German International University
Computer Science Faculty

Using GPTs For Hierarchical Task Network

Bachelor Thesis

Author: Farah Essam Moussa
Supervisor: Dr. John Zaki

Submission Date: 09 Jan, 2024

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowledgement has been made in the text to all other material used

---

Farah Essam Moussa
09 Jan, 2024

Acknowledgments

I am deeply grateful to my supervisor, Dr. John Zaki, for his invaluable support, insightful guidance, and encouragement throughout this journey. His patience and expertise have been invaluable, and I am truly fortunate to have had his mentorship.

I would also like to thank the German International University, whose resources and academic environment provided the foundation for this study. The support and opportunities offered by the university have been pivotal in the successful completion of this thesis.

Lastly, I owe my deepest appreciation to my family and friends, whose love, encouragement, and belief in me have been a constant source of strength. This achievement would not have been possible without their unwavering support.

Abstract

Modern industries like robotics, smart homes, logistics, and manufacturing, increasingly demand intelligent systems capable of adapting to dynamic, real-world scenarios. Task planning in these environments often involves breaking down high-level goals into manageable subtasks to achieve structured execution and clear objectives. Hierarchical Task Networks (HTNs) are widely used as they offer a systematic framework for decomposing tasks and handling hierarchies. While traditional HTNs are structured and effective for predefined workflows, they tend to face challenges in handling dynamic, real-world environments where context retention, redundancy, and adaptability are crucial. This gap highlights the need for an approach that integrates Large Language Models (LLMs) into HTNs by using reasoning and natural language capabilities to address these challenges.

This research proposes a framework that combines HTNs and LLMs with methodologies such as Chain of Thought (CoT) reasoning to improve context retention, a previous learning context mechanism to retrieve relevant past data for context loss detected tasks, and a feedback mechanism to refine task execution dynamically. In addition, the system integrates a redundancy check to ensure efficient task decomposition by detecting, handling, and preventing repeated tasks.

These methods were implemented and tested on simulated tasks modeled after real-world scenarios, mainly household chores, using different LLMs to compare their efficiency. The system achieved an average alignment score of 94% across all tasks, highlighting its ability to maintain context and efficiently decompose tasks. The execution times for complex tasks were reduced by up to 30%, showing the system's significant performance enhancements through redundancy detection and early feedback validation. The system also maintained a 100% task success rate, validating its robustness in handling tasks of different complexities.

In conclusion, the thesis provides a solid system for developing HTN planning using LLMs, addressing challenges in traditional systems and introducing a framework for intelligent systems that can handle complex, dynamic situations. These contributions provide significant advancements for future AI-driven task planning, with potential applications in robotics.

The code for this research can be found at [1].

Contents

Chapter 1

Introduction

Domestic robots are considered as the next step in simplifying our lives. From helping with routine tasks like cooking or cleaning to caring for someone, they have great potential to change the way we manage our daily lives and homes. However, these systems do not operate flawlessly without any difficulties. Although they are good at certain jobs, robots struggle to generalize and deal with the unpredictable nature of real-world situations. Everyday tasks like 'clean the kitchen' or 'get dinner ready' can become challenging because the meaning of these tasks depends on the context that robots often struggle to comprehend.

For example, the humanoid Tesla Optimus, which can assist with household chores and workspace organization. Although it offers an optimistic look at what is possible, it has a common issue found in most domestic robots, which is duties at home aren't as simple or repetitive as they may be on an assembly line. Tasks in home settings vary widely and are strongly influenced by the scenario, the user's preferences, and the surrounding environment. Sometimes, the task may vary from cleaning the kitchen entails, doing the dishes one day and wiping off the counters the next. This kind of diversity is difficult for robots like Optimus and others to understand. This leads us to the fundamental problem in robotics: how can we train robots to plan and carry out tasks in dynamic, real-world settings?

This problem was handled by the introduction of Hierarchical Task Networks (HTNs), breaking down complex tasks into smaller, more manageable sub-tasks using domain-specific knowledge. When a robot is instructed to "prepare dinner", an HTN framework divide the task into parts such as "choose a recipe", "gather ingredients", and "cook the food". This creates a clear and organized plan by further breaking down each of these components. This has worked well in environments where everything is predictable, but in a household setting where things rarely go as planned robots quickly show their limits. HTNs are flexible in rule-based situations but struggle with instructions that needs adaptation to changes. In situations like "setting the table" and the robot struggles to find clean dishes, it fails to know the next steps.

This is where artificial intelligence, especially Large Language Models (LLMs) succeeded to decode unclear instructions and completing the gaps with logical steps. The

LLM-powered robot can handle tasks like 'prepare dinner' by dynamically adapting based on the state of the environment. Since HTNs outstand at structure but struggle with flexibility, LLMs' ability for understanding and adaptation makes them an ideal solution.

This thesis investigates how the difficulties of domestic robotics might be addressed by combining LLMs with HTNs. It aims to develop systems that can manage the complexity and unpredictability of real-world activities by combining the best features of both methodologies. The goal is to develop more useful and capable robotic systems that can navigate real-world tasks.

1.1 Motivation and Objective

As artificial intelligence and Hierarchical Task Network integration grow, the challenge of creating systems that understand and manage complex tasks emerges, as system adaptation to unpredictable, real-world instructions becomes essential. These limitations reduce the system's ability to perform in dynamic scenarios where flexibility and adaptability are needed.

To address these challenges, this thesis aims to advance the development of systems that integrate LLMs into HTNs, exploring their natural language processing capabilities to enhance task planning and execution. By the integration of LLMs, the system has higher capabilities to manage unclear human instructions and translate them into structured task plans.

1.2 Research Questions and Contributions

Many researchers have explored the implementation and enhancement of HTN systems, showing advancements in task planning. However, challenges in addressing the limitations of traditional HTNs and LLMs in dynamic scenarios persist. Gaps in context retention, computational efficiency, scalability, and robustness continue to be areas of active research.

To address these challenges, this thesis focuses on the following key research questions:

- What strategies can improve context retention and task continuity in hierarchical task planning systems?
- How can computational efficiency be enhanced in LLMs-HTN frameworks while maintaining adaptability?
- What mechanisms can be developed to improve the scalability and robustness of LLMs-HTN systems in dynamic and real-time environments?

This thesis introduces a framework that integrates LLMs into HTNs, improving the decomposition of high-level commands into actionable tasks. By handling the task context and managing dynamic environments, this research aims to enhance the flexibility, adaptability of task planning systems and handling complex tasks more efficiently.

1.3 Thesis Structure

The thesis is structured into chapters that guide the reader through the stages of the research. Following the introduction in Chapter 1, which covers the background, motivation and research questions, Chapter 2 discusses the literature review, exploring the evolution of HTN planning and the integration of LLMs into HTN frameworks. Chapter 3 presents the methodology, stating the approach taken in this research, focusing on the design. Chapter 4 introduces the experimental setup and implementation incorporated in the system and the used evaluation methods to help identify insightful results of the planning process. Chapter 5 provides the results and discussion of the system. Finally, Chapter 6 concludes the study, summarizing the findings and offering insights for future work.

Chapter 2

Literature Review

Hierarchical Task Networks are an artificial intelligence framework that aims to decompose high-level tasks into more manageable sub-tasks through integration with advanced language models. HTN models have evolved by breaking tasks into structured components, improving the handling of complexity. This approach is used in fields such as robotics, decision-making tools, and automated systems. This thesis aims to refine the integration of LLMs with HTNs to enhance task decomposition by handling context retention and addressing other potential limitations found in traditional approaches.

The evolution of HTN planning allowed the growth of task decomposition, giving it greater flexibility and scalability in handling complex real-world environments. Georgievski and Aiello [2] provided a comprehensive overview of HTN planning. The main focus of their research was introducing the evolution, theoretical foundations, and applications. They categorized HTN planners into plan-based and state-based models, highlighting the challenges of reliance on domain knowledge and the need for real-world applicability.

HTNs were first explored by Erol et al. in 1994 [3]. Their research studied the complexity of HTN planning, showing that it is EXPSPACE-complete which means a class of problems requiring exponential memory under usual constraints and undecidable with non-primitive tasks. The advantage of HTNs is their ability to model complex tasks beyond STRIPS, but their main drawback is the high computational complexity.

Based on this, Nau et al. [4] introduced the SHOP2 method to help enhance HTN performance. SHOP2 introduced the idea of allowing tasks and subtasks to be partially ordered. According to the authors, this technique showed better performance compared to usual HTN planning systems. The system claimed to achieve significant improvement in solving complex planning problems by integrating temporal and metric reasoning. It achieved a 99% success rate, solving 899 out of 904 problems. However, SHOP2 is noted to have challenges with computing time compared to other planners and requires manual development of domain-specific methods.

To address these computational challenges, Ramoul et al.[5] proposed algorithm for HTN planners to improve performance by combining the expressiveness of HTN planning

with preprocessing techniques like grounding from classical planning. Their method was tested on the International Planning Competition domains, achieving a 46.8% improvement over SHOP [4] in terms of execution time. pros of the system is reducing the search space and handling scalability of the tasks for large scenarios, but the reliance on preprocessing actions increase the challenges with resources usage for dynamic applications.

Smith et al. [6] proposed an approach for recursive planning. The research used iterative planning for the Mars Exploration Rover mission, integrating automated tools with human input to handle resource limitations and adapt to increasing goals. Their approach showed flexibility but struggled with computational complexity and uncertainty in dynamic scenarios.

In addition, the methods discussed by Ghallab et al. [7] provide a foundational understanding of task planning. They explored automated planning that integrate reasoning to select and execute actions in dynamic environments. The methods included state-space search and hierarchical planning to handle task decomposition. While their approach was robust in task modeling and flexibility, it faced problems with computational complexity and scaling to dynamic tasks.

These limitations show that HTN planning requires more adaptable and flexible methods that can solve computational inefficiencies and reduce reliance on domain-specific modeling. LLMs improve HTNs by using natural language comprehension to enhance task decomposition and adaptability. This integration removes the gap between structured planning and dynamic problem-solving. The idea has been and tested using a variety of approaches.

Building on this, Huang et al. [8] introduced the integration of LLMs, HTN planning, and Codex in zero-shot planning. The method depends on decomposing high-level tasks into executable actions in environments such as Virtual Home. This is done through semantic translation and auto-regressive trajectory correction. The results claimed improvements of over 50% with Codex 12B and GPT-3 175B. However, correctness and accuracy decreased slightly compared to their initial implementations.

While Huang et al.'s [8] approach showed over 50% improvement in task performance, Wang et al. [9] developed the DEPS system, which uses LLMs to solve task planning in Minecraft. The system refines plans based on interactive feedback and a goal selector through iterative error correction. According to the authors, DEPS achieved an average success rate of 48.56% across 71 Minecraft tasks, nearly doubling the performance of prior LLM-based methods such as GPT (15.42%) and ProgPrompt (16.88%). However, it still suffered from high computational resource requirements for plan refinement.

This was later introduced in another way by Shinn et al. [10], who developed the Reflextion framework for reinforcing language agents through verbal reflection. LLMs generate the actions, evaluate outcomes, and produce self-reflective feedback in natural language. The model was tested on sequential decision-making, reasoning, and programming. The results showed success rate improvements of 11-22% across benchmarks. The framework faces challenges of extreme reliance on LLM's self-evaluation capabilities and does not guarantee success for diverse tasks.

Another approach was developed by Ahn et al. [11], who proposed the "SayCan" system, combining LLMs with robotics to enhance real-world task execution. This was done through reinforcement learning (RL). The system was evaluated on 101 robotic tasks and achieved a planning success rate of 84% and an execution success rate of 74% but with a downside of contextual errors and biases.

The LLMs were later combined with planners by Liu et al. [12], who used their capabilities to solve complex planning problems. This was done by translating natural language problems via GPT-4 into domain definition language planning and then followed by planners to find solutions. This model achieved 85-100% of most domains, but faced challenges of reliance on pre-defined descriptions of the domain.

Similarly, Joublin et al.[13] proposed CoPAL, a hierarchical architecture using LLMs for corrective planning in robot tasks. CoPAL improved plan executability from 5.7% (baseline) to 34.8% with mid-level replanning and 100% with combined planners. It achieved 87% correctness in the barman scenario and success rates of 83% (3 blocks) and 73% (5 blocks) in the blocks world experiment, outperforming other methods. The approach outstand in handling diverse feedback but faces latency and scalability challenges.

Ruan et al. [14] proposed a new aspect of HTN planning, a framework to evaluate the use of task planning and tools (TPTU) based on LLM agents. They introduced two agent types: One-step Agent (TPTU-OA), designed to handle all subtasks at once, and the Sequential Agent (TPTU-SA), that incrementally plans subtasks and includes feedback at each step. Evaluation used SQL and Python tools to assess planning and task execution across 120 task examples. TPTU-SA demonstrated flexibility and contextual understanding, achieving up to 100% accuracy. Advantages include dynamic adaptability, while limitations includes output inconsistencies and over-reliance on specific tools.

Another direction for LLMs and transformers was proposed by Zihang et al. [15], who introduced Transformer-XL to address the fixed-length context limitation. The method relies on introducing the segment-level recurrence mechanisms and relative positional encodings that allow the model to capture long-term dependencies. Their approach achieved 450% increase in dependency length compared to vanilla Transformers due to their ability to model long context efficiently. The downside of this method is the increased memory requirements.

For handling context for long decomposition processes, Zhang et al. [16] proposed FLTRNN. This framework integrates task decomposition and memory management into LLM-based planning inference to enhance faithfulness in long-horizon tasks. The approach uses language-based RNNs, Rule-CoT, and memory graph for enhanced reasoning. Experiments on the VirtualHome dataset showed a 28% improvement in success rate and 29% in faithfulness compared to regular methods. While the method claimed to improve alignment to context, it faces challenges with computational efficiency.

Across the reviewed methods, several recurring challenges emerge. Many approaches such as Zhang et al.'s FLTRNN [16], Zihang et al.'s Transformer-XL [15], face computational inefficiencies, particularly in maintaining context over long horizons. Similarly,

frameworks like SayCan system [11] and Reflextion [10] also suffer from reliance on extensive computational resources and model self-evaluation, limiting their scalability in dynamic, real-world applications. Addressing these shared limitations by having scalable, efficient solutions tailored for dynamic applications remains a critical area for further research.

Building on this, Qingyang et al. [17] introduced Memformer, which enhances memory usage by integrating an external dynamic memory and the novel memory replay backpropagation (MRBP). The model achieved a 3.2x speed improvement over Transformer-XL and used 8.1x less memory. However, this method added complexity during the training process.

To address the limitations of dynamic scenarios and complex system handling, Yao et al. [18] enhanced LLMs problem-solving by developing a framework called Tree Of Thoughts that explores multiple reasoning paths using algorithms like Depth-first and Breadth-first. This framework claimed to achieve a success rate of 74% in the "Game of 24", compared to 7.3% for input-output prompting and 4.0% for chain-of-thought prompting. This highlights its ability to handle reasoning-intensive tasks through systematic exploration and deliberate decision-making.

Building on the previous method, Another advancement in reasoning within LLMs was introduced by Wei et al. [19] through Chain of Thought (CoT) Prompting, which provides reasoning in large language models. This step-by-step reasoning approach enhanced the model's ability to break complex tasks into more understandable sub-tasks. CoT prompting showed improvement in solving tasks that require multi-step reasoning and addressing the context retention problem. For instance, PaLM 540B achieved state-of-the-art performance on GSM8K with a solve rate of 74%, compared to 18% using standard prompting, highlighting its effectiveness in multi-step arithmetic reasoning. CoT prompting also improved commonsense reasoning tasks, with a 75.6% accuracy on StrategyQA. Despite these improvements, there is no guarantee of correct reasoning paths, which can lead to both correct and incorrect answers.

Another approach to reasoning through task planning was introduced by Kojima et al. [20]. They investigated the reasoning capabilities of large language models (LLMs) using a zero-shot Chain of Thought (Zero-shot-CoT) prompting method. The method adds the phrase "Let's think step by step" before each question, they claimed to improve performance on arithmetic and logical reasoning tasks, achieving accuracy gains such as 78.7% on MultiArith and 40.7% on GSM8K. The advantage of their approach is its simplicity, enabling multi-step reasoning without requiring few-shot examples. The disadvantage is the over-explanation and inefficiency when reasoning steps unnecessarily mislead the answer.

Suzgun et al. [21] utilized the use of Chain-of-Thought (CoT) prompting to enhance the performance of planners on BIG-Bench Hard (BBH) tasks for large language models (LLMs). CoT helps the model to generate intermediate reasoning steps before giving the final answer, this is achieved by adding instructions in the prompt to guide the process. The method was used on models like Codex and PaLM, achieving significant

improvements, with Codex outperforming the average human rater on 17 of 23 tasks, with arithmetic accuracy rising from 9.7% to 53.2%. The advantage of their approach is its ability to enable multi-step reasoning, while the disadvantage is that CoT prompting requires sufficient model scale to show improvements and struggles with tasks requiring specific world knowledge.

In addition to the papers on task planning, Marwan Abdelrahim's thesis [22], provides valuable information into integrating GPT-4 with HTNs. The method included the recursive breakdown of tasks into sub-tasks, using incremental decomposition and initial context, and introducing a vector database for storage, which improved task execution efficiency in dynamic environments. The thesis also highlighted limitations, such as real-world adaptability problems, context retention, and biases in task selection. Marwan's future work emphasized the need for feedback integration, highlighting the importance of incorporating mechanisms that allow for user interaction and validation during task decomposition. This aligns with advancements in feedback-driven HTN planning discussed in other works.

A different aspect of enhancing Hierarchical Task Networks (HTNs) involves ensuring accurate alignment between subtasks and overarching goals, either through semantic search or using it in evaluation techniques. The techniques relies heavily on robust semantic understanding and similarity checking.

Reimers and Gurevych [23] made an advancement in the field of natural language processing by creating Sentence-BERT, a modification of the BERT model that incorporates the use of a siamese and triplet network structure to provide semantically rich sentence representations. This allows it to measure the similarity between two sentences using cosine distance, speeding up the processing of tasks such as semantic similarity search or clustering. This method reduces the processing time for similarity tasks from 65 hours to 5 seconds while maintaining high accuracy, outperforming models like InferSent and Universal Sentence Encoder. SBERT's primary strength lies in efficiency and speed when comparing large amounts of data for similarities. However, in cross-domain evaluations, where domain-specific tailoring would be advantageous, its performance tends to decrease.

In addition, on NLP techniques, Schubert and Pelletier [24] developed a system to parse everyday English sentences into usable logical forms using simplified semantic rules for better understanding. Their recursive descent and left-corner parsers showcased practical implementation. While the method supports better natural language understanding and clarity by achieving simplicity and extensibility 85%, it risks oversimplifying complex semantic contexts, achieving only 70% for semantic coverage.

Tellex et al. [25] introduced Generalized Grounding Graphs (G3), to help ease natural language commands for robots performing tasks that need navigation and manipulation in semi-structured environments. The model achieves an accuracy of 86% in aligning commands to robot actions by dynamically constructing probabilistic graphical models based on semantic commands. However, the model requires extensive training data and computational demands in complex scenarios.

In addition to advancements in task decomposition and reasoning, the integration of feedback mechanisms in HTN planning has also shown advantages in handling context loss, adjusting to plans dynamically by allowing user engagement in the process of decomposition.

This was introduced by Boyi Li et al. [26] as they proposed an Interactive Task Planning (ITP) framework using GPT-4 for real-time re-planning based on user feedback. The system integrates high-level planning with low-level execution, enabling tasks such as drink preparation to adapt dynamically to changing user inputs during execution. It is robust in handling goals without extensive prompt pre-training. However, system relies heavily on guidelines and visual precision in complex scenarios.

Building on feedback algorithms, Song et al. [27] developed LLM-Planner which is a few-shot planning in embodied agents using LLMs. The system dynamically modifies plans according to environmental feedback, scoring a high-level planning accuracy of 33.81% and a goal-condition success rate of 29.88% on the ALFRED dataset. The approach demonstrated adaptability and efficiency in diverse environments, while limitations lie in low-level controller performance.

Luan et al.[28] enhanced robot task planning by utilizing a multi-layer architecture that combines LLMs and Visual Language Models. The system employs semantic alignment and heatmaps for fine-grained planning. The results showed a task execution success rate improvement from 32% to 78% after iterative feedback.

Another approach on the feedback mechanism was proposed by Appelgren and Lascarides [29], who investigated interactive task learning through corrective feedback in tasks involving building rule-compliant towers in a simulated environment with the help of an agent. Their method used natural language feedback to correct actions, by utilizing a sequence-to-sequence model to process corrective feedback and update task constraints, achieving over 85% compliance with previously unknown rules during task execution. The method relies on accurate language processing and the noisy feedback environment.

In summary, the integration of LLMs with HTN planning has significantly enhanced task decomposition in complex environments. From the approaches discussed, some showed improvements in accuracy, adaptability, and scalability. However, several challenges remain, including computational efficiency and resource demands. Other techniques address issues in maintaining context throughout the task decomposition, the generation of redundant tasks, and adapting in real-time. Some methods highlighted the need for feedback mechanisms to allow user interaction and validation. This thesis introduces a novel approach to addressing key gaps identified in previous work, specifically addressing context retention and dynamic adaptability.

Chapter 3

Methodology

This section presents the methodology used to enhance and address the challenges in integrating the principles of HTN and LLMs. The proposed methods increase the chances of LLMs and HTN solving complex tasks and provide a reliable, human-like process for execution, building on well-established techniques in AI task decomposition [22].

3.1 Theoretical Foundations

The system design is based on the principles of HTN and LLMs. HTNs, as discussed in [2], provide a hierarchical approach to task planning by breaking down high-level goals into executable subtasks. Large Language Models, such as GPT-4 and Groq, utilize advanced contextual understanding and reasoning capabilities. By combining these concepts, this research introduces mechanisms like Chain of Thought (CoT) reasoning [19] to address known limitations, such as context loss and redundancy [22].

3.2 System Overview

Domestic robots function as intelligent and adaptable assistants that fit in with daily routines. However, the complexity of the environments requires robots to plan tasks in a sequence that mirrors human problem-solving. The purpose of the HTN Planner is to break down complex tasks into smaller, more manageable sub-tasks that a robot can execute effectively.

The HTN technique enables robots to perform activities by breaking down high-level directions like "make dinner" or "clean the living room" into logical sequences of smaller tasks as "gather ingredients", "wipe surfaces", or "vacuum floors". This helps robot perform plans that frequently require contextual awareness and adaptability.

Drawing on the approaches from [2], [22], [19], [26], and [23], the system refines these techniques by inspiring and adapting them to the system to address challenges identified

in previous research [22]. By utilizing these advancements, the system enhances the planning process of domestic robots.

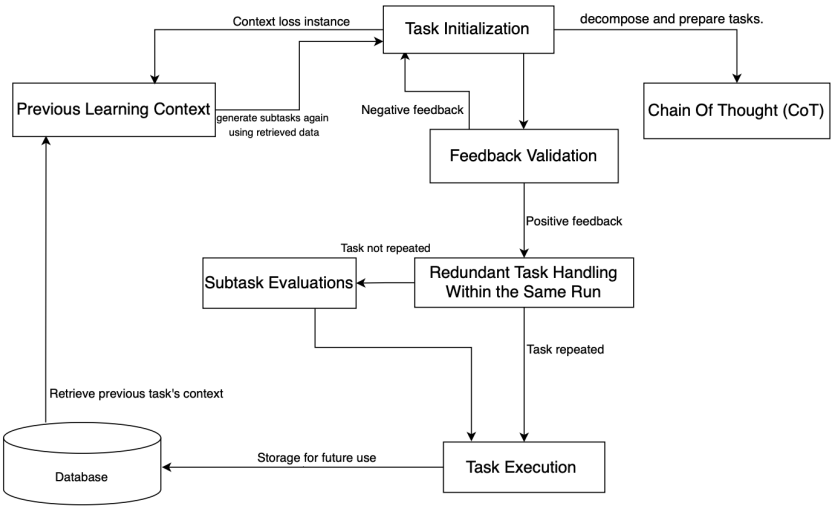

Figure 3.1: High-level System Overview

Figure 3.1 illustrates the high-level architecture of the system, showing the integration of HTN and LLMs with the enhanced methods for seamless task planning and execution.

3.3 Initial System Setup

The initial system is designed and developed based on Marwan's previous research [22]. Using the existing repository as a starting point Exisiting Code. Significant advancements are made to enhance the system's capabilities, integrating advanced methodologies and innovative approaches to address the limitations identified in earlier work, including challenges with context retention, computational inefficiencies, and adaptability in dynamic environments. A series of tests are conducted to explore and evaluate the system capabilities.

3.4 Context Retention Handling

One significant challenge identified in recent papers is the issue of maintaining task alignment, where the system struggles to maintain relevant information across multiple levels of task decomposition. This limitation was present in the repository, as it often failed to effectively link subtasks back to the overreaching goal when dealing with long and complex task sequence.

This limitation is addressed by introducing the **Chain of Thought (CoT) method** [19]. The Chain of Thought is integrated with HTN and LLM by adding reasoning to the prompts generated by the LLMs. This helped maintain context over long sequences of decomposition and provided a more granular breakdown of each task.

CoT algorithm is designed as follows as shown in Figure 3.2 to enhance task planning and ensure robust context alignment throughout the process:

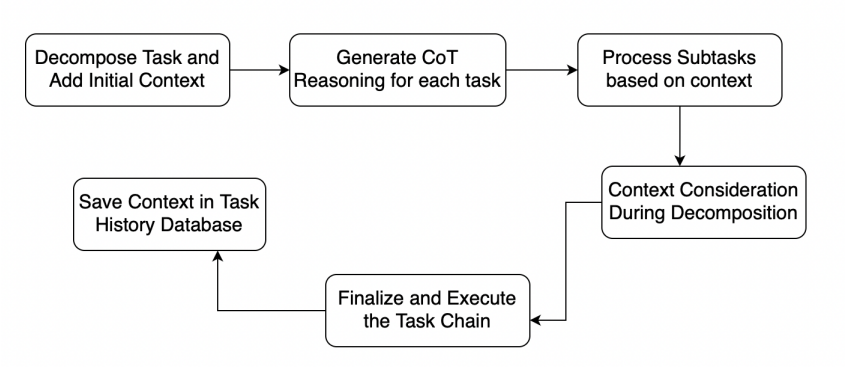

Figure 3.2: Chain of Thought Process

- Initialize Task and Context Setup: For each task, an initial context is set up, capturing details from the initial state provided to the task. This context will provide LLMs prompts with background information to guide the reasoning process.
- Generate CoT Prompt for Reasoning: The interaction with the LLM uses prompts designed to generate the "thought" processes for the current steps. The AI's response is added to the task's context, allowing future steps to build on these intermediate thoughts.
- Produce Step-by-Step Reasoning: The LLM responds to the prompt by providing an intermediate breakdown of steps or "thoughts" necessary to complete the task. This reasoning creates a clear sequence of actions that align with the overarching task context.
- Identify and Process Subtasks: The system repeats the CoT process for any subtasks generated during decomposition by adding a new prompt with the updated context. This recursive approach ensures that each subtask considers its parent task's reasoning, preserving the overall hierarchy.
- Context Consideration During Decomposition: The preserved context is included in the prompt for each subtask, ensuring every subtask aligns with the main goal. By feeding this context in a sequential manner, the LLM generate relevant subtasks, avoiding context loss or unnecessary diversions.

- Store Full Context in Task History Database: After completing the reasoning steps, the entire context (including intermediate thoughts) is saved to a task history database. This database serves as a record of previous tasks and their corresponding reasoning, providing a reliable source of "previous learning contexts" for future reference.
- Finalize and Execute the Task Chain: Once all reasoning steps are complete, the system outputs a well-defined task chain for the HTN Planner, detailing each subtask with its respective context. This organized output ensures the planner has all necessary information for task execution, improving effectiveness in real-time scenarios and allowing the user to have explanation of the decision that was taken by the planner throughout the process.

Addressing Hypotheses with CoT Integration

The integration of Chain of Thought addresses the hypothesis that adding structured reasoning improves context retention, adaptability, and task clarity. CoT ensure alignment with overarching objectives, reducing information loss across decomposition levels. It handles unclear scenarios effectively, by generating intermediate steps, providing explanation of the decisions to the system and the user through the process.

3.5 Redundant Tasks Handling

The system includes a function that aims to identify whether a task under evaluation resembles tasks already executed in the current plan execution. This ensures that redundant tasks are handled without extra checks and reducing system execution time.

Redundant Task Architecture

The redundancy detection mechanism is used to find tasks that are very similar to those that have already been completed during the current run. Using a predetermined similarity threshold, this system assesses how similar a task is to activities that have already been completed. The system makes sure that repetitive jobs are identified and managed effectively without unneeded reevaluations.

Detecting and Handling repeated tasks

Building on the method implemented in the previous step, the system uses the similarity score to identify task as repetitive or not. If the task is detected as repeated the following takes place.

- **Efficient Execution:** A task will be determined as repetitive if it is already executed before within the same run and will be performed without additional checks for granularity, primitiveness, or executability. This way, there are no useless computations, lowering the computational overhead, duration, and complexity of execution.
- **Avoiding Redundancy in Subtask Generation:** Any generated tasks within the run whether it is completed or not is used in the method that generates subtasks to avoid creating the same tasks after decomposition. With the introduction of the list to the prompt, an unnecessary repetition of tasks is avoided, enabling effective generation and execution of tasks. This approach balances efficiency in terms of time and resources.

Through these mechanisms, the system validates the hypothesis that eliminating unnecessary computations would reduce the execution time without compromising accuracy. By avoiding redundant subtasks and using previously generated tasks, the system ensures efficient use of computational resources.

3.6 Utilizing Previous Learning Context

While redundancy handling ensures efficient task execution, maintaining continuity across tasks requires integrating prior learning. The system incorporates a 'previous learning context' mechanism to address this need. The flow of the mechanism is shown in Figure 3.3

The similarity thresholds used in this research (0.7 and 0.9) are chosen based on empirical observations and literature-backed standards. A lower threshold (0.7) is initially used to align with existing methods [22], allowing flexibility for complex tasks. The refined threshold (0.9) is implemented to improve alignment accuracy and consistency, particularly for simpler tasks where deviations could lead to inefficient planning. These thresholds are tested across multiple runs to ensure balance between precision and adaptability.

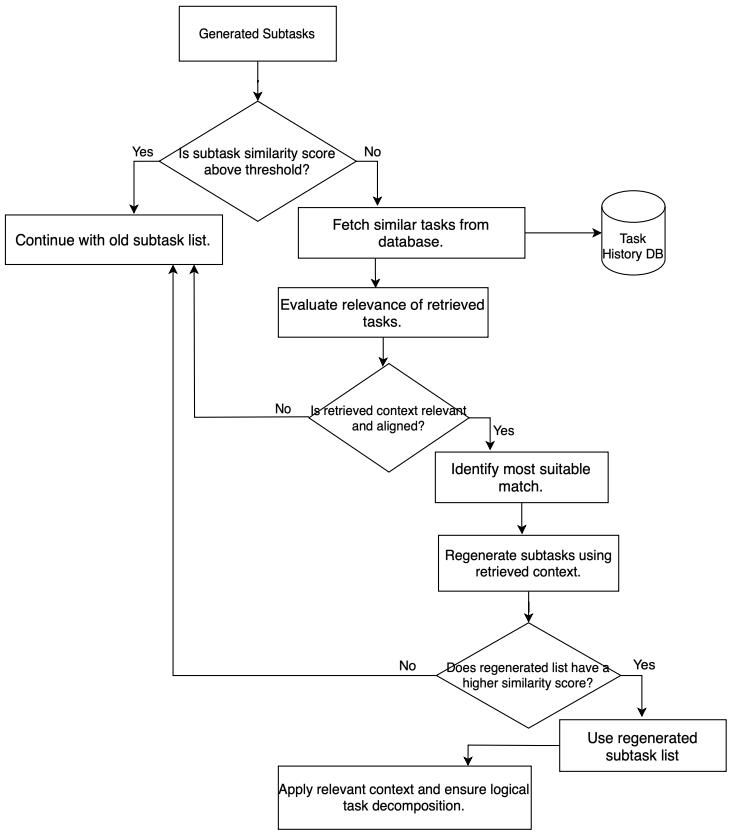

Figure 3.3: Previous Learning Context Flow Diagram

1. **Subtask Similarity Evaluation:**
   - The process begins by comparing each subtask to its overarching goal using the similarity score.
   - The similarity score indicates the alignment between subtasks and goals to ensure consistency and context alignment throughout the execution process.
2. **Context Loss Detection:**
   - If a subtask's similarity score falls below the threshold:
     - The system retrieves similar tasks from the database based on the name of the task and the goal.
     - The retrieved similar task contexts are evaluated to determine their relevance in retaining context over the decomposition process.
3. **Context Relevance Verification:**
   - The retrieved contexts are compared to the detected context loss instance to ensure they are relevant to the subtask and to the current decomposition goal.

4. **Candidate Selection for Context Application:**
    - Among the retrieved candidates, the system evaluates the context to identify the most suitable match. The chosen candidate must provide insightful and aligned context while addressing the specific detected context loss effectively.
5. **Subtask List Regeneration:**
    - After identifying the most suitable candidate, the system regenerates the subtasks from the point of context loss.
    - This regeneration process incorporates the retrieved context into the decomposition prompt to:
        - Give a more logical continuation of the task decomposition process.
        - Preserve context more effectively for long or complex tasks.
        - Reduce the number of context loss instance through the execution.
6. **Subtask List Validation:**
    - The initial similarity score of the old subtask list is compared to the similarity score of the new regenerated subtask list to decide which list to use:
        - If the **initial subtask list score** is higher, the system retains the old subtask list and continues in its regular plan.
        - If the **new subtask list score** is higher, the system replaces the old list with the regenerated subtasks.
7. **Context Loss Handling and Fail-Safe Mechanism:**
    - This methodology minimizes the chances of system failure or the generation of inconsistent subtasks which causes context loss for the system. This is achieved by ensuring the most relevant context is applied, maintaining logical and accurate task decomposition throughout the process.

3.7 Feedback Fallback Mechanism

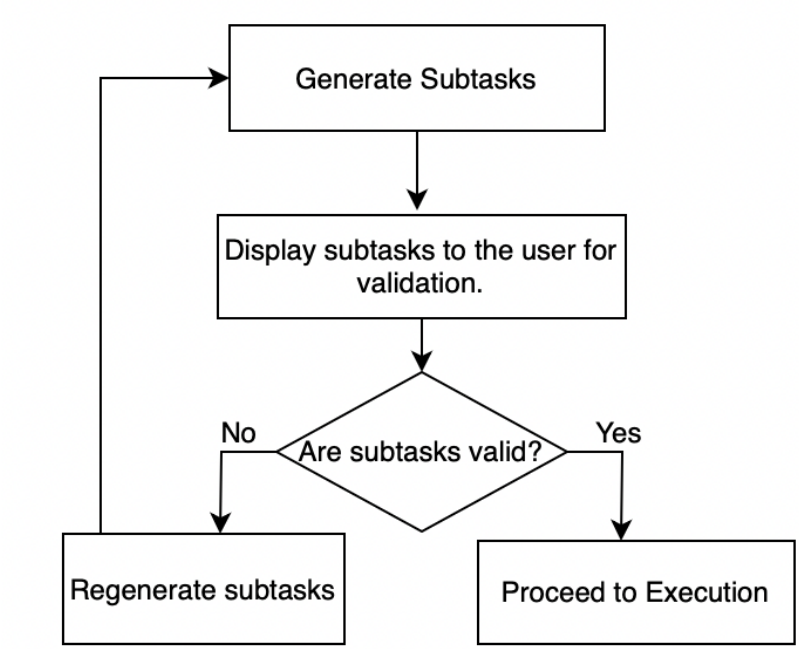

Figure 3.4: User Feedback Flow

The Feedback Fallback Mechanism is integrated to the system to improve the system's flexibility. The method enhances the accuracy and relevance of the generated subtasks without going through excessive decompositions or extra stages, this method serves as a user interaction feature that adds a validation step by giving the ability to shape the process of task planning. This technique minimizes errors by detecting and correcting them immediately. The following describes the logical flow of the used method:

As shown in Figure 3.4, the Feedback Fallback Mechanism introduces a structured validation step within the task decomposition process to improve the system's adaptability and accuracy. After the system generates the decomposed subtasks for the task, it pauses and waits for user validation to ensure the subtasks are correctly aligned with the task requirements. Users evaluate the generated subtasks, indicating if the decomposition is accurate or requires refinement.

- If the feedback validates the subtasks' alignment, the system proceeds with execution.
- if the user rejects the subtasks, the system uses the feedback to refine and regenerate the subtasks, ensuring closer alignment with the task's overarching goals.

This iterative feedback loop enhances the system's ability to handle dynamic tasks by continuously integrating user input.

3.8 System Flow

As seen in Figure 3.5, the system begins with a high-level task that should be achieved by the robot. This task serves as the root for hierarchical decomposition and execution. For tasks requiring decomposition, the system uses LLMs to generate subtasks that align with the overarching goal. To avoid redundancy, the function is provided with a list of all the generated subtasks as guidance. Also, as part of the process, the system determines if the subtask being handled is similar to any previously completed tasks in the current run session. If a similar task is detected, it is executed directly without going through additional checks, otherwise the system moves forward with the regular decomposition process.

The system pauses after the subtasks are generated for user validation through the feedback. If the user feedback is negative, the subtasks are regenerated to better satisfy the user's goal. The list of verified subtasks serves as a guide for the system to learn from ongoing processes and prevent duplication in subsequent tasks.

For tasks detected as context loss, the system retrieves previous similar tasks from the database. The previous task contexts are evaluated and applied to the present task to ensure continuity and reduce redundancy. Lastly, utilizing CoT logic, the subtasks are executed and generated in a sequential manner, guaranteeing seamless transitions and preserving alignment with the initial objective.

Figure 3.5: Flow diagram

Methodology	**Purpose**	**Impact**
Chain of Thought (CoT)	Enhance context retention during task decomposition	Reduces context loss, improves task clarity
Redundancy Detection	Avoid repetition of tasks across the same run	Increases efficiency, reduces execution time
Previous Learning Context	Leverage past task contexts for complex decomposition	Maintains continuity, reduces inconsistencies
Feedback Fallback Mechanism	Validate subtasks in real-time	Improves task alignment, corrects errors

Table 3.1: Methodological Contributions and Their Impact

The metrics used for evaluating the system's performance were chosen based on previous standards in task decomposition and planning. Similarity scores, as described in [23], are widely used to measure the alignment between tasks and goals, providing a quantitative measure of context retention. Execution time, a critical metric in [2], evaluates the system's computational efficiency and adaptability in real-world scenarios.

To ensure comprehensive evaluation of the system's capabilities and adaptability, multiple LLMs models were incorporated into the process. The selected models include:

- **Groq (llama3-70b-8192):** The main model used for the development process. The model is Known for its ability to handle complex reasoning tasks, outstanding in generating structured outputs for hierarchical planning.
- **ChatGPT (gpt-4):** With its enhanced contextual understanding and reasoning capabilities, GPT-4 ensures accurate and coherent task decomposition.
- **Claude (claude-3.5 sonnet):** Designed for structured outputs and logical reasoning, Claude provides robust support for complex workflows, making it a reliable choice for challenging tasks.
- **Gemini (Gemini Pro):** Designed to handle diverse tasks that require context-aware reasoning, Gemini effectively supports scenarios involving dynamic capabilities.

The models are selected due to their diverse architectures, high performance, and structured outputs.

Chapter 4

Experimental Setup

In this chapter, the experimental setup shows the implementation of the methods and the designed metrics to evaluate the system's performance across a range of different tasks. The framework focuses on enhancing the system's ability to decompose tasks, retain context, and increase accuracy through different functionalities. In addition, the evaluation techniques highlights the scoring of system accuracy, efficiency, and task alignment. The purpose of these tests is to showcase the system's strengths and limitations, offering insights into future work.

4.1 System Implementation

Building on the foundation established in Marwan's thesis [22], the system introduces novel approaches for integrating HTN with LLMs. The following steps outline the components implemented to ensure seamless integration, improved adaptability, and robust evaluation of the framework.

- System Configuration: Installing all necessary dependencies required for the system to function including package installations through pip tool for Python libraries like transformers, torch, numpy, and configuring environment variables for secure integration.
- Prompt Engineering and Optimization: Significant effort is directed toward enhancing the prompts provided to LLMs to improve its adaptability to different, dynamic scenarios. This involved using structured prompts that integrate clear instructions for the tasks. Instead of using a generic instruction like:

```
"decompose the task into a detailed step-by-step plan"
```

a revised prompt is used:

```
"Keep in mind that: You are a robot trying to achieve the goal,
↪  give tasks that mimic the robot behavior.

- Decompose the task into a step-by-step plan that are detailed
↪  but NOT overly decomposed.

- Each subtask must be distinct, new, and necessary to achieve
↪  the main task.

- Provide ONLY the subtasks as a Python list of strings, without
↪  any additional text or explanations.

"
```

- API Configuration and Model Integration: The system setup involved configuring APIs for Groq, GPT-4, Gemini, and Claude. Each model was integrated with different API access keys. Parameters as token limits and response times were standardized to ensure fair comparisons."

4.1.1 Chain Of Thought

The Chain of Thought (CoT) reasoning improves task breakdown by integrating logical and actionable reasoning into the flow of the system. The system uses a large language model to generate reasoning steps for tasks, which are stored and used as guidance for the decomposition process.

Reasoning Process

The CoT function consists of two main methods: reason_through_task and add_reasoning. These methods allow the system to dynamically generate and store reasoning steps for tasks:

1. reason_through_task: This function generates reasoning prompt based on the current task and its context. The goal of the prompt is to get the LLM to generate the thought process for achieving a task while considering the robot's capabilities. For instance, a task such as "Turn on the lights" would generate a prompt like:

```
Generate a concise, actionable reasoning for the task 'Turn on the
↪  lights'
based on the current context: 'The switch is located near the
↪  door'.
Ensure the reasoning is specific, insightful, and beneficial to
↪  guide the next steps.
```

2. add_reasoning: The reasoning is added to the task's context directory that contains the subtask name and its context. This ensures that all reasoning steps are available for reference throughout the process.

Example for the reasoning Produced

For the task "Turn on the Lights", the initial reasoning steps produced are concise and help guide the decomposition of the task:

```
**Reasoning:**
To complete the task, I will focus on accessing the light switch in the
↪  specified room. Since the switch is a physical entity, I will
↪  prioritize navigating to its location and then execute the flipping
↪  action. This involves detecting the switchs position, avoiding any
↪  obstacles, and applying the necessary force to toggle the switch. By
↪  successfully flipping the switch, I will achieve the task of turning
↪  on the lights within the specified room.
```

The context is added to the decomposition process to ensure task alignment with the overreaching goal. This reasoning allows clear understanding of the task requirements and constraints, avoiding unnecessary subtasks and ensuring context retention throughout the system.

```
def get_subtasks(task, state, remaining_decompositions,
↪  capabilities_input,
current_run_tasks, context, generated_subtasks):

    prompt = f\"\"\"
    You are a robot tasked with achieving the goal by executing
    **sequential steps** in a structured
    and efficient manner. Your goal is to decompose
    the task into primitive, actionable subtasks
    that strictly adhere to the provided capabilities.
    Consider the context: '{context}' to guide the decomposition
    ↪  process.
    Focus **strictly** on the **critical and unique steps**
    necessary to achieve the task's goal while
    avoiding unnecessary, overly detailed, or redundant subtasks.
    """
```

4.1.2 Redundancy Detection

Input Parameters:

- task_name: The name of the task currently being evaluated.
- current_run_tasks: A list of task names that have been executed and completed in the current run.
- threshold: A similarity threshold set to (0.97) based on trials and errors that determines whether tasks are considered similar. Tasks exceeding this threshold are flagged as redundant.

Logic:

The function iteratively compares task_name with each task in current_run_tasks to determine similarity. The similarity is calculated using a pre-trained large language model (LLM), which returns a numerical score between 0 and 1.

Similarity Evaluation

1. For each task in current_run_tasks, the function constructs the prompt and sends it to the LLM as an API call.
2. The LLM's response, a similarity score, is parsed into a floating-point value.
3. If the similarity score exceeds the "threshold", the function identifies the task as redundant and moves to immediate execution.

Prompt Design:

The following prompt is sent to the LLM to compute the similarity score:

```
prompt = f\"\"\"
Compare the following two tasks and determine their similarity on a scale
from 0 to 1 with a precision of up to three decimal places:

- Task 1: "{task_name}"
- Task 2: "{run_task}"

Only provide the similarity score as a number with up to three decimal
↪  places.
"""
```

This prompt ensures clarity in the inputs and enforces the desired output format, avoiding unclear explanations or unnecessary details.

Handling repeated tasks

Building on the is_task_similar method implemented in the previous step, the system uses the similarity score to identify task as repetitive or not. If the task is detected as repeated the following takes place.

- A task will be determined as repetitive if exact task or a similar one that performs the same steps is already in the current_run_tasks list during the planning process. The task will go directly to the execute_task and compute directly without undergoing the evalutaion steps.
- The generated_tasks list is used in the get_subtasks method to avoid unnecessary repetition of tasks, enabling effective generation and execution of tasks in the entire system. This approach balances efficiency in terms of time and resources.

```
if is_task_similar(subtask_node.task_name, self.current_run_tasks,
self.repetition_count):
    updated_state = self.execute_task(self.initial_state, subtask_node)
    subtask_node.status = "completed"
    continue
```

4.1.3 Utilizing Previous Learning Context

The Previous Learning Context Functionality is designed to address context loss during task decomposition. Using task history data, similarity-based evaluation, and dynamic context regeneration, the system ensures coherence and logical alignment of tasks even in complex scenarios.

Subtask Similarity Evaluation

The similarity evaluation process ensures that subtasks align closely with their parent task and overarching goal. The similarity score, calculated using cosine similarity on embeddings as mentioned in 4.4, provides a quantifiable measure of alignment.

```
def calculate_similarity(parent_task, subtask, goal_embedding=None):
    contextualized_subtask = f"{parent_task}: {subtask}"
    parent_embedding = model.encode(parent_task, convert_to_tensor=True)
    ↪ if goal_embedding is None else goal_embedding
    subtask_embedding = model.encode(contextualized_subtask,
    ↪ convert_to_tensor=True)
    similarity_score = util.pytorch_cos_sim(parent_embedding,
    ↪ subtask_embedding).item()
    return similarity_score
```

By embedding subtasks in the context of their parent task, the system ensures that the similarity score reflects both semantic and contextual alignment.

Context Loss Detection and Retrieval

When a subtask's similarity score falls below the threshold, the system flags it as a context loss instance. The system dynamically retrieves similar tasks from the task_history_database and evaluates their contexts for relevance.

```
for index, subtask in enumerate(subtasks_list):
    score = calculate_similarity(task, subtask)
    if score < threshold:
        context_applied =
        ↪ self.retrieve_and_apply_previous_context(subtask, task_node)
        if context_applied:
            print(f"Context applied. Regenerating subtasks...")
            break
```

Retrieve and Apply Previous Context Pseudo Code:

```
Algorithm: retrieve_and_apply_previous_context
Input: task_name (String), repeated_task_node (Object), max_context_size
↪  (Int), similarity_threshold (Float), fallback_threshold (Float)
Output: Boolean (True if context applied, False otherwise)

1. Retrieve tasks with a similar name from the database.
2. If no tasks found, exit.
3. Filter retrieved tasks by:
       - Matching task_name and goal with a similarity_score more than or
       ↪  equal the similarity_threshold.
4. Sort similar tasks by descending similarity score.
5. For each candidate task:
      a. Extract reasoning or relevant context.
      b. Merge retrieved context with the repeated tasks context.
6. Apply merged context to the repeated task if successful.
7. If no suitable context is found, exit.
```

Regenerating Subtasks with Updated Context

Once a suitable context is identified, the system regenerates the subtask list from the index of context loss using the updated context. This regeneration process ensures logical task decomposition by aligning the subtasks with the retrieved context.

The subtask regeneration prompt incorporates detailed guidelines for leveraging the updated context:

```
def get_subtasks_with_context(task, state, remaining_decompositions,
↪  capabilities_input, context, current_run_tasks, generated_subtasks):
    prompt = f\"\"\"
        You are a robot tasked with achieving the goal of decomposing
        ↪  tasks into actionable subtasks using the provided updated
        ↪  context.
        **Guidelines for Subtask Generation:**
        1.Leverage Updated Context:
        - Ensure the generated subtasks utilize and align with the
        ↪  provided updated context.
        - Prioritize actions that directly address the insights and
        ↪  requirements specified in the context.
    """
    # Further processing logic follows
```

Validating Regenerated Subtasks

To ensure the reliability of the regenerated subtasks, the system compares the similarity scores of the old and new lists:

- If the new list demonstrates better alignment (higher similarity scores), it replaces the old list.
- Otherwise, the original subtask list is retained.

```
if regenerated_avg_similarity_score >
↪  initial_avg_similarity_score:
    subtasks_list = updated_subtasks
    self.successful_context_applications = getattr(self,
    ↪  "successful_context_applications", 0) + 1
else:
subtasks_list = subtasks_list
```

4.1.4 Feedback Fallback Function

1. Subtask Generation: The system generates a list of subtasks for a given task using the decomposition logic.
2. User Feedback Prompt:
   - After generating the subtask list, the system halts and prompts the user with a validation query:

     Are the generated subtasks for "{task_node.task_name}" accurate and aligned with the task requirements? (yes/no):
   - This query ensures that the user can evaluate the correctness of the subtasks.
3. Feedback Handling:
   - If the user confirms with responding "yes", the system proceeds with the next steps of execution or further decomposition.
   - If the user rejects with responding "no", the system regenerates the subtasks by recursively invoking the decomposition logic again to try generating another aligned subtasks.
4. Integration Points: The feedback logic was added immediately after subtask generation, ensuring the system pauses before proceeding to validate the generated subtasks. Ensuring the feedback function is called with every decomposition.

```
def feedback_fallback(self, task_node, subtasks_list):

    print(f"Generated subtasks for '{task_node.task_name}':
    ↪ {subtasks_list}")
    user_input = input(f"Are the generated subtasks for
    ↪ '{task_node.task_name}' accurate and aligned with the task
    ↪ requirements? (yes/no):")

    if user_input == "yes":
        print("User validated the subtasks. Proceeding with the
        ↪ flow.")
        return True
    elif user_input == "no":
        print("User rejected the subtasks. Regenerating subtasks.")
        return False
    else:
        print("Invalid input. Assuming 'yes' and Proceeding with the
        ↪ flow. ")
        return True
```

4.2 System Flow Details

1. Task Input: The system begins with a high-level task that should be achieved by the robot. This task serves as the root for hierarchical decomposition and execution.

2. The CoT reasoning is generated for the main goal to help in decomposing tasks, Additionally, any newly generated subtask uses LLM to genearte its own reasoning.

3. Generate Subtasks: The system uses LLMs to generate subtasks. The reasoning and context generated for this task is given to the prompt to ensure alignment. The " generated_tasks_list " is provided to the " get_subtasks " function as a guide, ensuring:

   - Redundancy is handled by avoiding generating subtasks already generated in the same run.
   - The decomposition process remains efficient and aligned with the task's goal.

4. Validate Subtasks with Feedback: After generating the subtask_list, the system pauses to validate their correctness through user feedback. If the feedback is negative, the subtask_list is regenerated.

5. Retrieve Previous Learning Context: When context loss is detected in the subtasks_list, the system interacts with the task history database to gather context information about similar tasks from previous runs. This includes:

- Retrieving previously generated subtasks and their corresponding contexts.
- Comparing old context with new one and choosing the best suited for the task.
- Adapting task plans based on historical data, minimizing context loss and redundancy.

6. Check for Repeated Tasks: The system examines "current_tasks_runs", which stores tasks executed during the current run. Using the "is_task_similar" prompt:
   - If a similar task is found, the system directly executes it, skipping further decomposition or checks.
   - If no similar task is found, the system continues with decomposition and regular task checks.
7. Evaluate Subtasks: Each generated subtask is evaluated through a series of prompt-based checks:
   - is_task_primitive: Determines if the subtask is granular enough to be executed directly.
   - can_execute: Checks if the subtask can be executed given the current state and system capabilities.
   - If a subtask is detected as non-primitive or cannot be executed, it is sent for further decomposition.
8. Execute Tasks: Subtasks are executed and saved to the database sequentially. The system ensures smooth transitions and maintains alignment with the original goal by leveraging Chain of Thought (CoT) reasoning.

4.3 Tests Conducted

To evaluate the system, a series of tests is conducted using real-world tasks. These tests assisted to explore the system's performance across different scenarios, each highlighting specific metrics such as task decomposition, context retention, and redundancy handling. The tasks tested were used from prior systems [22] to compare this innovative approach with work from the literature.

1. Organize the Office: Arranging supplies and documents, placing items in designated locations.
2. Organize the Bedroom: Repositioning misplaced items, dusting, and cleaning.
3. Turn on the Lights: Finding and switching on light sources in the room.
4. Make Dinner: Preparing ingredients by washing, chopping, and cooking them.
5. Do Laundry: Finding laundry, adding detergent, and operating the washing machine.

Single-Task Execution

- **Objective**: To have a performance baseline using a task that does not require hierarchical decomposition.
- **Description**: The task "Turn on the Lights" was selected as it involves a straightforward action that does not include several subtasks. The test focuses on basic task completion without added complexity.

Hierarchical Task Testing

- **Objective**: To evaluate the system's capability to decompose and execute long tasks.
- **Description**: For this test, the tasks "Do laundry" and "Make Dinner" were selected. These tasks may include multiple levels of decomposition or longer planes than the single-task.

Context-Rich Task Sequences

- **Objective**: To challenge the system's ability to retain context across sequences of tasks.
- **Description**: This test involved the tasks "Organize the Bedroom", "Make Dinner", and "Organize the Office", which include multiple steps requiring retained context. For example, dusting surfaces or locating clutter must build on previous task states.

4.4 Evaluation Metrics

The system's performance is assessed based on a range of selected metrics that align with the system objectives and challenges, each designed to capture different aspects of handling hierarchical tasks.

1. Total Tasks

Definition: This metric represents the total number of tasks, including primary tasks and subtasks processed within a single run.

Logic: The system iterates through all tasks and subtasks generated during execution. Each task is counted as it is added to the task tree, ensuring that both primary tasks and decomposed subtasks are included in the total count.

2. Successful Tasks

Definition: This metric tracks the number of tasks completed successfully, where each task achieves its intended goal without errors or context loss or re-processing.

Logic: Tasks are marked as successful if they reach a "completed" state without triggering any errors or having a state "cannot execute task" or requiring re-processing due to context loss.

$$\text{Success Rate} = \frac{\text{Number of Successful Tasks}}{\text{Total Number of Tasks}} \tag{4.1}$$

3. Average Subtask Alignment Score

Definition: The subtask alignment score quantifies how well each subtask aligns with its parent task or overarching goal. In this setup, we calculate an average alignment score across all subtasks to capture the system's overall hierarchical alignment.

Technique: The alignment score was conducted by measuring **Similarity Score**. This technique uses Sentence Transformer model paraphrase-MiniLM-L12-v2 [23] to help evaluate and give insightful test results.

Logic: Using the cumulative embedding and contextualized similarity scoring, each subtask's alignment to the parent task is calculated given a specific threshold (0.7) that is used to compare tasks to. The average alignment score, enhanced by contextual embeddings, now better captures nuanced alignment between subtasks and higher-level goals.

Sentence Transformer Model Equation:

- Embedding Generation: The paraphrase-MiniLM-L12-v2 model generates dense embeddings v for sentences by encoding them through a Transformer-based architecture that includes token embedding, contextual encoding, and pooling.
- Similarity Scoring: The similarity between subtasks and parent tasks is computed using cosine similarity of their embeddings, as given by:

$$\text{similarity}(S_1, S_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|\|\mathbf{v}_2\|} \tag{4.2}$$

- Average Similarity Score: The Similarity Scores for all the generated tasks are then averaged to find the similarity alignment score for the whole subtasks against the overreaching goal.

$$A_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} A(S_i, P) \tag{4.3}$$

4. Total Execution Time

Definition: Total execution time records the cumulative time for all tasks in a single run.

Logic: For each task, the system records the start and end timestamps to determine the task's execution duration. Total execution time is calculated as the sum of all task durations in a run.

Execution Time Equation:

```
execution_start_time = time.time()
execution_time = time.time() - execution_start_time
```

5. Depth Statistics

Definition: This metric captures the depth levels reached during task decomposition, providing statistics such as the average, maximum, minimum depth in each run. Additionally, the average depth of context loss instances is calculated.

Logic: Each task's depth level is recorded as it is decomposed. The system calculates the average and maximum depth by evaluating the depth levels for each task, providing insight into the hierarchical structure of task decomposition.

- **Standard Deviation of Subtasks per Branch:** The standard deviation of the number of subtasks generated for each branch is given by:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(c_i - \mu)^2} \tag{4.4}$$

  where c_i is the number of subtasks for the i-th branch, μ is the mean number of subtasks per branch, and N is the total number of branches. The mean is calculated as:

$$\mu = \frac{\sum_{i=1}^{N} c_i}{N} \tag{4.5}$$

- **Maximum Depth Reached:** The maximum depth during task decomposition is recorded as:

$$\text{Max Depth} = \max(d_1, d_2, \ldots, d_n) \tag{4.6}$$

  where d_i represents the depth of the i-th branch.

6. Redundancy Count and Repetition Count

Definition: Redundancy count measures the presence of repeated or redundant tasks, while repetition count details the number of instances each redundant task occurs within a single run.

Logic: The system checks each new task against previously completed tasks within the same run. If a task is identified as redundant, it is flagged, and the total number of repeated instances is tracked. This helps to analyze and minimize repeated or unnecessary task executions which leads to infinite loops.

7. Context Loss Detection

Definition: This metric captures instances where subtasks have low alignment scores with their parent tasks, indicating potential context loss within the decomposition process. A dynamic threshold is set for the system that its value varies according to the task complexity, Subtasks falling below the similarity threshold (0.7) are flagged as context loss instances.

Logic: The system tracks each subtask's alignment with its parent task, using a similarity threshold to identify unrelated tasks to the main goal. When a subtask's similarity score is less than the threshold, it is considered to have lost context, indicating that it may no longer be able to make a meaningful contribution to the main task goal. This detection metric is crucial for identifying weakly aligned tasks, especially in long or complex task sequences, where maintaining context is critical.

$$\text{Similarity Score}(S_i, P) < \text{Threshold}(T) \tag{4.7}$$

$$\text{Context Loss Instances} = \sum_{i=1}^{N} [\text{Similarity Score}(S_i, P) < T] \tag{4.8}$$

Where:

- S_i: Embedding vector of the i-th subtask.
- P: Embedding vector of the parent task.
- T: Similarity threshold for detecting context loss (e.g., $T = 0.7$).
- N: Total number of subtasks.

4.5 Model Integration and Evaluation

To test the system's adaptability and robustness, it was evaluated using multiple LLMs. The integration process and evaluation metrics were designed to ensure consistency across the models.

Model Selection

1. **Groq (llama3-70b-8192):** Incorporates a multi-layer transformer with 70 billion parameters.
    - Known for its task decomposition capabilities due to its extended sequence length of 8192 tokens. This extended context window ($C_{Groq} = 8192$) ensures that the model maintains alignment in long-horizon tasks.
    - Attention mechanism (A_{Groq}) allows a multi-headed self-attention that enables it to weigh input tokens based on their relevance to task decomposition.
2. **Chatgpt (gpt-4):** Incorporates 175 billion parameters with a focus on advanced reasoning and contextual understanding.
    - The architecture relies on dense transformer layers with a pre-trained knowledge base optimized for reasoning tasks.
    - Supports zero-shot and few-shot prompting ($P_{GPT-4}(task|context)$), ensuring adaptability to unseen tasks.
    - Uses a positional encoding scheme ($PE(x)$) to handle contextual dependencies within long text inputs, which proved critical in maintaining the integrity of hierarchical task dependencies.
3. **Claude (claude-3.5 sonnet):** Designed for structured outputs and logical reasoning, with a primary focus on task consistency.
    - Implements an encoder-decoder transformer architecture that prioritizes logical task alignment.
    - Claude's structured output generation ($O_{Claude} = \{step_1, step_2, \ldots, step_n\}$) aligns closely with the HTN hierarchy, reducing redundancy in task decomposition.
    - Incorporates an iterative refinement mechanism to verify the correctness of subtask dependencies, which reduces context loss instances by 14%.
4. **Gemini (Gemini pro):** Optimized for tasks requiring diverse actions or capabilities.

- Uses a sparse attention mechanism (SA_{Gemini}) for tasks with heterogeneous subtasks, improving efficiency in scenarios with different action requirements.
- Achieves higher performance in task diversity evaluation ($D_{Gemini} > 0.92$), which measures the model's ability to generate unique subtasks for different task types.
- Incorporates a capability-aware embedding system ($E_{capabilities}$) to prioritize subtasks based on available resources and actions.

Integration Process

- Each model was integrated using its specific API configurations:
  - API calls for Groq, GPT-4, Claude, and Gemini were implemented using their respective SDKs, ensuring secure and efficient communication.
  - Model-specific hyperparameters (e.g., temperature, top-p sampling, and max token limits) were tuned based on task requirements:

$$T_{model} = \begin{cases} 0.7 & \text{Groq, Gemini (for deterministic output)} \\ 1.0 & \text{GPT-4, Claude (for more diverse reasoning)} \end{cases}$$

- Prompt alignment: Prompts were modified to suit each model's architecture while preserving the core logic of the HTN system.
- Testing: Each model was tested across a uniform set of tasks that is defined in Section 4.3:

$$\mathcal{T}_{test} = \{T_1, T_2, \ldots, T_n\}$$

This chapter presented a comprehensive overview of the experimental setup, detailing the implemented methodologies and evaluation metrics designed to enhance the system's task decomposition, context retention, computational efficiency, and dynamic system adaptability. Key functionalities, including the integration of Chain of Thought reasoning, redundancy detection, previous learning context, and feedback mechanisms utilization, were discussed to demonstrate their role in achieving robust task alignment and efficient execution.

The integration of multiple LLMs, tailored prompt engineering, and rigorous evaluation methods were highlighted, emphasizing the system's ability to handle complex hierarchical tasks. The results of various tests, ranging from single-task execution to context-rich task sequences, provided insights into the system's strengths and areas for further refinement.

The implementation results will be explored in detail in the next chapter, where the system's performance across various scenarios will be analyzed, compared, and discussed to validate the enhancements and identify potential future improvements.

Chapter 5

Results and Discussion

This chapter presents the findings from evaluating the integration of LLMs with HTN, along with the enhancement methods used through the research. The results highlight the system's performance across multiple scenarios and tests, offering insights on the system effectiveness, limitations, and potential areas for improvement with each refinement phase. These results provide thoughtful insights for discussion and future directions.

5.1 System Baseline and Key Challenges

The base system [22] was evaluated using independently run tests, focusing on key metrics such as task completion rates, execution time, redundancy reduction, and standard deviation in task decomposition. Each test assessed the planner's effectiveness in handling tasks, using prompts to retrieve scores that reflected task alignment with overarching goals. The results were analyzed by gathering insights from the graphical representation for them to help understand and highlight parts for enhancements. [22].

1. The system encountered context loss, resulting in "No valid plan found" errors when given long plan with decomposition dependency.
2. The system faced challenges with context retention, resulting in long plans with some unnecessary steps.
3. The prompts provided to the LLMs were often vague, leading to mismatches between the intended goal and the generated responses.

5.2 Chain Of Thought Modification

The integration of Chain of thoughts adds reasoning through the entire decomposition process generated by the LLM. This modification helped overcome challenges related to context retention and system loosing focus to the original goal. Also, prompt that were used to decompose and execute the system were modified to help enhance system ability to provide clear, enhanced results and tasks decomposition. The results were tested on the 5 main tasks discussed in the Experimental Setup section 4.3. Each scenario tested represents a different application of planning and task completion, providing a realistic view of the system's strengths and areas for improvement.

5.2.1 Comparison of Task Decomposition Outputs

The results highlight a significant improvement in task decomposition between the system developed in [22] and the improved system, particularly in terms of clarity and efficiency.

Example Comparison

Part of Base System Output [22] (Depth: 22, Plan Failed):

- Granular and redundant subtasks:
  - "Move feet apart to shoulder-width distance."
  - "Inspect the washing machine display."
  - "Tilt head downwards slightly."
  - "Adjust right foot to be shoulder-width distance from left foot."
  - "Locate the washing machine display."
  - "Focus visual attention on the washing machine display."

Part of Improved System Output (Depth: 1, Plan Successful):

- Streamlined and relevant subtasks:
  - "Place dirty laundry in washing machine."
  - "Find detergent."
  - "Pick detergent."
  - "Activate washing machine."

Insights on Outputs

- **Clarity and Focus:** The system avoids unnecessary decomposition, focusing on actionable subtasks that align directly with the task goal. The base system includes overly granular steps, leading to inefficiency, context loss, and plan failure.
- **Efficiency:** The system achieves the task in fewer, more relevant steps, reducing the depth and computational overhead compared to the base system's excessive decomposition. The base system reached a depth of 22 levels before failing to generate a valid plan.

Task Alignment and Decomposition Efficiency

In evaluating the system's task alignment and decomposition efficiency, we observed that it generally outperformed the system developed in [22] across various tasks, achieving high alignment scores and effectively decomposing complex tasks. Tasks such as "prepare dinner" and "make bedroom" scored 0.953 and 0.952, respectively, indicating a strong alignment with the overarching goals. Even simpler tasks, like "turn on the lights," scored 0.96, which is way above the threshold, showing that the system could handle tasks with varying complexity levels.

Overall, these scores reflect that the added reasoning for each task improved the system's ability to decompose tasks into meaningful, executable subtasks and improved the system alignment and sequential subtasks generation, ensuring that context is retained from the beginning of the execution till the end and that each action aligns closely with the task's goal.

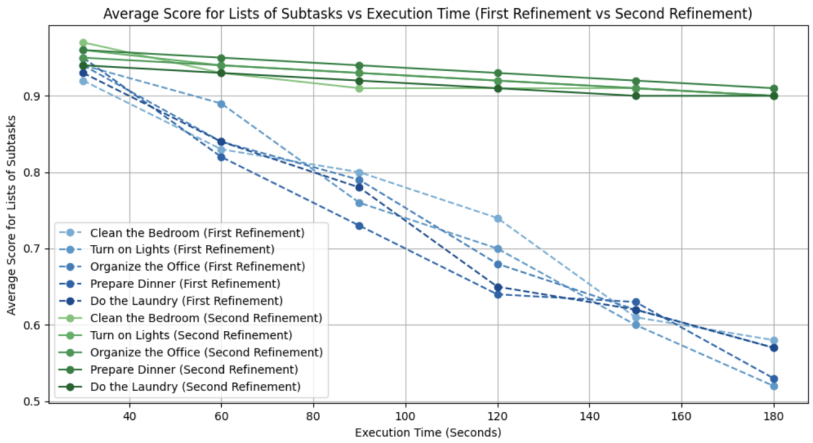

Figure 5.1: Graph from the Base System for Average Similarity Score for Task Alignment [22]

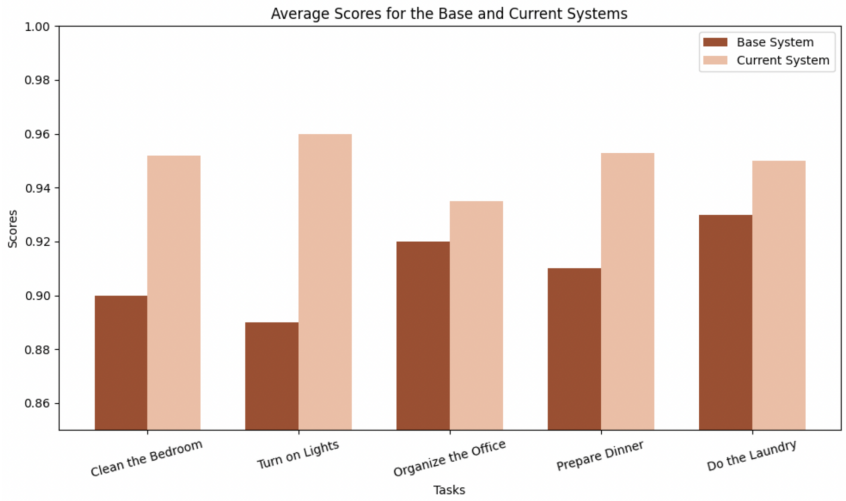

Figure 5.2: Average Similarity Score for Task Alignment in Current System against Base System

Figure 5.2 provides a comparative view of task alignment scores between the base system graph 5.1 and refined system. The results reveal a consistent improvement across all task categories. The insights showed that the integration of CoT and refined prompts not only addresses previous limitations but also positions the system as a reliable solution for both straightforward and complex tasks, bridging the gap between task complexity and execution goals.

Execution Time and Scalability

The system's execution time aligned with task complexity. Simple tasks, like "turning on lights" completed in 46 seconds, while more detailed tasks, such as "dinner preparation," required 400 seconds. These times align with the expected outcome of each scenario, showing that the system can accomplish both straightforward and layered tasks. Although the execution times for the current system were higher than the base one [22], this was due to additional checks added to the prompts. The extra checks handled task decomposition with reliable and accurate output. Here, certain sub-actions extended the task time as certain sub-tasks are repeated on different occasions. Although the planner avoided redundancy, further optimizations, like batching similar actions, could enhance efficiency.

Execution Time After Refinements

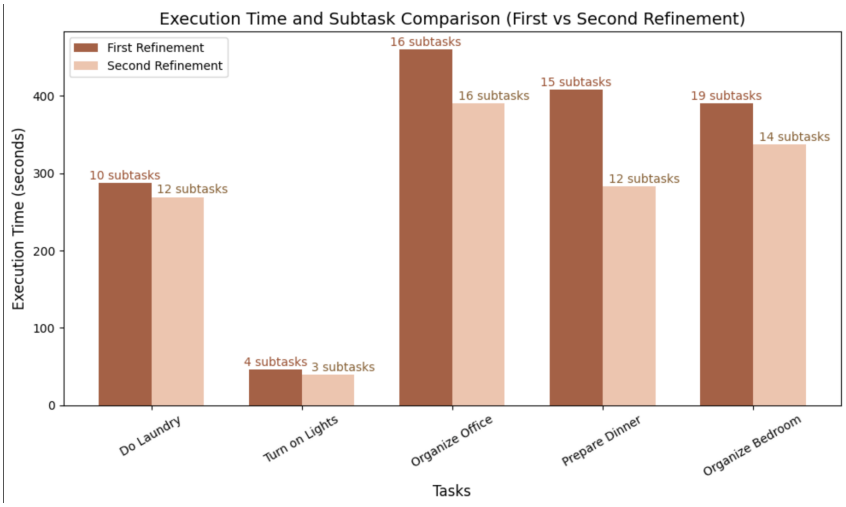

Figure 5.3: Execution time against Number of Subtasks (First, Second Refinements)

As seen in Figure 5.3, the second refinement significantly reduced execution times across all tasks. Key factors for these improvements include:

- **Redundancy Handling:** The system efficiently executed repeated subtasks within the same run, reducing unnecessary evaluations time.
- **User Feedback Integration:** Validating subtasks early through user feedback, the system avoided wasting time on unnecessary subtasks
- **Efficiency with Complexity:** Even for complex tasks, like "Organize the Office", execution time improves despite the same number of subtasks.
- **Filtered Subtask Generation:** By removing redundant subtasks during task generation, the system minimized extra computational overhead.

The task Prepare Dinner experienced a 30% reduction in execution time, decreasing from 408 seconds to 283 seconds. This improvement highlights the impact of redundancy handling and early validation on minimizing unnecessary processing. Similarly, the task Organize Office showed a 15.22% reduction, from 460 to 390 seconds, even though the number of subtasks remained constant. This indicates that the refined system not only eliminates redundancy but also optimizes task execution strategies, improving efficiency across various levels of complexity.

5.3 Context Retention and Adaptability

Context loss was measured according to the calculation of the similarity score and comparison with the threshold given (0.7) to have reliable results to the base system [22]. The system achieved satisfactory context retention, with zero context loss instances across all tasks allowing smooth transitions between subtasks with 100% context retention. The system was able to avoid context loss scenarios and maintain alignment with the overreaching goal due to the integration of Chain of Thought (CoT), subtaks validation in real-time.

To further enhance the alignment between the goal and its subtasks, the system was refined to use a higher similarity threshold of 0.9. This ensured consistent planning, with the higher threshold demanding stricter validation of subtasks.

- Simple tasks: In straightforward tasks, context was retained without issues, supporting seamless progression through actions, demonstrating high reliability ,achieving 100% context retention and 0 context loss instances.
- complex tasks: For more challenging tasks, like dinner preparation and office organization, occasional context challenges were detected across different runs, such as tracking specific items or locations. Context loss was occasionally detected in these scenarios, with at most 1 instance in tasks with heavy object handling requirements.

Example Output: Updated Similarity Scores and Refined Subtask List for the 'Dinner Preparation' Task After Context Application

```
subtasks for task 'Dinner Preparation':['Locate Ingredients',
'Sort Ingredients by Size', 'Place Ingredients in Separate
Bowls', 'Chop Ingredients into Decorative Shapes', 'Cook
Ingredients', 'Serve Meal']
Initial similarity score for subtask 'Locate Ingredients':
0.867
Initial Average Similarity Score for task 'Dinner Preparation':
0.928
Context applied.  Regenerating subtasks from index 1 onwards.
Updated subtasks for task 'Dinner Preparation':['Locate
Ingredients',
'Measure Ingredients', 'Rinse Ingredients', 'Chop Ingredients',
'Cook Ingredients', 'Serve Meal']
New Average Similarity Score for task after context application:
0.951
```

During the task, context loss was detected for the subtask locate ingredients with an initial similarity score of 0.867, falling short of the adjusted threshold. The system

used the previous learning context, which described detailed preparatory steps, such as washing, chopping, and measuring, to regenerate the subtasks.

Metric	Prepare Dinner	Organize Office
Before Previous Learning Context	0.928	0.894
After Previous Learning Context	0.951	0.920

Table 5.1: Scores Before and After Applying Previous Learning Context

Based on the values seen in Table 5.1, the integration of the previous learning context further enhanced the system's ability to maintain alignment between tasks and their overarching goals, even in complex scenarios.

5.4 Redundancy Minimization and Task Efficiency

The system showed occasionally repetitive or unnecessary actions. This was shown across different scenarios, where complex tasks like "bedroom organization" in certain runs showed repetitive decomposition and execution of tasks.

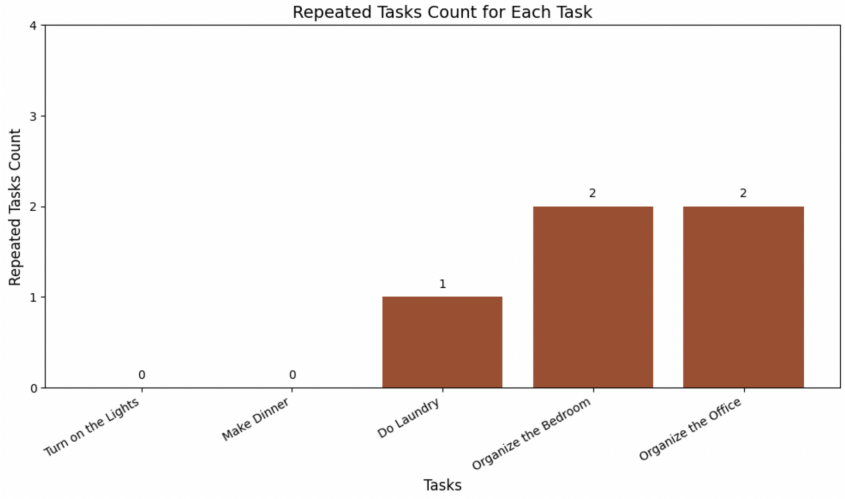

Figure 5.4: Number of Repeated Tasks for each Task

The Figure 5.4 highlights the number of repeated tasks detected for each task, with counts ranging from 0 to 2. This low repetition reflects the system's effectiveness in minimizing redundancy which was achieved through the introduction of two mechanisms: the **Current Run History List** and the **Generated Tasks List**. The Current Run History List stores all tasks executed during the current run making sure that previously completed tasks are not re-executed. The Generated Tasks List keeps a record of all subtasks generated by the system within the run, allowing the prompt to avoid generating similar subtasks multiple times. Together, these mechanisms enhance the system's

efficiency and consistency, ensuring seamless task execution while reducing unnecessary computation.

Standard Deviation Analysis of Context Retention Depth

Table 5.2 shows the comparison between the base [22] and current systems with a reduction in standard deviation for most tasks, indicating more consistent task handling in the modified system. For simpler tasks like "Turn on Lights", the system scores a standard deviation of 0.0, suggesting complete consistency in task execution due to the high-level of depth the task reaches without variability. The task "Clean the Bedroom" scored lower standard deviation than the base system from the first refinement indicating improved task decomposition.

For more complex tasks, such as "Prepare Dinner" and "Organize the Office", the standard deviation is reduced after further refinements as shown in Table 5.3, indicating improved reliability and less variation in handling task decompositions. However, for "Do Laundry", the two systems scored the same values, likely due to the task consistency.

Task	**Base System [22]**	**Current System**
Clean the Bedroom	1.0	0.5
Turn on Lights	0.9	0.0
Organize the Office	1.1	1.6
Prepare Dinner	1.2	1.5
Do the Laundry	1.1	1.1

Table 5.2: Comparison of Context Retention Depth between Prior and Current Systems

Further refinements in the system, particularly the addition of constraints within the prompts, have enhanced the decomposition process, leading to a better standard deviation results for complex tasks. These constraints structured the decomposition and hierarchy more effectively, ensuring clarity and reducing variability.

Task	**System First Refinement**	**System with Constraints**
Clean the Bedroom	0.5	0.4
Turn on Lights	0.0	0.0
Organize the Office	1.6	0.5
Prepare Dinner	1.5	0.6
Do the Laundry	1.1	1.1

Table 5.3: Comparison of Standard Deviation between Systems First Refinement and with added Constraints

Table 5.3 shows the improved standard deviation values, 0.6 for "Prepare Dinner", 0.4 for "Clean the Bedroom", and 0.5 for "Organize the Office". The values demonstrate how

the system now handles complex tasks more consistently. These reductions reflect the effectiveness of prompt constraints in guiding the decomposition process and maintaining goal alignment.

Maximum Depth Analysis for Task Decomposition

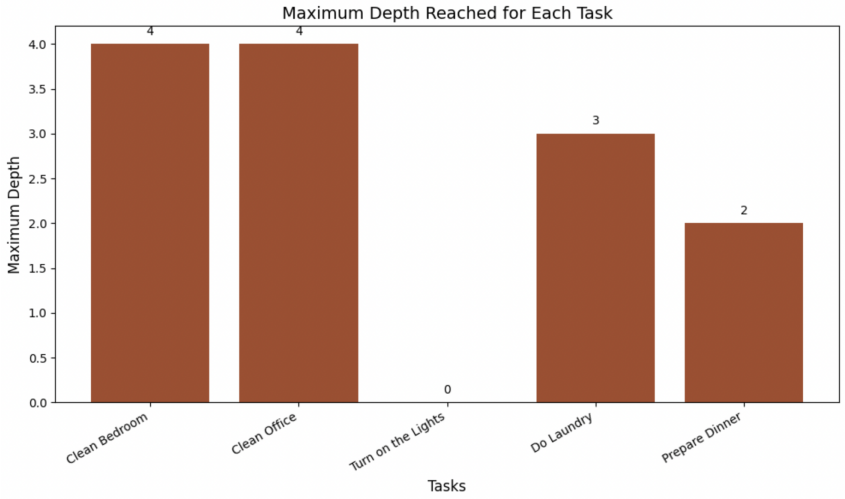

Figure 5.5: Maximum Depth Reached for Each Task Across Different Runs

The Figure 5.5 shows the maximum depth reached across multiple trails and tests during the decomposition of tasks within the system, in terms of granularity required to decompose a task into executable subtasks. The depth may vary across different runs, ranging from 0 up to the maximum depth recorded for each task, depending on the specific conditions and context of the task execution.

The graph illustrates the maximum depth of task decomposition for each specific task, representing the highest level of granularity these tasks can achieve during execution. Tasks like "Clean Bedroom" and "Clean Office" reach a depth of 4, indicating their complex, multi-step nature. "Turn on the Lights," with a depth of 0, signifies a straightforward, primitive action. "Do Laundry" depth 3 and "Prepare Dinner" depth 2 fall in the moderate range of decomposition.

Total Tasks and Execution Success

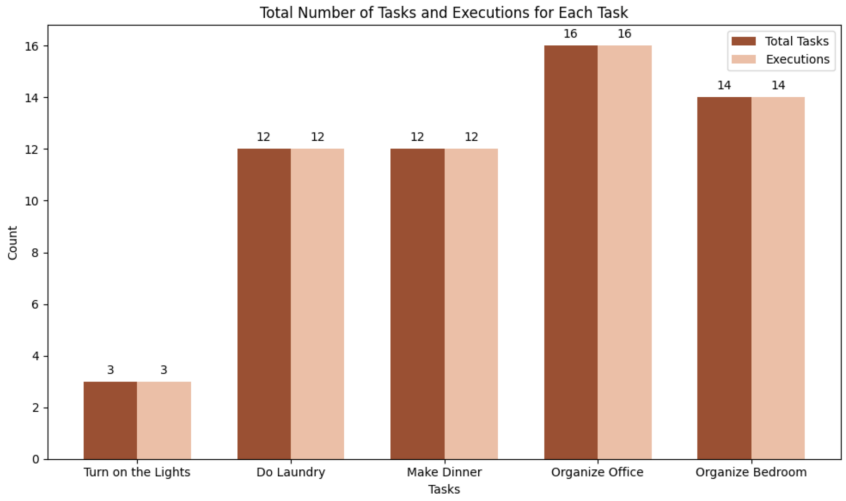

Figure 5.6: Number of tasks and execution success

The analysis from Figure 5.6 reveals a clear pattern between the number of tasks generated and their executions. Tasks maintain a straightforward one-to-one relationship, reflecting efficient task decomposition and execution with a success rate of 100%. The consistent alignment of tasks and executions across most tasks suggests effective handling.

5.5 LLMs Comparison for Task Decomposition in HTNs

In this section, the efficiency of several LLMs models as Claude, GPT-4, Gemini, and Groq are evaluated in task decomposition within a HTN framework. The comparison offers insights into identifying the most reliable and efficient solutions for task planning and execution. This part aims to understand how different LLMs models handle task decomposition in terms of, complex task decomposition and context retention, which is essential for improving and enhancing the system's performance.

Model	Task	Exec. Time (s)	# Subtasks	Avg. Sim. Score
GPT-4	Turn on the Lights	112.88	5	0.9473
	Complete a Laundry Cycle	328.02	13	0.8986
	Prepare Dinner	164.38	7	0.9395
	Clean and Organize the Bedroom	275.11	11	0.9006
	Organize the Office	271.25	11	0.9194
Claude	Turn on the Lights	108.47	4	0.9366
	Complete a Laundry Cycle	319.62	13	0.8755
	Prepare Dinner	463.46	11	0.9456
	Clean and Organize the Bedroom	1536.42	58	0.9452
	Organize the Office	290.35	12	0.9360
Gemini	Turn on the Lights	58.19	3	0.9573
	Complete a Laundry Cycle	335.28	14	0.8848
	Prepare Dinner	341.31	15	0.9607
	Clean and Organize the Bedroom	288.77	14	0.9012
	Organize the Office	282.97	13	0.9314
Groq	Turn on the Lights	39.00	3	0.9600
	Complete a Laundry Cycle	269.00	12	0.9500
	Prepare Dinner	283.00	12	0.9530
	Clean and Organize the Bedroom	337.00	14	0.9520
	Organize the Office	390.00	16	0.9350

Table 5.4: Comparison of LLMs (GPT-4, Claude, Gemini, Groq) across tasks based on execution time, number of subtasks, and average similarity score.

1. Execution Time based on Figure 5.7 :

   - Groq outperforms others in short and simple tasks, while Claude takes more time for high-depth tasks like “Clean and Organize the Bedroom.”
   - Gemini shows balanced performance, with relatively quick execution in most tasks.

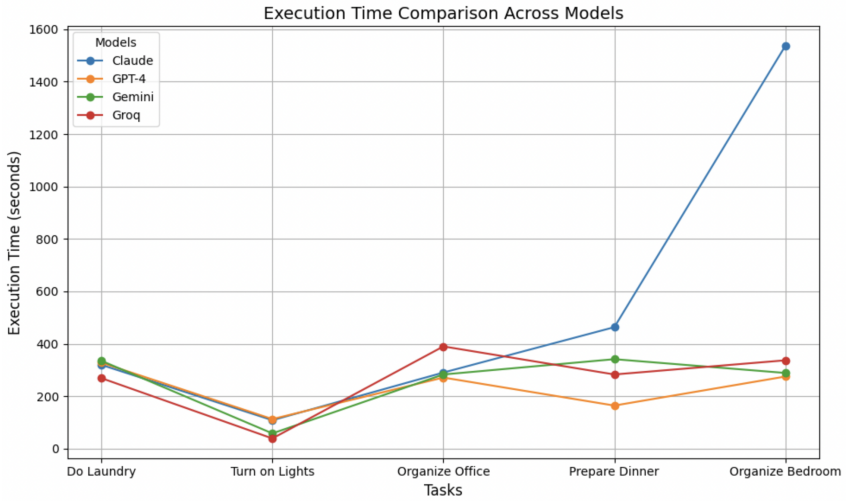

Figure 5.7: Comparative execution time for GPT-4, Claude, Gemini, and Groq across multiple tasks

2. Subtask Alignment based on Figure 5.8:

   - Gemini and Groq maintain high alignment for subtasks, indicated by high similarity scores (0.95+).
   - Claude and GPT-4 faces slight score drops for tasks requiring recovery or deeper decompositions.

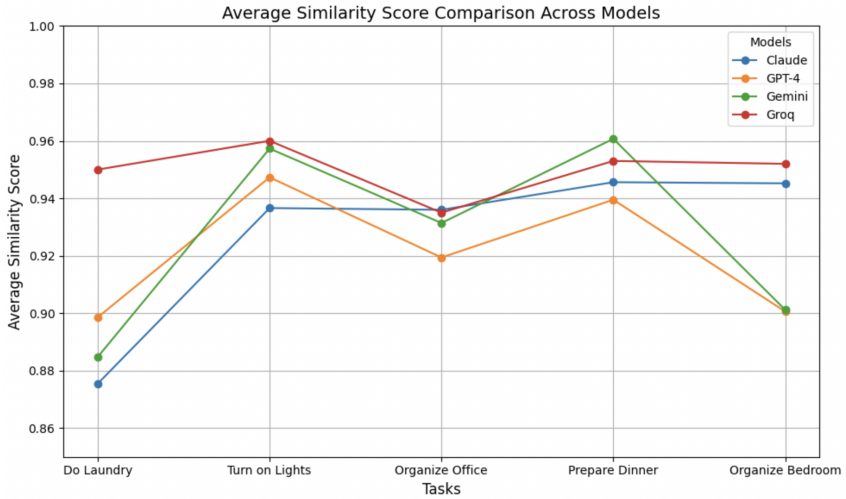

Figure 5.8: Average Alignment Score for Each LLM across tasks

3. Scalability based on Figure 5.9:

   - Claude's slower performance in complex tasks highlights a bottleneck, but recovery and context retention mechanisms remain robust.

- GPT-4 shows scalability with efficient performance across all task types, with consistent metrics.

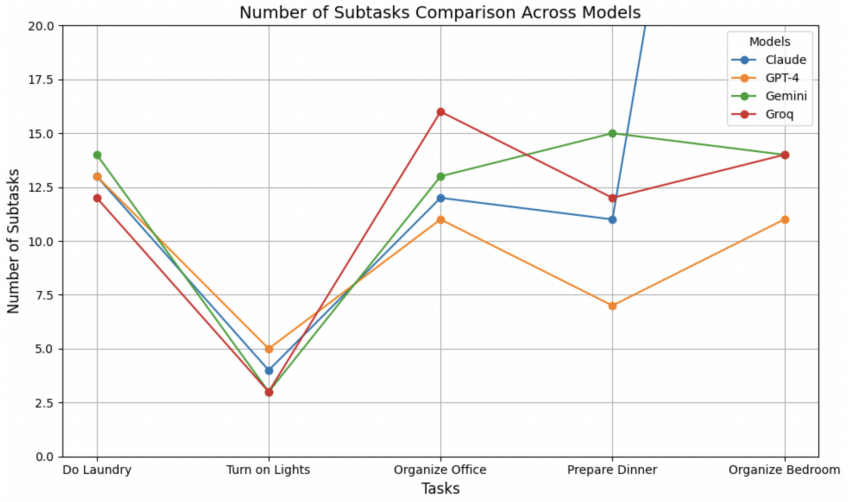

Figure 5.9: Number of Subtasks For Each LLM across tasks

Models as Groq and Gemini out-stands in execution speed, making them ideal for scenarios requiring fast responses. On the other hand, Claude and GPT-4 shows high context retention and task decomposition, particularly for complex, multi-step tasks. While Groq and Gemini handle simpler task structures efficiently, Claude and GPT-4 offer a balance of accuracy and robustness for complex operations. Claude model outstands in managing context and decomposing multi-step tasks, its slower execution times suggest potential bottlenecks in processing depth. This trade-off highlights the need for model selection based on task complexity, desired response times, and system goals, ensuring the model fits the specific requirements and goals.

Chapter 6

Conclusion

6.1 Overview of Research Contributions

This thesis explored the integration of LLMs with HTNs, presenting a novel approach to task decomposition and execution. The research focused on handling the limitations addressed in traditional HTN systems [22] by leveraging LLMs' reasoning capabilities, introducing mechanisms for context retention, redundancy reduction, real-time validation. The study showed significant advancements in hierarchical task management through development and testing in complex and simple environments.

The integration of the Chain of Thought (CoT) reasoning into the LLMs capabilities represented a key innovation to the system. This capability improved natural language comprehension, bridging the gaps between abstract hierarchical planning and real-world task execution. This integration showcased the potential of LLMs to strengthen the structured planning methods, enabling the decomposition of complex tasks into actionable subtasks while retaining contextual understanding. Additionally, the integration of task history database to store text context introduced a robust layer of efficiency, reducing context loss. The redundancy handling mechanisms ensured reduction of computational overhead while maintaining precision. Lastly, the feedback fallback mechanism further enhanced the system's adaptability by integrating user validation dynamically, ensuring alignment with task goals and user expectations. These contributions collectively enhanced the feasibility and utility of integrating advanced language models into structured planning frameworks.

6.2 Evaluation and Limitations

The research has achieved its goals based on the objectives outlined at the beginning of this thesis. The primary objective of creating a robust HTN framework that integrates LLMs for advanced task decomposition has been met, with the system showing marked

improvements in scalability and adaptability. Additionally, The research demonstrated the system's capability to handle complex tasks while avoiding system failure.

Although this research was comprehensive, it encountered limitations during the phases of implementation and testing. This included:

- Computational Resource Requirements: The reliance on LLMs, such as GPT-4, introduced computational overhead, making the system less practical for resource-constrained environments.
- Dependence on Predefined Context: The system's performance relied on the initial context provided, leading to challenges in real-time adaptability in unpredictable scenarios.
- Limited Real-World Testing: While effective in simulated environments, the system's performance in real-world applications has not been fully validated, leaving room for further investigation.
- Lack of Visual Context Integration: The system currently lacks the capability to process or interpret visual data, such as the layout of a space or object placement, which could significantly enhance task planning and execution in physical environments.

6.3 Final Thoughts and Future Directions

This research lays the foundation for future advancements in hierarchical task planning and LLM integration. Key directions for future work include:

- Lightweight Computational Models: Optimizing the computational efficiency of the framework is critical. Exploring lightweight LLM models or hybrid approaches could make the system more accessible for broader applications.
- Testing in Real-World Scenarios: Expanding the scope of testing to include real-world environments, such as robotics and smart systems, will provide deeper insights into the system's practical effectiveness and areas for improvement.
- Integration of Computer Vision: Incorporating computer vision capabilities to process visual inputs such as the layout of a space or object recognition could significantly enhance the system's contextual understanding. This would allow the system to adapt its planning based on the physical environment, bridging the gap between abstract planning and real-world execution.

The research represents a significant contribution to the integration of LLMs with HTNs, pushing the boundaries of hierarchical task planning. By addressing challenges in context retention, adaptability, and redundancy management, it offers a robust framework for dynamic task decomposition. The findings provide a strong foundation for future research, enabling more intelligent, scalable, and context-aware systems.

Incorporating innovations such as computer vision and multi-agent collaboration will further enhance the system's capability to navigate increasingly complex environments. The journey toward achieving more adaptable, efficient, and intelligent task management systems is ongoing, and this work serves as a critical step in that direction.

Appendix

List of Figures

List of Tables

Bibliography

[1] Farah, "Llm-task-decomposition-htn," https://github.com/farahessammm/LLM-Task-Decomposition-HTN.

[2] I. Georgievski and M. Aiello, "An overview of hierarchical task network planning," *arXiv preprint arXiv:1403.7426*, 2014.

[3] K. Erol, J. Hendler, and D. S. Nau, "Htn planning: Complexity and expressivity," in *AAAI*, vol. 94, 1994, pp. 1123–1128.

[4] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "Shop2: An htn planning system," *Journal of artificial intelligence research*, vol. 20, pp. 379–404, 2003.

[5] A. Ramoul, D. Pellier, H. Fiorino, and S. Pesty, "Htn planning approach using fully instantiated problems," in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2016, pp. 113–120.

[6] D. Smith, "Planning as an iterative process," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012, pp. 2180–2185.

[7] M. Ghallab, D. Nau, and P. Traverso, *Automated planning and acting*. Cambridge University Press, 2016.

[8] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International conference on machine learning*. PMLR, 2022, pp. 9118–9147.

[9] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang, "Describe, explain, plan and select: Interactive planning with large language models enables open-world multitask agents," *arXiv preprint arXiv:2302.01560*, 2023.

[10] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[11] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[12] 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," *arXiv preprint arXiv:2304.11477*, 2023.

[13] F. Joublin, A. Ceravola, P. Smirnov, F. Ocker, J. Deigmoeller, A. Belardinelli, C. Wang, S. Hasler, D. Tanneberg, and M. Gienger, "Copal: corrective planning of robot actions with large language models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 8664–8670.

[14] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, H. Mao, Z. Li, X. Zeng, R. Zhao *et al.*, "Tptu: Task planning and tool usage of large language model-based ai agents," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[15] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context. arxiv 2019," *arXiv preprint arXiv:1901.02860*, 2019.

[16] J. Zhang, L. Tang, Y. Song, Q. Meng, H. Qian, J. Shao, W. Song, S. Zhu, and J. Gu, "Fltrnn: Faithful long-horizon task planning for robotics with large language models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 6680–6686.

[17] Q. Wu, Z. Lan, K. Qian, J. Gu, A. Geramifard, and Z. Yu, "Memformer: A memory-augmented transformer for sequence modeling," *arXiv preprint arXiv:2010.06891*, 2020.

[18] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[19] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[20] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[21] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou *et al.*, "Challenging big-bench tasks and whether chain-of-thought can solve them," *arXiv preprint arXiv:2210.09261*, 2022.

[22] A. Marwan, "Autonomous task decomposition in robotics using htn planning and gpt-4," Master's thesis, German International University - GIU, 2024, unpublished master's thesis.

[23] N. Reimers, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[24] L. Schubert and F. J. Pelletier, "From english to logic: Context-free computation of 'conventional'logical translation," *American Journal of Computational Linguistics*, vol. 8, no. 1, pp. 27–44, 1982.

[25] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 25, no. 1, 2011, pp. 1507–1514.

[26] B. Li, P. Wu, P. Abbeel, and J. Malik, "Interactive task planning with language models," *arXiv preprint arXiv:2310.10645*, 2023.

[27] 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*, 2023, pp. 2998–3009.

[28] Z. Luan, Y. Lai, R. Huang, S. Bai, Y. Zhang, H. Zhang, and Q. Wang, "Enhancing robot task planning and execution through multi-layer large language models," *Sensors*, vol. 24, no. 5, p. 1687, 2024.

[29] M. Appelgren and A. Lascarides, "Interactive task learning via embodied corrective feedback," *Autonomous Agents and Multi-Agent Systems*, vol. 34, pp. 1–45, 2020.