimates, and uncertainty measures based on the results of the evaluation.

Numerical backpropagation is a standard step in MCTS that updates the statistics of all nodes and edges along the trajectory from the terminal node s_T back to the root s_0. Following (O'Donoghue et al., 2018; Oren et al., 2022), we apply an uncertainty-aware Bellman update to backpropagate uncertainty measures alongside the value returns.

$$\nu^i(s_k, a) = \hat{R}(s_k, a) + \gamma \hat{V}\left(s_{k+1}^i\right) \tag{7}$$

$$\begin{aligned}\text{Var}\left(\nu^i(s_k, a)\right) = \text{Var}\left(\hat{R}(s_k, a)\right) \\ + \gamma^2 \text{Var}\left(\hat{V}\left(s_{k+1}^i\right)\right)\end{aligned} \tag{8}$$

The Q-value is updated as in standard MCTS by averaging returns over all trajectories that include the edge (s_k, a):

$$q_{\hat{M}}(s_k, a) = \frac{1}{N(s_k, a)} \sum_i \nu^i(s_k, a) \tag{9}$$

$$\begin{aligned}&\sqrt{\text{Var}\left(q_{\hat{M}}(s_k, a)\right)} \\ &\leq \frac{1}{N(s_k, a)} \sum_i \sqrt{\text{Var}\left(\nu^i(s_k, a)\right)}\end{aligned} \tag{10}$$

Based on Theorem 1 in (Oren et al., 2022), the optimal Q-value across all possible value estimate models M is upper-bounded by the average standard deviation of the returns with high probability:

$$\begin{aligned}&\mathbb{P}\Big(Q^*(s_k, a) \leq \\ &\max_\pi q_{\hat{M}}(s_k, a) + \sqrt{\text{Var}\big(q_{\hat{M}}(s_k, a)\big)}\Big) \geq 1 - \delta\end{aligned}$$

Therefore, we use the derived upper bound as the estimate of $Q^*(s_k, a)$, which is then applied in the selection phase (see Equation 1) to guide subsequent search decisions. We claim that there is a trade-off between uncertainty and reward in MCTS selection.

In addition to numerical statistics, we also backpropagate error information from the evaluation to the layer of Mathematical formulation and Assumption. Each terminal node s_T^i is associated with error explanations $e^i_{\text{feasibility}}$ and $e^i_{\text{consistency}}$ if the corresponding checks failed. As we backpropagate, we aggregate these error explanations at each ancestor node s_k along the trajectory and augment one revised node based on the error message. The augmented node will inherit the statistical information from the ill node.

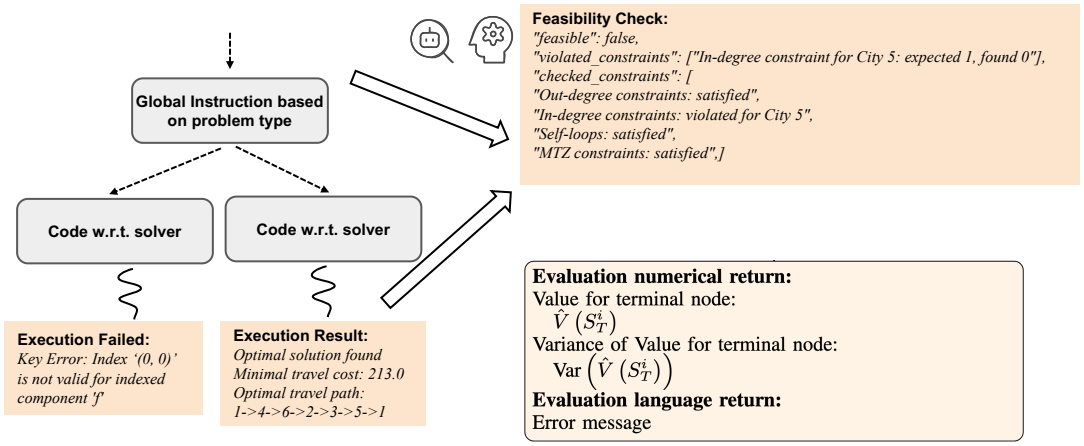

Figure 3: Value Estimate Pipeline.

Table 1: Solution Accuracy Comparison

	NL4Opt	NLP4LP	MamoEasy	ComplexOR	MamoComplex	IndustryOR
Prompt-based (GPT4)						
Chain-of-Experts (ICLR'24) 64	n.r.	n.r.	38	40	n.r.	n.r.
OptiMus (ICML'24)	n.r.	n.r.	n.r.	67	n.r.	n.r.
Training-based (Qwen)						
ORLM (OR'25)	86	n.r.	85	n.r.	44	25
OptMath (ICML'25)	96	n.r.	90	n.r.	54	31
LLMOPT (ICLR'25)	93	n.r.	97	73	68	46
MCTS-based						
Autoformulation (GPT-4) (ICML'25)	93	n.r.	n.r.	72	62	48
UMCTS (GPT-4o mini)	**96**	**97**	**98**	**83**	**78**	**56**

n.r.: not reported. The best results in each column are bolded.

4 Experiments

To evaluate the effectiveness and efficiency of UMCTS, we conduct comprehensive experiments across six benchmark datasets, comparing our approach with prompt-based, learning-based and search-based baselines. Our study is designed to answer the following key research questions:

RQ1: Effectiveness. How does UMCTS perform in terms of solution accuracy compared to state-of-the-art methods across diverse optimization problem types and complexities?

RQ2: Efficiency. How does UMCTS balance solution quality with computational cost, particularly in terms of token usage and inference time with search-based methods?

RQ3: Reliability. How does the uncertainty back-propagation in UMCTS improve the reliability of generated formulations and solutions?

4.1 Experimental Setup

Dataset setting. We evaluate UMCTS on six public benchmarks: NL4Opt (Ramamonjison et al., 2022), NLP4LP (AhmadiTeshnizi et al., 2023), MamoEasy, MamoComplex (Huang et al., 2024), ComplexOR (Xiao et al., 2023), and IndustryOR (Tang et al., 2024). We manually verify and correct errors in these datasets, such as missing information, conflicting objectives, and most importantly, incorrect ground-truth solutions that would mislead model evaluation. The corrected version of these datasets is provided for reproducibility[1].

Baselines. We compare UMCTS with three categories of baselines: (i) Prompt-based methods including Chain-of-Experts (Xiao et al., 2023) and OptiMUS (AhmadiTeshnizi et al., 2023); We also include a directly ask model and a sequential ask model for comparison and verify the claims in Section 3.1. (ii) Learning-based methods including LLMOPT (Jiang et al., 2024), OptMath (Lu et al., 2025), and ORLM (Tang et al., 2024); (iii) Search-based methods including Autoformulation (Astorga et al., 2024). Considering the reproducibility and the cost of using advanced LLMs, we borrow

[1] GitHub Repository

the numbers reported in their papers for the baselines and note the LLM version used. Our model UMCTS is implemented with GPT-4o-mini and compares it with a direct ask and sequential ask baselines using the same model.

4.2 Results and Analysis

Effectiveness on solution accuracy. Table 1 summarizes the solution accuracy of UMCTS compared to state-of-the-art baselines across six benchmark datasets. UMCTS achieves the highest solution accuracy on all datasets, demonstrating its effectiveness in generating correct and feasible formulations. Notably, UMCTS attains a solution accuracy of 78% on MamoComplex, significantly outperforming the best baseline (LLMOPT at 62%) by a margin of 16 percentage points. This improvement is particularly impressive given MamoComplex's complexity and diversity, indicating UMCTS's robustness across various problem types.

Ablation Study. We consider two model variants to verify the effectiveness of our proposed UMCTS. *Direct Ask* is to directly prompt the LLM to generate the complete formulation and code in one step, without any decomposition or search. *Sequential Ask* decomposes the generation into the sequential stages but does not use MCTS for search. Both variants use the same LLM (GPT-4o-mini) as UMCTS for a fair comparison, and results are shown in Table 2. UMCTS shows consistent improvements across all three complex datasets. The gains are most pronounced on MamoComplex, where UMCTS attains 78% accuracy, outperforming direct ask (48%) by 30 points and sequential ask (36%) by 42 points. These results highlight the benefits of UMCTS's structured search and uncertainty-aware evaluation in enhancing solution quality compared to straightforward prompting strategies.

Table 2: Solution Accuracy Comparison with GPT-4o-mini

	ComplexOR	**MamoComplex**	**IndustryOR**
	GPT-4o mini		
Direct Ask	78	48	39
Sequential Ask	67	36	35
UMCTS	**83** (↑ 5)	**78** (↑ 30)	**56** (↑ 17)

RQ2: Efficiency. For the test time scaling methods, the token usage is the key factor reflecting efficiency. We compare the token usage and solution accuracy of UMCTS with Autoformulation (Astorga et al., 2024) in Table 3. UMCTS achieves a solution accuracy of 78% on MamoComplex with only 64,367 tokens, significantly lower than Autoformulation (80,245 tokens). For our model with GPT-4o-mini API, the average inference token cost is $0.019 USD, while Autoformulation with GPT-4 API costs $0.320 USD per instance. This demonstrates that UMCTS effectively balances solution quality with computational cost, making it a more efficient choice for optimization problem solving.

Table 3: Token Usage and Solution Accuracy Comparison on MamoComplex

	# Token / SA% / Cost $
Autoformulation (GPT-4) (ICML'25)	80,245 / 62 / 0.320
MCTS (GPT-4o mini)	**54,213 / 70 / 0.016**
UMCTS (GPT-4o mini)	**64,367 / 78 / 0.019**

RQ3: Reliability. To evaluate the reliability of uncertainty backpropagation, we compare UMCTS with a variant that disables this component (denoted as MCTS). As shown in Table 3, UMCTS attains a solution accuracy of 78% on MamoComplex, exceeding MCTS (70%) by 8 percentage points, albeit with a modest increase in token usage. This result suggests that propagating uncertainty signals during search provides useful guidance, enabling the exploration to escape local optima. For example, we found that UMCTS can avoid generating "lazy solutions", which satisfy the hard constraints but not the optimality, such as setting all decision variables to zero in maximum flow problems. The uncertainty-aware evaluation helps detect and avoid such suboptimal formulations, improving the overall reliability of the generated solutions.

5 Conclusion

We present UMCTS, an uncertainty-aware tree search framework that integrates LLM with well-established solvers to automatically solve optimization problems. UMCTS avoids fine-tuning LLMs on task-specific data and instead leverages the pre-trained knowledge of LLMs through an effective and efficient search strategy built upon uncertainty quantification, semantic-equivalence pruning, and solver-based feasibility checks. UMCTS achieves state-of-the-art solution accuracy, improved reliability, and reduced computational cost compared to prompt-based, learning-based and search-based approaches.

6 Limitations

Although UMCTS demonstrates strong performance, several limitations remain. First, the current evaluation relies primarily on feasibility, which is a necessary but not sufficient condition for optimality. Future work could investigate methods to better approximate or iteratively verify solution optimality. Second, the global instruction component (the first layer of the tree) has substantial influence, and it would be valuable to study model variants where deeper layers are generated by weaker LLMs to save cost. Third, ambiguity in natural language descriptions can lead to multiple valid interpretations, and developing ways to incorporate common knowledge for aligning assumptions with real-world scenarios remains an open challenge.

7 Acknowledgments

This work is partially supported by the National Science Foundation (NSF) under Grants No.2515265, 2220574, and 2107449.

References

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2023. Optimus: Optimization modeling using mip solvers and large language models. *arXiv preprint arXiv:2310.06116*.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. 2024. Autoformulation of mathematical optimization models using llms. *arXiv preprint arXiv:2411.01679*.

Ashok D Belegundu and Tirupathi R Chandrupatla. 2019. *Optimization concepts and applications in engineering*. Cambridge University Press.

Yi Chen, Aimin Zhou, and Swagatam Das. 2021. Utilizing dependence among variables in evolutionary algorithms for mixed-integer programming: A case study on multi-objective constrained portfolio optimization. *Swarm and Evolutionary Computation*, 66:100928.

Erwin J Delgado, Xavier Cabezas, Carlos Martin-Barreiro, Víctor Leiva, and Fernando Rojas. 2022. An equity-based optimization model to solve the location problem for healthcare centers applied to hospital beds and covid-19 vaccination. *Mathematics*, 10(11):1825.

Sumanth Doddapaneni, Mohammed Safi Ur Rahman Khan, Sshubam Verma, and Mitesh M Khapra. 2024. Finding blind spots in evaluator llms with interpretable checklists. *arXiv preprint arXiv:2406.13439*.

Gurobi Optimization. 2024. State of mathematical optimization report. https://www.gurobi.com/resources/report-state-of-mathematical-optimization-2024/. Accessed: Sep. 2025.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2024. Mamo: a mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144*.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2025. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2678–2710.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. 2024. Llmopt: Learning to define and solve general optimization problems from scratch. *arXiv preprint arXiv:2410.13213*.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. 2023. Large language models for supply chain optimization. *arXiv preprint arXiv:2307.03875*.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. 2025. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Forty-second International Conference on Machine Learning*.

Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Guojun Peng, Zhiguang Cao, Yining Ma, and Yue-Jiao Gong. 2024. Llamoco: Instruction tuning of large language models for optimization code generation. *arXiv preprint arXiv:2403.01131*.

Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. 2020. The second type of uncertainty in monte carlo tree search. *arXiv preprint arXiv:2005.09645*.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. 2018. The uncertainty bellman equation and exploration. In *International conference on machine learning*, pages 3836–3845.

Yaniv Oren, Villiam Vadocz, Matthijs TJ Spaan, and Wendelin Böhmer. 2022. Epistemic monte carlo tree search. *arXiv preprint arXiv:2210.13455*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. 2022. Nl4opt competition: Formulating optimization problems based on their natural language descriptions.

In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 189–203. PMLR.

Christopher D Rosin. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230.

Zhengyang Tang, Chenyu Huang, Xin Zheng, Shixi Hu, Zizhuo Wang, Dongdong Ge, and Benyou Wang. 2024. Orlm: Training large language models for optimization modeling. *arXiv preprint arXiv:2405.17743*.

Raghav Thind, Youran Sun, Ling Liang, and Haizhao Yang. 2025. Optimai: Optimization from natural language using llm-powered ai agents. *arXiv preprint arXiv:2504.16918*.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and 1 others. 2023. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*.

Yusuke Yamauchi, Taro Yano, and Masafumi Oyamada. 2025. An empirical study of llm-as-a-judge: How design choices impact evaluation reliability. *arXiv preprint arXiv:2506.13639*.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. 2025. Optibench meets resocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*.

Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The shift from models to compound ai systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. 2025. What, how, where, and how well? a survey on test-time scaling in large language models. *arXiv preprint arXiv:2503.24235*.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan. 2023. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. 2025. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. In *Forty-second International Conference on Machine Learning*.

A Appendix

A.1 Related Work on Optimization Problem Solving with LLMs

Existing methods for using LLMs to solve optimization problems fall into prompt-based, learning-based, and search-based approaches.

Prompt-based systems explicitly steer an LLM to produce a mathematical formulation, generate solver code, and run it. Chain-of-Experts (Xiao et al., 2023) adopts a multi-agent setting coordinated by a conductor. There is an interpreter that extracts domain terms and intent, a modeler writes the formulation, a coder produces executable solver code, and a reviewer checks and revises. OptiMUS (AhmadiTeshnizi et al., 2023) also uses multiple agents, but follows a different workflow. It first converts the problem text into a structured record of parameters, objectives, constraints, and context. A manager then cycles clause by clause through a formulator, programmer, and evaluator to get the solution. OptimAI (Thind et al., 2025) adopts a plan-before-code multi-agent pipeline with a formulator, planner, coder, and code critic, and adds UCB-based debug scheduling to switch among plans. These methods rely on carefully tuned agent roles and prompt templates, limiting their generalization ability on unfamiliar optimization tasks.

Learning-based methods collect training data and fine-tune pretrained models to create task-specific language models. While they can generate more accurate mathematical formulations, they are computationally expensive. LLMOPT (Jiang et al., 2024) defines a "five-element" intermediate schema for optimization problems and fine-tunes a base model with multi-instruction supervision, alignment, and self-correction. The training dataset uses dual labeling, where experts generate gold five-element labels and solver code with assistance from LLMs. OptMATH (Lu et al., 2025) constructs a large synthetic corpus through a bidirectional pipeline and produces triplets of natural language, mathematical formulation, and problem data.

Search-based methods are built on test-time scaling, which allocates extra computation at inference to draw more reasoning ability from a fixed language model (Zhang et al., 2025). Tree-search

methods such as MCTS have shown marked gains on mathematical and reasoning benchmarks (Zhang et al., 2023; Huang et al., 2025; Zheng et al., 2025). For optimization solving, AutoFormulation (Astorga et al., 2024) decomposes the formulation into sets, variables, parameters, objective, and constraints, builds a search tree over these components, and sequentially generates each part conditioned on the previously generated ones. At each step, it samples multiple candidates per component. Whether the solver can generate feasible numerical values based on the complete component or not is set as the numerical reward to guide the search, which substantially increases token consumption.

A.2 Methodology Details

Mathematical Model for Optimization Problems. *Sets* are collections of related items, such as products, locations, or time periods. *Variables* represent the decision points to be optimized, like quantities to produce or routes to take. *Parameters* are fixed inputs that define the problem context, such as costs, capacities, and demands. The *Objective* is the function to be minimized or maximized, reflecting the goal of the optimization (e.g., minimizing cost or maximizing profit). *Constraints* are the rules that limit the values of the variables, including equality and inequality constraints (e.g., resource limits or demand fulfillment).

Backpropagation Illustration of UMCTS We provide a detailed illustration of the backpropagation process in UMCTS as Figure 4, highlighting how uncertainty is integrated into the updates. The backpropagation phase updates the statistics of all nodes and edges along the trajectory from the terminal node s_T back to the root s_0, as well as the error information. This includes updating visit counts, Q-values, value estimates, and uncertainty measures based on the results of the evaluation.

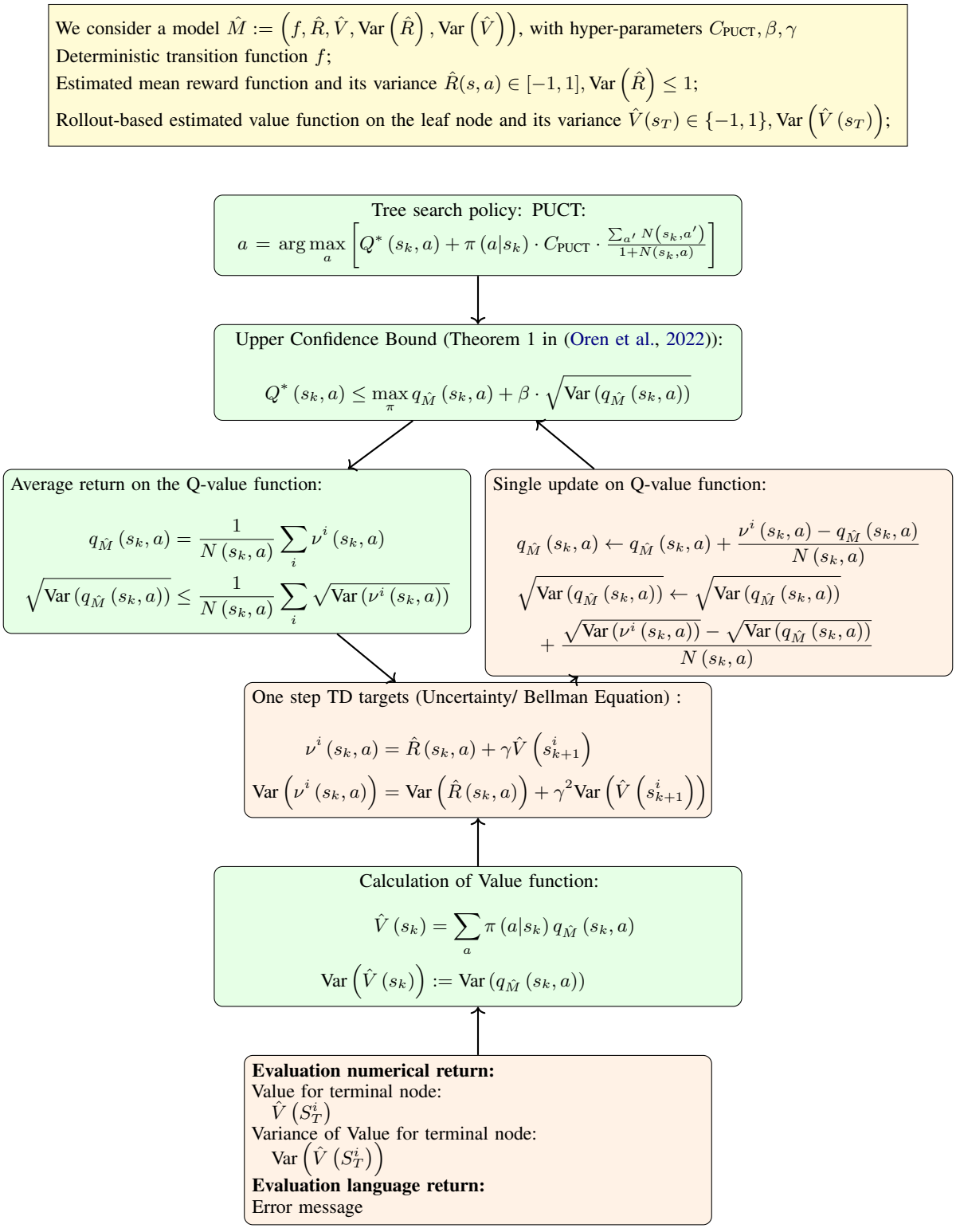

Figure 4: Epistemic Uncertainty Propagation in Action Selection and Value Backup