Enhancing Language Models with Tool Augmentation for Multi-Step Queries in the Context of Physical Health: An Empirical Study

Bachelor Thesis

presented by
Joel Arenz
Matriculation Number 1738583

submitted to the
Data and Web Science Group
Prof. Dr. Christian Bizer
University of Mannheim

August 2024

Abstract

The advent of Large Language Models (LLMs) has revolutionized natural language processing, yet these models face limitations in complex reasoning and specialized knowledge application. This study explores the potential of tool-augmented LLMs as autonomous agents, specifically in the context of physical health assistance. Four distinct configurations of GPT-4o are introduced, each designed to enhance the model's ability to solve multi-step tasks: Chain of Thought with tool incorporation (CoT+), Planning and Solving (PS), Self Reflect (SeRe), and a combination of Planning, Solving, and Reflecting (PS + SeRe). To assess these configurations, a dataset of 75 multi-step use cases in the physical health domain is created. The results indicate that the Self Reflect (SeRe) configuration achieved the highest success rate, as the Reflect step allowed the model to improve upon its initial answers. Interestingly, simply splitting the reasoning process into Planning and Solving stages did not yield notable improvements. However, the addition of the Reflect module enhanced performance across various tasks. These findings suggest that incorporating reflective processes in tool-augmented LLMs is a promising avenue for developing more effective autonomous agents for complex multi-step reasoning tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The advent of Large Language Models (LLMs) has marked a significant milestone in the field of artificial intelligence, revolutionizing natural language processing and generation. These models, trained on vast amounts of textual data, have demonstrated remarkable capabilities in understanding context, generating human-like text, and performing a wide range of language-based tasks. As LLMs continue to evolve, they are pushing the boundaries of what's possible in human-computer interaction and automated reasoning.

Alongside the development of LLMs, the concept of autonomous agents has gained traction in AI research. These agents are envisioned as frameworks where the LLM serves as their "brain". The LLM receives input from the environment and makes decisions taking actions to potentially influence the environment. This loop enables them to act autonomously. Through tool-augmentation of LLMs, this idea becomes more realistic. By equipping LLMs with access to external tools and specialized knowledge bases, these models can overcome some of their inherent limitations and expand their problem-solving capabilities. This approach allows LLMs to leverage domain-specific tools and information, bridging the gap between general language understanding and specialized task execution.

Current research in this area, which will be explored in more detail in the related work section, has already demonstrated promising results. Various studies have shown how tool-augmented LLMs can perform complex tasks, from coding and data analysis to creative problem-solving and decision-making.

1.1 Problem Statement

While Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language generation and reasoning, they still face significant limitations that hinder their development into fully autonomous agents. Some of these limitations stem from their probabilistic nature, resulting in challenges such as inconsistent arithmetic skills and the generation of hallucinations. Other constraints are due to technical limitations. Those limitations will be explored in greater detail in section 2.3.

While LLMs excel in general tasks, a critical problem impeding their evolution into autonomous agents is their lack of domain-specific knowledge. This limitation can be partially mitigated by augmenting LLMs with tools that enhance their knowledge base, such as external databases. Models can access these tools through function calling capabilities, although the extent of this ability varies between open-source and closed-source models [42].

Similarly, existing domain-specific tools can be made accessible to LLMs, enabling them to perform actions relevant to particular fields. However, the approach of augmenting LLMs with tools introduces a new challenge: as the context in which the model operates expands significantly with the integration of multiple tools, it becomes increasingly difficult for the model to maintain a comprehensive overview of the task at hand.

This difficulty in managing expanded contexts leads to a particular struggle with multi-step tasks. While the typical Chain of Thought (CoT) approach aims to break down complex problems into multiple sub-tasks, the effective granularity of this method is constrained by the model's context size. As a result, LLMs often struggle to maintain coherence and accuracy across extended reasoning chains, especially when these chains involve multiple tool interactions and domain-specific knowledge applications expanding the model's context.

1.2 Background and Context

Multi-step problems are characterized by a chain of reasoning steps, where the output of one step serves as the input for the subsequent step. This sequential nature requires models to maintain coherence and accuracy throughout an extended reasoning process.

To address the challenges posed by multi-step problems and to leverage the potential of tool augmentation, this research explores four distinct configurations designed to enhance the reasoning capabilities of LLMs:

1. **CoT+**: CoT approach with a focus on incorporating tools in the reasoning steps

2. **PS**: Separating reasoning into Planning and Solving

3. **SeRe**: Adding Self Reflection to CoT+
4. **PS + SeRe**: Combining Planning, Solving, and Reflecting

These configurations represent different strategies for enhancing LLM performance on multi-step tasks, each with its own potential advantages and challenges. By exploring these approaches, this research aims to contribute to the ongoing efforts to develop more capable and reliable AI systems for complex problem-solving tasks.

1.3 Research Question and Objectives

The potential for tool-augmented Large Language Models (ALMs) to revolutionize problem-solving in complex domains is significant. There is a wide range of highly specialized tools for humans to use to enhance their capabilities. The prospect of LLMs acting as the intelligent ”brain” capable of autonomously utilizing these tools to solve complex problems presents immense potential. To advance the research in this domain, this study aims to address the following primary research question: “To what extent can a tool-augmented Large Language Model effectively serve as a personal health assistant in solving multi-step tasks, and how do different prompting and module configurations impact its performance?”

The key objective of this research is to compare the four distinct configurations previously introduced, and their influence on the model’s success rate. By implementing and testing these configurations across a dataset of 75 multi-step use cases, the goal is to evaluate their performance and identify the strengths and weaknesses of each approach. Furthermore, efficiency will be taken into account evaluating those configurations.

1.4 Structure Overview

This thesis is organized into several key chapters that systematically address the research question. First the historic and technical background of LLMs is introduced in chapter 2. Then work related to the subject with a focus on autonomous agents is presented in chapter 3. Subsequently, chapter 4 explains the methodology of the experiment, followed by chapter 5 presenting and discussing the results thoroughly. This also includes a comparison of the four introduced configuration with an Error Analysis on SeRe, the best performing configuration.

Chapter 2

Theoretical Framework

The Theoretical Framework chapter provides a comprehensive overview of Large Language Models (LLMs) and their development. It begins by tracing the evolution of LLMs in section 2.1, followed by quick presentation of the Transformer architecture that underpins modern language models in section 2.2. The next section 2.3 then discusses the current limitations of LLMs, setting the stage for an examination of the development of state-of-the-art models in section 2.4. A significant portion of the chapter is dedicated to the enhancement of LLMs in section 2.5, detailing various strategies and techniques for improving their performance. The final section 2.6 bridges the gap between tool use and the concept of AI agents, exploring the definition and key components of agent architectures.

2.1 Evolution of LLMs

The development of LLMs has progressed through several distinct stages, each representing a significant advancement in natural language processing capabilities. Initially, Statistical Language Models (SLMs) were developed based on statistical learning methods, with the fundamental approach of predicting the next word based on the most recent context [47]. These models laid the foundation for probabilistic language understanding.

The field then advanced to Neural Language Models (NLMs), which characterize the probability of a word sequence using neural networks such as multi-layer perceptrons (MLPs) and recurrent neural networks (RNNs). This shift allowed for improved context understanding and more nuanced predictions compared to their statistical predecessors.

A significant leap forward came with the introduction of Pre-trained Language Models (PLMs), which utilize pre-trained context-aware word representations as general-purpose semantic features. The transformer architecture, introduced by Vaswani et al. in 2017 [34], played a crucial role in this development. PLMs demonstrated substantial performance improvements across various NLP tasks.

The current stage in this evolution is represented by Large Language Models, which are characterized by their increased scale and capacity. These models are developed through the scaling of PLMs and have shown improved performance on downstream tasks. Notably, at certain parameter scales, LLMs exhibit what researchers term ”emergent abilities” – capabilities that are not present in smaller models but are present in larger models [38]. These abilities were not explicitly programmed but arise as a consequence of the model’s scale.

This progression in language model development illustrates how increases in model scale and capacity have led to enhanced performance and the emergence of new capabilities. The evolution from simple statistical methods to complex neural architectures has been marked by continuous improvements in the ability to process and generate human-like text, with each stage building upon the technological and theoretical advancements of its predecessors.

2.2 The Transformer Architecture

As the core statistical nature of NLP models does not change with the introduction of the transformer architecture, the input has to be converted from textual representation to statistical representation of text. This helps the model to understand and process the input more efficiently and effectively. This is achieved through a process called embedding where words are transformed into numerical vectors. These embeddings are compact representations of the semantic relationship between words.

To further expand on this concept, the Transformer architecture introduces a self-attention layer that allows each word in a sequence to attend to all other words [34]. This enables the capturing of complex relationships and dependencies. Another big part of the transformer architecture’s success is its ability to process sequences in parallel (unlike recurrent neural networks). This makes it highly scalable while still being efficient. This combination enables the development of increasingly large and powerful language models.

2.3 Limitations of LLMs

The most popular limitation of LLMs is hallucinations. As characterized in the literature as ”the generation of content that is nonsensical or unfaithful to the provided source” [10]. While the context in which the model hallucinates plays a big role in whether it is acceptable or not, in general the occurrence of hallucinations makes LLMs seem unfaithful, especially in the eyes of the general public. However, in research more limitations are important [19]. For example:

1. LLMs don't have a state or memory, limiting their potential use.
2. They are stochastic/probabilistic. While there are parameters to limit the variability in response to the same prompt sent multiple times, getting the same response generated each time cannot be guaranteed.
3. LLMs also lack dynamic information and have no access to external data sources.
4. Additionally they are very large, which makes them very expensive.

Since all of these limitations play a critical role in the widespread use of LLMs, there is a lot of research trying to eliminate these limitations.

2.4 Development to current State-of-the-Art Models

The transformer architecture has enabled a significant increase in both model and data size for current Large Language Models (LLMs). This scaling has been a key factor in the remarkable capabilities demonstrated by state-of-the-art models. Two important aspects of scaling in LLMs are predictable scaling and task-level predictability [47].

Predictable scaling refers to the ability to reliably estimate the performance of larger models based on that of smaller models. This concept has proven invaluable in guiding the training of LLMs, allowing researchers and developers to make informed decisions about resource allocation and model design. The existence of scaling laws suggests that performance improvements can be anticipated as model size increases, providing a roadmap for future development. Task-level predictability, however, presents a more complex challenge as the practical applications of LLMs are more concerned with performance on specific tasks than with language modeling loss.

As language models scale to unprecedented sizes, researchers have observed a phenomenon not predicted by scaling laws: the sudden appearance of ”emergent abilities.” These capabilities unexpectedly manifest when models reach certain, undefined, parameter thresholds. The emergence of these abilities leads to significant and sometimes dramatic improvements in model performance, beyond what would be expected from simple scaling [39]. Emergent abilities are characterized by their unpredictability and their substantial impact on model performance. Three notable emergent abilities observed in large language models are:

1. **In-context Learning**: This ability allows models to perform tasks based on instructions and examples provided within the input context, without requiring additional training. The model effectively ”learns” from the given demonstrations and applies this knowledge to the presented instance of the task.

2. **Instruction Following**: Large models demonstrate an enhanced capacity to understand and execute tasks described through natural language instructions. This ability enables models to perform well on previously unseen tasks when given clear directions, significantly expanding their versatility.

3. **Step-by-step Reasoning**: While complex problem-solving requiring multiple reasoning steps has traditionally been challenging for AI, large language models with this emergent ability can break down and approach intricate problems in a more systematic manner. When properly prompted, these models can articulate their thought process, making their reasoning more transparent and often more accurate.

These emergent abilities not only enhance the performance of large language models but also broaden their potential applications across various domains.

This discovery in NLP research has led to a never before seen rise in performance and prominence of Large Language models. One of the most prominent examples is the GPT (Generative Pre-trained Transformer) series, developed by OpenAI. The latest major iteration GPT-4, has demonstrated remarkable capabilities across a wide range of tasks [21], from natural language understanding and generation to complex problem-solving. Its ability to understand and generate human-like text has set new benchmarks in the field.

Another significant model is Google's PaLM (Pathways Language Model), which uses a technique called pathways to efficiently train on diverse datasets. PaLM has shown impressive performance on tasks requiring reasoning, including multi-step reasoning [5].

These state-of-the-art models are characterized by their massive scale, often containing billions of parameters, and their ability to perform a wide variety of tasks without task-specific fine-tuning. They exhibit impressive few-shot and zero-shot learning capabilities, allowing them to adapt to new tasks with minimal or no additional training. However, the development of these models also raises important considerations. Their size and computational requirements pose challenges in terms of energy consumption and accessibility. Additionally, as these models become more powerful, questions about their ethical use, potential biases, and societal impacts arise and are widely discussed.

2.5 Enhancement of LLMs

As discussed earlier in this chapter (see section 2.3), Large Language Models (LLMs) face several limitations despite their impressive capabilities. These constraints underscore the critical importance of enhancing LLMs to overcome their current shortcomings and expand their potential applications. This section delves into various strategies for augmenting LLMs, exploring techniques that aim to improve their reasoning, problem-solving, and self-correction abilities.

2.5.1 Importance of Enhancing LLMs

While these models can do a lot, and they're certainly impressive, they still have some significant limitations. The models can still produce non-factual content that seems plausible at first glance. These errors are often referred to as hallucinations [21]. While these errors are avoidable, Mialon et al. argue that they are caused by a fundamental flaw in LLM [18]. This is the nature of statistical language modeling, given a single parametric model and a limited context. Besides these consistency problems, the reasoning of LLMs can't yet be interpreted and fully understood, especially when it comes to emergent abilities

2.5.2 Planning, Reflecting, Refining

Planning and Reflection especially plays a vital role in overcoming LLMs limitations when the LLM is augmented with tools. As those tools alone can be used to increase factuality and therefore trustworthiness of the LLM. However, to reach this, the tools have to be used in an efficient and effective way. To foster the tool use capabilities of LLMs, different steps can be taken. The first step of improving the LLMs reasoning process overall and to incorporate tools better, is to improve its planning capabilities. Huang et al. [9] present a taxonomy of enhancing the planning capabilities of LLMs in five categories. Those categories however are not exclusive, but interconnectable:

1. **Task Decomposition**: Focuses on dividing the complex task into smaller, less complicated sub-tasks.

2. **Multi-Plan Selection**: This approach generates various alternative plans for the same task. The best one is then selected.

3. **External Planner-Aided Planning**: An external planner is used to outsource the planning process.

4. **Reflection Planner-Aided Planning**: This method introduces improving the planning ability of the model through reflection and refinement.

5. **Memory-augmented Planning**: The mode is augmented with an external knowledge base enhancing the planning capability.

Task Decomposition

The idea of enhancing reasoning of LLMs by following the Divide-and-Conquer principle was first introduced by Wei et al. [39] as Chain-of-thought prompting (CoT). CoT prompting is one of the most popular and useful approaches. It enhances the model's performance on complex reasoning tasks by prompting it to rely on intermediate reasoning steps [39]. These steps bridge the gap between input and output. One straightforward method for triggering a chain of thought is by including a simple instruction, such as ”Let's think by step,” in the prompt. In the referenced paper, it was demonstrated that this approach reduces the LLM's likelihood of making reasoning errors. Although chain of thought prompting is a valuable technique, as evidenced by the findings of Qian et al [25] and Bian et al. [1], it still encounters challenges in tasks such as mathematical reasoning and multi-step question answering.

Another approach following the Divide-and-Conquer principle is Plan-and-Solve by Wang et al. [36]. The idea here is a progression from CoT by emphasizing the aspect of creating and following a plan. The CoT prompt is refined to focus more on not only thinking step by step but to create a plan that considers sub tasks by adding “Let's first devise a plan”. Wang et al. show that this leads to an increase of performance in reasoning. Similar improvement was also observed by the ReAct approach [46] by Yao et al. They decouple reasoning and planning and alternate these two steps.

However the probability of generating a perfect plan directly from only the task description decreases as the required number of sub tasks, the task is divided in, increases [37]. To tackle this problem, providing the model with feedback after each sub task is executed and giving the model the opportunity to further improve its plan helps in improving the task selection and ultimately the model's performance.

Multi-Plan Selection

Approaches in the second category expand the planning capabilities by just generating not one but multiple alternative plans. In this spirit, the Tree-of-Thoughts [45] method spans a tree of alternative plans where each branch represents a thought.

To decide on the next course of action going down a branch the model uses self-evaluation and backtracking. Later the most optimal plan is selected and carried out. While this multi-plan approach fosters a broader exploration it also increases computational demands and cost.

Reflection Planner-Aided Planning

The core idea of approaches in this category is to aid the planning capabilities of the model by providing it with some form of feedback on its reasoning. Pan et al. introduce a framework for creating such a feedback cycle [22]. The framework incorporates three distinct models, the Language Model which performs tasks by planning. The Critic Model that generates feedback based on the output of the Language Model, and the Refine Model which uses the feedback and refines the Language Models output or the Language Model itself. This way hallucinations, unfaithful reasoning, flawed code or bad behavior in the Language Model's output can be corrected. There are two ways that feedback can be gathered. Either, the LLM itself can be utilized to criticize its own behavior and send feedback. This can be used iteratively taking the previous output as the new input. Or the model could use external feedback, provided by other models, tools or external knowledge sources. Another dimension to consider with Self reflection, is the format of the feedback and when to use the feedback to improve the model. A specific implementation following this framework is Self-Refine by Madaan et al. [17]. This approach improves the initial output of the LLM by iteratively providing feedback to the output, then refining it. For the generation, feedback, refinement cycle the same model is utilized. When testing the performance of GPT-4 using this approach, the authors demonstrate that with one-step generation they are able to improve the model's performance by around 20%. This approach is similar to one of the four configurations used in this paper's experiment (for results refer to section 5.2.2). Another method in this category is Reflexion [29]. This extends the ReAct approach by incorporating an evaluation step to provide feedback. The feedback is saved to a memory buffer that the model can access to improve its decision-making in subsequent trials.

2.5.3 Tool Augmentation

As previously mentioned, one method of obtaining feedback on the model's behavior is by utilizing tools. These tools represent an additional way of enhancing Language Models to mitigate their common weaknesses. A tool is defined as an external module that can be called by the model, and whose output is considered while generating the model's response. LLMs that are augmented with tools are called Augmented Language Models (ALMs) [18]. While a significant amount of research focuses on discovering different prompting strategies, the utilization of tools presents many possibilities for LLMs. They allow the model to access real time data, get automated feedback, and take actions in their environment. To fully comprehend the diversity and potential of tools, they can be classified into the following categories.

1. **Iterative LM Calling:** This intuitive method involves repeatedly calling the model to refine its output iteratively. While it offers a simple improvement technique, it may potentially produce significant computational overhead.

2. **Information Retrieval:** These tools augment the model's memory, helping to avoid non-factual and outdated information. This can be achieved through specific retrievers, querying search engines, or general web searching and navigation. By expanding the model's knowledge base in real-time, these tools enhance the accuracy and relevance of responses.

3. **Symbolic Computation and Code Interpretation:** This category addresses the numerical and logical shortcomings of LLMs. A prominent example is a code interpreter, which can handle arithmetic reasoning and execute complex computations. By offloading such tasks to specialized tools, the model ensures correct calculations while expanding its problem-solving capabilities.

4. **Virtual and Physical World Interaction:** Unlike tools that enrich the model's context with external information, this category includes tools capable of performing actions that affect the virtual or physical world. These tools extend the model's influence beyond information processing, allowing it to interact with and manipulate its environment.

One of the main limitations of using LMs today is their lack of up-to-date knowledge and their lack of use-case-specific information. These limitations can be tackled by using Retrieval Augmented Generation (RAG). RAG is the process of incorporating non-parametric knowledge (not included in the model's parameters in training) while generating language [13]. An interesting aspect of providing the model with knowledge it is not familiar with during training, is when knowledge conflicts occur. Jin et al. describe many forms of knowledge conflicts in their paper and how they could be resolved. They observe that RALMs (Retrieval-augmented Language Models) tend to favor their own internal memory even when correct evidence is provided. Furthermore they show that models prefer evidence that appears more frequently, again aligning with the statistical nature of the models [11]. As the potential of an ALM exceeds the potential of a LM, new benchmarks need to be created to evaluate the performance of those enhanced models (for benchmarks refer to section 3.3).

2.6 From Tool Use to Agents

The concept of autonomous agents as personal assistants has long captured the human imagination, prominently featured in science fiction and theoretical discourse alike. With the advent of large language models (LLMs) and their substantial leaps in performance, this concept has moved closer to reality. The following sections introduce one definition of autonomous agents and the key components of agent architectures.

2.6.1 Definition of Agents

Large Language Models augmented with external tools, present a compelling vision of autonomous agents capable of human-like behavior and sophisticated reasoning skills. At the core of an autonomous agent is the LLM, functioning as its ”brain.” This brain interprets inputs, makes decisions, and controls the agent’s actions. The LLM’s decision-making process is driven by its ability to understand and generate human-like text, allowing it to interact with users and the environment in a seemingly intelligent manner. When the LLM decides on an action, it leverages external tools to execute these actions, thereby affecting its surroundings. This interaction forms a feedback loop: the agent takes actions based on its understanding, these actions influence the environment, and the changes in the environment are perceived and processed by the agent to inform future decisions. This dynamic allows the agent not only to perform tasks but also to adapt and learn from its experiences, enhancing its utility and effectiveness over time.

2.6.2 Key Components of Agent Architectures

To bridge the gap between traditional LLMs and autonomous agents, it is essential to enhance LLMs by providing a suitable agent architecture. Wang et al. propose a framework comprising profiling, memory, planning, and action modules [35].

The Profiling module’s role is to initialize the Agent with its designated role. This requires pre-defining the role and gathering general information about the user, such as personal details and preferences. Creating these Agent profiles can be done manually or by utilizing another LLM to generate them. The profile then serves to instruct the Agent, guiding its behavior and reasoning.

The Memory module serves to store relevant information for the Agent. LLMs often struggle with the lack of up-to-date and use-case-specific information due to their limited context. To address this, when an agent is tasked with a specific role rather than functioning as a general-purpose assistant, it can be augmented with critical, role-specific knowledge. This enhancement can be likened to human biology by differentiating between short-term and long-term memory. Knowledge should be partitioned into short-term and long-term memory based on its frequency of use. Additionally, the agent should be capable of not only retrieving information from its memory but also writing new information into it. This ability allows the Agent to build a long-term relationship with the user or other agents.

Another important aspect of the Memory module is the structure of the stored information. The same principles that apply to the model's context processing are relevant here. Information can be stored in various formats such as natural language, databases, embeddings, or structured lists, each varying in retrieval speed and efficiency.

The Planning module guides the Agent's reasoning by breaking down the use case into smaller subtasks. Planning can be categorized into two types, with or without feedback. In feedback-based planning, the model receives feedback during the planning process and adjusts the plan accordingly. This approach helps the agent develop long-term strategies for tackling complex tasks. Given that the agent acts as a human assistant, it can also communicate with the user to gather human feedback.

Once the agent has produced a plan, the Action module is responsible for executing it and transforming planned actions into specific outcomes. The Action module directly interacts with the agent's environment, including but not limited to the user. To achieve the goals of each subtask, the agent utilizes the tools it is augmented with. The specific tool to be used can either be predetermined in the plan or selected by the Action module. After an action is taken, its immediate impact should be observed and reported back to facilitate evaluation and advance the plan.

Chapter 3

Related Work

The Related Work chapter examines the current landscape of research and applications in the field of autonomous agents, building upon the theoretical foundations discussed in the previous chapter. This comprehensive review begins in section 3.1 with an exploration of agent construction methodologies. Section 3.2 delves into the practical applications of these agents, with a particular focus on two key areas: social simulation and industrial automation. These subsections highlight the diverse ways in which agents are being utilized to solve complex problems and improve processes across different domains. The chapter then addresses the crucial aspect of agent evaluation in section 3.3, discussing the metrics, methodologies, and challenges involved in assessing the performance and capabilities of autonomous agents. This section provides insights into how researchers measure the effectiveness of different agent architectures and implementations. Finally, section 3.4 looks ahead to the future of AI agents, exploring potential post-attention architectures and the ethical considerations that arise as these technologies continue to evolve and integrate into various aspects of society and industry.

3.1 Construction of Agents

The construction of autonomous agents capable of interacting with their environment and completing complex tasks has been an area of significant research in recent years. This section explores various approaches to augmenting Large Language Models (LLMs) with additional capabilities, enabling them to function as more sophisticated agents. From web browsing and visual perception to tool usage and API integration, these advancements aim to overcome the limitations of traditional LLMs and create more versatile and effective AI systems.

Nakano et al. [20] used a web environment that can be navigated by natural language, to enable GPT-3 to browse the web autonomously searching for information. The model was previously trained on tasks in the environment using imitation learning with humans. The model's capabilities were then optimized by human feedback. The models ability to search the web autonomously improved the relevancy of its answers, as the model could use factual and up-to-date information in its reasoning process.

Song et al. [30] adds to this by enabling the agents in their paper to perceive visual input of the virtual environment they are situated in. Those agents having a physical or virtual presence in their environment makes them embodied agents. Their LLM-Planner fosters embodied instruction following. As the state of the environment might now be known before, it is necessary that the plan can be influenced based on observations of the environment made by the agent itself. This makes the plan grounded as it is based on real observations from the physical or virtual environment.

For more general tool learning, Toolformer [28] uses models from the LLaMA subset, which are much smaller than GPT-3. They provide a handful of human-written examples of how to use the API to the model, which then trains itself in a supervised way with a dataset containing tasks where an API call is potentially useful. The tools they provide are a simple QA, calculator, calendar, machine translator, and Wikipedia search tools. In contrast to other approaches of teaching the model to use tools, this approach requires less human supervision and also does not require the model to know the tool call beforehand. However, the dataset they use consists only of one-step tasks, limiting the meaningfulness of their results to achieve similar performance to the much larger model GPT-3.

Ruan et al [27] augmented a LLM with similar tools. Besides the weather query and the Wikipedia tool, they also present the model with the possibility to use generators for Java, SQL, or Python code. This augmented model is then tested on two-step reasoning tasks, which limits the meaningfulness of the results, since real use cases often require more than just two reasoning steps.

Gorilla [23] is a fine-tuned LLM designed to generate input arguments for API calls and mitigate the hallucination problem with external API calls. However, it does not execute real APIs and offers a poor variety of APIs.

Similar to Gorilla, ToolAlpaca [33] also does not call real-world AP. Tang et al. create a dataset that spans different environments to fine-tune a much smaller model than e.g. GPT-4. In this way, they are able to achieve similar performance to GPT-3.5 in generalized tool usage scenarios. Their approach to acquiring and verifying their dataset requires less manual effort than that of ToolLLM.

In a more general sense of augmenting LLMs, TaskMatrix.AI [15] presents an ecosystem that connects basic models with millions of APIs for task completion. Liang et al. present their framework with four different modules. The "Multimodal

Conversation Foundation Model" is responsible for communicating with the user and generating action codes based on understanding the user's goals. The "API Platform" is a platform for users to provide APIs while providing a unified API documentation. The API Selector module then selects relevant APIs. The API Executor then executes the generated action codes by calling the relevant API.

While TaskMatrix.AI is a theoretical framework, Song et al. [31] augment LLMs with APIs following the already existent REST structure. Together with a specialized API executor, this approach improves the performance of LLMs on complex tasks. This approach is much more compatible with real-world applications because the REST architecture and APIs that follow it already exist.

Similarly, Quin et al. have developed the ToolLLM framework [26] to holistically facilitate the use of tools in LLMs. ToolLLM includes data construction, model training, and evaluation. The first step is to build a dataset of about 16,000 APIs from 49 different categories that follow the RESTful API architecture. ChatGPT is then used to automatically generate instructions for these APIs. Additionally, ChatGPT is tasked with creating a valuable solution sequence for each instruction. The model can be fine-tuned to recommend appropriate APIs for tasks by evaluating it with its evaluator ToolEval. In this way, the LLM is even able to use APIs it has not seen before.

Other research focuses more on specific approaches of augmenting LLMs to address the limitations of LLMs. The augmentation of PLMs with knowledge-based Knowledge Graphs (KG) is a research area with a history of investigation. They have already been successfully employed for tasks in NLP, including named entity recognition, relation extraction, and question answering. Since LLMs primarily scale the size of parameters and training data from PLMs, their model architecture and training methods remain largely unchanged. Therefore, the transition from KGPLM to KGLLMs should be feasible. [12] This would provide the same benefits to LLMs, namely greater capability and truthfulness, while offering the advantage of a dynamic knowledge graph.

ToolkenGPT [8] presents another interesting approach to make tool calling more reliable. It mitigates the problems arising from the limited context length, which allows only a few shots of demonstration, leading to suboptimal understanding of the tools. Also, if there are numerous tools to choose from, in-context learning could fail completely. Therefore, ToolkenGPT represents each tool as a token and learns an embedding for it, allowing extensive demonstration data for learning the toolken embedding. This led to a significant improvement in the performance of LLMs in many task categories.

For cases where the user query is of a multimodal nature extended with visual content Gao et al. [7] present AssistGPT. The diversity of visual content in the user query is reflected in the possible reasoning paths and the flexible inputs and intermediate results. For that, AssistGPT includes a planner, an inspector as an efficient memory manager, and a learner that helps to find the optimal solution. Interestingly, the learner records successful explorations for use as in-context examples. The goal of AssistGPT is to dynamically engage various tools to answer multimodal user queries. These tools include image recognition and labeling, region grounding, and temporal grounding.

Finally, another interesting but flawed approach is presented as CREATOR by Quian et al. [24]. Based on the idea that models could create their own tools, they present a framework that allows the LLM to do just that. The framework is divided into four distinct phases. In the creation stage, the first, the model is asked to abstract from solving the given mathematical problem, but instead to create a function that it can later use to solve that problem. In this way, the model is induced to create its own tool. In subsequent steps, the tool is then selected, executed, and refined. However, this approach may be specific to the domain of math problems, as functions for other, more general problems may be much harder or impossible for LLMs to create at this point.

While the construction of these advanced agents represents a significant step forward in AI capabilities, their true value lies in their practical applications across various fields. The following section explores how these augmented LLMs and autonomous agents are being deployed in real-world scenarios, demonstrating their potential to revolutionize multiple domains and tackle complex challenges in ways previously not possible.

3.2 Applications of Agents

Integrating tools into LLM-system, these systems to not only process and generate human-like text but also to perform specialized tasks. Through the exploration of its applications in a variety of fields, insight can be gained into how this technology is transforming various industries and disciplines. This exploration highlights the versatility and potential of tool-augmented LLMs, showcasing their ability to solve complex problems, enhance decision-making processes, and push the boundaries of what's possible in AI-assisted work. Understanding these applications provides valuable context for LLM development, as the needs of different domains can be very unique.

3.2.1 Social Simulation

Recent studies demonstrate the versatility of LLM-based agents in simulating complex social phenomena. ”AgentVerse” [3] introduces a framework for multi-agent collaboration, allowing researchers to study emergent behaviors and group dynamics. It revealed that LLM-based agents can develop sophisticated collaboration strategies and exhibit emergent social behaviors without explicit programming. ”Epidemic Modeling with Generative Agents” [40] applies these technologies to epidemiology, enhancing our understanding of disease spread by incorporating nuanced human behaviors. The study found that incorporating individual agent behaviors led to more accurate predictions of disease spread compared to traditional models. ”Are you in a Masquerade?” [14] explores the potential impact of LLM-driven social bots on online social networks, assessing their influence on information spread and opinion formation. This research discovered that LLM-driven bots could significantly influence online discussions and potentially manipulate user opinions, highlighting the need for advanced detection methods.

3.2.2 Industrial Automation

In the domain of cybersecurity, Deng et al. [6] explore the application of Augmented Language Models (ALMs) to penetration testing, a field that has traditionally resisted automation due to its complexity and the high level of expertise required. Their research assesses the capabilities of ALMs in utilizing the same tools employed by human professionals in penetration testing workflows. While ALMs demonstrate proficiency in specific sub-tasks, they struggle with synthesizing information for long-term strategy formulation, a crucial aspect of effective penetration testing. To address these limitations, the authors introduce a novel architecture comprising three self-interacting modules. This structure aims to mitigate context loss and maintain extended reasoning sequences, similar to the Agent framework discussed earlier (see chapter 2). The architecture includes a Reasoning Module for planning and user interaction, a Generation Module for executing specific sub-tasks, and a Parsing Module for inter-module communication. This approach showcases how tool-augmented LLMs can be adapted to highly specialized and complex domains, potentially revolutionizing fields that have historically relied heavily on human expertise. However, it also highlights the ongoing challenges in developing AI systems capable of the holistic, long-term reasoning required in sophisticated cybersecurity operations.

Theory Proving is a similarly highly specific domain relying heavily on human expertise. However, Yang et al. [43] demonstrate the effectiveness of utilizing ALMs in this domain through their work on LeanDojo and ReProver. Their system leverages an LLM as the core reasoning engine, augmented with a retrieval mechanism to address a key challenge in theorem proving: premise selection. The LLM is used to understand and generate formal mathematical statements and proof steps, while the retrieval component allows it to access a vast library of mathematical knowledge. This retrieval-augmented approach enables the model to select relevant premises from a large corpus, significantly improving its ability to construct valid proofs. LeanDojo extracts data from the Lean theorem prover, including premise annotations, which are then used to train the retrieval component. The resulting system, ReProver, demonstrates superior performance in theorem proving tasks compared to non-retrieval baselines and GPT-4.

In the field of chemistry, Bran et al. introduce ChemCrow [2], an LLM powered Chemistry Agent utilizing computational tools in chemistry. ChemCrow represents one of the first chemistry-related LLM agents capable of interacting with the physical world, capable of successfully planning and synthesizing chemical structures. Beyond this practical application, the system also excels in solving complex chemistry reasoning tasks. When compared to GPT-4, ChemCrow demonstrates superior performance, especially in more complex chemistry problems. This highlights the advantage of specialized, tool-augmented LLMs over generalist models in domain-specific applications. Interestingly, when an evaluator model based on GPT-4 was tasked with assessing answers from both ChemCrow and GPT-4, it showed a preference for GPT-4's responses. Bran et al. attribute this to the more fluent and complete-looking nature of GPT-4's outputs. However, this observation also raises questions about potential biases in evaluation, as discussed earlier in the context of tool augmentation. The shared knowledge base between GPT-4 as an evaluator and as a respondent could influence the assessment, potentially favoring familiar patterns of expression over specialized, tool-derived insights. While ChemCrow demonstrates significant potential, the researchers note that its performance is limited by the quality and quantity of available chemistry tools. This underscores the importance of developing and integrating high-quality, domain-specific tools to fully leverage the capabilities of augmented LLMs in specialized fields.

Ziems et al. [49] explore the integration of LLMs into the Computational Social Science (CSS) research pipeline, evaluating 13 different models across 25 benchmarks. Their study demonstrates that LLMs can effectively serve as data annotators, supporting human annotators and explaining underlying textual attributes across various social science disciplines. The research addresses four key aspects: viability, model selection, domain utility, and functionality of LLMs in CSS. While LLMs show promise in augmenting CSS research processes, they face challenges with technical terminology and large label spaces. The study concludes that LLMs can significantly enhance but not entirely replace traditional CSS methods. This research highlights both the potential and limitations of tool-augmented LLMs in social science.

3.3 Evaluation of Agents

In the history of Natural Language Processing (NLP) many different benchmarks have been developed. However, these benchmarks are not necessarily equipped to assess ALMs holistically. Given the diverse set of capabilities of ALMs and the correspondingly diverse range of potential applications, the development of new benchmarks is a necessity.

The WebShop benchmark [44] focuses on creating scalable, interactive environments for grounding language models in real-world online shopping tasks. It combines Reinforcement Learning (RL) and Imitation Learning (IL) to develop agents capable of understanding text instructions and purchasing products on e-commerce websites. Key features include realistic text and images, a diverse interaction space, and challenges such as search query generation, strategic exploration of connected websites, and robust language understanding. The results of the evaluations demonstrate that the WebShop agents significantly outperform the heuristic baseline and exhibit effective sim-to-real transfer. Despite shifts in product data and dynamics (search engine behavior), the agents achieve comparable performance in real-world settings.

WebArena criticizes other benchmarks, like WebShop, for over-simplifying real-world situations and therefore losing their meaningfulness [48]. To address this weakness, WebArena builds an environment setup with fully functional websites from four common domains: e-commerce, social forum discussion, collaborative software development, and content management. The environment is enriched with tools and external knowledge bases. To ensure reproducibility, the websites used are not live but a standalone environment. However, this again neglects struggles like CAPTCHAs, unpredictable content and configuration changes, that autonomous agents would face in real life scenarios.

By comparing the performance of open-source LLMs against GPT-4 on single- and multi-step tasks, Xu et al. [42] demonstrate a severe gap between open-source and closed models. Xu et al. use Model alignment, In-context demonstration, and system prompts to boost the performance of the open-source models. To evaluate the impact of these methods, they developed ToolBench, consisting of five newly curated datasets and three existing datasets, including WebShop. They also deliver predefined test cases for quantitative evaluation, making ToolBench unique from other tool manipulation research benchmarks. Overall the results of enhancing OSS models with these techniques seems promising, as they were able to significantly lower the gap, even though their results suggest, that there is still a long way to go to reach the performance of closed models. By conducting ablation studies they delve deeper to find that while 3-shot in-context demonstration and model alignment help increase performance across all tasks, the system prompt only helps with simpler tasks with relatively fewer API calls.

Another benchmark that incorporates WebShop into its dataset is AgentBench. Liu et al. [16] bring together a total of eight datasets from the categories code, game, and web, to assess agent's reasoning and decision-making abilities. AgentBench challenges agents in a multi-step openended generation setting. They test multiple commercial models against open-source models, yet to find out that there is a large gap in performance between these two approaches. From the commercial models GPT-4 performs best. To boost performance, the authors suggest introducing code training and alignment training over high-quality data.

Deng et al. focus on the use of LLMs in penetration testing [6]. Their benchmark encompasses a diverse set of tasks varying in difficulty while also tracking incremental progress. As this is what other benchmarks in the same domain lack to closely reflect real-world penetration tests. As a reaction to the struggles LLMs are facing during penetration testing, as presented in section 3.2.2, Deng et al. introduce their own PentestGPT-LLM, which is based on GPT-3.5 or GPT-4 and leverages the interplay of three different LLM modules. In this architecture of LLMs, the reasoning module takes the role of planning the process and dividing it into subtasks. In the process of planning the chain-of-thought (CoT) approach is applied. Each subtask is executed by a single instance of the generation module. Finally the parsing module operates as a supportive interface for exchanging information between the user and the LLM, and between the modules. This strategy leads to substantially better testing results for the PentestGPT-LLM when compared against their own baseline model (GPT-3.5 and GPT-4).

In the context of the Genotopia framework [41] and its Agent sharing platform GentPool, Xu et al. also introduce GentBench. During construction of the dataset, each problem was tested with GPT-3.5-turbo without tool use, and only challenges where GPT-3.5-turbo failed were added to the benchmark. This automatically excludes easier challenges that don't require external tools to solve. GentBench serves as one component of the Gentopia framework, which tries to enable collaborative work with customizable agents. GentBench tries to evaluate Agents holistically by probing performance across diverse dimensions such as, reasoning, knowledge, safety, multilingual capabilities, robustness, memory, and efficiency. To evaluate those agents without the risk of overfitting or evaluation bias, the GentBench is partitioned into a public and private subset. To foster the usage of the Gentopia framework, Agents can only be evaluated with the private challenges, when the agent is to be merged into GentPool.

Another fascinating dimension of LLMs is the existence of social knowledge in the model. SOCKET [4] is a benchmark containing 58 NLP tasks that tests LLMs in humor & sarcasm, offensiveness, sentiment & emotion, trustworthiness, and other social factors. Choi et al. understand social knowledge as a necessity for LLMs to work on tasks, like detecting sarcasm, that require social information. In their tests on the benchmark even big models only performs moderately at best. One interesting observation is, that politeness shows transfer with many of the offensive and hate speech detection tasks in the benchmark.

3.4 Future Direction and Ethical Considerations

This section gives a quick outlook in the potential future of LLMs not based on the transformer architecture and finishes with talking about ethical considerations that should be considered when doing research and working on Large Language Models.

3.4.1 Post-attention Architectures

Since its introduction, the Transformer Architecture has been the state of the art approach for NLP models. However, there is other research on the horizon presenting new architecture types. As attention is the main driver of the Transformer Architecture this new line of work is labeled as post-attention. One class of models following a different approach are the State Space Models (SSMs). Some models following this approach are Mamba and Striped Hyena [19]. They prove to be competitive in terms of performance and efficiency, while also addressing the limitation of context length of models based on the Transformer Architecture. This new category of Models focuses on enabling the models to be able to process more context. This could especially lead to improvements in agents, as more context that is considered by the model should improve their applicability in complex scenarios just as pentesting [6]. Furthermore retrieval-augmented LLMs could benefit from this to a high extent.

3.4.2 Ethical Considerations

The rapid advancement of Large Language Models has also given rise to ethical concerns that researchers must address. In their paper "TrustLLM" [32], Sun et al. present a comprehensive framework for assessing the trustworthiness of LLMs, outlining eight key principles: Truthfulness, Safety, Fairness, Robustness, Privacy, Machine Ethics, Transparency, Accountability To operationalize these principles, Sun et al. introduced the TrustLLM benchmark, which evaluates LLMs based on these ethical criteria. Notably, their research revealed a correlation between LLMs' task performance and their adherence to trustworthiness principles. This finding suggests that ethical considerations are not merely ancillary but may be integral to the overall effectiveness of LLMs. The TrustLLM framework emphasizes the importance of considering ethical implications throughout the development and deployment of LLMs. Therefore these principles should be taken into consideration in the development of autonomous agents. As models and agents become increasingly integrated into various aspects of society, addressing ethical concerns becomes paramount to ensure their responsible and beneficial use.

Chapter 4

Methodology

This section introduces the methodology employed to assess the capacity of augmented LMs to respond to user queries within the context of physical health. The first section 4.1 presents the experiment's design. Followed by the description of the model used (section 4.2), the tools the model is augmented with (section 4.3), and the use cases to test the model's capability (section 4.4). Subsequently, section 4.5 introduces the configurations in greater detail. Finally, section 4.1 and 4.7 describe the process of one run and how the model's performance is evaluated.

4.1 Experimental Design

The objective of this experiment is to evaluate the effectiveness and efficiency of a tool-augmented language model in solving real-life, multi-step user problems within the domain of physical health. Given the rapid scientific progress and discrepancy between scientific knowledge and that of the average person in this field, an Augmented Language Model (ALM) serving as a personal assistant has significant potential for improving people's lives.

In order to reach a meaningful verdict regarding the efficacy and efficiency of ALMs in assisting the user, the experiment investigates the performance of four different configurations over 75 multi-step use cases with authentic user queries. The efficacy of the model is evaluated manually, with the success rate serving as the primary metric for assessment. The efficiency of the model in utilizing tools is assessed by monitoring the selection of tools, the selection of target-oriented arguments, and the generation of the model's response.

4.2 Model

The model employed in the experiment is GPT-4o (version gpt-4o-2024-05-13). At the time of writing it is the most recent model developed by OpenAI, which is capable of performing function calls. Function calling allows the model to be augmented with tools. To foster the communication between the tools and the model, the model is fine-tuned to better understand data in the JSON format.

To access the model for the experiment, the OpenAI ChatCompletion API[1] is utilized. This interface allows developers to integrate the model into the experiment in a manner that is tailored to their specific requirements. This way the model's temperature can be set to a custom value. To ensure the reproducibility of this experiment, the model's temperature is set to 0.

4.3 Tools

To enhance the model's capabilities to solve real life user queries in the domain of physical health, the model is augmented with 15 tools (see Table 4.1). Those tools enable the LLM to acquire domain specific information about workouts, nutrition and recipes. In order to augment GPT-4o with the tools, a description of the tools in JSON including their name, arguments, and the output is passed to the model.

When the model decides to call a tool, it collects the necessary arguments and returns the tool call formatted in JSON including the functions name and the arguments. This information is used to call the function that was independently implemented. Therefore, the model does not call the tool directly but provides the necessary information for the function to execute the tool call. In order to solve the use cases correctly, the model needs to chain multiple tool calls together, as the reasoning requires multiple reasoning steps chained together.

To illustrate the process of chaining multiple tool calls, a possible scenario is where the model needs to retrieve nutritional information for a specific ingredient. This task requires two sequential tool calls:

1. First, the model calls the 'get_ingredient_id' tool, which takes the ingredient's name as a string input. This tool returns the unique identifier for the ingredient in the database.

2. Using this identifier, the model then calls the 'get_ingredient_nutrition' tool. This second tool requires three arguments: the ingredient ID (obtained from the previous step), the amount of the ingredient, and the unit of measurement.

[1] `https://platform.openai.com/docs/guides/chat-completions`

Name	Parameters	Returns
get_longitude_and_latitude	city, country	longitude, latitude
get_temperature	longitude, latitude	temperature
get_workout	muscles, intensity_level, equipment	list of exercises
get_actor_id	name	actor_id
get_movie_by_actor_id	actor_id	movie
get_macros	age, gender, height, weight, activity_level, goal	nutritional information
get_calculation	a_number, b_number, calc_type	result
find_by_nutrients	calories, protein, carbs, fat, number	number of recipes
recipe_search	query, add_recipe_nutrition, cuisine, recipe_type, number, include_ingredients, exclude_ingredients, show_ingredients, offset	number of recipes
extract_recipes	url	recipe information
search_ingredient	query	ingredient_id
get_ingredient_nutrition	ingredient_id, amount, unit	nutritional information
convert_amounts	ingredient_name, source_amount, source_unit, target_unit	converted amounts
post_to_shopping_list	item	-
get_items_from_storage	-	list of items in storage

Table 4.1: List of Tools with their Arguments and what they return to the Model

The later tool then returns comprehensive nutritional information for the specified amount of the ingredient.

This example demonstrates how the model must strategically chain tool calls to gather the necessary information. The output from the first tool becomes a crucial input for the second, enabling the model to provide accurate and detailed nutritional data to the user. Such multi-step processes are common in addressing complex problems by using multiple tools.

To maintain experimental stability when utilizing external APIs for information retrieval, robust error handling for each tool call that integrates an external API has been implemented. This ensures that if any issues arise during API interactions, the model is notified and can respond appropriately. In cases where the model misuses an API, the resulting error message is relayed back to the model, allowing it to adjust its approach and reattempt the tool usage to obtain the desired output. This error-handling system enhances the model's resilience and adaptability.

With these tools and error-handling mechanisms in place, it can now be explored how they can be applied in practice through a series of use cases.

4.4 Use Cases

The model's performance is tested on how effectively and efficiently it is capable of solving the 75 use cases (see Appendix A) presented to the model. The structure and characteristics of the use cases, the method of creating them, and the categorization is explained in this section.

> ”Today I had two servings of Crepes Suzette for breakfast, how much percent of my daily calories did I already eat? To cover **between 15 and 25 percent** of the remaining calories please provide me with a recipe of the same cuisine as Crepes”

Figure 4.1: User Query of Use Case 43

Each use case is characterized by one distinct user query that is drawn from real-life scenarios. The user's requirements that can be derived from the user's query constitute the ground truth of the use case. The solution sequence illustrates a singular execution workflow of how all necessary information can be acquired and actions can be taken to successfully solve the use case. All 75 use cases are multi-step use cases, meaning that multiple reasoning steps together with the appropriate tool call need to be chained together to align with every user requirement defined in the ground truth. On average the model would need four tool calls to solve the use case correctly.

The creation of the dataset with the use cases can be broken down into multiple steps:

1. **Tool Selection**: 15 tools that provide the model with domain specific knowledge and capabilities were selected

2. **Initial Use Case Creation**: Leveraging the potential of combining multiple tools, 25 unique use cases that require multi-step reasoning and tool use were created.

3. **Baseline Evaluation**: To assess the potential impact of tool augmentation, ChatGPT (a non-augmented model based on the related model GPT-4) was tasked to solve the initial use cases. By comparing ChatGPT's answers to the ground truth and solution sequence of the initial use cases, this step provides an idea of the room for improvement that tool augmentation could offer.

4. **First Iteration**: To acquire more information on the potential of using tools, GPT-4o augmented with the 15 tools was tasked to solve the use cases.

5. **Use Case Refinement**: Based on the insights gathered by the execution of the use cases by ChatGPT and GPT-4o, all 25 use cases were fine-tuned, adjusting their difficulty, the necessity of tool usage, and the potential for improvement through tool augmentation.

6. **Expanding the Use Cases**: To increase the statistical significance when using the dataset, the use cases were slightly altered while keeping the adjusted base versions. This includes changes to the formulation of the user queries and the solution sequences. That way the number of use cases was expanded from 25 to 75.

This robust approach to use case development provides a solid foundation for the experiment, aligning closely with the research objective in assessing the performance of an ALM in the field of physical health.

Subset	Name	Use Cases included
1	Temperature-Based Decision Making	1 to 9
2	Integrated Workout and Nutrition Planning	10 to 18
3	Activity Level Adjustment	19 to 27
4	Breakfast-Based Nutritional Planning	28 to 43
5	Dinner with Friends with dietary Restrictions	44 to 52
6	Multi-Day Meal Planning	53 to 61
7	Storage-Based Nutrition	62 to 75

Table 4.2: Use Cases divided into relevant Subsets

To facilitate a better understanding of the subdomains present in the dataset, the use cases can be classified into seven subsets (see Table 4.2).

1. Temperature-Based Decision Making: This subset requires the model to recommend either an outdoor workout or a movie to watch starring a specific actor based on the current temperature at the user's location.

2. Integrated Workout and Nutrition Planning: Here, the model must provide both a workout recommendation and a recipe that meets the user's daily nutritional requirements.

3. Activity Level Adjustment: These cases challenge the model to adapt the user's activity level based on specific scenarios. For instance, despite a general activity rating of 3 out of 5, a day spent sitting at a desk might necessitate

adjusting the activity level to 1, consequently affecting daily macronutrient requirements influencing the recipes that the model should provide.

4. Breakfast-Based Nutritional Planning: This subset presents various breakfast scenarios, requiring the model to adjust the nutritional recommendations for the remainder of the day accordingly.

5. Dinner with Friends with dietary Restrictions: These cases involve planning a dinner for friends, requiring the model to extract ingredients from a given URL and account for specific dietary restrictions or eating habits.

6. Multi-Day Meal Planning: This subset tasks the model with creating comprehensive meal plans spanning several days, incorporating multiple dietary restrictions and preferences.

7. Storage-Based Nutrition and Recipe Recommendations: The final subset prompts the model to utilize information about items in storage, either to calculate their nutritional values or to recommend recipes based on available ingredients.

Each subset is designed to test the model's ability to integrate multiple tools, adapt to varying scenarios, and provide comprehensive solutions to complex, real-world health and nutrition queries.

Personal Information

Martin lives in Mannheim, Germany. He is 22 years old, male, 175 centimeters tall, and weighs 70 kgs. Martin does not like to eat the same food twice a day. Martin is moderately active and his goal is to maintain his weight and muscles. He is a beginner at working out.

Figure 4.2: Personal Information about the User Martin that is appended to every System Prompt to inform the Model

The majority of the information required to successfully resolve the use case is provided by the user's query or the tool calls. Nevertheless, general personal information about the user Martin that is not intended to be communicated to the model for each use case is provided in the system prompt. The primary function of the system prompt is to provide instructions to the model regarding its behavior. This concept will be further elaborated upon in the subsequent section.

4.5 Configurations

In addition to carrying general information about the user, the model's role and behavior in the reasoning process is instructed by the system prompt. The model is instructed by the system prompt one step before it receives the user query. The four configurations tested in the experiment mainly differ by the system prompt used to instruct the model.

CoT+

You are a personal assistant to Martin. **Work out a plan to use tools effectively** to help Martin and present your plan to Martin. **Then carry out the plan step by step**.

Figure 4.3: System Prompt to elicit Reasoning similar to zero-shot CoT+ but with a Focus on Incorporating Tools

The first configuration extends the idea of CoT (as introduced in chapter 2) to adapt it to the new possibilities the model gathered by augmenting it with tools. While the basic CoT prompt adds "Let's think step by step" to the end of the system prompt to foster step by step reasoning, the new configuration "CoT with Tools" (CoT+) adds the new dimension of using tools to enhance reasoning, by telling the model specifically to include tool calls in its reasoning process. The enhanced system prompt for CoT+ is shown in Figure 4.3.

Planner

You are a planner. Your role is to generate a plan that solves the user's request. The plan should specify multiple tool calls that lead to a final answer that complies with the user's requirements. You are not supposed to execute the plan, just return the plan! The user is Martin. [personal information about Martin].

Figure 4.4: System Prompt to instruct the Planner Module

The second configuration, "Planning and Solving" (PS), while also prompting the model to include tools in the reasoning process like CoT+, splits the reasoning in two parts by initiating two separate iterations of the model with two different system prompts. First, the Planner module is instructed to formulate a Plan including multiple Tool Calls providing information or action that are vital to comply with the user's requirements. This adheres to the multi-step characteristic of the use cases. Then the second iteration, the Solver module, is tasked to carry out that plan by following the described steps and function calls. The Solver is again reminded to pay extra attention to complying with all user requirements described in the user query.

> **Solver**
>
> You are a Solver. Your role is to execute a plan that is provided to you. The plan includes multiple tool calls that support you getting all the information necessary to generate a final answer. The final answer has to comply to all of the user's requirements. [personal information about Martin].

Figure 4.5: System Prompt to instruct the Solver Module

During the creation of the dataset in the first iteration of the experiment, the Planner module often tried to not only create a plan but to solve the use case by itself. To prevent this, two measures were taken. First, the system message explicitly states again, that the planner is not supposed to execute its plan, but just to return it. Further, when the user query is input to the model, "Please provide a high level plan for how this user query could be solved." is set before the user's query. These two steps did lower the frequency of the Planner also executing the plan drastically.

The third configuration, "Self Reflect" (SeRe), also adds an additional iteration to the CoT+ approach. First the model is instructed to generate a final answer by also using the CoT+ system prompt (see Figure 4.3). Then however, the Reflect module is instructed by the Reflect system prompt (see Figure 4.6). The Reflect module is tasked to evaluate the response generated by the first CoT+ iteration. Reflect is specifically prompted to evaluate the first model's performance based on its choices of using tools effectively to generate the final answer, and the response's compliance with the user requirements. Based on the evaluation, Reflect is instructed to either return "sufficient", if the first model's response was sufficient. Or generate a plan on how to improve upon the response and carry it out.

By appending another layer of reflection to the CoT+ configuration, it is assumed that this will lead to mistakes by CoT+ being detected and resolved, and therefore lead to an increased Success Rate in comparison to CoT+.

The fourth configuration combines PS and SeRe and is therefore called "Planing and Solving and Self Reflect" (PS + SeRe). The process starts with the Planner module, instructed with the same prompt, formulating a plan. Then this plan is

Reflect

Your role is to evaluate the answer to a user's query by another model. You are handed the user's query, the tool calls the model used to answer to the user, and the final answer. Your job is to evaluate the model's performance in using tool calls effectively to generate a final answer, compliant with the user's requirements. You can assume that the result of each function call with the specific arguments is correct, and repeating it won't change the result. If you find the performance to be sufficient, only answer with 'sufficient'. If not, make a plan on how to comply with the user's requirements better and carry it out step by step. [personal information about Martin].

Figure 4.6: System Prompt to instruct the Reflect Module

carried out by the Solver module. With the addition of SeRe to PS, now the Reflect module is appended to the PS approach. When reflecting on the response of the Planning and Solving part of the process, the Plan is also present in the Reflect module's context. The Reflect module again either returns "sufficient" or tries to improve upon the initial response by generating a plan and then carrying it out. By combining both PS and SeRe into one configuration, the aspiration is to further improve the model's performance.

4.6 Experimental Procedure

This section describes an example run using the Self Reflect configuration as shown in Figure 4.7. First the model is augmented with the 15 tools. For this specific use case only a subset of 4 different tool calls are required to generate a response complying to the user's requirements specified by the user query as in Figure 4.1.

With the first message in the conversion history, the system prompt, the model is instructed on its role while also providing general information about Martin, the user. Equipped with the general user information, instructions, and information retrieval and reasoning capabilities, the model now decides internally about the tool calls that could help the model at generating a sufficient response. The first objective is to get the nutritional needs of Martin by using the get_macros tool. Parallely, the model calls the recipe_search tool to get the nutritional information about Martin's breakfast. Now that the model knows the total amount of calories

Martin needs per day and the caloric value of his breakfast, it can calculate the range of calories for the recipe to recommend. By then calling the recipe_search function again, but with another set of arguments, the model can acquire multiple recipes from the French cuisine and select one falling in the caloric range calculated before. For the detailed solution sequence refer to Appendix A.4 Use Case 43. After the first response is generated, it is immediately passed to the Reflect module. If the first response complies to all requirements no further action is initiated by the Reflect module. However if it does not, the Reflect module tries to improve upon the first answer.

To enable evaluation of the run presented above, the model's response to the user, every tool call initiated together with the arguments and the output, error messages, and the tokens used are logged.

4.7 Evaluation Process

As the model should serve the user as a personal assistant with human-like features, the evaluation of its performance is challenging. To evaluate the model's performance on the real-life inspired dataset, a lot of consideration has to go into deciding on the model's success in complying to the user's requirements. Additionally, there is not only a singular solution sequence that leads the model to generating a sufficient response. That is why the evaluation of the configurations performance to calculate their Success Rate (SR) has to be done manually.

Another important aspect of judging the model's performance is its efficiency while reasoning to generate the final response. To foster the comparability of the four configurations in the dimension of efficiency, the model's token usage is tracked during the experiment. Even though the manual evaluation form limits the extent of the experiment, it further enables the evaluator to get a deep understanding of how the model works on the challenges.

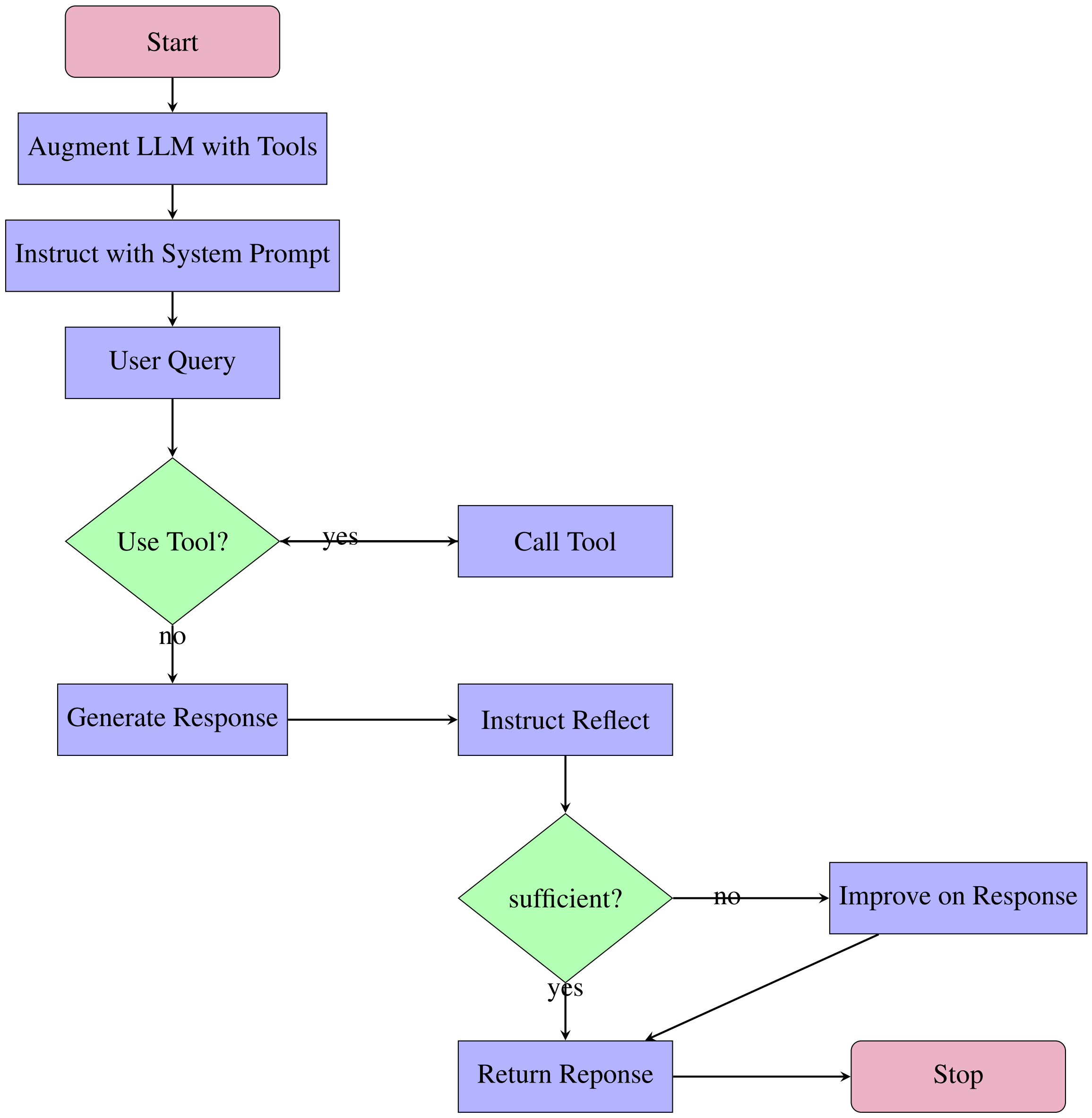

Figure 4.7: Process of the Self Reflect Configuration

Chapter 5

Experimental Evaluation

This chapter presents and discusses the results of the experiment. The first section 5.1 introduces the results from the first iteration used to further improve the use case dataset and the configurations. In Section 5.2 the setting of the experiment is shortly repeated and the Results are displayed shortly after in Section 5.2.1. The following Section 5.2.2 presents the result with adjusted baselines and in Section 5.2.3 shows the adjusted results in relation to the subset introduced previously. After presenting all results, Section 5.3 compares the configuration's performances over the experiment and Section 5.4 finally provides a detailed Error Analysis.

5.1 Initial Iteration

As previously elicited in chapter 4, the first iteration of the experiment was conducted to gather more information about the model's and the configuration's performance on solving the initial 25 use cases. This new intel was then used to revise and further enhance the use cases and configurations. Figure 5.1 presents the results of the first iteration. It shows the Success Rate (SR) of solving the use case correctly of three different configurations. The success rate is the percentage of correctly solved use cases out of all use cases. The initial iteration tested the model in the ChatCompletion API without and with CoT prompt and the Assistants API with CoT prompt configuration. The highest SR scores the configuration using the ChatCompletion API together with the CoT prompt. The only difference to the worst configurations is the use of the CoT enhanced prompt. The usage of the Assistants API together with the CoT prompt lead to a decrease in SR of 4%.

The information on the difficulties the model faced when solving those use cases were used to create more variations of the initial use cases to increase the size of the dataset. Furthermore the two worst performing configurations, Chat-

Configuration	**Success Rate**
ChatCompletion API without CoT prompt	40%
ChatCompletion API with CoT prompt	48%
Assistants API with CoT prompt	44%

Table 5.1: Initial Results of Testing GPT-4o in three different Configurations on 25 Use Cases

Completion API without CoT and Assistants API with CoT were discontinued and the best performing configuration, ChatCompletion API with CoT was kept as the new baseline. Additionally the simple CoT prompt was further refined to align more with the tool calling capabilities of the tool-augmented language model resulting in the CoT+ prompt and configuration.

5.2 Main Experimental Results

The main experiment focuses on testing the performance of GPT-4o on 75 multi-step use cases by employing four configurations:

1. **CoT+**: Chain of Thought with a Focus on Planning with Tool Incorporation
2. **PS**: Planning and Solving
3. **SeRe**: Self Reflect
4. **Ps + SeRe**: Planning and Solving + Self Reflect

The following sections first present and analyze the results of the experiment. Then the four configurations are compared. Finally, an error analysis is performed.

5.2.1 Results

Configuration	**Success Rate (SR)**	Δ
CoT+	30.67%	-
PS	30.67%	-
SeRe	46.67%	+16.00%
PS + SeRe	38.67%	+8.00%

Table 5.2: Results of the Main Experiment presenting the Success Rate of the four Configurations

Figure 5.2 displays the results comparing the performance of the four configurations. While CoT+ serves as the baseline for the experiment and the development of the other three configurations, both CoT+ and PS scored the same SR of 30.67%. Next in line, SeRe performed better on the dataset than both previous configurations, scoring 46.67%. The fourth configuration PS + SeRe manages a SR of 38.67%. These results clearly single out the SeRe approach of adding an additional reflection step to the CoT+ prompt as the best performing configuration with a gap to the second best of 8%.

To better understand the 16% gap between Self Reflect and its base prompt CoT+, the effectiveness of the Reflect step is analyzed. In the first step of SeRe, where the model is asked to solve the problem using the CoT+ prompt, it can be observed that in 7 out of 12 use cases that SeRe solved correctly compared to CoT+, the first step already solved the use cases correctly, leading to the correct answer. This correctness is attributed to the randomness present in the model and not to the second reflection step. In only 5 of the 12 cases did the configurations rely on the reflect step to improve the initial incorrect answer from CoT+ and still return the correct answer. After this discrepancy was discovered, the same analysis was performed for the PS and PS + SeRe configurations. In this case, 4 use cases could be attributed to randomness.

5.2.2 Adjusted Results

Configuration	**Success Rate (SR)**	**Adjusted SR**	**Δ**
CoT+	30.67%	33.33%	+2.66%
PS	30.67%	33.33%	+2.66%
SeRe	46.67%	42.00%	-4.67%
PS + SeRe	38.67%	37.33%	-1.34%

Table 5.3: Results after adjusting the Baseline Configurations

To address the issue of the baseline models (CoT+ and PS) randomly performing better when being part of the more advanced configurations, the Success Rates of all four configurations are adjusted to take randomness into account. First the performance of the CoT+ and PS configurations are separated from their respective advanced configurations (SeRe, PS + SeRe) by extracting their performance before the Reflect step. Together with their respective baseline performance (CoT+ and PS) a new average is calculated. This new base average is then used as the SR for the baseline configurations (CoT+ and PS). To get the adjusted SR for the advanced configurations, the use cases were the Reflect Step led to a correction over the intermediate response are added back on.

Subset	Use Cases	CoT+	PS	SeRe	PS + SeRe
1	1 to 9	**89%**	72%	**89%**	72%
2	10 to 18	11%	**28%**	11%	**28%**
3	19 to 27	0%	0%	**22%**	11%
4	28 to 43	41%	25%	**53%**	31%
5	44 to 52	11%	**22%**	11%	**22%**
6	53 to 61	17%	22%	**28%**	22%
7	62 to 75	**61%**	54%	**61%**	**61%**
	average	35%	33%	**42%**	37%

Table 5.4: Adjusted Success Rate across the Subsets defined in 4.2. Notable Performances are highlighted.

The adjusted Success Rates of the four configurations are shown in Figure 5.3. Additionally the delta between the non adjusted and the adjusted SR is also stated. While the adjustment of the SR did not change the order of the configurations in regards to the SR, the gap between them decreased notably. SeRe is still the most successful configuration with a SR of 42%. The gap towards PS + SeRe decreased by 1.34% resulting in a Success Rate of 37.33%. Again, the PS approach did not yield a higher SR when compared to the overall baseline of CoT+. Both achieve the same SR of 33.33% and improve after the adjustment by 2.66%. In addition to mitigating the impact on performance difference caused by pure coincidence when comparing the advanced configurations to their respective baseline configurations, this increase in sample size for the baseline configurations also increases the meaningfulness of their SR.

5.2.3 Adjusted Results by Subset

To further enhance the understanding of the model's behavior when facing a diverse set of user queries and therefore very diverse user requirements, this section presents and discusses the adjusted results in relation to the seven subset introduced in section 4.2. The Success Rate of the four configurations with adjusted baselines by subsets is displayed in Table 5.4

While SeRe archives the highest average Success Rate of 42% across all use cases, SeRe also scores the highest SR in five out of seven subsets. In Subset 4, SeRe clearly outperforms the other configurations by scoring a SR of 53%. This represents a gap of 12% towards CoT+, and 22% to PS + Sere and 28% PS. In order to illustrate the superior performance of SeRe, two runs of use cases belonging to subset 4 are presented.

Query: ”Today I had two servings of Crepes Suzette for breakfast, how much protein do I still need to eat today?”
Solution Sequence:

1. recipe_search (query: Crepes Suzette, add_recipe_nutrition: True)
   −> ... 11.49g of protein ...
2. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain)
   −> ... 159g of protein ...
3. potentially calculations

Ground truth 136.02 g of protein

Figure 5.1: User Query, Solution Sequence, and Ground Truth for Use Case 29

Use Case 29: The user query and the associated ground truth as shown in Figure 5.1 constitute the requirements for the model. The initial step is to ascertain the protein value of two servings of Crêpes Suzette. Subsequently, the daily protein requirement for the user must be determined. This data is then employed in the calculation of the protein value that the user is required to consume on that day.

In both instances where the CoT+ approach is used to generate an answer (CoT+ and SeRe), the CoT+ iterations failed to utilize the designated recipe_search function. Instead, they treated ”Crêpes Suzette” as an ingredient and consequently employed the get_ingredient_nutrition-tool. This does not yield a result, as ”Crepes Suzette” is not, in fact, an ingredient. Subsequently, both CoT+ iterations refrain from attempting any further actions and instead solicit the user’s input regarding the desired course of action.

In contrast, the two configurations utilizing the Planner module, operate in the opposite manner. In both instances, the Planner module elicits the consequent Solver module to utilize the recipe_search function to get the information on the protein value of two servings of Crepes Suzette. This is a stark contrast to the CoT+ based responses, and it can be assumed that the Planner’s distanced view of the matter helped it to recognize the true nature of ”Crepes Suzette” being a recipe. Both PS basedconfigurations, PS and PS + SeRe, proceed with the plan and fulfill the use case correctly. However, after the Reflect model evaluates the intermediate response, it correctly determines that the response is not sufficient and generates a plan to use the recipe_search function as an alternative. This results in the SeRe configuration providing an accurate reflection and returning the appropriate response. In this case, SeRe effectively addresses a limitation of the CoT+ approach.

Tool Calls

1. CoT+: recipe_search (cuisine='French', add_recipe_nutrition=True)
   −> Baked Ratatouille with 1031 calories
2. PS: recipe_search (... query='Main Course')
   −> Classic French Onion Soup with 412 calories
3. SeRe: recipe_search (cuisine='French', add_recipe_nutrition=True)
   −> Baked Ratatouille with 1031 calories
4. Reflect: recipe_search (... number=5)
   −> Creamy Ratatouille Over Penne with 308 calories
5. PS + SeRe: recipe_search (... query='dinner')
   −> Classic French Onion Soup with 412 calories

Ground truth French recipe with between 198 and 331 calories.

Figure 5.2: Tool Calls by the four Configurations gathering Information to reply to the User Query of Use Case 43

Use Case 43: This use case already served as an example in the methodology part (see section 4.4). The objective here is to calculate the percentage of the daily calorie intake that would remain after consuming two servings of the "Crepes Suzette" breakfast. Moreover, a recipe must be recommended that falls within the calculated calorie range of 198 to 331 calories (15% to 25% of the total calories consumed), and that also belongs to French cuisine. All configurations' recipe recommendations, with the exception of SeRe, fail to comply with the specified range of calories. The two unique recipes recommended exceed the specified limit. The same is true of the intermediate result produced by the SeRe configuration.

The Reflect step detects this error and prompts the model to call the function 'recipe_search' again, this time setting the 'number' argument, which indicates the number of recipes the function should return, to 5. With the objective of obtaining a more diverse set of recipes, this approach increases the likelihood of a recipe being returned that aligns with the desired caloric values. The same result could have been achieved by utilizing the 'offset' parameter and setting it to 1, thus bypassing the initial result. It can be observed that the three configurations that fail to achieve the desired outcome do not appear to exhibit a high degree of flexibility in their utilization of the tool. Instead, they appear to adhere to a relatively inflexible reasoning process. Nevertheless, in the event of an unsuccessful outcome, the approach remains unaltered, resulting in the generation of the final answer, regardless of its alignment with the user's requirements.

Query: ”I need a meal plan for the next three days. In the next three days I want to load up on carbs so I want to eat 20 percent more carbs than I normally would. I don't want to eat the same lunch or dinner twice. For this time it is most important to me to reach my carbs goal, less important to meet the rest of the macros. I am also willing to eat not only three times a day. Just make sure my carbs are met.”
Ground truth: Meal Plan supporting 20% more carbs with not recommending the same lunch or dinner twice.

Figure 5.3: User Query and Ground Truth for Use Case 58

Similar behavior can be observed when examining use cases from other subsets as well.

Use Case 58: In this use case from subset 6, the user seeks a meal plan that will enable him to achieve his objective of consuming 20% more carbohydrates than his typical intake, while also ensuring that the same lunch or dinner is not repeated in the span of the meal plan (see Figure 5.3). The CoT+ configuration is able to identify three recipes with high carbohydrate values that satisfy the increased carbohydrate requirement, and distributes these across the three days. Consequently, the three recipes are repeated on each of the three days, thereby failing to meet the second requirement. In contrast, PS follows its plan of finding a diverse set of recipes while accepting the repetitive usage of the recipe_search tool. However, the meal plan it returns is just a list of the searched-for recipes, without the nutritional values of the recipes. Consequently, when the Reflect step assesses the meal plan, it does not identify the nutritional values as absent and deems the meal plan to be sufficient.

It can be assumed that had the initial response included inadequate nutritional values, the Reflect module would have identified this deficiency. Nevertheless, this reveals a limitation of the Reflect step. In the event that the preceding model fails to incorporate essential information into the intermediate response, the Reflect model may prioritize the content of the first answer more than it should, failing to consider the preceding steps with the requisite degree of attention.

This, however, is not the case with SeRe. The intermediate answer is identical to the CoT+ one, but it identifies a deficiency in the diversity of recipes. To enhance the quality of the answer, the Reflect model searches for additional recipes and distributes them across three days to ensure the inclusion of a meal plan that aligns with the user's requirements. This illustrates a scenario where the PS approach, employed in the generation of the intermediate answer, can impede the Reflect model's ability to fulfill its intended function. This offers a potential explanation for the observed superiority of the SeRe configuration, which appears less complex at first glance, over the Ps + SeRe variant, particularly when considering the entire dataset and its constituent subsets.

Query: "How many days could I maintain my weight if i don't go shopping?"
Storage: fruit, dairy and nuts
Solution Sequence:

1. get_macros (age=22, gender='male', height=175, weight=70, activity_level=3, goal='maintain')
   −> calorie: 2564, protein: 159, fat: 75, carbs: 314
2. get_items_from_storage ()
   −> items in storage
3. for all 9 ingredients search_ingredients (query='item')
   −> ingredient_id of item
4. for all 9 ingredients get_ingredient_nutrition (ingredient_id)
   −> calorie, protein, fat, and carbs value of ingredient
5. potentially calculations

Ground truth: 4.47 days

Figure 5.4: User Query, Storage Information, Solution Sequence and Ground truth of Use Case 73

In testing the PS + SeRe approach against the 75 use cases, the Reflect module correctly transformed three instances of an initially incorrect intermediate answer by PS into a final correct answer. One illustrative example of this can be observed in the following use case.

Use Case 73: In this use case, PS generates its final answer by gathering all the necessary information to make a plan, and then taking the appropriate steps. However, in responding to the user's query, the system incorrectly inputs the daily macro requirements of the user as the added macros of all items in storage. This results in the response to the user query being precisely one day, which is inaccurate with respect to the ground truth of 4.47 days. Such an outcome may be achieved by aggregating the nutritional data for all items in storage and subsequently dividing the calorie, carbohydrate, fat, and protein values by their respective daily requirements. Both CoT+ based configurations achieved the correct answer from the get go. While this was not the case for PS modules, in the PS + SeRe configuration that error was correctly identified by the Reflect step. The Reflect module made the correct plan and carried it out to finally return the correct answer. Once again the extra step of the Reflect module, this time with PS as the basis, turned out to be crucial for providing a response complying with the user's requirements.

The considerable enhancement in performance exhibited by configurations augmented with the Self Reflect step in comparison to their individual baseline of CoT+ and PS is a highly encouraging outcome. In these instances, the Self Reflect step was correctly identified as the source of an error in the intermediate answer. A plan was devised to rectify the issue and subsequently executed with success.

5.3 Comparison of Configurations

This section will present a summary of the previously presented results and contextualize them in relation to the efficiency of the four configurations in achieving their respective success rates. After adjusting the results to account for the larger sample size of the base configurations (CoT+ and PS), SeRe achieves a success rate of 42%. As evidenced by the concrete examples of how the Self Reflect step enhances the performance of the model in accordance with user requirements, SeRe represents a promising approach to improving the accuracy of the model's response

Configuration	**completion_tokens**	**prompt_tokens**	**total_tokens**
CoT+	53.505	852.280	905.785
PS	109.748	1.159.449	1.269.197
SeRe	84.036	1.158.146	1.242.182
PS + SeRe	138.355	1.627.838	1.766.193

Table 5.5: Token Usage of the four Configurations during the Experiment

Table 5.5 provides a further dimension of analysis by displaying the completion and prompt tokens used by each configuration in the experiment in relation to the token usage of CoT+. The context provided to the model constitutes the prompt tokens. Completion tokens represent tokens utilized by the model to progress from the prompt tokens towards the generation of a response. Furthermore, this encompasses all tokens utilized for tool calls.

The consistent rise in token usage across configurations demonstrates that incorporating additional steps does, in fact, result in increased token usage. When utilizing GPT-4o via the API, the pricing is calculated on a token basis. Consequently, an increase in the number of tokens associated with a given configuration will result in a proportional increase in the associated cost.

The incorporation of the Planner model as an additional phase within the PS configuration process results in a 105.12% increase in completion tokens used. This is anticipated, as the amount of steps double and so do the amount of tokens used more or less.

Nevertheless, the introduction of a second step that assesses the initial response with the SeRe configuration merely results in an increase of 57% in completion tokens in comparison to CoT+. The comparatively minor increase for SeRe in comparison to PS can be attributed to the fact that the Reflect model of SeRe does not always necessitate the generation of an entirely new answer. Instead, it may simply declare the preceding answer to be sufficient, thereby requiring a drastically reduced number of tokens. For instance, no tool calls are required for this classification. This is consistent with the incorporation of the Self Reflect concept into PS in PS + SeRe, where the increase in completion tokens is 158.58%.

As proven in previous sections, the SeRe configuration demonstrates the greatest improvement in performance, while also presenting the lowest increase in token usage in relation to the CoT+ configuration. This further supports the assertion that the SeRe approach is a promising method of increasing performance. However as the success of 42% is far from perfect, the next section discusses some common pitfalls that the SeRe suffers from.

5.4 Error Analysis

This section provides a more detailed analysis of the errors that occurred in the experiment conducted with the SeRe configuration. First, nine error categories are defined, classified by the function and parameter selection and the final response. Subsequently, the categorized errors are presented, divided into the previously introduced subsets (see Table 5.6). Then, the error types that occurred with notable frequency are discussed in greater detail by providing specific examples.

5.4.1 Error Categories

This section introduces the error categories that were used to classify the 96 total errors that occurred in the SeRe configuration. One use case can be classified as multiple error types.

Incorrect Function Selection

There are 29 errors that fall in the category of incorrect function calls. By going into detail on what the SeRe configuration struggles with, three categories were defined.

1. **Incomplete Solution Sequence (ISS)**: This error category comprises cases where the model failed to give the correct answer, as one of the solution sequences of calling functions was not followed. This leads to missing information in the final response of the model.
2. **Wrong Function Call (WFC)**: This category includes cases where the model chose to call a tool that does provide any progress towards generating the final answer. The model's reasoning behind choosing this tool can't always be clearly pointed out.
3. **Inefficient Function Calls (IFC)**: This comprises errors where the model called functions repetitively without any variation of parameters or other information leading to this decision.

Incorrect Parameter Selection

There are a total of 26 errors that can be led back to incorrect parameters. The following two categories are used to further describe those cases in more detail:

4. **Sticking to failing Parameters (SFP)**: This category includes errors where the model kept on calling the same functions with the same parameters without any deviation. This was especially the case when the Reflect model tried functions the previous model already called again to check them. This type of error was already very noticeable in pre-testing and the system prompt of the Reflect model therefore specifically calls for the model to see the function calls as definitive.
5. **Incorrect Parameter Selection (IPS)**: An error is put into this category when the model selects parameters that don't fit in any possible solution sequence, or meets false decisions on parameters based on the context present at the moment of selection.

Incorrect Response

There are a total of 37 errors falling in the category of incorrect responses. The following four categories have been established to further divide those to foster a deeper understanding.

6. **Reflect Loop (RL)**: In these cases the Reflect model is stuck in a loop of improving the previous answer. This loop leads to the model at one point stop generating and not emit a sufficient response.
7. **Misinterpretation of User Requirements (MUR)**: In these cases the model misinterprets the user's query, not understanding what the user's intent is.
8. **Reliance on previous Knowledge (RPK)**: This category includes errors where the model relies on its own knowledge on the matter not trying to get more specific knowledge with function calls enough.
9. **Ignoring Requirements (IR)**: This category encompasses instances where the model's inability to comprehend the user's intent, whether intentional or unintentional, results in the model's failure to fulfill a requirement specifically stated by the user in the query or the system prompt. This is also the case when the model claims to fulfill all requirements while the response does not.

5.4.2 Analysis

This section provides a more detailed analysis of some of the most common errors that occurred in the SeRe configuration. The overall frequency of error types is presented in Table 5.6.

The first notable error type that occurred often in the second subset is the ”Sticking to Failing Parameters” error. In all eight instances in which this error occurred, the model was prompted to recommend one or more exercises that either focus on the upper body, the entire body, or muscles in the upper body. The three variations of the prompt were employed in an attempt to prompt the model to utilize the tool, get_workout. The parameter ”muscle,” however, is defined to the model as a specific muscle that the exercise returned should target. The solution sequence employs this information to target specific muscles, such as the biceps, chest, or legs, based on the formulation indicated in the user query. The model however without exception selected the three prompt variations mentioned before as the muscle parameter. This resulted in the tool failing to return any exercise. The Reflect model also exhibited this issue, and still executed the same tool call again, thereby perpetuating the failure of the parameters. In five of these eight

		Function			Parameter		Response			
subset	#	ISS	WFC	IFC	SFP	IPS	RL	MUR	RPK	IR
1	1	0	0	1	1	0	0	0	1	1
2	8	0	0	0	8	1	0	0	5	1
3	7	7	0	4	0	3	0	0	0	7
4	7	0	7	2	4	1	2	1	2	0
5	7	1	0	0	0	5	1	1	0	4
6	6	4	0	3	0	3	0	2	0	5
7	4	4	0	0	0	0	0	4	0	0

Table 5.6: Table showcasing the Frequency of Error Types occurring in the Subsets of the SeRe Configuration. Column '#' shows the Amount of incorrectly solved Use Cases in the respective Subset

occasions, the model then relied on its own knowledge about specific workouts. Consequently, the final answer recommended exercises that had not been gathered by the get_workout tool.

The third subset involved supplying the user with recipes that meet his daily nutritional requirements. This resulted in two types of error, ”Incomplete Solution Sequence” and ”Ignoring Requirements,” occurring seven times each. In four cases, the model did not select the appropriate value for the ”activity_level” parameter of the get_macros tool, and therefore was not able to correctly calculate the macronutrient need. Failing to provide recipes that fulfill the nutritional need however was the even more prominent error. In seven out of nine cases the model failed to utilize an adequate number of tool calls to obtain the necessary information about recommended recipes. After selecting three recipes for breakfast, dinner, lunch, and occasionally a snack, the model stopped searching for recipes. Subsequently, the nutritional information was either aggregated and presented to the user or not mentioned at all in the final answer.

Whether the model includes nutritional information about the recipes presented in its final answer directly correlates with the capability of the Reflect step to detect an error. In instances where the initial model did not provide the aggregated nutritional information but instead listed the names of the selected recipes, the response was deemed sufficient. However, even when the Reflect model attempted to enhance the initial response, it merely resulted in the model expanding the nutritional value of the recipes slightly with a recipe, yet still failing to meet the requisite amount. Consequently, when the model recommends that the user's selected recipes are inadequate to meet the specified nutritional requirements, despite the model having previously obtained these values through tool usage, an IR error has occurred.

In subset 4, the model is tasked with either providing additional nutritional information or recommending recipes based on the user's breakfast. In 6 out 7 use cases, the breakfast includes Crêpes. However, the model treats Crêpes as an ingredient, employing the get_ingredient_id and get_ingredient_nutrition functions to obtain the nutritional information about Crêpes. This results in the tool returning implausibly low values for macronutrients. This decision is classified as an error of the "Wrong Function Call" type. This error already occurred in the initial trial of the experiment. Modifying the breakfast to exclude Crêpes but include Crêpes Suzette prompted the model to recognize this as a distinct recipe.

This illustrates the sensitive nature of language and the model's capacity to comprehend the user's intent. This exemplifies a recurring issue: the user indicating that they consumed Crêpes for breakfast is a more plausible scenario than the user explicitly stating the specific recipe they used. An additional possibility is that the user may also mention specific ingredients, such as apples or nuts, in conjunction with mentioning Crêpes. To investigate the model's behavior in this case, one user query requests the protein value of the breakfast consisting of Crêpes, nuts, and an apple. However, incorporating explicit ingredients into the breakfast did not prompt the model to distinguish between the recipe of Crêpes and the ingredients of apple and nuts.

In subset 5, the user prompts the model to extract a recipe from an url for his dinner with friends later. Based on their dietary restrictions and the items Martin has in his storage, the model is prompted to adjust the recipes ingredients and take the storage into account when posting the shopping list.

In five out of seven cases, the model posts items to the shopping list that either are already in storage in sufficient quantities or posts the full amount as mentioned in the recipe, rather than accounting for the quantity of the ingredient already in storage. Consequently, this action is classified an IPS error. While in some use cases the items in storage are a lot, it may be assumed that providing the model with an excess of contextual information, even if it is essential, could result in the model failing to consider general reasoning and user requirements like dietary restrictions or items that don't need to be posted to the shopping list.

Subset 6 presents a comparable challenge to that encountered in subset 3. In Subset 6, the model is prompted to create a meal plan that adheres to multiple specified requirements, either those provided in the user query or those indicated by the system prompt. In four out of six cases, the model exhibited deficiencies in providing sufficient nutritional values in its recipes and in continuing the search for recipes. This was also observed when the user query indicates that it would be acceptable to recommend more than three meals per day.

In two instances, the model demonstrated an inability to accurately comprehend the user's precise requirements. In instances where the user indicates a preference for avoiding repetition in their dietary habits, the model frequently recommends the same recipe for both lunch and dinner, subsequently asserting that this aligns with the user's stated preferences. This serves to illustrate the potential consequences of using imprecise language in the user query. This is, however, a challenge that ALMs must address when functioning as a personal assistant to a human user.

Nevertheless, an even bigger challenge for the model was its inability to comprehend the user query, which was represented by use cases drawn from subset 7. In this instance, the model correctly identified the protein item with the highest density in the storage facility. However, it did not provide a subsequent recommendation, despite the user's explicit request for a recipe incorporating this item.

The model demonstrated deficiencies in all three areas: calling the function correctly, selecting parameters, and putting together a correct response. Some error types that were prevalent in one subset were not observed in others. This is indicative of the particular nature of the user's requirements and the challenges that the model faces as a result. There are multiple potential avenues for enhancing the efficacy of the SeRe configuration. They differentiate between improvements that are specific to a given dataset and those that are not. It is evident that the base model, GPT-4o, is deficient in certain functional capabilities, particularly in the selection of the optimal tool and the determination of the most suitable parameters. This deficiency is particularly pronounced when the necessary information is already available. Nevertheless, future modifications to the system prompts, instructing the model to devote greater attention to common errors, may prove beneficial. This, however, is dataset-specific and will not result in the model becoming a better personal agent for general tasks

Chapter 6

Conclusion

This thesis has explored the performance of tool-augmented Large Language Models on multi-step tasks, with a specific focus on the domain of physical health. A comprehensive examination of the theoretical background and the related work with a focus on autonomous agents was conducted. Furthermore various prompting strategies to enhance model performance were investigated. The experiment implemented four distinct configurations: Chain of Thought with a focus on planning with tool incorporation (CoT+), a two module approach implementing a Planner and a Solver to divide the reasoning and acting process (PS), the Self Reflect (SeRe) configuration that augmented the CoT+ approach with adding a step of Reflection at the end, and PS + SeRe, a combination of the three modules (Planner, Solver, Reflect). These configurations were tested using GPT-4o augmented with 15 tools, on a dataset of 75 realistic multi-step use cases. The findings of the experiment yielded several crucial insights:

1. The Self Reflect (SeRe) configuration demonstrated superior performance, outperforming other configurations by a notable margin. This suggests that incorporating a reflection step in the reasoning process can lead to improved outcomes in multi-step tasks.

2. Contrary to initial expectations, the simple division of the reasoning process into separate Planning and Solving modules did not result in an increase in success rate. This indicates that task segmentation into two separate specialized modules may not be sufficient to enhance performance in complex, multi-step scenarios.

3. A pattern where configurations based on similar prompting strategies (either CoT+ or PS) performed comparably within their groups but showed marked differences when compared to the other group was discovered. This suggests

that the fundamental prompting approach has a notable impact on model performance, potentially more so than the specific modular arrangement. This could be due to the differences in use cases, as this phenomenon appeared when comparing the performance on specific subsets of the dataset.

These results offer an avenue for further research and development in the domain of personal autonomous agents providing advice on physical health-related topics. Further research in this area has the potential to yield outstanding benefits for the general public, as there is much to be gained from the dissemination of accurate scientific information to the consumer. Moreover, this research yields intriguing insights into the optimal configuration of multiple modules to enhance the reasoning capabilities of the models. In conclusion, this study contributes to our understanding of how different prompting strategies and modular configurations can influence the performance of tool-augmented LLMs on multi-step tasks. As AI systems continue to evolve and are increasingly applied to complex real-world problems, insights from this research can guide the development of more effective and reliable AI agents.

Bibliography

[1] Bian, N., Han, X., Sun, L., Lin, H., Lu, Y., He, B., Jiang, S., Dong, B.: ChatGPT Is a Knowledgeable but Inexperienced Solver: An Investigation of Commonsense Problem in Large Language Models. In: Calzolari, N., Kan, M.Y., Hoste, V., Lenci, A., Sakti, S., Xue, N. (eds.) Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024). pp. 3098–3110. ELRA and ICCL, Torino, Italia (May 2024), `https://aclanthology.org/2024.lrec-main.276`

[2] Bran, A.M., Cox, S., Schilter, O., Baldassari, C., White, A.D., Schwaller, P.: ChemCrow: Augmenting large-language models with chemistry tools (Oct 2023). https://doi.org/10.48550/arXiv.2304.05376, `http://arxiv.org/abs/2304.05376`, arXiv:2304.05376 [physics, stat]

[3] Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.M., Yu, H., Lu, Y., Hung, Y.H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., Zhou, J.: AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors (Oct 2023). https://doi.org/10.48550/arXiv.2308.10848, `http://arxiv.org/abs/2308.10848`, arXiv:2308.10848 [cs]

[4] Choi, M., Pei, J., Kumar, S., Shu, C., Jurgens, D.: Do LLMs Understand Social Knowledge? Evaluating the Sociability of Large Language Models with SocKET Benchmark. In: Bouamor, H., Pino, J., Bali, K. (eds.) Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. pp. 11370–11403. Association for Computational Linguistics, Singapore (Dec 2023). https://doi.org/10.18653/v1/2023.emnlp-main.699, `https://aclanthology.org/2023.emnlp-main.699`

[5] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., et al.: PaLM: Scaling Language Modeling with Pathways. Journal of Machine Learning Research **24**(240), 1–113 (2023), `http://jmlr.org/papers/v24/22-1144.html`

[6] Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., Rass, S.: PentestGPT: An LLM-empowered Automatic Penetration Testing Tool (Jun 2024). https://doi.org/10.48550/arXiv.2308.06782, `http://arxiv.org/abs/2308.06782`, arXiv:2308.06782 [cs]

[7] Gao, D., Ji, L., Zhou, L., Lin, K.Q., Chen, J., Fan, Z., Shou, M.Z.: AssistGPT: A General Multi-modal Assistant that can Plan, Execute, Inspect, and Learn (Jun 2023). https://doi.org/10.48550/arXiv.2306.08640, `http://arxiv.org/abs/2306.08640`, arXiv:2306.08640 [cs]

[8] Hao, S., Liu, T., Wang, Z., Hu, Z.: ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings. Advances in Neural Information Processing Systems **36**, 45870–45894 (Dec 2023), `https://proceedings.neurips.cc/paper_files/paper/2023/hash/8fd1a81c882cd45f64958da6284f4a3f-Abstract-Conference.html`

[9] Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., Chen, E.: Understanding the planning of LLM agents: A survey (Feb 2024). https://doi.org/10.48550/arXiv.2402.02716, `http://arxiv.org/abs/2402.02716`, arXiv:2402.02716 [cs]

[10] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Chen, D., Dai, W., Chan, H.S., Madotto, A., Fung, P.: Survey of Hallucination in Natural Language Generation. ACM Computing Surveys **55**(12), 1–38 (Dec 2023). https://doi.org/10.1145/3571730, `http://arxiv.org/abs/2202.03629`, arXiv:2202.03629 [cs]

[11] Jin, Z., Cao, P., Chen, Y., Liu, K., Jiang, X., Xu, J., Qiuxia, L., Zhao, J.: Tug-of-War between Knowledge: Exploring and Resolving Knowledge Conflicts in Retrieval-Augmented Language Models. In: Calzolari, N., Kan, M.Y., Hoste, V., Lenci, A., Sakti, S., Xue, N. (eds.) Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024). pp. 16867–16878. ELRA and ICCL, Torino, Italia (May 2024), `https://aclanthology.org/2024.lrec-main.1466`

[12] Kang, M., Lee, S., Baek, J., Kawaguchi, K., Hwang, S.J.: Knowledge-Augmented Reasoning Distillation for Small Language Models in Knowledge-Intensive Tasks. Advances in Neural Information Processing Systems **36**, 48573–48602 (Dec 2023), `https://proceedings.neurips.cc/paper_files/paper/2023/hash/`

97faedc90260eae5c400f92d5831c3d7-Abstract-Conference.html

[13] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: Advances in Neural Information Processing Systems. vol. 33, pp. 9459–9474. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[14] Li, S., Yang, J., Zhao, K.: Are you in a Masquerade? Exploring the Behavior and Impact of Large Language Model Driven Social Bots in Online Social Networks (Jul 2023). https://doi.org/10.48550/arXiv.2307.10337, http://arxiv.org/abs/2307.10337, arXiv:2307.10337 [cs]

[15] Liang, Y., Wu, C., Song, T., Wu, W., Xia, Y., Liu, Y., Ou, Y., Lu, S., Ji, L., Mao, S., Wang, Y., Shou, L., Gong, M., Duan, N.: TaskMatrix.AI: Completing Tasks by Connecting Foundation Models with Millions of APIs (Mar 2023). https://doi.org/10.48550/arXiv.2303.16434, http://arxiv.org/abs/2303.16434, arXiv:2303.16434 [cs]

[16] Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., Tang, J.: AgentBench: Evaluating LLMs as Agents (Oct 2023), https://openreview.net/forum?id=zAdUB0aCTQ

[17] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., et al.: Self-Refine: Iterative Refinement with Self-Feedback. Advances in Neural Information Processing Systems **36**, 46534–46594 (Dec 2023), https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html

[18] Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., Grave, E., LeCun, Y., Scialom, T.: Augmented Language Models: a Survey (Feb 2023). https://doi.org/10.48550/arXiv.2302.07842, http://arxiv.org/abs/2302.07842, arXiv:2302.07842 [cs]

[19] Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., Gao, J.: Large Language Models: A Survey (Feb 2024).

https://doi.org/10.48550/arXiv.2402.06196, `http://arxiv.org/abs/2402.06196`, arXiv:2402.06196 [cs]

[20] Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., Schulman, J.: WebGPT: Browser-assisted question-answering with human feedback (Jun 2022). https://doi.org/10.48550/arXiv.2112.09332, `http://arxiv.org/abs/2112.09332`, arXiv:2112.09332 [cs]

[21] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., other: GPT-4 Technical Report (Mar 2024). https://doi.org/10.48550/arXiv.2303.08774, `http://arxiv.org/abs/2303.08774`, arXiv:2303.08774 [cs]

[22] Pan, L., Saxon, M., Xu, W., Nathani, D., Wang, X., Wang, W.Y.: Automatically Correcting Large Language Models: Surveying the landscape of diverse self-correction strategies (Aug 2023), `http://arxiv.org/abs/2308.03188`, arXiv:2308.03188 [cs]

[23] Patil, S.G., Zhang, T., Wang, X., Gonzalez, J.E.: Gorilla: Large Language Model Connected with Massive APIs (May 2023). https://doi.org/10.48550/arXiv.2305.15334, `http://arxiv.org/abs/2305.15334`, arXiv:2305.15334 [cs]

[24] Qian, C., Han, C., Fung, Y.R., Qin, Y., Liu, Z., Ji, H.: CREATOR: 2023 Findings of the Association for Computational Linguistics: EMNLP 2023. Findings of the Association for Computational Linguistics pp. 6922–6939 (2023), `http://www.scopus.com/inward/record.url?scp=85183289703&partnerID=8YFLogxK`, publisher: Association for Computational Linguistics (ACL)

[25] Qian, J., Wang, H., Li, Z., Li, S., Yan, X.: Limitations of Language Models in Arithmetic and Symbolic Induction (Aug 2022), `http://arxiv.org/abs/2208.05051`, arXiv:2208.05051 [cs]

[26] Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., et al.: ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs (Oct 2023), `https://openreview.net/forum?id=dHng2O0Jjr`

[27] Ruan, J., Chen, Y., Zhang, B., Xu, Z., Bao, T., Qing, D.G., Shiwei, S., Mao, H., Li, Z., Zeng, X., Zhao, R.: TPTU: Task Planning and Tool Usage of Large

Language Model-based AI Agents (Nov 2023), https://openreview.net/forum?id=GrkgKtOjaH

[28] Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language Models Can Teach Themselves to Use Tools. Advances in Neural Information Processing Systems **36**, 68539–68551 (Dec 2023), https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[29] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems **36**, 8634–8652 (Dec 2023), https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html

[30] Song, C.H., Wu, J., Washington, C., Sadler, B.M., Chao, W.L., Su, Y.: LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. pp. 2998–3009 (2023), https://openaccess.thecvf.com/content/ICCV2023/html/Song_LLM-Planner_Few-Shot_Grounded_Planning_for_Embodied_Agents_with_Large_Language_ICCV_2023_paper.html

[31] Song, Y., Xiong, W., Zhu, D., Wu, W., Qian, H., Song, M., Huang, H., Li, C., Wang, K., Yao, R., Tian, Y., Li, S.: RestGPT: Connecting Large Language Models with Real-World RESTful APIs (Aug 2023). https://doi.org/10.48550/arXiv.2306.06624, http://arxiv.org/abs/2306.06624, arXiv:2306.06624 [cs]

[32] Sun, L., Huang, Y., Wang, H., Wu, S., Zhang, Q., Li, Y., Gao, C., Huang, Y., Lyu, W., Zhang, Y., et al.: TrustLLM: Trustworthiness in Large Language Models (Mar 2024). https://doi.org/10.48550/arXiv.2401.05561, http://arxiv.org/abs/2401.05561, arXiv:2401.05561 [cs]

[33] Tang, Q., Deng, Z., Lin, H., Han, X., Liang, Q., Cao, B., Sun, L.: ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases (Sep 2023). https://doi.org/10.48550/arXiv.2306.05301, http://arxiv.org/abs/2306.05301, arXiv:2306.05301 [cs]

[34] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, u., Polosukhin, I.: Attention is All you Need. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[35] Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W.X., Wei, Z., Wen, J.R.: A Survey on Large Language Model based Autonomous Agents (Sep 2023). https://doi.org/10.48550/arXiv.2308.11432, http://arxiv.org/abs/2308.11432, arXiv:2308.11432 [cs]

[36] Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R.K.W., Lim, E.P.: Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In: Rogers, A., Boyd-Graber, J., Okazaki, N. (eds.) Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 2609–2634. Association for Computational Linguistics, Toronto, Canada (Jul 2023). https://doi.org/10.18653/v1/2023.acl-long.147, https://aclanthology.org/2023.acl-long.147

[37] Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., Liang, Y., CraftJarvis, T.: Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. pp. 34153–34189. NIPS '23, Curran Associates Inc., Red Hook, NY, USA (2024)

[38] Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E.H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., Fedus, W.: Emergent Abilities of Large Language Models (Oct 2022). https://doi.org/10.48550/arXiv.2206.07682, http://arxiv.org/abs/2206.07682, arXiv:2206.07682 [cs]

[39] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q.V., Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems **35**, 24824–24837 (Dec 2022), https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[40] Williams, R., Hosseinichimeh, N., Majumdar, A., Ghaffarzadegan, N.: Epidemic Modeling with Generative Agents (Jul 2023). https://doi.org/10.48550/arXiv.2307.04986, http://arxiv.org/abs/2307.04986, arXiv:2307.04986 [nlin, physics:physics, q-fin]

[41] Xu, B., Liu, X., Shen, H., Han, Z., Li, Y., Yue, M., Peng, Z., Liu, Y., Yao, Z., Xu, D.: Gentopia: A Collaborative Platform for Tool-Augmented LLMs (Aug 2023). https://doi.org/10.48550/arXiv.2308.04030, http://arxiv.org/abs/2308.04030, arXiv:2308.04030 [cs]

[42] Xu, Q., Hong, F., Li, B., Hu, C., Chen, Z., Zhang, J.: On the Tool Manipulation Capability of Open-sourced Large Language Models (Nov 2023), https://openreview.net/forum?id=d5ogyvdl1X

[43] Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R.J., Anandkumar, A.: LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. Advances in Neural Information Processing Systems **36**, 21573–21612 (Dec 2023), https://proceedings.neurips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html

[44] Yao, S., Chen, H., Yang, J., Narasimhan, K.: WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. Advances in Neural Information Processing Systems **35**, 20744–20757 (Dec 2022), https://proceedings.neurips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html

[45] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., Narasimhan, K.: Tree of Thoughts: Deliberate Problem Solving with Large Language Models. Advances in Neural Information Processing Systems **36**, 11809–11822 (Dec 2023), https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html

[46] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K.R., Cao, Y.: ReAct: Synergizing Reasoning and Acting in Language Models (Sep 2022), https://openreview.net/forum?id=WE_vluYUL-X

[47] Zhao, W.X., Zhou, K., Li, J., Tang, T., et al.: A Survey of Large Language Models (Nov 2023). https://doi.org/10.48550/arXiv.2303.18223, `http://arxiv.org/abs/2303.18223`, arXiv:2303.18223 [cs]

[48] Zhou, S., Xu, F.F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., Neubig, G.: WebArena: A Realistic Web Environment for Building Autonomous Agents (Apr 2024). https://doi.org/10.48550/arXiv.2307.13854, `http://arxiv.org/abs/2307.13854`, arXiv:2307.13854 [cs]

[49] Ziems, C., Held, W., Shaikh, O., Chen, J., Zhang, Z., Yang, D.: Can Large Language Models Transform Computational Social Science? Computational Linguistics **50**(1), 237–291 (Mar 2024). https://doi.org/10.1162/coli$_{a0}$0502, , place: Cambridge, MA Publisher: MIT Press

Appendix A

Use Cases

A.1 Subset 1: Temperature-Based Decision Making

Use Case 1

Query: ”If it is warmer than (CURRENT_TEMP + 1) degrees, I want to do a workout for my biceps using the ez-bar. If it isn’t, recommend a movie starring Arnold Schwarzenegger to watch.”

Solution Sequence:

1. get_longitude_and_latitude (city: Mannheim, country: Germany) −> longitude: 8.4673098, latitude: 49.4892913
2. get_temperature (longitude: 8.4673098, latitude: 49.4892913) −> temperature
3. get_actor_id (Name: Arnold Schwarzenegger) −> actor_id
4. get_movie_by_actor_id (actor_id: nm0000216) −> Movie

textbfGround truth: Movie by Arnold Schwarzenegger

Use Case 2

Query: ”If it is warmer than (CURRENT_TEMP + 1) degrees, I want to do a workout for my biceps using the ez-bar. If it isn’t, **i want to watch a movie** with Arnold Schwarzenegger.”

Use Case 3

Query: ”If it is warmer than (CURRENT_TEMP + 1) degrees, I want to do a workout for my biceps using the ez-bar. If it isn’t, recommend **something with Paul Mescal to watch**.”

Use Case 4

Query: ”If it is warmer than (CURRENT_TEMP - 1) degrees, I want to do a workout for my biceps using the ez-bar. If it isn’t, recommend a movie starring Arnold Schwarzenegger to watch.”

Ground truth: Exercise working on the biceps using the ez-bar with the intensity level for a beginner.

Use Case 5

Query: ”If it is warmer than (CURRENT_TEMP - 1) degrees, I want to do a **workout outside for my biceps using some equipment that is easy to bring to the park**. If it isn’t, recommend a movie starring Arnold Schwarzenegger to watch.”

Use Case 6

Query: ”If it is warmer than (CURRENT_TEMP - 1) degrees, I want to do a **workout outside consisting of three exercises for different muscles using some equipment that is easy to bring to the park**. If it isn’t, recommend a movie starring Arnold Schwarzenegger to watch.”

Use Case 7

Query: ”If it is warmer than (CURRENT_TEMP) degrees, I want to do a workout for my biceps using the ez-bar. If it isn’t, recommend a movie starring Arnold Schwarzenegger to watch.”

Ground truth: Movie by Arnold Schwarzenegger

Intention: Is to test how the model performs when the threshold is exactly the current temperature.

Use Case 8

Query: ”If it is warmer than (CURRENT_TEMP) degrees, I want to do a workout for my biceps using the ez-bar. If it isn’t, recommend a movie starring **Paul Mescal** to watch.”

Use Case 9

Query: ”If it is warmer than (CURRENT_TEMP) degrees, I want to do a workout for my biceps using the ez-bar. If it isn’t, recommend a movie starring **Quentin Tarantino** to watch.”

A.2 Subset 2: Integrated Workout and Nutrition Planning

Use Case 10

Query: ”Please recommend an exercise using the ez-bar to work on my biceps. Then provide a recipe that fulfills my dietary needs for today. I already had 1931cal, 111g of protein, 272g carbs and 55g fats.”

Solution Sequence:

1. get_workout (muscles: Biceps, equipment: ez-Bar, intensity_level: Beginner) −> exercise
2. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
3. get_calculation (given nutrition minus result from get_macros) −> result of calculation
4. find_by_nutrients (calories: 433, protein: 28, carbs: 32, fat: 10) (minimum) −> recipe fulfilling nutritional specification

Ground truth: Exercise using the ez-bar for the biceps and sufficient recipe
Intention: Testing if model adjusts the activity_level

Use Case 11

Query: ”Please recommend **three exercises to work on my appearance**. Then provide a recipe that fulfills my dietary needs for today. I already had 1931cal, 111g of protein, 272g carbs and 55g fats.”

Use Case 12

Query: ”Please recommend an exercise using the ez-bar to work on my biceps. Then provide some recipes that fulfill my dietary needs for today. I already had **1610cal, 80g of protein, 220g carbs and 40g fats**.”

Use Case 13

Query: ”Please recommend a balanced workout consisting of 6 exercises focusing on my upper body. Then provide a recipe that fulfills my dietary needs for today. I already had 1931cal, 111g of protein, 272g carbs and 55g fats.”

Intention: Testing what the model defines as a balanced workout consisting of 6 exercises.

Use Case 14

Query: ”Please recommend a balanced workout consisting of 6 exercises **focusing on my whole body**. Then provide a recipe that fulfills my dietary needs for today. I already had 1931cal, 111g of protein, 272g carbs and 55g fats.”

Use Case 15

Query: ”Please recommend a balanced workout consisting of 6 exercises focusing on my upper body. Then provide some recipes that fulfill my dietary needs for today. I already had **1610cal, 80g of protein, 220g carbs and 40g fats**.”

Use Case 16

Query: ”Please recommend a balanced workout consisting of 6 exercises focusing on my upper body. I want to use a minimum of three different kinds of equipment. Then provide a recipe that fulfills my dietary needs for today. I already had 1931cal, 111g of protein, 272g carbs and 55g fats.”

Solution Sequence:

1. 6 x get_workout (muscles: balances muscles, equipment: three different, intensity_level: Beginner) −> exercise
2. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
3. get_calculation (given nutrition minus result from get_macros) −> result of calculation
4. find_by_nutrients (calories: 433, protein: 28, carbs: 32, fat: 10) (minimum) −> recipe fulfilling nutritional specification

Ground truth: Balanced workout consisting of 6 exercises focusing on the upper body with three different equipment and sufficient recipe

Intention: Adding restriction with three different equipment.

Use Case 17

Query: ”Please recommend a balanced workout consisting of 6 exercises focusing on **muscles** in my upper body. I want to use a minimum of three different kinds of equipment. Then provide a recipe that fulfills my dietary needs for today. I already had 1931cal, 111g of protein, 272g carbs and 55g fats.”

Use Case 18

Query: ”Please recommend a balanced workout consisting of 6 exercises focusing on my upper body. I want to use a minimum of three different kinds of equipment. Then provide some recipes that fulfill my dietary needs for today. I already had **1610cal, 80g of protein, 220g carbs and 40g fats**.”

A.3 Subset 3: Activity Level Adjustment

Use Case 19

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk the whole day. Please provide me with recipes for the day.”

Solution Sequence:

1. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 1, goal: maintain) −> daily need of calories, protein, carbs, and fat
2. either 3 x find_by_nutrients (calories: 1/3, protein: 1/3, carbs: 1/3, fat: 1/3) −> recipe fulfilling nutritional specification
3. or recipe_search (add_recipe_nutrition: Yes , recipe_type: breakfast/lunch/dinner) −> recipe fulfilling specification

Ground truth: Sufficient meal plan
Intention: Adjusted activity_level and following lower nutritional need

Use Case 20

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk the whole day. Please provide me with **enough recipes** for the day.”

Use Case 21

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk the whole day. Please provide me with **enough recipes for the day to fulfill my protein need**.”

Use Case 22

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk the whole day. Please provide me with recipes for the day. Keep in mind, my goal is to maintain my weight.”

Solution Sequence:

1. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 1, goal: maintain) −> daily need of calories, protein, carbs, and fat
2. either 3 x find_by_nutrients (calories: 1/3, protein: 1/3, carbs: 1/3, fat: 1/3) −> recipe fulfilling nutritional specification
3. or recipe_search (add_recipe_nutrition: Yes , recipe_type: breakfast/lunch/dinner/snack) −> recipe fulfilling specification

Ground truth: Sufficient meal plan

Intention: Triggering the model to take the goal to maintain the weight into account, as it didn't call the get_macros function before

Use Case 23

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk the whole day. Please provide me with recipes for the day. **Make sure my macros for the day are met**!”

Use Case 24

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk the whole day. Please provide me with **enough recipes for the day to meet my macros**!”

Use Case 25

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk until dinner. After that I will go to a spinning class. Please provide me with recipes for the day.”

Solution Sequence:

1. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 5, goal: maintain) −> daily need of calories, protein, carbs, and fat
2. either 3 x find_by_nutrients (calories: 1/3, protein: 1/3, carbs: 1/3, fat: 1/3) −> recipe fulfilling nutritional specification
3. or recipe_search (add_recipe_nutrition: Yes , recipe_type: breakfast/lunch/dinner) −> recipe fulfilling specification

Ground truth: Sufficient meal plan

Intention: Taking the high intensity spinning class into account for the activity_level

Use Case 26

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk until dinner. After that I will go to my spinning class **which is really exhausting for me**. Please provide me with recipes for the day.”

Use Case 27

Query: ”Tomorrow will be a very busy day at the office, so I will sit at my desk until dinner. After that I will go to a spinning class. Please provide me with recipes for the day. **Because the spinning class is so exhausting I want to have a late evening snack with at least 15g of protein.**”

A.4 Subset 4: Breakfast-Based Nutrition Planning

Use Case 28

Query: ”Today I had two servings of Crepes for breakfast, how much protein do I still need to eat today?”

Solution Sequence:

1. recipe_search (query: Crepes, add_recipe_nutrition: True) −> recipe fulfilling specification
2. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
3. potentially calculation

Intention: Fall back to base activity_level

Use Case 29

Query: ”Today I had two servings of **Crepes Suzette** for breakfast, how much protein do I still need to eat today?”

Use Case 30

Query: ”Today I had two servings of **Crepes Suzette and oatmeal with water and one cup of blueberries** for breakfast, how much protein do I still need to eat today?”

Use Case 31

Query: ”Today I had one serving of Crepes, an Apple, and 1 cup of Walnuts for breakfast, how much protein do I still need to eat today?”

Solution Sequence:

1. recipe_search (query: Crepes, add_recipe_nutrition: True) –> recipe fulfilling specification
2. 2 x search_ingredient (query: Apple / Walnut) –> ingredient_id
3. 2 x get_ingredient_nutrition (ingredient_id: , amount: , unit:) –> nutritional information of ingredient
4. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) –> daily need of calories, protein, carbs, and fat
5. potentially calculation

Intention: Combining recipe_search and get_ingredient_nutrition as treating Crepes as an ingredient leads to bad results

Use Case 32

Query: ”Today I had one serving of Crepes, an Apple, and **two servings of Scrambled eggs** for breakfast, how much protein do I still need to eat today?

Use Case 33

Query: ”Today I had one serving of Crepes, an Apple, and **two servings of British Muffins** for breakfast, how much protein do I still need to eat today?”

Use Case 34

Query: ”Today me and my wife shared three servings of Crepes for breakfast, how much protein do I still need to eat today?”

Solution Sequence:

1. recipe_search (query: Crepes, add_recipe_nutrition: True) −> recipe fulfilling specification
2. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
3. potentially calculation

Intention: Using the information about servings from recipe_search to calculate individual protein intake

Use Case 35

Query: ”Today me and my wife shared three servings of Crepes **equally** for breakfast, how much protein do I still need to eat today?”

Use Case 36

Query: ”Today me and my wife equally shared three servings of **Crepes Suzette** for breakfast, how much protein do I still need to eat today?”

Use Case 37

Query: ”Today me and my wife had three servings of Crepes for breakfast. I ate half of them. How much protein do I still need to eat today?”

Solution Sequence:

1. recipe_search (query: Crepes, add_recipe_nutrition: True) −> recipe fulfilling specification
2. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
3. potentially calculation

Intention: Different form of calculation

Use Case 38

Query: ”Today I had two servings of Crepes for breakfast, how much **percent of my daily calories** did I already eat?”

Use Case 39

Query: ”Today I had two servings of **Crepes Suzette** for breakfast, how much percent of my daily calories did I already eat?”

Use Case 40

Query: ”Today I had two servings of **Scrambled eggs** for breakfast, how much percent of my daily calories did I already eat?”

Use Case 41

Query: ”Today I had two servings of Crepes for breakfast, how much percent of my daily calories did I already eat? To cover at least 20 percent of the remaining calories please provide me with a recipe of the same cuisine as Crepes”

Solution Sequence:

1. recipe_search (query: Crepes, add_recipe_nutrition: True) −> recipe fulfilling specification
2. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
3. potentially calculation

Intention: Different form of calculation

Use Case 42

Query: ”Today I had two servings of **Crepes Suzette** for breakfast, how much percent of my daily calories did I already eat? To cover at least 20 percent of the remaining calories please provide me with a recipe of the same cuisine as Crepes”

Use Case 43

Query: ”Today I had two servings of Crepes Suzette for breakfast, how much percent of my daily calories did I already eat? To cover **between 15 and 25 percent**

of the remaining calories please provide me with a recipe of the same cuisine as Crepes”

Solution Sequence:

1. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> ”calories”: 2564, ”protein”: 159, ”fat”: 75, ”carbs”: 314
2. recipe_search (query: Crepes Suzette, add_recipe_nutrition: True) −> ”Crepes Suzette”, ”servings: 4”, ”name”: ”Calories”, ”amount”: 618.91, ”unit”: ”cal”, ”percentOfDailyNeeds”: 30.95 + more
3. get_calculation (’a_number’: 618.91, ’b_number’: 2, ’calc_type’: ’multiply’) −> 1237.82
4. ...
5. get_calculation (’a_number’: 1326.18, ’b_number’: 0.15, ’calc_type’: ’multiply’) −> 198.927
6. get_calculation (’a_number’: 1326.18, ’b_number’: 0.25, ’calc_type’: ’multiply’) −> 331.545
7. recipe_search (cuisine: French, add_recipe_nutrition: True, number = 2) −> ”Baked Ratatouille”, ”servings: 1”, ”name”: ”Calories”, ”amount”: 1031.89, ”unit”: ”cal”, ”percentOfDailyNeeds”: 51.59 + more, ”Creamy Ratatouille Over Penne”, ”servings: 2”, ”name”: ”Calories”, ”amount”: 308.03, ”unit”: ”cal”, ”percentOfDailyNeeds”: 15.4 + more

Ground truth: Recipe with calorie value between 198 and 331 cal

A.5 Subset 5: Dinner with Friends with dietary Restrictions

Use Case 44

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? One friend is vegetarian. Recommend a vegetarian alternative to meat, and show the difference in macros.”

Solution Sequence:

1. extract_recipe (url: https://www.acozykitchen.com/spaghetti-meatballs) −> information about recipe (e.g. title, ingredients, time)
2. search_ingredient (query: beef / pork / lentils) −> ingredient_id
3. get_ingredient_nutrition (ingredient_id: , amount: , unit:) −> nutritional information of ingredient

Ground truth: Dinner time and difference in macros comparing pork and beef to vegetarian alternative

Use Case 45

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? **Half of my friends are vegetarian. The rest wants to eat meat though**. Recommend a vegetarian alternative to meat, and show the difference in macros.”

Use Case 46

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping and **do not have a shopping list so far**. For when should I invite my friends over for dinner? Half of my friends are vegetarian. The rest wants to eat meat though. Recommend a vegetarian alternative to meat, and show the difference in macros.”

Use Case 47

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? One friend is vegetarian. Recommend a vegetarian alternative to meat, show the difference in macros, and add everything I need to buy to the shopping list.”

Storage: 6 onions

Ground truth: Give time for when to invite friends over based on recipe information and add items to the shopping list. Taking the vegetarian restriction into account. No meat should be added to the list but a vegetarian alternative.

Intention: Test if the model itself realizes items could be in storage before adding them to the shopping list

Use Case 48

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? One friend is vegetarian. Recommend a vegetarian alternative to meat, show the difference in macros, and add everything I need to buy to the shopping list. **I think i have tofu at home though**.”

Storage: 6 onions, 3 cloves of garlic, 600g of tofu

Use Case 49

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? Half of my friends are vegetarian. Recommend a vegetarian alternative to meat, show the difference in macros, and add everything I need to buy to the shopping list. **We want to do half of the meal in vegetarian and in another pot half of it with meat. One friend also is lactose intolerant is this a problem? If yes how should we alter the vegetarian pot to account for his intolerance**”

Use Case 50

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? One friend is vegetarian. Recommend a vegetarian alternative to meat, show the difference in macros, and add everything I need to buy to the shopping list. Also take into account what I already have at home.”

Storage: 5 onions, 3 cloves of garlic, 1 l of olive oil

Solution Sequence:

1. extract_recipe (url: https://www.acozykitchen.com/spaghetti-meatballs) −> information about recipe (title=”Spaghetti and Meatballs”, ingredients=List of ingredient including 1/2 yellow onion, 6 garlic cloves, 1 tablespoon neutral oil, 1 pound ground pork, 1 pound ground beef), total_time = 45)

2. get_items_from_storage () −> Items from storage in JSON (5 onions, 3 cloves of garlic, 1 l of olive oil)

3. search_ingredient (query: [insert vegetarian alternative]) −> ingredient_id of vegetarian alternative

4. search_ingredient (query: ground Beef) −>
   ingredient_id ground beef = 10219

5. search_ingredient (query: ground Pork) −>
   ingredient_id ground Pork = 10023572

6. get_ingredient_nutrition (ingredient_id: X, amount: 1, unit: pound) −> nutritional information of X

7. get_ingredient_nutrition (ingredient_id: 10219, amount: 1, unit: pound) −> nutritional information of ground beef

8. get_ingredient_nutrition (ingredient_id: 10023572, amount: 1, unit: pound) −> nutritional information of ground pork

9. post_to_shopping_list (items (not onion, 3 cloves of garlic, 1 tablespoon neutral oil) −> None

Ground truth: Give time for when to invite friends over based on recipe information and add only items that are not in storage to the shopping list

Intention: Items not in storage but needed for the recipe are neutral oil, onions and 6 cloves of garlic. However, Olive oil and 3 cloves of garlic are in storage.

Use Case 51

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? One friend is vegetarian. Recommend a vegetarian alternative to meat, show the difference in macros, and add everything I need to buy to the shopping list. Also take into account what I already have at home.”

Storage: 5 red onions, 3 pieces of garlic, 1 l of olive oil

Use Case 52

Query: ”Tonight I want to cook this: https://www.acozykitchen.com/spaghetti-meatballs. Before starting to cook at 7pm, I need to go grocery shopping. For when should I invite my friends over for dinner? One friend is vegetarian. Recommend a vegetarian alternative to meat, show the difference in macros, and add everything I need to buy to the shopping list. Also take into account what I already have at home.”

Storage: 5 onions, 3 pieces of garlic, 1 l of olive oil, 1kg of fusilli

A.6 Subset 6: Multi-Day Meal Planning

Use Case 53

Query: ”For the next three days I need a meal plan with three meals a day. I like to eat meat once a day, however I don’t want to eat the same lunch or dinner twice and also I dislike eating the same kind of meat twice in three days.”

Solution Sequence:

1. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
2. either recipe_search (add_recipe_nutrition: True, recipe_type: breakfast/ lunch/dinner, number: 3) −> recipe fulfilling specification
3. or find_by_nutrients (calories, protein, carbs, fat) −> recipe fulfilling nutritional specification

Ground truth: Sufficient meal plan

Intention: General meal plan use case with preference of not eating the same thing twice. Also testing whether the model asks for macros need if not reminded by the user.

Use Case 54

Query: ”For the next three days I need a meal plan with three meals a day. I like to eat meat once a day, however I don’t like to eat the same kind of meat twice during the three days. During one day I don’t like to eat the same lunch or dinner twice.”

Use Case 55

Query: ”For the next three days I need a meal plan with three meals a day. I like to eat meat once a day, however I don’t want to eat the same lunch or dinner twice and also I dislike eating the same kind of meat twice in three days. Also I had way too much cheese the last days, so **please don’t include cheese** in the recipes.”

Use Case 56

Query: ”I need a meal plan for the next three days. In the next three days I want to load up on carbs so I want to eat 20 percent more carbs than I normally would. I don’t want to eat the same lunch or dinner twice.”

Solution Sequence:

1. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 3, goal: maintain) −> daily need of calories, protein, carbs, and fat
2. get_calculation (+ 20% carbs) −> result of calculation
3. either recipe_search (add_recipe_nutrition: True, recipe_type: breakfast/ lunch/dinner, number: 3) −> recipe fulfilling specification
4. or find_by_nutrients (calories, protein, carbs, fat) −> recipe fulfilling nutritional specification

Ground truth: Sufficient meal plan
Intention: Testing how the model adjusts to 20% more carbs.

Use Case 57

Query: ”I need a meal plan for the next three days. In the next three days I want to load up on carbs so I want to eat 20 percent more carbs than I normally would. I don't want to eat the same lunch or dinner twice. For this time it is **most important to me to reach my carbs goal**, less important to meet the rest of the macros.”

Use Case 58

Query: ”I need a meal plan for the next three days. In the next three days I want to load up on carbs so I want to eat 20 percent more carbs than I normally would. I don't want to eat the same lunch or dinner twice. For this time it is most important to me to reach my carbs goal, less important to meet the rest of the macros. **I am also willing to eat not only three times a day. Just make sure my carbs are met.**”

Use Case 59

Query: ”I need a meal plan for the next three days. In the next three days I want to load up on carbs so I want to eat 20 percent more carbs than I normally would. I will also do heavy endurance training in that time. I don't want to eat the same lunch or dinner twice.”

Solution Sequence:

1. get_macros (age: 22, gender: male, height: 175, weight: 70, activity_level: 5, goal: maintain) −> daily need of calories, protein, carbs, and fat
2. get_calculation (+ 20% carbs) −> result of calculation

3. either recipe_search (add_recipe_nutrition: True, recipe_type: breakfast/ lunch/dinner, number: 3) −> recipe fulfilling specification
4. or find_by_nutrients (calories, protein, carbs, fat) −> recipe fulfilling nutritional specification

Ground truth: Sufficient meal plan
Intention: Testing whether model adjusts the activity_level due to heavy endurance training.

Use Case 60

Query: ”I need a meal plan for the next three days. In the next three days I want to load up on carbs so I want to eat 20 percent more carbs than I normally would. I will also do heavy endurance training during these three days. I don’t want to eat the same lunch or dinner twice. **Make sure that i hit my macro goal!**”

Use Case 61

Query: ”I need a meal plan for the next three days. I want to load up on carbs so I want to eat 20 percent more carbs than I normally would. On the three days I will be very active in preparing for a triathlon, therefore I will do heavy endurance training every day. Also **I don’t want to eat the same lunch or dinner twice**”

A.7 Subset 7: Storage-Based Nutrition

Use Case 62

Query: ”Please provide me with a recipe using the least expensive vegetable and meat in my storage.”
Storage: A lot of vegetables, fruit, dairy, nuts and meat
Solution Sequence:

1. get_items_from_storage () −> Items from storage in JSON
2. recipe_search (include_ingredients: cheapest vegetable) −> recipe fulfilling specification

Ground truth: Using the cheapest vegetable
Intention: Testing the model’s capability of retrieving and processing formatted storage information.

Use Case 63

Query: ”please help me prepare a **three-course meal for my family**. Each course should contain at least one ingredient from my storage. Please choose the most expensive vegetable, the most expensive meat and the most expensive fruit.”

Storage: A lot of vegetables, fruit, dairy, nuts and meat

Use Case 64

Query: ”please help me prepare a three-course meal for my family. Each course should contain at least one ingredient from my storage. Please choose the most expensive vegetable, the most expensive meat and the most expensive fruit. Also **prepare a shopping list** for me.”

Storage: A lot of vegetables, fruit, dairy, nuts and meat

Use Case 65

Query: ”Please provide me with a recipe using the least expensive vegetable relative to weight in my storage.”

Storage: A lot of vegetables

Solution Sequence:

1. get_items_from_storage () −> Items from storage in JSON
2. recipe_search (include_ingredients: cheapest vegetable) −> recipe fulfilling specification

Ground truth: Using the cheapest vegetable

Intention: Same as previous use case but with a lot more items in storage

Use Case 66

Query: ”Please provide me with a recipe using the least expensive vegetable relative to weight in my storage. **For items that are in storage in pieces, calculate their weight first.**”

Storage: A lot of vegetables

Use Case 67

Query: ”How much percent of my daily protein need can I cover with a recipe with the protein densest product in my storage?”

Storage: kiwi, milk, yogurt, cashews

Solution Sequence:

1. get_items_from_storage () −> Items from storage in JSON
2. recipe_search (include_ingredients: cheapest vegetable) −> recipe fulfilling specification

Use Case 68

Query: "How much percent of my daily protein need can I cover with a recipe with the **calorie densest** product in my storage?"

Storage: kiwi, milk, yogurt, cashews

Use Case 69

Query: "During my bike ride on Saturday I need to eat **30g of carbs to fuel. However I want to carry as little weight as possible with me on the ride. Also I will not have time to go grocery shopping before.**"

Storage: kiwi, milk, yogurt, cashews

Use Case 70

Query: "I either want to eat one serving of Crepes, Waffles. Which one should I make to reduce the total weight of my storage the most?"

Storage: Flour, Butter, Milk

Solution Sequence:

1. recipe_search (query: Crepe, add_recipe_nutrition: , cuisine: , recipe_type: , number: , include_ingredients: , exclude_ingredients) −> recipe fulfilling specification

Use Case 71

Query: "I either want **to make a recipe** for one serving of Crepes or Waffles. Which one should I make to reduce the total weight of my storage the most?"

Storage: Flour, Butter, Milk

Use Case 72

Query: "I either want to make a recipe for Crepes or Waffles. Which one should I make to reduce the total weight of my storage the most and what would the **differences in macros** be?"

Storage: Flour, Butter, Milk

Use Case 73

Query: ”How many days could I maintain my weight if i don’t go shopping?”
Storage: fruit, dairy and nuts
Solution Sequence:

1. get_macros (age=22, gender=’male’, height=175, weight=70, activity_level=3, goal=’maintain’)
−> calorie: 2564, protein: 159, fat: 75, carbs: 314

2. get_items_from_storage ()
−> items in storage

3. for all 9 ingredients search_ingredients (query=’item’)
−> ingredient_id of item

4. for all 9 ingredients get_ingredient_nutrition (ingredient_id) −> calorie, protein, fat, and carbs value of ingredient

5. potentially calculations

Use Case 74

Query: ”For my bike ride on Saturday i need to go buy something to fuel on the bike. The snack should have a lot of carbs and little fat and sugar. Which of the following suite these requirements best: banana, rice cake, beer, nuts or bagels?”

Use Case 75

Query: ”For my bike ride on Saturday i need to go buy something to fuel on the bike. The snack should have a lot of carbs and little fat and sugar. Out of the following: banana, rice cake, beer, nuts or bagels? Which has the highest carbs to weight ratio?”

Appendix B

Program Code and Experimental Results

The source code, documentation, the experimental results and a pdf version of the thesis are available at:

`https://github.com/Joel-Arenz/Tool-Augmented-LLM`

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 01.08.2024 Unterschrift

[illegible]