UNIVERSITÄT ZU LÜBECK

Using AutoGPT for Information Retrieval Agents

Nutzung von AutoGPT für Informations-Recherche Agenten

Masterarbeit

verfasst am
Institut für Informationssysteme

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Jakob Horbank

ausgegeben und betreut von
Prof. Dr. Ralf Möller

Lübeck, den 15. April 2024

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Jakob Horbank

Zusammenfassung

Abstract

Abstract

Abstract

Contents

1

Introduction

Encoding knowledge in natural language is an everyday process for humans, but has been a problem for machines for decades.

In recent years, there has been a large increase in language modeling capabilities. After the proposal of the transformer architecture [15], which removed all sequential components from previous language modeling techniques, the architecture and its components were applied to all kinds of language modeling problems.

Humans need to retrieve information from somewhere constantly. It can be a task like digging out an old and hidden memory or finding a book that contains the desired information. Improving the efficiency of tasks like the latter has been extensively researched in the information retrieval area in the last decades. With the introduction of Internet search engines and data repositories, the barrier to accessing information has been drastically improved.

1.1 Contributions of this Thesis

Three main contributions of this Thesis are

1. An analysis of AutoGPT and its default agent for information retrieval tasks.
2. An agent for information retrieval that uses retrieval augmented generation.
3. A way to benchmark an information retrieval agent using the AutoGPT benchmarking system with a synthetic IR dataset

1.2 Related Work

There are different attempts at creating LLM outputs with sources. Perplexity AI hosts a question-answering service, that gives sources from websites.

LLM Agent Tools

AutoGPT was the first attempt to make LLM autonomous, but other projects have since been published. Each project focuses on different aspects of agent behavior

BabyAGI is a minimal task-driven autonomous agent, similar to the nanoGPT project.

LLM Information Retrieval Tools

Since the release of more capable language models, Many developers work on tools that use these models for information retrieval

In a variety of application contexts, the answers of an assistant have to be correct. This is especially true in research contexts. A model that hallucinates isn't feasible in this case. But while hallucination can be reduced with fine-tuning, it can not be eliminated. Perplexity AI is an online service that leverages a language model to provide a search that generates an answer from different sources on the internet. The content of the answer is then linked to the found sources, so the user can see and verify the result.

As this is a proprietary closed-source product accessible through a web interface, this is not useful to create our custom research assistant. The provided API simply hosts different language models and allows prompting them. There is no possibility of fine-tuning.

1.3 Structure of this Thesis

In chapter 2 I will introduce the main topics are relevant to this work. Chapter 3 will then give an introduction to AutoGPT and its core elements, as well as an analysis of the shipped information retrieval capabilities. The RAG Agent will be presented in chapter 4. In chapter 5, I use the AutoGPT benchmarking system to evaluate the RAG Agent on generated test prompts.

2

Backgrounds

Different branches of research were used to build upon this work. These topics and how they are connected to my work are explained in more detail in this section.

A core concept in the creation of artificial intelligence is *agents*. An introduction to agents is given in section 2.2. In section 2.1 I will describe the process of *information retrieval* on a high level. Finally, in section 2.3 the topic of *language modeling* and the current developments of large language models are explained in more detail.

2.1 Information Retrieval

Retrieval of information is essential for humans. Information can be anything, like things seen by the eye, thoughts or an article from a book. Retrieving the best documents for a query from a large database filled with information of different kinds is a classic problem in computer science. There has been extensive research on database architectures and indexing algorithms. Popular applications of IR algorithms are search engines that enable fast access to billions of documents on the internet. Before retrieving information from an information retrieval system, the data is preprocessed and stored. Then a user query initiates a series of steps that can rewrite the query, retrieve relevant documents, re-rank the relevant documents and finally present the information to the user.

Information Retrieval Steps

Information retrieval systems consist of multiple steps. Typically, the user interacts with an information retrieval system by writing out a query that describes the retrieval topics. A problem with queries written by humans is that they are not optimized for the following steps of the retrieval system. Human writing sentences can include spelling mistakes missing words or even wrong descriptions of information. In IR systems researchers try to mitigate such problems with query rewriting. A core query rewriting technique is to expand the query with more words to improve the retrieval accuracy.

To retrieve documents that correlate to the search query, different algorithms and models have been proposed. For a long time, functions like the BM25 [9] were often used

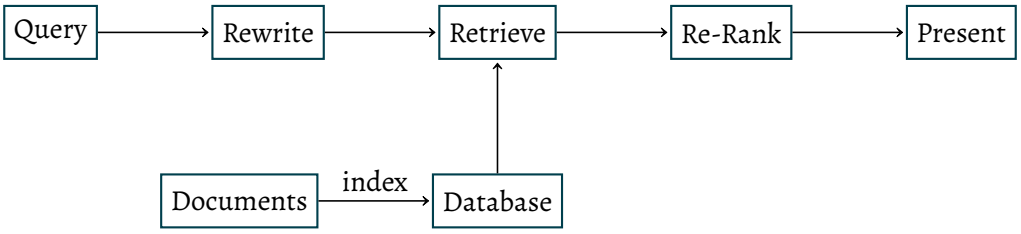

Figure 2.1: Every information retrieval process starts with a user query. The database is then queried to retrieve relevant documents to the query. Before presenting an answer to the user, the retrieved documents are ranked by another relevance measure.

to retrieve the best matching documents from a corpus. Term-based bag-of-words functions like BM25 rank documents based on word occurrences in every document. With neural network approaches becoming more popular, the focus has shifted toward mapping documents to high-dimensional vectors called embeddings. A similarity matrix can then be used to calculate the distance between to embeddings corresponding to the query and a document.

After a set of relevant documents was retrieved with an efficient retrieval method, the documents are re-ranked by their relevance. This time the algorithms used for ranking the documents are specialized towards quality rather than efficiency. In addition, the re-ranking phase can include task-specific ranking strategies to meet user demands.

In the last step, the information is presented to the user. Large language models have become a popular method to create user-friendly responses from retrieved documents.

Before documents can be retrieved from a database, they have to be stored. How documents are stored is a crucial part of creating an information retrieval system. While some data sources are delivered in structured formats like JSON or XML, academic texts are published in journals, articles or conference proceedings. All of these mediums are distributed as PDFs which is an unstructured data format. The PDF format does not save the content A preprocessing step is required before storing the documents to extract information from unstructured formats. There are different techniques to preprocess text before storing it. With *tokenization*, sentences are split up into words, phrases or symbols. Words with little information such as articles and prepositions are removed from the word list. The words are called *stop words*. Furthermore, a *stemmer* can be used to group words that have the same stem. All of these preprocessing steps help to capture the semantics of a text or sentence.

Evaluation

The evaluation of an information retrieval system is not a trivial task, as it is not clear what the human user searches for. Different information retrieval evaluation strategies are categorized into subjective and objective evaluations.

Objective evaluations use metrics that can be computed and compared over time.

Subjective evaluations are needed because

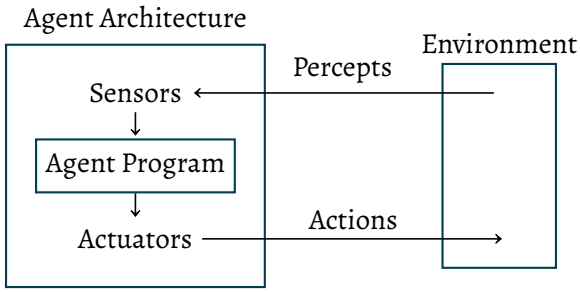

Figure 2.2: An agent as defined in Russel and Norvig. The agent program runs on the agent architecture. Agents perceive their environment through sensors and act upon it using actuators. The agent program continuously maps perceptions to action and defines the agent type.

2.2 Agents

Rationality is viewed as a core component of intelligence. In computer science, computational entities that act rationally are called agents. What it means to act rational is, and will stay an open research question for a long time. However, to use the notion of an agent in practical work a more concrete definition has emerged. An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators [10]. A human perceives the world through eyes and can act with hands or speech, a software can perceive and act through software interfaces. Both examples can be called agents.

A key step when constructing an agent is to define the *task environment*. The task environment consists of a performance measure, the environment, the actuators and the sensors [10]. Environments can be physical but also purely virtual, we are then using a software agent.

Agent Types

An agent consists of an agent architecture and an agent program. While the agent architecture makes perceptions and actions through sensors and actuators available, the agent program implements the mapping from perceptions to actions. Agent programs can be categorized into four basic types that almost all agents are based on [10]. What kind of agent is useful depends on the given task environment.

Simple reflex agents Simple reflex agents are the simplest kind of agent. They choose actions based on the current perceptions looking at the perception history. This means that even a bit of unobservability can break the agent.

Model-based reflex agents Model-based reflex agents deal with partially observable environments by keeping an internal state of the world. This internal state depends on the perception history. Modeling physical or mental states is a complex topic that is heavily researched.

Goal-based agents In many cases, the perception history is not enough to choose the best action. To be able to do that a goal is required. Goal-based agents search for action sequences that end in a goal state. This process is called planning.

Utility-based agents Even with a goal in mind, there are cases where more than one action sequence leads to the goal state. To decide which action to select, utility-based agents choose the action that maximizes the expected utility. The utility function of the agent should ideally match the performance measure of the task environment.

To improve the performance of an agent it *has to learn*. All agent types can be extended to learning agents. For an agent, learning means modifying each component such that it better aligns with feedback from a new critic component [10]. As a result of these modifications, the agent performance improves.

Large Language Model Agents

With the rise of large language models and their impressive capabilities in various language tasks, efforts began to leverage them in agent systems. Experiments to give GPT the option to access tools [12, 11] showed that it was possible.

Mention AutoGPT and link to chapter

Four key reasons why large language models are suitable as agent brains are outlined in [17]. Agents should act autonomously without direct interventions from humans. The generative capabilities and dynamically adjusted outputs based on the input fit that characteristic.

Move agent backgrounds from agent chapter to here?

For practical applications [19], a distinction between static and dynamic agents is proposed. A static agent is characterized as a fixed pipeline that mimics the user behavior. While this approach works, it cannot deal with complex and sometimes random human actions. The other type of agent can dynamically execute actions that are presented to it. The most prominent way of using LLM for agents is to build a prompt that contains all the information about the task, and then ask it to propose the next action. The prompt can contain descriptions of possible abilities, the task, guidelines, information about the environment and more.

For LLM agents we do not have all the components of a classical agent that were listed in the previous section about agent types. [13]

2.3 Language Modeling

Language modeling is an important research area of computer science. In recent years, developments have significantly sped up with the introduction of deep learning into language modeling. The transformer network [15] has been the base for many advancements in recent years. Deriving from the transformer network, massively scaled-up pre-trained transformer models [2] enabled a big leap in the capabilities of language processing models.

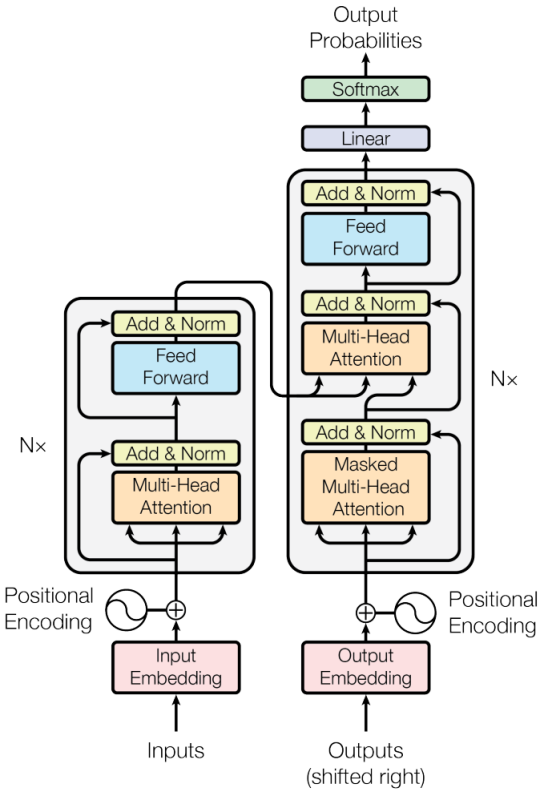

Figure 2.3: The encoder-decoder architecture of a full transformer network [15]. The encoder (left) can attend overall positions to learn rich embeddings. The decoder (right) can generate output sequences. The first decoder attention layer can only attend to previous positions of the input sequence, while the second can attend to the outputs of the encoder. This combines a sequence-to-sequence with autoregressive properties into the decoder.

Transformer Networks

Using recurrent structures such as recurrent neural networks and long short-term memory [3] was the dominant strategy in sequence modeling before transformer networks [15]. Every sentence token was represented as a hidden state that is a function of all previous hidden states. While this approach has a reasonable motivation, the sequential nature constrains computation speed for a single training example. This limitation is especially hindering for longer sequences, where batching the training data is only possible to a memory limit. Attempts to minimize sequential computation included convolutional networks that can be computed in parallel [5]. For these models, however, the number of operations required to relate two tokens grows in the distance of their positions in the sequence. Attention mechanisms allow modeling token relationships independent of their distance in a sequence.

The transformer network [15] shown in Figure 2.3 was the first model that removed all recurrent structures and only relies on attention mechanisms. In particular, they use multi-headed self-attention layers. Attention mechanisms learn dependencies between

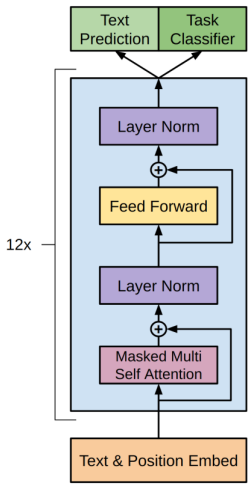

Figure 2.4: The GPT architecture [8]. A transformer encoder without connections to a previous encoder is used. The attention layer can only attend to previous positions in the input sequence. The decoder is stacked 12 times before generating the final output.

tokens in sentences without regard to their distance. Their non-sequential characteristics allow for better parallelization. Self-attention is an attention mechanism that computes relations between different positions of the same sequence to find a representation of the sequence.

Since the transformer architecture does not use any recurrent structures, the order of the tokens in the sequence must be manually injected. Positional encodings are added to the input embeddings to achieve this.

Like most language modeling networks, a transformer consists of an encoder and a decoder. The encoder has two sub-layers, a multi-head attention block and a fully connected feed-forward block. The decoder is similar to the encoder but has an additional multi-head attention layer that attends to the outputs of the encoder. The first attention layer is masked, and the decoder input sequence is shifted to the right, so only previous positions can be attended by the decoder.

The full architecture has the capabilities of a classic sequence-to-sequence model used for tasks like language translation. All positions in the decoder can attend to all positions in the encoder. For other tasks, however, the individual encoder and decoder sections are a better fit, as explained in the sections 2.3 and 2.3.

Generative Pre-Trained Transformers

Decoder-only transformers are the base of generative pre-trained transformers (GPT). The decoder used for GPT does not rely on the outputs of an encoder, because it is designed for generation tasks. In Figure 2.4 we can see that only the masked multi-head attention layer is kept in comparison to the full transformer network. The simple architecture enables faster pre-training and fine-tuning.

GPT is trained in two phases. In the unsupervised pre-training phase, the model is

trained with big datasets of unlabeled text data. The training objective here is to accurately generate the next token of a sequence.

After the first pre-training phase, the model needs to be fine-tuned for a specific task. By including custom tokens in the fine-tuning dataset, the decoder can learn to handle a variety of different tasks such as text generation or classification.

Newer research on generation models focuses on scaling the network. The *scaling law* states that by simply increasing the number of parameters of the network, the model capabilities increase. Furthermore, after a certain level of scaling, abilities emerge that the model was not directly trained on.

As large language models have reached parameter counts over a billion, only very few companies have the hardware to train or even run inference tasks.

Embedding Models

The left-to-right nature of the transformer encoder suits text generation tasks that only depend on previous tokens. However, there are dependencies between tokens in both directions, which means a decoder-only model can not capture all relations.

The amount of knowledge learned in the pre-training phase is encoded in embeddings. Embeddings are high-dimensional vectors that represent a sequence of words or tokens. Semantically similar documents are represented with embedding vectors that have a small distance, while semantically different documents have a high distance.

BERT [4] uses the encoder part of a transformer network, to create rich embeddings of input sequences. In pre-training, the bidirectional encoder is trained with two unsupervised tasks. For the first task, random tokens in the sequence are masked out to predict the masked tokens from the remaining tokens in the sequence. In the second task, sentence-level relationships are learned by predicting the next sentence. After pre-training, BERT can be fine-tuned on different downstream tasks leveraging the rich bidirectional embeddings learned in pre-training.

Instruction-Tuned Models

Language models are trained to predict the next token of a sequence. While a lot of knowledge is captured in the weights of the model, most information on the internet is not formatted in a conversational style. Because of this, language models need to be prompted in a specific way to be effective. The prompt needs to be written such that its continuation yields the desired output. This is not optimal for human users, as one would rather write in a conversational style. The training objective of large language models is different from the objective "follow the user's instructions helpfully and safely" [7]. Researchers say that the language model is not *aligned*. A popular attempt at aligning language models is reinforcement learning with human feedback (RLHF) [7].

Instruct models are fine-tuned versions of language models. Capturing the intent of the user is a key challenge for language models. This process is called *alignment*. A popular approach to the alignment problem is reinforcement learning with human feedback (RLHF) [7]. Handcrafted prompts are used to fine-tune GPT-3. The outputs of the model are collected into a set and ranked by humans. This set is then used to train a reward

model. With this reward model, the language model is further fine-tuned. The resulting model is called *InstructGPT* and performs better than the baseline GPT-3 model.

Large language models are trained to predict the next token of a sequence, not to follow the instructions of the user. This leads to some unwanted results such as toxic, harmful or fabricated answers that are not true.

3

Analysis of AutoGPT for Information Retrieval Tasks

The general notion of an agent in computer science was introduced in section 2.2. AutoGPT is an open-source project that tries to 'make GPT fully autonomous'. It started as a collection of loose scripts but quickly gained a lot of interest in the open-source community and grew into a much bigger and now-funded project. Initially, it only contained the AutoGPT agent that I will analyze in this chapter but over time has seen multiple additions. An agent framework called *forge*, a benchmarking system that was put in place to host an agent hackathon and creation of the *agent protocol*. The agent protocol is an attempt by the community to standardize agent applications.

The AutoGPT was not built with a focus on information retrieval, which is why I will analyze the information retrieval capabilities in this chapter.

GPT language models are used to control an agent that works towards reaching a stated goal. The project contains a core agent with a predefined set of abilities. Additionally, a baseline SDK is being developed to build custom agents.

3.1 LLM Agents Backgrounds

Move section into background chapter

The concept of agents in computer science is not new. An agent is a system that acts towards reaching a goal in an environment. Agents can be implemented as software as physical robots or even humans. In the same way, different environments are possible such as the real physical world, a web browser or a simulation. The agent needs ways to sense its environment, which can be done by sensors in a physical environment or programmatically in a software environment. A task has to be specified to the agent, so it knows what his goal is. It can then employ different strategies to reach that goal. These strategies consist of planning steps to execute. While acting out these steps, the environment will probably change as time passes, so an agent needs to reevaluate its plan and the contained steps.

The increasing capabilities of large language models have led to research implementing them into agent systems.

3.2 Project Overview

- Default agent
- Forge agent
- evo ninja
- Benchmarking
- ACtivity Decline
- GUI

Originally, the default agent lived in the computer terminal and was controlled through the command line. Later, an option to start a server that serves the agent protocol was added. The user can interact with the agent through a GUI frontend running in the browser.

The web interface lists all conversations an agent had and provides a chat interface for the selected one. When the user sends an input to the agent, a task or step request is made to the server running the agent. Furthermore, the web interface can be used to start benchmarks for an agent. A single test or a complete test suite consisting of stages can be started. The GUI has no support for uploading documents to an agent. Because of this, documents for information retrieval would need to be placed in specific folder locations such that the agent can access them before it is started.

AutoGPT agents use workspaces in which they can act. Each agent has its own workspace and can not modify files outside it by default. If the user wants the agent to have access to documents he needs to place them into its workspace by hand. Often, the workspace is also used to save intermediate information to text files. Some abilities produce large outputs, and the limited context window of large language models can be exceeded quickly if all intermediate information is appended to the prompt.

3.3 Architecture Overview

How should this differ from the Forge and IR agent overview?

The AutoGPT agent is modeled after the classic agent architecture. After the start, the user is asked to enter a task the AutoGPT agent should perform. Then the agent enters a loop of prompting the LLM, executing the proposed action handling the result of the action and updating the agent state.

The LLM is prompted in a structured way. A base prompt template is defined and populated with current information before each prompting step. The information includes the task at hand, a list of possible actions, a history of previous actions and their results and some extra statements that are there to guide the language model. As the answer needs to be parsed, the system prompt defines a fixed format the LLM should answer in. The answer consists of the thoughts and the proposed next action.

AutoGPT is divided into four modules. The *brain* is the main module that controls the agent. In AutoGPT this is realized by prompting the language model in a structured way.

Using the chat system prompt, the language model is prompted to answer in a structured format. Different techniques are implemented in this structure. The language model is forced not only to plan the next step but also to explain the choice for the chosen step and to add self-criticism. An extra output for the human user is also returned. The second part of the answer is the actual next action with the needed arguments. The action makes up the second module of the agent. In this module, the abilities of the agent are defined. These can be file operations, database queries or web search functionalities. The third module is the *memory*. Memories are modeled after humans which have short and long-term memory. Short-term memory can be implemented as an in-memory list of messages to the language model. Long-term memory needs persistent storage such as a database. A popular option for language models is vector databases that work with embeddings.

Currently, the AutoGPT agent is implemented for OpenAI GPT models. The prompts are tailored in ways that benefit the characteristics of GPT-3 and GPT-4. Switching to a different model is not difficult from a software perspective, but semantically poses a challenge. If one knows the message format for a specific model the input can be adjusted. However, the same prompting techniques do not necessarily work for all language models. The challenge is to switch between different prompting styles, as every model needs to be prompted differently. For example, OpenAI models benefit from profile sentences like "You are an expert in computer science", while Anthropic Claude does not...

3.4 Information Retrieval Capabilities

The AutoGPT Agent has different abilities that can be utilized for information retrieval. It can search the web and operate on files, execute code and

The web search is implemented by a two-step process. First, a search API like DuckDuckGo is called to get a list of relevant pages. Then the page contents are scraped with a headless browser. It is possible to read and write text files. Other document types are processed by basic text extraction tools to get the plain text.

For longer files such as scientific journals the extracted text is too long for the language model. The AutoGPT agent cannot chunk the text into smaller chunks or store it in a database. This is a limitation that needs to be addressed for information retrieval tasks over a research database repository.

Having a vector database would enable techniques such as retrieval augmented generation. The agent would get a prompt with a question over the RDR and choose an action to start a semantic search over the vector database. The result of the search is the chunks that are semantically closest to the question. These chunks can then be included as context for the LLM prompt to generate an answer.

The default agent tends to search the web for information. We want an agent that prioritizes information that is present in the research repository. This needs to be addressed in the prompting techniques of the agent.

4

Retrieval Augmented Generation Agent

Large language models are excellent at generating text in a variety of styles. But when prompted about specific topics the answer quality suffers. Different approaches try to tackle this problem. A promising method is retrieval augmented generation, which combines in-context learning prompting techniques with a vector database. In this section, I present an agent that is designed to answer user prompts in a retrieval-augmented fashion.

To create the information retrieval agent, I used the Forge SDK [1] that is included in AutoGPT. The Forge agent SDK is a tool to create a custom agent without having to write the boilerplate code. In comparison with the AutoGPT agent, it comes with less predefined abilities and has no initial logic. On the contrary, fewer parts can break, and it is not under an ongoing re-factoring process, compared to the AutoGPT agent. Additionally, the AutoGPT agent is not optimized for information retrieval over a set of documents but operates more as a general-purpose agent.

I extended the Forge agent with abilities that are needed for retrieval augmented generation. To store the documents for retrieval, I used a *Chroma* vector database. The 'ingest' ability of the agent can be used to ingest a PDF file that is saved in the workspace. The PDF is converted into plain text and then chunked into smaller documents using the *SentenceSplitter*, which produces chunks of roughly equal length while respecting sentence boundaries.

4.1 Methods To Improve LLM Generation

Different approaches try to embed information sources into the generated text. One approach is to fine-tune the language model on a dataset that contains the information. The creation of fine-tuning data is an expensive task, as data needs to be gathered and cleaned to produce good results in training. There is some emerging work on generating synthetic datasets by using large language models as data augmentation tools. But even if the fine-tuning data quality is sufficient, it is still a challenge to make an LLM expert in a specific domain. Rather than learning new knowledge, fine-tuning is best suited to guide the model toward a certain answering style. Furthermore, fine-tuning models with billions of parameters is only possible on expensive hardware and therefore not feasible

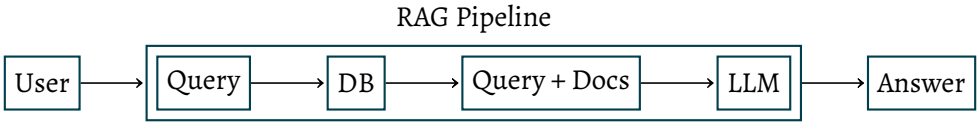

Figure 4.1: Instead of directly answering the user prompt with an LLM, a retrieval augmentation system leverages context to generate better answers. After receiving the user prompt, it is used to retrieve relevant documents from a database. These documents are appended to the language model prompt as context. The model is instructed to answer only using the provided context.

for on-demand tasks.

Following up on the challenges of fine-tuning researchers have searched for ways to optimize prompts to get the best model performance. These methods make use of a phenomenon called in-context learning. In-context learning describes the observation that giving a large language model more context surrounding the prompt can drastically improve the response quality. [16] found that adding a sentence that suggests step-by-step thinking to solve a problem makes the model better at solving logical questions. For GPT models, [18] showed an improvement after giving the language model a hint that it is an expert in a certain topic. The findings in the area of prompting techniques and in-context learning are suggesting that prompt optimizations have a high potential of getting the most out of large language models.

A more promising approach is retrieval augmented generation [6]. Before generating an answer to a prompt, a vector database is searched for relevant information. The prompt then includes the information of the returned documents as context.

4.2 Retrieval Augmented Generation

Large language models can embed a large amount of knowledge into the weights during the pre-training phase. It is possible to generate good answers for different kinds of prompts. But when it comes to specific domain knowledge, the exactness of large language models starts to degrade.

When trying to prompt models about very specific topics common problems such as hallucinations start to show. Not only does the model generate wrong information, it does so with strong confidence. A promising method to make the model answer based on specific sources is called retrieval augmented generation (RAG).

The RAG method combines an information store with in-context learning. A corpus of documents is stored in a database. When the user prompts the system the database is queried with that prompt matching documents are returned. These documents are then included in the prompt to the LLM as context.

To store the information a vector database is used. A vector database uses embedding models to embed each document into a high-dimensional vector space. After a user query, the query string gets embedded by the same model, and then a metric such as the cosine similarity is used to find semantically close documents. The top-k documents are

then returned.

4.3 Agent

I will first explain the default Forge agent. Then will describe how I built the information retrieval agent.

The Forge Agent

To make collaboration with agents easier, the open-source community created the agent protocol. The Agent Protocol defines an API schema that handles the communication with an agent. On a high level, the protocol defines endpoints to create a task and to trigger the next step for the task. The two important concepts of the agent protocol are tasks and steps:

Task A task describes a goal for the agent. A task has an input prompt and contains a list of steps.

Step A step describes a single action of the agent. A step can have custom input or copy the task input. Additionally, there is a variable that signals if this is the last step. If this variable is false, then the next step is requested automatically after completing the current. Every step has to be linked to a parent task.

The Forge SDK handles the boilerplate code that implements the agent protocol. On running, the server with the corresponding endpoints gets started, and the agent can be used over the API endpoints. AutoGPT also comes with a chatbot web app that builds the appropriate HTTP requests to the agent endpoints. The user of the Forge SDK has to create the actual agent logic, create custom prompt templates for the used model and add abilities to interact with external resources. Because the Forge SDK is still under development and by no means a polished product, some internals also need to be tweaked to achieve the desired agent behavior.

By default, the Forge agent comes with abilities to read and write text files and to search a search engine as well as scraping a webpage. Each ability is specific by a name that the LLM can use to call it. Additionally, a short description of what the ability does, input parameters and return types are described. The information about an ability is formatted into a string before adding it to the step prompt.

File System The file system abilities allow operations on files located in the workspace of the agent. The agent can only operate in the defined workspace, this prevents unwanted effects when the agent proposes unexpected actions. Abilities to read, write and list files are present. All the abilities have a path parameter that the large language model has to populate when generating a step proposal with actions from this category.

Web The web search functionality is split up into abilities to call a search engine and to read a webpage The search engine ability requires a search string that is sent to the engine. For the web page ability a URL is needed, and if specific information should be extracted the LLM also has to provide a question.

```
Reply only in JSON with the following format:

{
      \"thoughts\": {
            \"text\":  \"thoughts\",
            \"reasoning\": \"reasoning behind thoughts\",
            \"plan\": \"- short bulleted\
                        - list that conveys\
                        - long-term plan\",
            \"criticism\": \"constructive self-criticism\",
            \"speak\": \"thoughts summary to say to user\",
      },
      \"ability\": {
            \"name\": \"ability name\",
            \"args\": {
                  \"arg1\": \"value1", etc...
            }
      }
}
```

Listing 4.2: The system format of the Forge agent. The language model is asked to only answer in this format. The thoughts before creating the output for the user (speak), the LLM generates reasoning, a plan and criticism. After the model generated its thoughts, it generates an ability proposal with the corresponding arguments.

Finish The "finish" ability terminates the agent loop. The agent is asked to choose this ability if the initial user prompt can be answered and give a reason.

The Forge agent comes with a built-in template engine that is used to populate the prompts before sending them to the large language model. Some templates are included by default:

Task-Step This template is used to create the prompt that is sent to the language model for each step. It includes the current task description and placeholders for extra information. The template always needs to be populated with the list of available abilities, allowing the language model to choose one of them.

System-Format The system format defines how the model should respond to prompts. It is shown in Listing 4.2 The responses of the model need to be parsed according to this definition. Language models have different capabilities in answering in a structured manner, so the format has to be tuned for every model.

Techniques Techniques is a collection of prompting techniques that have been shown to improve generation quality. By default, the Forge agent has templates for few-shot, expert and chain-of-thought prompting.

The core of an agent is its logic. The Forge agent comes without any logic, its default behavior is to write a boilerplate text into a file in the workspace.

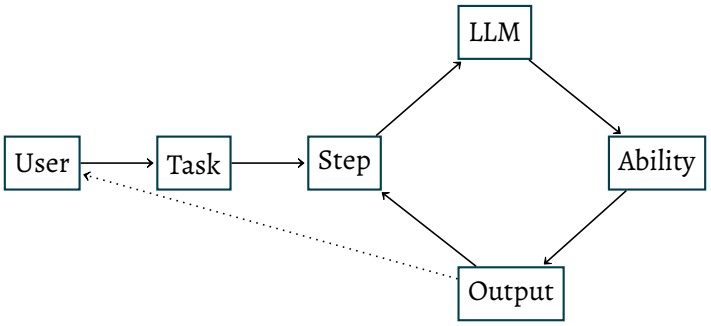

Figure 4.3: The information retrieval agent receives the user prompt at the start of each cycle. Next, the step prompt is constructed with the action history, list of abilities and extra information. After the LLM responds to the step prompt, the proposed action is executed and the step output is presented to the user. The loop ends if the LLM proposes the 'finish' ability.

Agent Memory

Similar to the view of human memory, different implementations of memories for agents have been proposed. For short-term memory, the message history format used by instruct language models can be used. The last n messages of a conversation between the user and the agent are recorded and change the generation behavior of the LLM that controls the agent. For example, a researcher could

Long-term memory is generally represented as some external store where information can be stored and retrieved. This memory can include past conversations between the agent and the user, as well as additional data that the agent should have access to. The information can be present from the beginning or added at runtime by a user prompt.

For large language model applications vector databases are a popular choice, because they integrate well into the embedding-based language modeling concept. Vector databases store data together with their high-dimensional embedding. Embedding models that are specifically trained to produce rich embeddings are used. The agent uses a *chromadb* vector database [14].

Agent Main Loop

The agent program is defined in the agent step execution function. Every time a new step request is given this function is called. For the first user input, a task is created, then the agent proceeds to complete steps until the task goal is reached. A single cycle of the agent follows a fixed set of actions. First, the step input is updated for the current step. The user can give updated instructions in each step to further guide the agent. If no new input is given, the agent uses the input of the parent task as the step input.

After the input is updated the agent constructs the message history for the language model. The first message is always the same system format specification. The agent uses the default Forge agent format shown in Listing 4.2. The second message is the populated step prompt, which describes the current perceived environment of the agent. When the

step prompt is constructed, it is sent to the language model. The generated answer is parsed into the thoughts and ability parts.

The response is parsed into the thoughts of the LLM and the proposed ability. If a valid ability is proposed, it is executed.

Step Prompt

The main step of the agent is to populate its step prompt template As denoted in section 2.2, an agent perceives its environment through sensors. Our agent describes the current environment in the step prompt. An example step prompt is shown in Listing 4.4. It includes the current step input and instructions on how to answer. The available resources are denoted, and the action history is appended.

If the agent is in its first step, the action history is empty. Otherwise, a list of the previous actions is compiled to give the agent a sense of its current state inside the task context. Each entry has the proposed ability with its parameters and the ability output. If the ability produced an error, this is also denoted. In the best practices section, the agent is prompted to react to errors and to not use the same action with the same arguments again.

In addition to the action history, the available abilities are added to the prompt. The abilities enable the agent to act out its decisions in the environment. For each ability, the name, parameters, return type and a short description is given.

Abilities

In section 2.2, I described how agents use actuators to perform actions in the environment. The IR agent is a software agent that uses functions to modify its workspace. In particular, it has abilities that enable retrieval augmentation capabilities. These abilities will be described in the following.

Ingest The *Ingest* ability should be used when a document should be a source of information for the agent. A file with the specified filename has to exist in the agent workspace before calling this function. This ability converts the PDF into plain text, creates text chunks of the same length and calls the functions to embed them into the vector database.

Retrieve The *Retrieve* ability accepts to query string and an output file path. With the query string, the agent queries its memory and gets the most relevant documents for it. The documents are formatted and saved to the specified file path.

Answer This ability uses the contents of a text file to populate the augmented generation template. The populated template is then sent to the LLM to generate an answer based on that context.

As the LLM tends to choose web search abilities to collect information, I removed the corresponding abilities. The agent is now forced to retrieve information from local sources.

```
Answer as an Expert in Planner.
Your task is:

What characterized prompt book usage in the late eighteenth
century?

Answer in the provided format.

Your decisions must always be made independently without seeking user
    assistance. Play to your strengths as an LLM and pursue simple
    strategies with no legal complications.

## Resources

You can leverage access to the following resources:

- A vector database representing your memory. You can ingest documents into
    it and query it for relevant information.

## Abilities

You have access to the following abilities you can call:

- answer_with_context(prompt: str, context_file: str, output_file: str) ->
    None. Usage: Answer a prompt using content
from a textfile as context,

- retrieve_context_from_memory(query: string, output_file: string) ->
    string. Usage: Retrieve the most relevant information for a query and
    save it to a .txt file,

- finish(reason: string) -> None. Usage: Use this to shut down once you
    have accomplished all of your goals, or when there are insurmountable
    problems that make it impossible for you to finish your task.

...
## Best practices

- Prefer querying your memory to retrieve source before answering.

...
```

Listing 4.4: The step prompt is the core of each step the agent takes. It describes the current environment representation of the agent. First, the current task is presented to the agent. Next, available resources and abilities are described. Finally, some hints for best practices are denoted. In this example, there is no step history as this is the first step. A full step prompt example is show in

Language Model Response

The populated step prompt is sent to the language model. To react to the model answer, the agent parses the generated response. From the thoughts part of the answer, only the 'speak' string is used as an output to the user. The second part contains the action proposal for the step.

5

Benchmarking for an IR Agent

The evaluation of large language model agents is a difficult task, as evaluating LLMs themselves presents a challenge. There are different approaches to evaluating systems built around language models. Subjective evaluation is based on human feedback. As LLM systems are generally made to serve humans this is an important part of evaluation. On the other hand, quantitative metrics that can be computed are used for objective evaluation. Different metrics are used for different tasks. Another important method is benchmarks. Benchmarks are a set of tasks or an environment that the agent is to move in.

5.1 Existing IR Agent Benchmarks

As the space of possible agent domains is large, lots of different benchmarks were proposed. Simulation environments like

Although lots of benchmarks for LLM applications have been proposed, few benchmarks are designed to test the information retrieval capabilities of an agent. Some benchmarks are used to evaluate information retrieval in general and in diverse domains.

5.2 The AutoGPT Benchmarking System

To evaluate AutoGPT and other agent systems that implement the agent protocol, the AutoGPT project has implemented a benchmarking system. The system consists of a set of tasks that the agent has to complete. The tasks are designed to test different aspects of the agent and are divided into different topics. Some tasks depend on the previous successful completion of other tasks. A task consists of an input prompt and an expected output. The output is defined by certain words that should be contained.

- Level-based system
- Dependencies

5.3 Custom Benchmarks for Local IR over Journals

– Retrieval benchmarks

6

Results

While working with AutoGPT and developing the information retrieval agent, a couple of weaknesses of current LLM agent systems became visible. The most important challenge for LLM agents is the non-deterministic generation of large language models. It makes it significantly harder to build a software system around it because every time the language model is called, there has to be a fallback mechanism in case something fails. For fixed pipeline systems this is true as well, but the error handling is easier as the task steps are known beforehand.

6.1 Points of Failure

Using large language models as agent controllers is a recent idea, and therefore there is a lot of experimentation left to do. A challenge when developing such agents is the natural language interface that language models communicate with. While it makes describing the task easier for humans, it complicates the internal communication in the software. For example, AutoGPT wants a certain system format that the LLM should respond in. While this works most of the time, it is not guaranteed that the answer complies with the format. A small deviation such as a missing bracket can break the parsing process. Therefore, a lot of effort and time is put into creating better tools and frameworks that handle such mistakes, instead of doing actual research on the agent performance.

Another aspect is the non-deterministic generation of large language models. For the same prompt, large language models can generate a different answer. This answer will probably be semantically similar to the previous one, but when a model has to choose between two abilities, this small difference can decide if the agent succeeds or fails at completing the task. Like earlier, developers have to spend time handling these cases by re-prompting several times or putting fallback actions in place.

For a lot of tasks, a static pipeline is enough to meet the user demands. Almost all retrieval tools that work in production use a fixed pipeline.

6.2 Benchmarking Results

6.3 Subjective Evaluation

7

Conclusion

Using language model agents for information retrieval tasks is an interesting approach, but needs to overcome some key challenges for usage in real-world applications.

7.1 Future Work

The research area about large language models currently moves at a rapid pace. While writing this thesis, communities have focused more on agent applications of LLMs. OpenAI has released OpenAI Assistants, which.... The AutoGPT team is still working on making the use of local models feasible. In this work, only information retrieval over local documents was done. This could be extended with web searching capabilities to include external information on demand. For retrieval augmented generation, the ingestion of documents into the vector database is a crucial step. The popular chunking methods for unstructured data like PDFs simply split the text to get a certain chunk size. Methods for semantically splitting up unstructured data might improve the performance of retrieval augmentation pipelines.

Bibliography

[1] *AutoGPT Forge Github*. (Visited on 03/01/2024).

[2] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language Models Are Few-Shot Learners. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. (Visited on 03/08/2024).

[3] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. Dec. 11, 2014. DOI: 10.48550/arXiv.1412.3555. arXiv: 1412.3555 [cs]. (Visited on 03/06/2024). preprint.

[4] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of NAACL-HLT*. 2019.

[5] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. *Convolutional Sequence to Sequence Learning*. July 24, 2017. DOI: 10.48550/arXiv.1705.03122. arXiv: 1705.03122 [cs]. (Visited on 03/06/2024). preprint.

[6] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.

[7] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training Language Models to Follow Instructions with Human Feedback. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 27730–27744.

[8] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. *Improving Language Understanding by Generative Pre-Training*. OpenAI, 2018.

[9] Robertson, S. and Zaragoza, H. The Probabilistic Relevance Framework: BM25 and Beyond. In: *Foundations and Trends in Information Retrieval* 3(4):333–389, Apr. 1, 2009. ISSN: 1554-0669. DOI: 10.1561/1500000019. (Visited on 03/06/2024).

[10] Russel, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Fourth Edition, global edition. Pearson, 2022. ISBN: 978-1-292-40113-3.

[11] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. *Toolformer: Language Models Can Teach Themselves to Use Tools*. Feb. 9, 2023. DOI: 10.48550/arXiv.2302.04761. arXiv: 2302.04761 [cs]. (Visited on 03/18/2024). preprint.

[12] Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. *HuggingGPT: Solving AI Tasks with ChatGPT and Its Friends in Hugging Face*. Dec. 3, 2023. DOI: 10.48550/arXiv.2303.17580. arXiv: 2303.17580 [cs]. (Visited on 03/18/2024). preprint.

[13] Significant Gravitas *AutoGPT*. (Visited on 03/18/2024).

[14] *The AI-native Open-Source Embedding Database*. (Visited on 03/18/2024).

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention Is All You Need. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[16] Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 24824–24837.

[17] Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., et al. *The Rise and Potential of Large Language Model Based Agents: A Survey*. Sept. 19, 2023. DOI: 10.48550/arXiv.2309.07864. arXiv: 2309.07864 [cs]. (Visited on 02/29/2024). preprint.

[18] Xu, B., Yang, A., Lin, J., Wang, Q., Zhou, C., Zhang, Y., and Mao, Z. *ExpertPrompting: Instructing Large Language Models to Be Distinguished Experts*. May 23, 2023. DOI: 10.48550/arXiv.2305.14688. arXiv: 2305.14688 [cs]. (Visited on 03/14/2024). preprint.

[19] Zhu, Y., Yuan, H., Wang, S., Liu, J., Liu, W., Deng, C., Chen, H., Dou, Z., and Wen, J.-R. *Large Language Models for Information Retrieval: A Survey*. Jan. 19, 2024. DOI: 10.48550/arXiv.2308.07107. arXiv: 2308.07107 [cs]. (Visited on 03/06/2024). preprint.