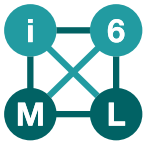

Chair of
Computer Science 6
(Machine Learning
and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.
Diese Arbeit wurde vorgelegt am Lehrstuhl für Maschinelles Lernen und Inferenz.

Novelty-based Tree-of-Thought Search for LLM Reasoning and Planning

Neuigkeit-basierte Tree-of-Thought-Suche für LLM-Planung und -Problemlösung

Bachelor Thesis
Bachelorarbeit

Presented by / Vorgelegt von

Leon Hamm
434986

Supervised by / Betreut von Zlatan Ajanovic, Ph.D..

1st Examiner / 1. Prüfer Prof. Hector Geffner, Ph.D.

2nd Examiner / 2. Prüfer Prof. Christopher Morris, Ph.D.

Aachen, September 23, 2025

Abstract

LLMs still have critical weaknesses in some reasoning and planning tasks. Although advances such as chain-of-thought, or more generally tree-of-thought through explicit prompting or more recently, reinforcement learning, have bettered performance, even these new techniques are brittle and have not achieved human-level reasoning and planning in all domains. Additionally, They also often suffer from high time and token cost. Inspired by the success of width-based search in planning, we explore how the concept of novelty can be transferred to language domains and how it can improve tree-of-thought reasoning. A tree of thoughts consists of possible "paths" of consecutive ideas or thoughts. These are generated by repeatedly prompting an LLM. In our paper a measurable concept of novelty is found that describes the uniqueness of a new node (thought) in comparison to nodes previously seen in search tree. Novelty is estimated by prompting an LLM and making use of embedded general knowledge from pre-training. This metric can then be used to prune branches and reduce the scope of the search. Although this method introduces more prompts, the overall token cost can be reduced. This procedure is tested and compared using several benchmarks in language-based planning and general reasoning.

Contents

1 Introduction **1**

2 Background **2**
2.1 Classical Planning . . . 2
2.2 Width-Based Planning . . . 2
2.3 Large Language Models . . . 3
2.4 Tree-of-Thought and other approaches . . . 4
2.5 LLMs in Planning . . . 4

3 Approach **6**
3.1 Tree of Thoughts . . . 6
3.2 Novelty-based Pruning . . . 7
3.2.1 Problem Width and Novelty Estimation . . . 8
3.2.2 Boolean Novelty Estimation . . . 8

4 Implementation **9**
4.1 Approach . . . 9
4.1.1 Tree of Thought . . . 9
4.1.2 Novelty Estimation and Pruning . . . 11
4.1.3 Other Implementation Details . . . 12
4.2 Evaluation Framework . . . 13

5 Evaluation **16**
5.1 Applicability and Potential . . . 16
5.2 Sub-task Evaluations . . . 17
5.3 Complete Approach Evaluation . . . 19
5.3.1 Naive Baseline . . . 20
5.3.2 Basic Tree-of-Thought . . . 20
5.3.3 Tree-of-Thought with Novelty Pruning . . . 22

6 Conclusion **26**

A Appendix **28**
A.1 Multiset Notation . . . 28
A.2 LLM Sub-task Examples . . . 28

A.3 LLM Repetition Example 30

List of Acronyms **32**

List of Figures **33**

List of Tables **34**

List of References **35**

1 Introduction

In recent years, large language models (LLMs) have garnered much attention for their surprising strengths. These so-called foundation models perform well in language-based problems of many different domains such as AP exams and coding [10, 25]. LLMs even display limited reasoning capabilities [23, 24, 38]. However, on closer inspection, it is clear that these models are incapable of complicated reasoning or long-term planning when used in a naive input-output fashion [33, 43]. Attempts to remedy this through more complicated methods such as Tree of Thoughts (ToT) [42] suffer from drawbacks such as high computation cost [18]. In ToT specifically, the cost stems from branching "thoughts" that lead to exponential runtime and token usage during the graph search.

Other methods of improving reasoning require reinforcement learning algorithms and human feedback, which usually includes an expensive training step [8, 19]. There are also some indications that too long Chain of Thoughts (CoTs) can lead to worse results [40].

On the other hand, classical algorithms such as Iterative Width (IW) have long been able to efficiently solve planning problems of limited width [20, 21]. IW achieves state-of-the-art performance by pruning states according to a measure of novelty and the problem's width. Width is defined such that the planner has a runtime that is exponential in the width of the problem. This is effective as it has been shown that many problem domains have low width. It stands to reason that the same is true for language-based planning tasks. The problem lies in defining a concept of width and novelty for the language domain.

Most implementations of classical planning algorithms also require the problem to be formalized in a specified manner. This rigid need for a certain problem structure has been a barrier to utilizing classical planning algorithms in many language-based domains.

In this thesis we hope to apply novelty-based pruning in ToT for planning and reasoning tasks. To this end, a planning algorithm using ToT is defined. It is comprised of four main tasks: action generation, successor state mapping, plan verification, and novelty estimation. The performance of an LLM in each of these sub-problems and the total algorithm will be evaluated similarly to the PlanBench [34] benchmark. The benchmark allows for side-by-side comparison of classical ground truths and alternative methods.

All code, prompts, and evaluation results for this thesis can be found in the GitHub repository[1].

[1] https://github.com/bobito25/bachelor-thesis

2 Background

2.1 Classical Planning

The classical planning problem is an important model for reasoning about actions [39]. It was devised in 1970s with the Stanford Research Institute Problem Solver (STRIPS) [9].
The topic of classical planning has been continuously researched since. Notably, in 1998, the Planning Domain Definition Language (PDDL) [1] was composed, a community standard, versions of which are used to this day.
The general approach assumes a state-based representation of the world. States are characterized by a collection of axioms, which can vary in type. The notation and problem schema devised by Lipovetzky and Geffner [21] in 2012 is used here. The set of all possibles states is denoted as S and an initial state $s_0 \in S$ is defined as a starting point. The agent or AI may then choose from a set of actions O that change the current state $s \in S$. Some restrictions or constraints can apply that limit which actions are allowed in a given state as given by $A(s) \subseteq O$. Any action $a \in A(s)$ will change the current state as specified by a successor function: $s' = f(a, s)$. The goal $S_G \subseteq S$ is defined as the set of target states that fulfill certain conditions. The planning problem can then be described as finding a sequence of allowed actions $a_1, ..., a_m$, a plan, that maps the input state to a goal state: the actions generate a series of states $s_0, s_1, ..., s_m$ where $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$, and $s_m \in S_G$. This plan can be optimal in qualities such as length. The planning problem can now be written as a tuple $\langle S, s_0, S_G, A, f \rangle$.
Most classical planners like Fast Forward [13] and Fast Downward [11] utilize heuristic search to find such optimal plans.

2.2 Width-Based Planning

In 2012, Lipovetzky and Geffner devised a new way of expressing the complexity and hardness of a planning problem [20, 21]. Their approach defines a width that provides a bound for the complexity of a problem. To do this, a concept of state novelty is needed. Every state s here is a subset of boolean features $\Phi(s) \subseteq F$. F is the set of all possible features as defined by the problem. Planning is then modeled as a tree search in the graph constructed by the initial state s_0 and all of its possible successors. The novelty $w(s)$ is the size of the smallest subset of features true in the current state s and false in all previously generated states S_s^-:

$$w(s) = \min_{\phi \subseteq \Phi(s), \phi \nsubseteq \Phi(s'), s' \in S_s^-} |\phi|. \tag{2.1}$$

This notion can be be exploited by width-based planners to prune states above a specified width k. The width $w(P)$ of a planning problem can then be simplified as the minimum novelty needed to solve the planning problem through breadth-first search while pruning all states with $w(s) > w(P)$. A width-based planner such as IW can find an optimal plan for a planning problem of width k in $O(|F|^k)$. Interestingly, most problems with a single goal fluent have a width less than or equal to two, leading to quadratic or even linear runtimes. Through problem serialization we are thus provided with an efficient planner that can quickly and optimally solve problems across many problem domains.

2.3 Large Language Models

Natural language processing has been a topic of extensive research ever since the 50s and 60s [6]. Recently, the field has advanced into natural language understanding using deep learning and neural networks instead of statistical modeling. The advent of pre-trained models using scalable transformer architecture [35] has catapulted the capabilities of current methods. Models of sufficient size (in parameters or compute) gain new, emergent abilities never seen before [37]. These models are focus of much recent research [44] and have been appropriately named large language models (LLMs). LLMs have gained the ability to learn in-context and excel at tasks such as conversation and exams. The ability to learn in-context allows a model to adapt to a new problem using not only what information it has been trained on but also new information contained in the context window at test time. While models have become more and more powerful, research has shown that how a model is used can severely impact performance [5, 38]. This has lead to a new field of research that tries to enhance the capabilities of LLMs by methods such as prompt engineering.

In the following, a notation close to the one devised by Yao et al. in 2023 is used. A pre-trained language model p with frozen parameters receives a sequence of prompt tokens $x = x[1], ..., x[n]$ and autoregressively predicts the next token by sampling from a probability distribution dependent on previous tokens: $x[n+1] \sim p(x[n+1]|x)$. This can be done repeatedly while adding generated tokens to already existing tokens to generate a completion $y = y[1], ..., y[m] = x[n+1], ..., x[n+m]$ for x. For simplicity, we will write this as $y \sim p(y|x)$, x and y being natural language texts that can be encoded using tokens. In the case of prompting techniques, inputs x are first mapped using a named function t^{name} before completion: $y \sim p^{name}(y|x) = p(y|t^{name}(x))$. The prompting function can prepend or append text and otherwise manipulate the input in a specified manner. Its name can be omitted if clear from context. In a naive problem-solution context, the problem is the prompt and the completion the solution. This naive method can be improved with few-shot prompting, where some example problem-solution pairs are added to the prompt [4]. Such techniques have been criticized for being brittle and overly reliant on manually tailoring prompts for the current domain [36] although it is also possible to generate such prompts autonomously [29].

2.4 Tree-of-Thought and other approaches

It may be surprising how much an LLM can achieve with auto-regressive left-to-right token prediction. Although their architecture is designed for text generation, they can even find solutions to some reasoning tasks. However, these models still have clear limitations. LLMs often do not ”think” about their answers and instead jump right to predicting the solution, akin to a human answering with the first things that comes to mind. They do not have a built-in way of deliberating on a problem. This leads to poor performance in tasks that require complicated reasoning or long-term planning [17].
In an attempt to remedy this, new methods have been developed. So far, large improvements were made by increasing train-time compute but recent approaches have focused on scaling test- or inference-time compute instead. Historically, this was first done explicitly by using prompting techniques.
One prominent and versatile technique is CoT [38]. It is a variant of few-shot prompting with the difference of an added reasoning path between example inputs and outputs. Remarkably, this simple trick is enough to significantly increase performance. A key factor here is the relevance and similarity of the presented examples to the problem domain. The design of such few-shot examples is usually done by a human, making domain transfer less efficient. A second aspect to note is the strict linearity of the approach. Newer methods such as ToT or Graph of Thoughts (GoT) [3] expand on this by modeling the reasoning steps as a tree or graph of thoughts. In practice, an LLM is repeatedly prompted to generate the next thought or reasoning step until a final output is produced. The model is then typically also used as an evaluation heuristic in breadth-first or depth-first search to find a solution. Additional modifications such as beam-search are also possible. These methods lead to improved performance in tasks such as language-based math reasoning but an important disadvantage is compute cost. These methods have high compute complexity as given by their branching factor [18].
More recent approaches have started training the LLM to implicitly use CoT on its own. New so-called large reasoning models such as OpenAI’s o1 [16] are fine-tuned using reinforcement learning to ”think” step by step. The disadvantage is that this requires large amount of compute resources and data.

2.5 LLMs in Planning

The achievements of LLMs have lead researchers to believe that perhaps an LLM might even be able to solve planning tasks. Efforts have been made to adapt these language models into frameworks that can solve planning problems [2, 14, 15]. The appeal is clear. LLMs being able to create and execute plans would open the path to many real world use cases. However, researchers disagree on what parts of planning the model should be responsible for.
There are currently two main approaches to using LLMs for planning.

The first uses the model in combination with some type of conventional and symbolic planner or verifier. Examples of tasks for the LLM include translating planning goals between natural language and a structured planning language [22, 41] or use as a decision-making heuristic [7, 28, 30].
The other approach omits a conventional planner and tries to solve the problem purely using the model and corrective measures [14, 27].
While both approaches improve upon the naive LLM baseline, the second approach especially has been criticized for its performance [17, 31]. According to these researchers, LLMs are overly reliant on fitting examples and should not be used as sole reasoners. Many approaches require a human to adapt the framework to the current domain and it is not trivial to create prompts such as few-shot reasoning examples automatically. That is why few-shot prompts will be avoided here whenever possible.

3 Approach

As discussed before, many classical planning problems have low width when serialized. However, in contrast to this supposed low hardness, LLMs still have difficulty with these problems [21, 33]. Our goal is to improve upon an existing method (Tree of Thoughts) to solve such problems by exploiting the low planning width.

3.1 Tree of Thoughts

We employ an LLM with ToT [42] as the planning problem solver. ToT relies on incremental steps for its nodes, which lets us interpret each ”thought” as a state. This means each node of the tree is a state s with the initial state s_0 as the root node. In this context any state is a natural language text that can be used to prompt the model. The goal is to find a series of thoughts through tree-search that leads to a valid solution to the problem. In this framework, the LLM can be responsible for four sub-tasks:

- action generation,
- successor state mapping,
- target state verification,
- and novelty estimation.

The first three will be described here since they are part of the baseline ToT, while the fourth will be discussed in the next section as it is the main novel concept. It should be noted that explicit action and successor state mapping is not part of the original ToT paper but is a construct taken from the planning domain to allow novelty over states.

Action Generation

The model acts as an action generator by using its common-sense heuristic and given knowledge about the problem to generate possible next actions in a given state. This can be done in multiple ways, most notably sampling and propose prompting. Sampling entails prompting the LLM multiple times independently for m possible next steps:

$$a_{i+1}^1 \sim p^{sample}(a_{i+1}|s_i), ..., a_{i+1}^m \sim p^{sample}(a_{i+1}|s_i). \tag{3.1}$$

In propose prompting, the LLM is asked to propose multiple different possible next steps within one completion:

$$a_{i+1}^1 ... a_{i+1}^m \sim p^{propose}(a_{i+1}^1 ... a_{i+1}^m | s_i). \tag{3.2}$$

This has the advantage of fewer duplicates and lower cost. It is similar to sampling while adding already generated states to the prompt:

$$a_{i+1}^1 \sim p(a_{i+1}|s_i), a_{i+1}^2 \sim p(a_{i+1}|s_i, a_{i+1}^1), ..., a_{i+1}^m \sim p(a_{i+1}|s_i, a_{i+1}^1, ..., a_{i+1}^{m-1}). \tag{3.3}$$

Successor State Mapping

After generating actions, the LLM must map these actions to states as a successor function:

$$s_{i+1} = f(a_{i+1}, s_i) \sim p^{successor}(s_{i+1}|a_{i+1}, s_i). \tag{3.4}$$

Different prompting methods can be used here, such as CoT. The resulting states are saved so they can be used for purposes such as novelty estimation.

Plan Verification

In each step the model must also also verify if the current state is a target state. This plan verification can be done either by trying to algorithmically parse the current state or by again prompting the LLM if a verifier is not available. The first option is domain dependent and presents the difficulty of forcing the model to output the state in a certain way, while the second can be unreliable due to the black-box nature of LLMs.

The performance of the model in any of these sub-tasks can be augmented using few-shot prompting with either domain specific or independent examples. It is also possible to combine action generation and successor state mapping into one step by asking the LLM to predict both or just either actions or states. This can be useful to reduce cost or in non-planning domains where no practical concepts of actions or states may exist.
The resulting tree can be built and searched by a chosen algorithm to find a solution to the given problem. This entire approach can, in principle, be applied to almost any reasoning or planning problem. A key disadvantage though, is the high time and token cost associated with LLM tree search. The core problem to be investigated is whether and how this tree search can be improved by pruning states according to some measure of novelty.

3.2 Novelty-based Pruning

To be able to efficiently prune states, the concept of novelty must be adapted. The main requirement of such an adaptation is an unstructured natural language input. Removing the

need for defining explicit atoms for each state and compelling the LLM to generate completions in a specific manner greatly simplifies the entire approach. However, the new concept of novelty must be analogous in some way to allow for worthwhile pruning. It must also be domain-agnostic and function as intended for any domain with states S.
It is assumed that states in any given domain have latent features $\Phi(s)$. Given a set of previous states $S_s^- \subseteq S$ and a new state $s \in S$ the new novelty $n(s|S_s^-)$ should correlate with the original novelty in some way.
Two approaches that may fulfill these requirements are proposed here.

3.2.1 Problem Width and Novelty Estimation

An LLM can be used as a novelty estimator: $n(s|S_s^-) \sim p^{nov}(n(s|S_s^-)|S_s^-, s)$ using a prompting function t^{nov}. The prompting function adds a wrapper that asks the language model to estimate the novelty of the new state is compared to the previous ones. However, an IW-like search as in classical planning is not prudent because of cost and missing guarantees for iteration. The LLM is used to estimate the width of the problem instance for this reason. These two estimators can then be used to prune any natural language states as in classic width-based pruning.
Problems may also arise when S_s^- is too large, leading to prohibitively expensive inference or lower estimator performance. This can be alleviated through sampling. In sampling, either m randomly selected states or the m last seen states can be used as a substitute for S_s^-.
This approach also relies on the fact that the model understands the concepts of novelty and width through its pre-training and prompt.

3.2.2 Boolean Novelty Estimation

In many cases, a concept of width or novelty may be difficult to define or apply. For this reason, a different method must be found that is more general. The LLM is asked whether the new state is novel, as a true or false question. This means the model effectively directly prunes the search tree according to its embedded idea of novelty. This method is very flexible and it could use the LLM's semantic understanding of text to, for example, give more weight to important features. The simplicity of this approach could also be beneficial for both the model's functioning and cost. Too large states can be handled in the same way as before.

4 Implementation

The subject of this section is the details of our specific implementation of all needed framework. First, the implementation of the main approach including ToT and novelty will be presented. Then, the evaluation framework will be discussed.

4.1 Approach

The implementation of the main approach can be divided, as previously done, into ToT and novelty estimation / pruning. Both have their own unique considerations.

4.1.1 Tree of Thought

Building on the Tree of Thought framework, a novel implementation was developed that introduces several modifications. The goals are performance, modularity, visualization, and universality.
Performance, measured by the speed at which the tree can be built, is important for rapid iterative development and testing. Modularity means that features should be able to be added and deactivated easily. This allows ablation tests and experimental designs. Visualization is also an important aspect important due to the high volume of requests and tree nodes. The logs of a tree can be very difficult to read, and visualization of resulting trees allows for quick assessments. The last goal is to keep the approach general and as domain-independent as possible. While this paper focuses on planning, it is but one application and other uses in reasoning tasks are feasible. The implementation takes care to preserve this property.

This is why the implementation allows two options for building the search tree. The first functions as discussed and specifically queries for the next action and then queries again to map the action to a successor state. The second option instead combines action generation and successor state mapping into one step. This makes it more general, because not all domains have sensible definitions for these tasks.

For both options, plan verification is also generalized to ensure that any goal is sufficient, not just a planning goal.

The prompts were designed to be domain-independent whenever possible. This also means no few-shot examples are used. This is helpful because it reduces overhead in utilizing the method. The only domain knowledge given is a hand-crafted context prompt that explains the rules of the domain.

In general, a prompting function will consist of multiple parts. An example prompt may be as follows:

```
You are a smart assistant that answers concisely.
You will be given a problem / question to solve.
You must output exactly the next step in solving the problem when prompted.
This is your problem statement:
{CONTEXT}
{PROBLEM STATEMENT}
Now you must find the next step.
Output only the next step in solving the problem or '[FINISHED]' if you are done
    → .
```

Any problem statement can easily be inserted while the context stays the same for a set domain. This prompt can be used to repeatedly query the LLM to obtain multiple possible next steps.

As small differences in prompts can have large differences in resulting behaviour, any reasonable evaluation must be done over several prompting functions. In the implementation, prompts can be easily exchanged via configs. A config contains all prompt parameters needed to run the approach. This allows for quick setup and testing of various prompting variants.

For example, if explicit CoT should be tested the last part of the prompt can easily be changed as such:

```
...
{PROBLEM STATEMENT}
Now you must find the next step. Give your reasoning and then '[OUTPUT]'
    → followed by the next step in solving the problem or '[FINISHED]' if you
    → are done.
```

When using explicit action-state-mapping, prompts might look like this:

```
...
{PROBLEM STATEMENT}
What is the next action to take based on the current state of the world to reach
    →  the goal? If the goal state is already reached, return "[FINISHED]".
{LLM ACTION RESPONSE}
What is the state of the world after applying this action? Output only the state
    →  of the world concisely, without any additional text.
```

In this case, this prompt must be queried in multiple steps. Responses such as the final result may also be extracted via an extra query such as in this example:

```
Extract the answer for the question
'{PROBLEM STATEMENT}'
from the answer
```

```
'{FINAL RESULT}'
Remove all reasoning and unnecessary text. Do not add any other text.
```

This is to avoid cluttering the prompt with many extraneous thoughts that may cause errors, perhaps by confusing the attention mechanism. One important use case for such an extraction is in leaf nodes of the tree. Every leaf represents a possible solution for the problem as given by the concatenation of all parent nodes. However, this may be an unnecessarily long string and contain extra information. The LLM can be queried to extract a concise solution. After extraction, a simple goal verification prompt may look like so:

```
Does the following answer correctly and completely solve the problem statement?
Your response must end with '[RESPONSE] yes' or '[RESPONSE] no'.
Problem statement:
{PROBLEM STATEMENT}
Answer:
{ANSWER}
```

While a more advanced prompt might add:

```
Redundant, unnecessary and additional information is fine as long as it is not
    → incorrect. The phrasing does not matter as long as the meaning is correct.
```

It is worth noting that only LLMs are used in the verification process and no symbolic models or plan verifiers. This only includes the ToT implementation and not the evaluation framework.

4.1.2 Novelty Estimation and Pruning

When using an LLM as a novelty estimator, the mechanisms are similar to the other prompts discussed so far. The main addition is a list of previously seen states.
Because a newly generated state always needs an up-to-date state list, the implementation can no longer be parallelized inside of a single run. Instead, the only parallelization is running multiple trees at the same time.

Since all states are strings, a prompt for boolean novelty estimation can be exceedingly simple:

```
Is '{new_state}' a novel state compared to this list of states '{
    → previous_states_str}'?
Respond with 'yes' or 'no'. Do not add any other text.
```

This will also work when directly asking for the next step instead of explicitly asking for states and actions but may lead to mixed results.

On the other hand, integer novelty estimation only makes sense when states are readily available. The instructions part of the prompt could look like this:

```
You are an expert in reasoning and search algorithms.

You are given:
Environment / rules: [describe the dynamics, constraints, or how the world
    → changes]
Initial state: [describe the starting situation]
Goal conditions: [list the desired target conditions or atoms]

Task: Estimate the problem width.
The width is the smallest number w of distinct conditions, variables, or
    → features that must be considered together in order to make systematic
    → progress toward the goal.
Width ≈ 1 means each goal or subgoal can be reached by tracking single
    → conditions independently.
Width ≈ 2 means pairs of conditions must be tracked jointly (dependencies
    → between two facts matter).
Width ≈ 3 means triples matter, and so on.

When reasoning, follow these steps:
Identify the key variables, features, or atoms that describe the problem.
Analyze the dependencies: can goals be achieved by considering features one at a
    →  time, or do combinations matter?
Estimate the minimal number of features that must be tracked jointly to
    → guarantee progress toward the goal.
```

The context, query state and goal atoms can be programmatically added to the instructions.

We avoid attempting to improve estimation performance through domain-specific prompts due to the main goal of generalizability as mentioned before.

To improve performance, different prompt templates for novelty estimation are tested using the implementation.

4.1.3 Other Implementation Details

Coming back to LLMs in general, an additional problem of these models is that, due to their nature, they are often used via the API of a large company such as OpenAI or Alphabet. This leads to a loss of control and changes in the backend may not be transparent. To ensure reproducibility and transparency, we use a self-hosted LLM that is available publicly: Qwen [32]. Tests were also done with other models to ensure that the approach works as intended with any LLM.

An option provided by newer Qwen models is a so-called thinking-mode. This allows the model to use the first part of its completion as hidden ”thinking” and only use the second

part as its normal output. The framework is designed so that this mode can be enabled and utilized as an additional enhancement. Of course, this comes at the cost of time and compute resources.

In general, compute resources and time are constraints on the amount of testing that can be done. For this reason, the base ToT framework is built to be highly parallel and in practice it is able to query a given model for all branches of a search tree simultaneously.

Since problem instances in the Blocksworld domain are often not solvable in just a few steps, breadth-first search as in standard IW is not advisable. Exponential growth leads to impractically large search trees, as confirmed by our tests. The hope then is, that depth-first search will more quickly find a solution and end the search. Other traversal strategies are also possible but not in the scope of this paper.

4.2 Evaluation Framework

As mentioned previously, the LLM is mainly responsible for 4 sub-tasks:

- action generation
- successor state mapping
- target state verification
- novelty estimation

These sub-tasks provide estimates for the performance of LLMs in planning and novelty estimation. Dividing the approach into its components and analyzing them separately improves understanding of the full approach.

For extensive evaluation of these sub-tasks, a testing framework inspired by PlanBench [34] was developed. The framework makes use of a planning domain simulator to easily generate an arbitrary number of test instances. The classical planning domain allows for a well-defined ground truth to many problems. The framework also allows bidirectional translation of PDDL and natural language using algorithmic parsing. Responses are compared to ground truths using a combination of LLM queries and this translation. For each sub-task, its implementation in the framework will be explained.

Action Generation

For each test instance, a random planning problem is generated. Then, a list of possible actions is generated using the simulator. The LLM is then prompted to list all possible actions given the current state. The current state can be given either in PDDL form or translated into natural language. The performance is measured by comparing the actions generated by the LLM to the ground truth of the simulator.
There is a variant to this task, in which the model is given the goal atoms and is queried to

generate one next action instead of all possible actions. The key difference here is that the model tries to find good actions instead of just possible actions. The single generated action can then be checked for validity and optimality. A valid action is allowed according to the Blocksworld rules and the optimal action is one that is the first action of an optimal plan. Examples for prompt templates and a concrete problem instance can be found in Appendix A.2.

Successor State Mapping

This sub-task uses the already generated states and actions from the previous sub-task. Each action is mapped to its corresponding successor task using the simulator. The LLM is prompted to do the same: map actions to resulting states. The results can be compared to the simulated states. This also has a variant where actions are mapped to states separately instead of together in one prompt and response.

Plan Verification

In a ToT run, the LLM will often only have the origin state and a series of actions (a plan) that may or may not be complete and lead to a goal state. This means two types of test instances are generated. The first half use the planner FastDownward [11] to generate a valid plan that leads to a goal state. The second half of the instances are generated using a random plan generator that guarantees the outputted plan does not reach a goal state. The LLM is thus prompted with an origin state and a (in)complete plan and must decide whether the plan leads to a state where all goal conditions are fulfilled.

Novelty Estimation

To start, a complete IW search on the origin state of every test instance is conducted and the lowest resulting width that reached a goal state is found. This will be called the instance width. To imitate a realistic intermittent search state, a partial width-based search in the state space is conducted. The search includes a classic width-based pruning mechanism that prunes any states with a novelty higher than the instance width. At a random point, the search is stopped and test instance is composed of the list of previous states and the new, current state.
There are two tasks for each of these instances.
In the first, the LLM is prompted to decide whether the new state is novel or whether it should be pruned. This can be compared to whether the state would have been pruned in the last iteration of IW which acts as the ground truth. The task is meant to evaluate the boolean novelty estimation capabilities of the model and to help design optimal prompts.
In the second, the model is prompted to estimate the novelty of the new state as an integer. This can be compared to the actual novelty of the state and represents the integer novelty estimation method.
One problem is that in this setting, most states are pruned because they are duplicates. The

actually difficult scenario of pruning states by comparing to the instance width does not occur often.
For this reason, there is also a more difficult variant in which the capabilities to prune are stress-tested. This variant only contains test instances where the new state would be pruned due to its novelty and not because it is a duplicate of a previous state. This means these query states are never a duplicate of a previous state and their novelty is always higher than the instance width.
Lastly, there is a further task to estimate the problem width of an instance which can again be compared to the classical ground truth.

5 Evaluation

Three main topics will be evaluated: the applicability and potential of our approach as measured by reasoning and planning problem hardness, the devised sub-tasks in comparison to their classical counterparts, and the performance of the entire approach against a naive LLM baseline and state-of-the-art methods.

5.1 Applicability and Potential

First, to assess the applicability and potential of our approach, the hardness of a reasoning problem used in the original ToT paper will be discussed and empirically calculated in terms of latent and estimated novelty.
As previously mentioned, research has shown that many common planning domains have a width below three when serialized. At the same time, many language-based reasoning problems can be modeled as a classical planning problem. It is plausible then that these problems also have a low width. In this thesis, the width of a popular LLM reasoning benchmark will be computed. To do this, the domain will be manually converted to a discrete state-based model. Using the IW algorithm, the effective width of the problem and a measure of the amount of prunable states can be obtained.
The first hypothesis is that many LLM reasoning problems also have a low width. The second is that a low latent width and a large amount of classically prunable latent states will lead to more pruning in our LLM-based approach.

Game of 24

The reasoning problem to be examined is Game of 24. In this domain, the solver is given four numbers $x_1, x_2, x_3, x_4 \in \mathbb{N}$ as inputs and must combine these numbers using addition, subtraction, multiplication, and division to reach the number 24. Each number and each result of a calculation may only be used once. A state in this domain is defined as the multiset of remaining, unused numbers R_i, the initial state being the first four numbers: $R_0 = \{\!\{x_1, x_2, x_3, x_4\}\!\}$. An action is the combination of two numbers in the state using one of the four basic arithmetic operations. The successor state is composed of the numbers not used in the action plus the number resulting from the operation. The goal state is one where there is only one number and that number is 24. However, this goal is currently not easily expressible using the current state features. We are missing a measure of how many numbers are left. Therefore an additional state feature is added that is equal to the size of R_i. Now, a goal state can be easily defined

using two atoms. Formally:

$$S = \{\!\{\mathbb{Q}\}\!\}^1 \times \mathbb{N} \tag{5.1}$$

$$s_0 = (\{\!\{x_1, x_2, x_3, x_4\}\!\}, 4) \tag{5.2}$$

$$s_i =: (R_i, r) \tag{5.3}$$

$$\Lambda = \{+, -, \cdot, \div\} \tag{5.4}$$

$$O = \mathbb{Q} \times \mathbb{Q} \times \Lambda \tag{5.5}$$

$$A(s_i) = A((R_i, r)) = \{(y_1, y_2, \lambda) \mid y_1 \in R_i,\ y_2 \in R_i \setminus \{y_1\},\ \lambda \in \Lambda,\ (\lambda = \div) \rightarrow (y_2 \neq 0),\ r > 1\} \tag{5.6}$$

$$f(a, s_i) = f((y_1, y_2, \lambda), (R_i, r)) = (\{y_1 \lambda y_2\} \uplus R_i \setminus \{y_1, y_2\}, r - 1) \tag{5.7}$$

$$S_G = \{\,(\{\!\{24\}\!\}, 1)\,\}. \tag{5.8}$$

The evaluation found that the Game of 24 domain has an average effective width of 1.74. There are no problem instances with a width higher than 3 and 88.2% of the states in the planning trees can be pruned on average. The distributions of these metrics can be seen in Figure 5.1. The problems had state trees made of 3698 states on average. These results show that a problem domain normally used for testing LLM reasoning has width similar to many classical planning domains.

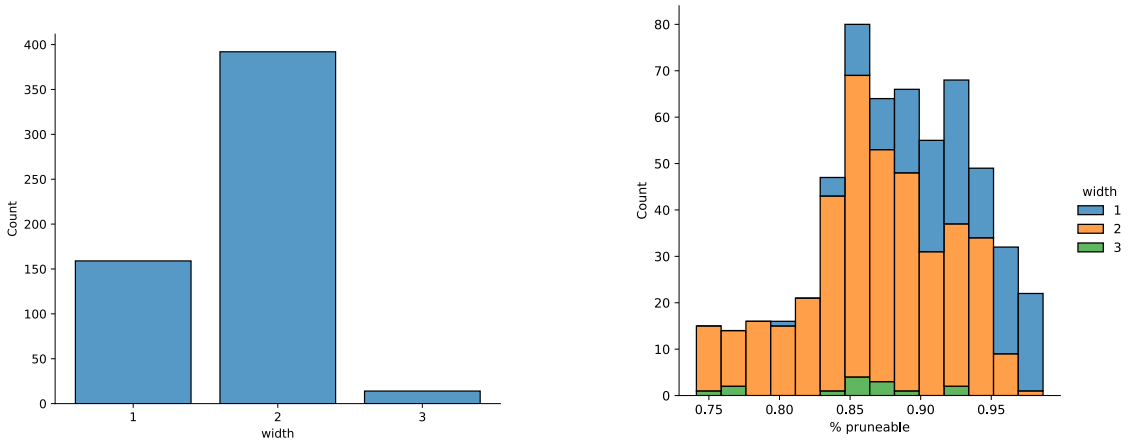

Figure 5.1: The distributions of calculated widths and percentage of pruneable states when using width-based search for all examined problem instances.

5.2 Sub-task Evaluations

The sub-tasks were evaluated using the Qwen3 14B model. It is open-source and small enough to be run locally. It is also relatively fast while still being able to compete with other state-of-the-art models of a similar size. This makes it ideal for the testing done in this paper.

[1]See Appendix A.1 for multiset notation definitions.

The testing of all four sub-tasks was done with two conditions: prompting and thinking.

Prompting can either be standard PDDL-like or translated to natural language. ”(ontable b)” becomes ”the blue block is on the table” and ”(on a c)” becomes ”the orange block is on top of the blue block”. Equivalent transformations are done for actions.
The goal is to see whether the LLM is better at dealing with structured, high information density or sparse, natural language.

The second condition is simple, the model’s thinking mode can either be turned on or off.

The results can be found in Table 5.1.

Table 5.1: Performance of LLM in Sub-tasks across prompting and thinking (NDAP: No Duplicates, All Pruned).

Sub-task	**Standard Prompting**		**Natural Language Prompting**	
	Normal	Thinking	Normal	Thinking
Action Generation	1/50	**50/50**	1/50	48/50
Action Generation Single Valid	22/50	41/50	23/50	**43/50**
Action Generation Single Optimal	12/50	33/50	12/50	**34/50**
Success State Mapping	15/50	38/50	34/50	**44/50**
Success State Mapping Separate	26/50	**47/50**	32/50	45/50
Goal State Verification	47/50	47/50	**48/50**	**48/50**
Novelty Pruning	31/50	**50/50**	26/50	**50/50**
Novelty Pruning NDAP	0/50	0/50	0/50	**1/50**
Novelty Estimation	25/50	**50/50**	28/50	49/50
Novelty Estimation NDAP	21/50	**32/50**	17/50	29/50
Problem Width Estimation	17/50	**27/50**	15/50	22/50

In most cases, thinking mode either improves upon or has similar results to its counterpart. This matches expectations, as the model was specifically trained to process inputs better using thinking tokens.

Natural language prompting also mostly improves or keeps performance. Standard PDDL prompting may perform better sometimes because it is more concise and easier to handle when working with a large number of states and atoms.

Interestingly, in thinking mode, the model performs better when asked to name possible actions in Action Generation than when asked to name the ”next action to solve the problem” in Action Generation Single. The added goal of choosing the right action leads to decreased validity scores but in return gives information on optimality.

Although the model correctly prunes in the Novelty Pruning sub-task, it completely fails at the more difficult variant. This is due to good duplicate recognition but poor other novelty understanding. When explicitly asking for the novelty in Novelty Estimation, it performs much

better. However, it is important to keep in mind that the novelty values only have few possible values. This is especially true for Problem Width Estimation where all problem instances have widths between one and three. This means the LLM does not perform very well in this task, likely because of the difficulty of the problem.

In summary, the LLM is able to complete most sub-tasks successfully, especially when configured optimally. The best results are usually achieved using natural language and thinking mode.

A iterative qualitative analysis was conducted to understand where the LLM goes wrong. One problem is the confusion of ”pick-up” and ”unstack”. According to the domain, blocks may only be picked up when they are on the table. If they are on another block they must be unstacked and cannot be picked up. The model has difficulty in deciding when to use the respective action, leading to lower performance in the Action Generation sub-task.

In an attempt to alleviate this, the context prompt was extended by an example:

```
For example, if I have a blue block on the table and a red block on top of it (
    → and clear), then I may NOT pick up the red block because it is on top of
    → another block. This means I must instead unstack it.
```

The effect of this manual prompt extension can be seen in Table 5.2. The performance increase is apparent.

Table 5.2: Performance of LLM in Action Generation Single with Prompt Extension.

Sub-task	**Standard Prompting**		**Natural Language Prompting**	
	Normal	Thinking	Normal	Thinking
Action Generation Single Valid	39/50	45/50	40/50	**50/50**
Action Generation Single Optimal	27/50	41/50	27/50	**46/50**

This case showcases the impact of prompts and the examples within them on LLM performance. In fact, even the prompts used for general sub-task evaluation prior to the extension were found by iterative testing of many different prompts. This iteration process took up much of the development time for this part of the evaluation. The prompt design procedure cannot be ignored either. Worse prompts often led to a complete collapse of performance.

5.3 Complete Approach Evaluation

Finally, the performance of the entire approach is measured on benchmarks in classical planning and LLM reasoning. The results will be compared to ToT baselines using multiple metrics. Important metrics include cost, accuracy, robustness, and adaptability. The main objectives are to:

- improve upon cost of regular ToT
- retain high accuracy similar to state-of-the-art approaches
- work towards robustness of classical methods
- make use of LLM flexibility

The main evaluations are performed in the planning domain, as that is the topic of this paper. This means problems are structured as follows: given an origin state and a set of goal atoms, a valid plan must be found that leads to a state that fulfills all goal conditions. Natural language problem statements will be used in all of the following to preserve generality.

5.3.1 Naive Baseline

The first baseline is the naive approach: simply prompting an LLM to solve the problem. The prompt will include domain-specific context, instructions to solve the problem, the origin state, and the goal.

The Blocksworld domain is used to evaluate this approach and all PDDL expressions are translated to natural language. It has five simple predicates and four actions that use up to two objects, making it a relatively simple planning domain. A given solution is accepted if it is a valid plan that reaches a goal state.

However, our main model for comparison: Qwen-14B, struggles to find valid plans, reaching the goal in only three of 50 test instances. Thinking mode improves its performance up to 23 out of 50 but it is clear that better methods are needed. For reference, OpenAI's GPT-4o-mini completes one task correctly and GPT-4o six of ten. GPT-4o's performance is likely explained by a significantly bigger number of parameters.

Table 5.3: Performance of Naive Baseline LLM.

Model	**Normal**	**Thinking**
Qwen3/14B	3/50	23/50
GPT-4o-mini	1/10	-
GPT-4o	6/10	-

5.3.2 Basic Tree-of-Thought

The second baseline is basic ToT. This approach will again be evaluated using Blocksworld.

When using the ToT approach, parameters must be given. The two most important are *max_depth* and *branch_factor*. The first constrains the maximum depth of the search tree while the latter specifies how many successors (child nodes) each state has.

This is where we run into our first problem. Due to the inherent inefficiency of ToT, it is not feasible to run many tests with a high depth. This is why the test instances were filtered to

only contain problems with solution plans of length 8 and below. This allows us to set our parameters as: $max\_depth = 8$ and $branch\_factor = 2$. The Blocksworld instances used in the following have four blocks and an average plan length of $\tilde{6}$.

In addition to these basic parameters, ToT can also be configured in diverse ways. One way, for example, is choosing whether the tree should be traversed depth-first or breath-first. A second, is deciding how to phrase the prompts. A third, whether to explicitly ask for actions and states or simply combine them into abstract steps.

This approach also suffers from the same problems as the naive baseline. Separating the problem into smaller steps does not always help the LLM understand the domain.

Additionally, repeated and recursive prompting can lead to disastrous chain reactions. One ill-fitting response can lead to bad subsequent responses. Quality can quickly decline and the model may, for example, respond by repeating the last parts of the prompt (see example in Appendix A.3). For this reason, responses that break the output length limit are checked for mass repetitions and subsequently retried with a higher frequency penalty parameter.

In fact, the base ToT approach without any examples does not solve a single Blocksworld problem instance successfully. Through qualitative analysis, we can find that one main problem is the previously mentioned difference between picking up and unstacking. The context is again extended by the example as before. This and further iterative testing of prompting configurations allowed for working prompts to be found. In the end, the final configuration was able to solve 11 of 50 problem instances without any other examples. This high variability in performance depending on the prompts highlights the fragility of LLM prompting. Activating thinking mode increases the amount of solved instances to 94%. Both methods consume large amounts of tokens with an average run using 229k or 330k tokens for non thinking and thinking respectively.

Next, explicit action-state-mapping will be evaluated. We expected this method to lead to higher performance but the opposite is the case. The new method does not solve any instances when utilizing the initial prompts and takes even longer to tune. However, when using the improved prompts, the explicit method does achieve similar although subpar performance to the previous method. A possible reason is that more prompts/queries per ToT allow more mistakes.

The results of all base ToT tests can be found in Table 5.4

Table 5.4: Performance of Base ToT.

Method	**Normal**		**Thinking**	
	Performance	Avg. Token Usage	Performance	Avg. Token Usage
Direct	11/50	229k	47/50	330k
Explicit State Action	7/50	438k	45/50	686k

5.3.3 Tree-of-Thought with Novelty Pruning

The previously formulated approaches of improving ToT by pruning with some measure of novelty will be evaluated and compared to the baseline.

Problem Width and Novelty Estimation Pruning

The instance width and integer novelty estimation approach is evaluated on the same 50 test instances used before and the results are shown in Table 5.5. Both breadth-first and depth-first search are evaluated.

Table 5.5: Performance of ToT with Width Estimation and Novelty Pruning.

Method	**Normal**		**Thinking**	
	Performance	Avg. Token Usage	Performance	Avg. Token Usage
Depth-first	1/50	40k	37/50	39k
Breadth-first	0/50	37k	38/50	80k

This method seems to rely heavily on thinking mode and solves almost no instances without. Using thinking however, yields results that starkly reduce token costs while reducing performance moderately. Additionally, this method does not generalize to domains outside of planning. This is why the rest of the evaluation will focus on boolean novelty pruning.

Boolean Novelty Pruning

Using the simple boolean novelty prompt previously discussed, results are found as shown in Table 5.6. These are then compared to base ToT in Figure 5.2 and Figure 5.3.

Table 5.6: Performance of ToT with Boolean Novelty Pruning.

Method	**Normal**		**Thinking**	
	Performance	Avg. Token Usage	Performance	Avg. Token Usage
Direct	5/50	146k	42/50	18k
Explicit State Action	5/50	1038k	33/50	43k

The success of novelty pruning seems to depend, in large part, on the previous base performance. Configurations that performed the best before, such as direct with thinking, still perform the best but use much less tokens. The same cannot be said for other configurations that sometimes even use more tokens than before due to the novelty queries. In all cases, pruning significantly reduces the number of states generated.

Using novelty pruning, the tree can also be evaluated using breadth-first search instead of depth-first search. This was not possible without pruning due to the prohibitive token/compute cost. However, this does not lead to improved performance as can be seen in Table 5.7. The

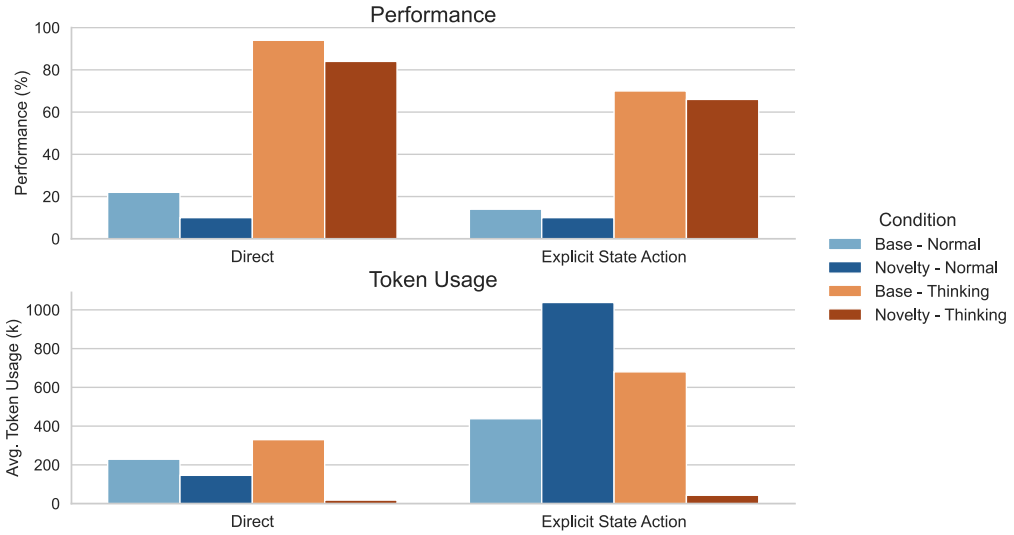

Figure 5.2: Comparison of performance and average token cost of Base ToT and Novelty Pruning with 50 test instances.

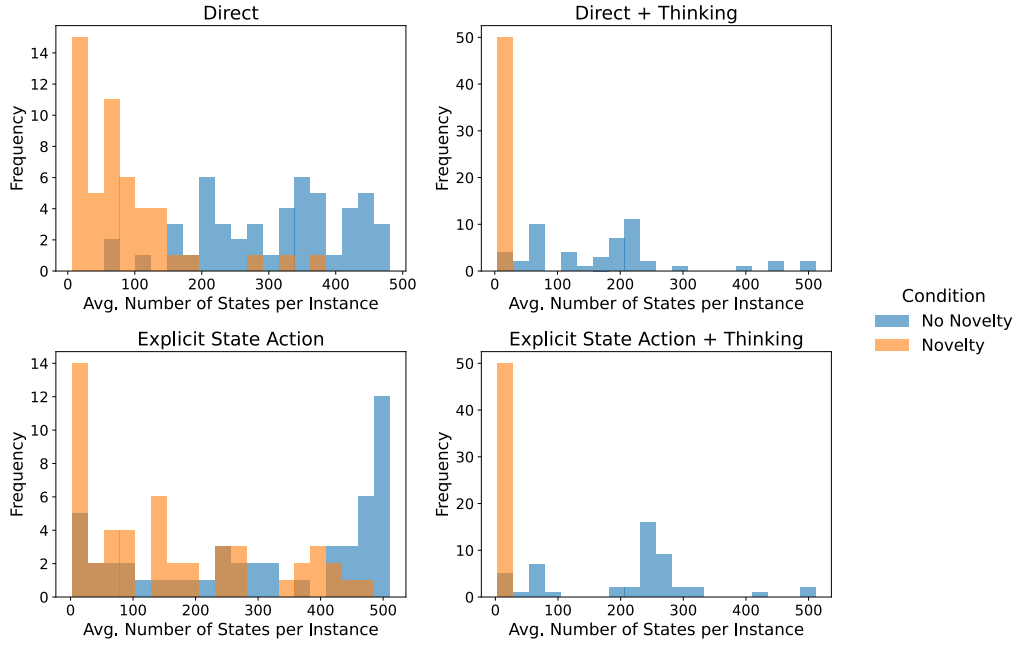

Figure 5.3: Distribution of average number of states per instance of different ToT configurations.

reason for this is most likely that breadth-first search allows more room for error since more states are generated and pruned on average. Depth-first search can quickly find solutions for easy problems without wasting compute.

All of the tests so far have been in the Blocksworld domain. To show the generalizability of the approach, the tests were run again in the also well-known Logistics domain [26]. The objective

Table 5.7: Performance of ToT using Breadth-First Search with Novelty Pruning.

Method	**Normal**		**Thinking**	
	Performance	Avg. Token Usage	Performance	Avg. Token Usage
Direct	4/50	51k	34/50	29k
Explicit State Action	5/50	829k	27/50	89k

in this domain is to bring one or more packages to their destination across cities using trucks and airplanes. The results for this domain can be found in Table 5.8.

Table 5.8: Comparison of Base vs. Novelty Pruning in Logistics Domain (ATU: Avg. Token Usage).

Method	**Normal**				**Thinking**			
	Base		Nov. Pruning		Base		Nov. Pruning	
	Perf.	ATU	Perf.	ATU	Perf.	ATU	Perf.	ATU
Direct	24/50	101k	19/50	12k	47/50	69k	48/50	9k
Explicit State Action	19/50	100k	24/50	83k	48/50	204k	24/50	43k

As visible in the data, the approach easily generalizes to this domain with some configurations performing even better than in Blocksworld. The only manual input for the domain change was the handwritten context prompt explaining the environment. No further prompt tuning was done, showing that the prompts found previously generalize to other planning domains.

To test how well the approach transfers to domains outside of planning, all tests were run on the MATH benchmark [12] using a maximum depth of 6. This benchmark contains various math problems from high school mathematics competitions and was filtered to only contain questions of the highest difficulty (Level 5). Due to the low depth, all tests could be run using depth-first and breadth-first search. Again, no prompts were changed to accommodate the new problem domain. The results can be found in Table 5.9. When not using thinking mode,

Table 5.9: Comparison of Base vs. Novelty Pruning on MATH Benchmark with Depth-First Search (DFS) and Breadth-First Search (BFS). ATU: Avg. Token Usage, ESA: Explicit State Action

Search	**Method**	**Normal**				**Thinking**			
		Base		Nov. Pruning		Base		Nov. Pruning	
		Perf.	ATU	Perf.	ATU	Perf.	ATU	Perf.	ATU
DFS	Direct	9/50	55k	8/50	128k	38/50	9k	42/50	10k
	ESA	17/50	9k	8/50	11k	36/50	12k	35/50	11k
BFS	Direct	9/50	49k	8/50	134k	49/50	15k	45/50	15k
	ESA	21/50	13k	16/50	10k	47/50	22k	46/50	21k

explicit state action mapping seems to improve performance and token costs. In reality, the

frozen prompts lead to smaller steps in reasoning for direct prompting than for explicit state action. This lead to much deeper search trees and thus higher token costs. The boolean novelty pruning seems not to change either performance or token cost much. This is not because states were not pruned, instead the pruned states were compensated by tokens used for novelty prompts. These results could feasibly be improved through changes in prompting which will not be done here.

6 Conclusion

The main goal of this thesis was to lower token costs of ToT by pruning using a LLM novelty estimator.
The potential of novelty pruning in reasoning tasks outside of classical planning was motivated through qualitative testing of a common reasoning problem. The idea is to utilize the embedded knowledge in LLMs to make use of this potential.
A general implementation was achieved that allows relatively easy experimentation and prompt tuning but requires minimal manual work when switching domains. Using an additional evaluation framework, the sub-components of the method were examined and some problems or hurdles were found. The proposed approach was then tested and compared to the baseline ToT.
Different configurations of prompt templates and LLM parameters were tested. It was quickly clear that these configurations make a big difference in performance and cost. The new method of novelty pruning was found to work as intended in configurations that performed well in the baseline.
In these cases, the model was able to estimate novelties in a way that pruned most states while still allowing the search to reach a valid solution. The implementation was even found to generalize to other problem domains.
In the other cases, the model performed much worse, solving close to zero problem instances, and average costs sometimes increased over 100%. This instability and dependency on many factors made the configuration process arduous and time-consuming. The unreliable black-box nature seems to be fundamental characteristics of current approaches that utilize repeated or recursive prompting of LLMs.
Despite these drawbacks, the advantage of adaptability to new domains and understanding of natural language should not be understated and is completely missing in many classical planning approaches.
This approach was inspired by width-based pruning in classical planning but the resulting method can be said not to have very much in common with the original. No guarantees are made to either solutions or the way novelty is calculated. But the novelty sub-task evaluation do imply that the model estimates novelty such that pruning is done similarly when configured correctly.
In total, this work described and evaluated an approach that successfully improves upon the existing ToT when well configured. In the process, the extensive evaluation provided insight into problems that hinder methods that utilize LLMs in general.

Further work could explore how goal serialization or other estimators could improve this approach or whether it would work with better base models instead of thinking.

A Appendix

A.1 Multiset Notation

A multiset is a set in which the same elements can appear multiple times. It can be created on the basis of a normal set S:

$$M = \{\!\{S\}\!\}. \tag{A.1}$$

M can now contain multiple copies of all elements in S.
Elements of a normal set can be added to a existing multiset as follows:

$$M' = S \uplus M. \tag{A.2}$$

M' now contains one more copy of every element in S than it did previously.
Subtraction is also possible:

$$M' = M \setminus S. \tag{A.3}$$

In this case, M' contains one fewer copy of each element in S, provided that at least one copy of that element was present in M.

A.2 LLM Sub-task Examples

In the following, all prompts are automatically computed, except for a static context block. This is an example for an action generation prompt in the Blocksworld domain.
The template:

```
<context>
Output exactly the next action for solving the problem. Be concise and specific.
State:
<query_state>
Goal:
<goal_atoms>
```

And the instantiation:

```
I am playing with a set of blocks where I need to arrange the blocks into stacks
    → . Here are the actions I can do:
Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block
I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear.
A block is clear if the block has no other blocks on top of it and if the block
    → is not picked up.
I can only unstack a block from on top of another block if the block I am
    → unstacking was really on  top of the other block.
I can only unstack a block from on top of another block if the block I am
    → unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being
    →  stacked.
I can only stack a block on top of another block if the block onto which I am
    → stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.
Once you stack a block on top of a second block, the second block is no longer
    → clear.

Output exactly the next action for solving the problem. Be concise and specific.

State:
the orange block is on top of the red block, the blue block is clear, the yellow
    →  block is clear, the yellow block is on the table, the hand is empty, the
    → blue block is on the table, the red block is on the table, the orange
    → block is clear

Goal:
the blue block is on top of the red block
```

and its completion:

```
unstack orange from red
```

A.3 LLM Repetition Example

This is an example where the LLM gets stuck in a repitition loop. In this case "Since the yellow block is on top of the blue block, we need to unstack the yellow block from the blue block. However, this is not possible unless the blue block is clear. Therefore, we must first ensure that the blue block is clear".
This is the beginning of the full response:

```
The current state is that the yellow block is on top of the blue block, and
  → the blue block is not clear. The goal is to have the blue block on top
  →  of the yellow block and the yellow block on top of the red block. To
  → achieve this, we must first unstack the yellow block from the blue
  → block. However, this is not possible unless the blue block is clear.
  → Since the yellow block is on top of the blue block, the blue block is
  → not clear. To make the blue block clear, we need to unstack the yellow
  →  block from it. This seems like a contradiction, but the key is that
  → the yellow block is on top of the blue block, and the blue block is
  → not clear. Therefore, we must unstack the yellow block from the blue
  → block to make the blue block clear. However, this is not possible
  → unless the blue block is clear. Therefore, we must first ensure that
  → the blue block is clear. Since the yellow block is on top of the blue
  → block, we need to unstack the yellow block from the blue block.
  → However, this is not possible unless the blue block is clear.
  → Therefore, we must first ensure that the blue block is clear. Since
  → the yellow block is on top of the blue block, we need to unstack the
  → yellow block from the blue block. However, this is not possible unless
  →  the blue block is clear. Therefore, we must first ensure that the
  → blue block is clear. Since the yellow block is on top of the blue
  → block, we need to unstack the yellow block from the blue block.
  → However, this is not possible unless the blue block is clear.
  → Therefore, we must first ensure that the blue block is clear. Since
  → the yellow block is on top of the blue block, we need to unstack the
  → yellow block from the blue block. However, this is not possible unless
  →  the blue block is clear. Therefore, we must first ensure that the
  → blue block is clear. Since the yellow block is on top of the blue
  → block, we need to unstack the yellow block from the blue block.
  → However, this is not possible unless the blue block is clear.
  → Therefore, we must first ensure that the blue block is clear. Since
  → the yellow block is on top of the blue block, we need to unstack the
  → yellow block from the blue block. However, this is not possible unless
  →  the blue block is clear. Therefore, we must first ensure that the
  → blue block is clear. Since the yellow block is on top of the blue
  → block, we need to unstack the yellow block from the blue block.
```

```
→ However, this is not possible unless the blue block is clear.
→ Therefore, we must first ensure that the blue block is clear. Since
→ the yellow block is on top of the blue block, we need to unstack the
→ yellow block from the blue block. However, this is not possible unless
→  the blue block is clear. Therefore, we must first ensure that the
→ blue block is clear. Since the yellow block is on top of the blue
→ block, we need to unstack the yellow block from the blue block.
→ However, this is not possible unless the blue block is clear.
→ Therefore, we must first ensure that the blue block is clear. Since
→ the yellow block is on top of the blue block, we need to unstack the
→ yellow block from the blue block. However, this is not possible unless
→  the blue block is clear. Therefore, we must first ensure that the
→ blue block is clear. Since the yellow block is on top of the blue
→ block, we need to unstack the yellow block from the blue block.
→ However, this is not possible unless the blue block is clear. [...]
```

List of Acronyms

CoT	Chain of Thought
GoT	Graph of Thoughts
IW	Iterative Width
LLM	large language model
PDDL	Planning Domain Definition Language
STRIPS	Stanford Research Institute Problem Solver
ToT	Tree of Thoughts

List of Figures

5.1 The distributions of calculated widths and percentage of pruneable states when using width-based search for all examined problem instances. 17
5.2 Comparison of performance and average token cost of Base ToT and Novelty Pruning with 50 test instances. 23
5.3 Distribution of average number of states per instance of different ToT configurations. 23

List of Tables

5.1 Performance of LLM in Sub-tasks across prompting and thinking (NDAP: No Duplicates, All Pruned). 18
5.2 Performance of LLM in Action Generation Single with Prompt Extension. . . . 19
5.3 Performance of Naive Baseline LLM. 20
5.4 Performance of Base ToT. 21
5.5 Performance of ToT with Width Estimation and Novelty Pruning. 22
5.6 Performance of ToT with Boolean Novelty Pruning. 22
5.7 Performance of ToT using Breadth-First Search with Novelty Pruning. 24
5.8 Comparison of Base vs. Novelty Pruning in Logistics Domain (ATU: Avg. Token Usage). 24
5.9 Comparison of Base vs. Novelty Pruning on MATH Benchmark with Depth-First Search (DFS) and Breadth-First Search (BFS). ATU: Avg. Token Usage, ESA: Explicit State Action 24

List of References

[1] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson, *et al.*, "Pddl| the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[2] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K. H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can, not as i say: Grounding language in robotic affordances," in *Proceedings of Machine Learning Research*, vol. 205, ML Research Press, 2023, pp. 287–318.

[3] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, *et al.*, "Graph of thoughts: Solving elaborate problems with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 17 682–17 690.

[4] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[5] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," vol. 15, no. 3, Mar. 2024, ISSN: 2157-6904. DOI: 10.1145/3641289. [Online]. Available: https://doi.org/10.1145/3641289.

[6] K. R. Chowdhary, "Natural language processing," in Springer India, 2020, pp. 603–649, ISBN: 978-81-322-3972-7. DOI: 10.1007/978-81-322-3972-7_19. [Online]. Available: https://doi.org/10.1007/978-81-322-3972-7_19.

[7] G. Dagan, F. Keller, and A. Lascarides, "Dynamic planning with a llm," Aug. 2023.

[8] DeepSeek-AI, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *CoRR*, vol. abs/2501.12948, Jan. 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.12948.

[9] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971, ISSN: 00043702. DOI: 10.1016/0004-3702(71)90010-5.

[10] "Gpt-4 technical report," OpenAI, Tech. Rep., Mar. 2023.

[11] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006, ISSN: 10769757. DOI: 10.1613/jair.1705.

[12] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.

[13] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 263–312, 2001, ISSN: 10769757. DOI: 10.1613/jair.855.

[14] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162, PMLR, Dec. 2022, pp. 9118–9147. [Online]. Available: https://proceedings.mlr.press/v162/huang22a.html.

[15] X. Huang, W. Liu, X. Chen, X. Wang, H. Wang, D. Lian, Y. Wang, R. Tang, and E. Chen, "Understanding the planning of llm agents: A survey," Feb. 2024.

[16] A. Jaech, A. Kalai, A. Lerer, A. Richardson, A. El-Kishky, A. Low, A. Helyar, A. Madry, A. Beutel, A. Carney, *et al.*, "Openai o1 system card," *arXiv preprint arXiv:2412.16720*, 2024.

[17] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. Saldyt, and A. Murthy, "Llms can't plan, but can help planning in llm-modulo frameworks," Feb. 2024.

[18] M. Katz, H. Kokel, K. Srinivas, and S. Sohrabi, "Thought of search: Planning with language models through the lens of efficiency," Apr. 2024.

[19] R. Kirk, I. Mediratta, C. Nalmpantis, J. Luketina, E. Hambro, E. Grefenstette, and R. Raileanu, *Understanding the effects of rlhf on llm generalisation and diversity*, 2024. arXiv: 2310.06452 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2310.06452.

[20] N. Lipovetzky, "Planning for novelty: Width-based algorithms for common problems in control, planning and reinforcement learning," Jun. 2021.

[21] N. Lipovetzky and H. Geffner, "Width and serialization of classical planning problems," in *Frontiers in Artificial Intelligence and Applications*, vol. 242, IOS Press BV, 2012, pp. 540–545. DOI: 10.3233/978-1-61499-098-7-540.

[22] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "Llm+p: Empowering large language models with optimal planning proficiency," Apr. 2023.

[23] H. Liu, R. Ning, Z. Teng, J. Liu, Q. Zhou, and Y. Zhang, "Evaluating the logical reasoning ability of chatgpt and gpt-4," Apr. 2023.

[24] H. liu, Z. Teng, R. Ning, J. Liu, Q. Zhou, and Y. Zhang, "Glore: Evaluating logical reasoning of large language models," Oct. 2023.

[25] "Livebench: A challenging, contamination-free llm benchmark," Jun. 2024.

[26] D. M. McDermott, "The 1998 ai planning systems competition," *AI Magazine*, vol. 21, no. 2, p. 35, Jun. 2000. DOI: 10.1609/aimag.v21i2.1506. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1506.

[27] S. S. Raman, V. Cohen, E. Rosen, I. Idrees, D. Paulius, and S. Tellex, "Planning with large language models via corrective re-prompting," in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. [Online]. Available: https://openreview.net/forum?id=cMDMRBe1TKs.

[28] S. P. Sharan, F. Pittaluga, V. K. B. G, and M. Chandraker, "Llm-assist: Enhancing closed-loop planning with language-based reasoning," Dec. 2023.

[29] Y. Shi and X. Hu, "Automatic prompt generation and optimization by leveraging large language models to enhance few-shot learning in biomedical tasks," in *2024 IEEE International Conference on Big Data (BigData)*, IEEE, Dec. 2024, pp. 1645–1654. DOI: 10.1109/BigData62323.2024.10825237.

[30] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2023, pp. 2998–3009.

[31] K. Stechly, K. Valmeekam, and S. Kambhampati, "Chain of thoughtlessness? an analysis of cot in planning," May 2024.

[32] Q. Team, *Qwen3 technical report*, 2025. arXiv: 2505.09388 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2505.09388.

[33] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati, "On the planning abilities of large language models - a critical investigation," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 75 993–76 005. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/efb2072a358cefb75886a315a6fcf880-Paper-Conference.pdf.

[34] K. Valmeekam, A. Olmo, M. Marquez, S. Sreedharan, and S. Kambhampati, "Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change," in *Advances in Neural Information Processing Systems*, vol. 36, Neural information processing systems foundation, 2023.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762.

[36] M. Verma, S. Bhambri, and S. Kambhampati, "On the brittle foundations of react prompting for agentic large language models," May 2024.

[37] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," Jun. 2022.

[38] J. Wei, X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 24 824–24 837. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

[39] D. E. Wilkins, *Practical planning: extending the classical AI planning paradigm*. Elsevier, 2014.

[40] Y. Wu, Y. Wang, Z. Ye, T. Du, S. Jegelka, and Y. Wang, *When more is less: Understanding chain-of-thought length in llms*, 2025. arXiv: 2502.07266 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2502.07266.

[41] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, "Translating natural language to planning goals with large-language models," Feb. 2023.

[42] 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*, vol. 36, Neural information processing systems foundation, 2023.

[43] M. Zečević, M. Willig, D. S. Dhami, and K. Kersting, "Causal parrots: Large language models may talk causality but are not causal," Aug. 2023.

[44] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," Mar. 2023.