HUMAN WEB BROWSING SIMULATION

SIMULACE PROCHÁZENÍ WEBU ČLOVĚKEM

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JÁCHYM DOLEŽAL**
AUTOR PRÁCE

SUPERVISOR **Ing. RADEK HRANICKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

Bachelor's Thesis Assignment

153702

Institut:	Department of Information Systems (DIFS)
Student:	**Doležal Jáchym**
Programme:	Information Technology
Title:	**Human web browsing simulation**
Category:	Web
Academic year:	2023/24

Assignment:

1. Study technologies for automated web crawling (Selenium, Puppeteer, WebdriverIO, etc.)
2. Learn how to simulate the behaviour of a real person visiting the web.
3. In consultation with the supervisor, design a tool to simulate a person's visit to the website. The resulting solution can be used for testing web servers or generating datasets of web requests. The tool will be configurable, with multiple human behaviour profiles to choose from, configure the frequency of requests, etc. Try to achieve as much similarity to organic traffic as possible.
4. Implement the proposed tool.
5. Experimentally verify the usability of the developed solution.
6. Evaluate the results obtained.

Literature:

- Kant, K., Tewari, V., & Iyer, R. (2002). Geist: A web traffic generation tool. In *Computer Performance Evaluation: Modelling Techniques and Tools: 12th International Conference, TOOLS 2002 London, UK, April 14–17, 2002 Proceedings 12* (pp. 227-232). Springer Berlin Heidelberg.
- Barford, P., & Crovella, M. (1998, June). Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems* (pp. 151-160).
- Han, X. P., Zhou, T., & Wang, B. H. (2008). Modeling human dynamics with adaptive interest. New Journal of Physics, 10(7), 073010.
- Matam, S., & Jain, J. (2017). *Pro Apache JMeter: web application performance testing*. Apress.
- Zhu, W., Gao, H., He, Z., Qin, J., & Han, B. (2019). A hybrid approach for recognizing web crawlers. In *Wireless Algorithms, Systems, and Applications: 14th International Conference, WASA 2019, Honolulu, HI, USA, June 24–26, 2019, Proceedings 14* (pp. 507-519). Springer International Publishing.

Requirements for the semestral defence:
Points 1 to 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Hranický Radek, Ing., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	9.5.2024
Approval date:	30.10.2023

Abstract

This work introduces a promising tool for automated web navigation and achieving specific goals based on decisions made by a Large Language Model using information from a current page. The results of the simulator with model GPT 4 Turbo demonstrate the tool's effectiveness, achieving over 80% success in completing predefined goals. The results show the usability of this tool in real use cases.

Abstrakt

Tato práce představuje slibný nástroj pro automatizovanou webovou navigaci a plnění specifických cílů podle rozhodnutí daných velkým jazykovým modelem, který používá aktuální informace z dané stránky. Výsledky simulátoru s modelem GPT 4 Turbo demonstrují efektivitu tohoto nástroje, který dosahuje úspěšnosti přes 80% v dokončování předem definovaných cílů. Výsledky dokazují použiteljnost tohoto nástroje v reálných případech užití.

Keywords

Human behavior simulation, Transformers, OpenAI, Web technologies, Technological advancements, Large Language Models, Website automation

Klíčová slova

Simulace lidského chování, Transformátory, OpenAI, Webové technologie, Technologický pokrok, Velké jazykové modely, Automatizace webových stránek

Reference

DOLEŽAL, Jáchym. *Human web browsing simulation.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Radek Hranický, Ph.D.

Rozšířený abstrakt

Cílem této práce bylo vytvořit simulátor schopný napodobit člověka při procházení webu. Při využití technologií, jakou jsou velké jazykové modely, zejména modelu GPT-4 Turbo od společnosti OpenAI, se tato práce snaží překročit propast v technologiích umožnující simulaci lidského chování při procházení webu. Jazykové modely i přes svá omezení, jakou jsou omezené kontextové okna a výskyt halucinací, představují slibnou technologii pro simulaci lidského chování díky schopnosti rozumět přirozené lidské řeči a možnosti modelování lidských rozhodnutí.

Výsledkem této práce je simulátor, který simuluje lidské interakce na webových stránkach, založené na textové specifikaci cíle a webové stránky. Tento nástroj by mohl najít uplantění v mnoha oblastech, od testování webových stránek až po rozpoznávání webových botů a vývoj aplikací poháněné umělou inteligencí.

Simulátor představen v této práci umožňuje plnění předem zadaného cíle na dané webové stránce. V jádru simulátoru se nachází jazykový model, který který vybírá vhodné akce pro splnění daného cíle na základě předchozích akcí a aktuálních informacích o dané stránce. Pomocí využití knihoven LangChain a Selenium je aplikace vytvořena a schopna plnění širokého množství cílů na různých webových stránkách. Cíle se mohou týkat webové navigace a získávání dat z webových stránek.

Závěry experimentů v této práci ukázaly, že simulátor s modelem GPT-4 Turbo dosahuje úspěšnosti nad 80%. Současně výsledky při porovnání s lidskou navigací na webu prokázali shodu rozhodování mezi simulátorem a člověkem. Závěr práce naznačuje, jak by mohl být tento nástroj využit v realných případech užití a zdůrazňují potenciál pro další vývoj.

Budoucí výzkum by se mohl zaměřit na testování simulátoru na širší a detailnější škále webů a srovnání výsledků s detailnější datovou sadou lidské interakce na webu. Dále závěr práce mluví o možnosti rozšíření simulátoru kde by byl schopný vytvářet své vlastní cíle na základě definovaného profila uživatele.

Human web browsing simulation

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Hranického Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . .
Jáchym Doležal
May 8, 2024

Acknowledgements

I would like to extend my heartfelt gratitude to my supervisor, Ing. Radek Hranický Ph.D., for his invaluable guidance and insights throughout this project. His encouragement helped me to push this project forward. I would also like to express my gratitude to my family and girlfriend, whose constant support was essential for completing this project.

Contents

Chapter 1

Introduction

The merits of technological advancements were usually the automation of tasks too repetitive for humans. With advancements in Machine Learning and AI models, further opportunities for automation are possible. Websites typically have unique structures based on their purpose. Even two given websites of similar categories have completely different structures. Therefore, it is a challenging task to create an algorithm that would orientate inside such a website. To create a simulator that can simulate humans browsing a website, choosing a technology that can analyze text and create action based on the given technology is crucial. The tasks of NLP and NLU are among the fastest branches in computer science. The invention of Transformers in 2017 [30] sped up the field advancements. Nowadays, many influential models are capable of understanding language and simulating humans.

This thesis takes the creation of a simulator that can simulate human web browsing. The task of simulating human behavior is a very complex one. The most complex model nowadays is GPT-4 created by OpenAI, a large language model trained on a big chunk of the internet. The model has the best understanding of humans yet created. LLM models are the most promising technology created for simulating human behavior. However, the LLMs come with new challenges. With their limited context and occurrence of hallucinations, it is sometimes challenging for them to complete given goals. Furthermore, the models are moderately slow. The ideal approach is to use models for tasks that are otherwise extremely challenging. The thesis will provide a detailed view of how to design and implement such a simulator. Additionally, it will also explore the history and mechanisms of LLM technology.

The motivation for this research is driven by the rapid advancement in web technologies and the complexity of human-web interactions. In today's digital-centric world, understanding and replicating human behavior in web environments is increasingly important for enhancing user experience and developing efficient web-based services. This study addresses the gap in technologies and methodologies for accurately simulating human browsing behavior. By leveraging Large Language Models (LLMs) as the decision-making core, supplemented by other components, this research aims to develop a simulator that closely mirrors human interactions with web pages.

This study's significance lies in its potential to advance the field of human-computer interaction, offering insights and practical solutions for web design, usability, web testing, web bot detection, and AI development. While focused specifically on web browsing behavior, the implications of this research extend to broader applications in digital behavior analysis and AI ethics.

The proposed tool aims to create a simulator capable of web navigation and retrieval based on a textual specification of a goal on a previously unvisited website. This work shows the usability of LLms in human simulation tasks on the web. The work can help to create an advanced tool for web testing, or it can be used in different use cases for human simulation on the web or automatized web navigation and data retrieval. The results of the experiments 6 show the high capability of the simulator with an accuracy, averaging 82.1% success rate with the best model and showing similarities with human web navigation.

Chapter 2 discusses the related work to this thesis, analyzing the previous approaches and comparing them with the proposed approach by this thesis. Chapter 3 presents the technical and historical background of LLMs and their overview. Furthermore, Chapter 3.3 describes both technologies that will be used for the implementation of the simulator and technology used for web control or the integration of LLMs into applications. Chapter 4 explains the components of a simulator proposed in this thesis. The chapter describes the scope of the work and explains the whole architecture of the simulator. The following Chapter 5 describes each component from the implementation perspective. Chapter 6 shows ways in which the tool was tested and presents the achieved results. Lastly, Chapter 7 summarizes the findings of this work and future development.

Chapter 2

Related work

This chapter will briefly discuss existing approaches for simulating human behavior on the web and human simulation techniques in general. The following text should provide a better context of the field and a better starting point for the thesis. Because combining human simulation and web browsing is not a common thesis, few papers discuss both topics. However, The works still have relevant information about web agents and human simulation.

2.1 Web prediction and web agents

Web bot development has always been an ongoing 'arms race' between bad bots crawling the web, good bots making Google search work as it works, and security teams trying to detect the bad bots. The team of Y. Yang et al. [33] proposes a new approach of impersonator bots browsing previously unvisited websites. In their paper, they explain their approach of using a prediction model based on interest-driven theory. Their proposed decision model HBB-IDT introduces capabilities for web page analysis, measuring 'user' interest and page selection. The model can analyze previously unvisited websites based on metrics and measures of interest. Next, the successor page is selected by comparing the metrics like the user's current interest in the theme, theme similarity between the current page and successor page, visibility and closeness, and whether the pages are similar based on the content and theme.

The prediction of users' web-browsing behavior application of the Markov model is a paper by Momoun A. and Issa Khalil [5]. Their analysis of the Markov and all-Kth Markov models gives a unique viewpoint on web-browsing techniques. Researchers propose an improved version of the Markov model that is originally memory intensive. Their optimized version helps better predict the following web pages in web prediction based on previously visited pages.

The question of how to create and develop behavior-based detection is the topic of a paper by Jing Jin et al. [14] called Evasive Bots Masquerading as Human Beings on the Web gives better insight into how web bots function to give other researchers information on how to fight bad bots working on similar principles. Specifically, the authors discuss and characterize the approach for evading bot detection and, at the same time, propose techniques to detect bots implemented on the new approach.

The paper written by Boleslaw K. Szymanski et al. [27], A Method for Indexing Web Pages Using Web Bots, isn't connected to human simulation on the web but provides information about bots on the web and how they are, in this case, used for indexing web pages, which is the crucial role of the majority of 'good' bot crawlers used by Google.

2.2 Human simulation techniques

The work of Gati Aher et al.[2] proposes a new technique to test Language model capabilities to simulate human behavior in the introduced so-called TE (Turing Experiment). Using Large Language Models to Simulate Multiple Humans and Replicate Human Subject Studies

In recent years, the rise of machine learning and deep learning models has given birth to many highly intelligent social bots on social networks. These bots show how human behavior can be simulated by these models as most people on the internet no longer recognize whether a human person or an artificial agent produces the tweets they read. The paper of Terrence Adams, AI-Powered Social Bots [1], demonstrates the power of these AI-powered social bots' capability to simulate human behavior.

Tao Zhou et al., [37] in their paper „Towards the Understanding of Human Dynamics," discuss and challenge the way of modeling characteristics of human behavior and activity patterns by statistical characteristics. They argue that the assumption that human dynamics can be described by the Poisson model. The paper shows empirical evidence that human dynamics follow non-Poisson statistics. The papers provide insight into understanding human dynamics, which are important for creating accurate models for human simulation. The paper indicates that statistical models for human simulation might fail for certain contexts in environments where the context is dynamic.

In recent research, advancements in software engineering practices have emerged by applying deep learning and large language models (LLMs). One notable contribution in this field is the paper 'CHATDEV: Leveraging Large Language Models for Chat-Powered Software Development' by Chen Qian et al. [22], which introduces an innovative paradigm for software development that employs LLMs throughout the entire development process. CHATDEV utilizes a virtual chat-powered approach that mirrors the established waterfall model, breaking the development process into designing, coding, testing, and documenting stages. Each stage engages a team of agents to facilitate collaborative dialogue, enabling efficient resolution of specific subtasks. The results indicate remarkable efficacy in software generation, with the entire development process completed in under seven minutes at less than one dollar. This research opens up possibilities for integrating LLMs into software development, which may be relevant to developing human simulators for web browsing.

Chapter 3

Theoretical Background

This chapter will contain information supporting knowledge about LLMs and the technology used for web testing for web interaction. Even though the thesis will focus on using LLMs for human modelling, it is also important to mention similar algorithms and approaches the LLMs are built on. This includes NLP algorithms and Transformers.

3.1 Transformers

Transformers were the breaking invention that revolutionized the architecture of Deep Learning models. Transformers were first introduced in 2017 in *Attention is all you need* paper [30] by Ashish Vaswani et al. Apart from RNN models, Transformers can create connections between words using *Attention* mechanism introduced in the original Paper. Researchers developed a new memory system using Self-attention that improved how to connect words in a particular sequence. Hence, transformers are more effective with less training and yield better results than older architectures. The core aspects of the Transformer architecture are the backbone of current advanced model architectures and Deep Learning models in general.

Furthermore, transformers helped to unify the approach to Deep learning across multiple fields as the architecture of transformers was integrated into systems implementing computer vision, natural language processing, etc.

The first transformer architecture introduced by Google researchers [30] was made from the encoder and decoder components. Thus, it was mainly aimed at predicting a sequence of characters based on the input from another sequence. This approach can, for example, work for language translation between two different languages. Future implementations of transformer architectures may involve systems with solely an encoder or only a decoder. This modification shifts their use case towards context comprehension within sequences for encoder-only or the generation of tokens on preceding sequences in decoder-only models.

The standard transformer architecture can be seen at 3.1, which is the original diagram from the original paper [30]. To understand why the underlying mechanics of the architecture were the new state of the art for many deep learning tasks, the following subsection will explore the essential parts of transformer architecture in detail.

3.1.1 How Transformers work?

This subsection will briefly go through the major processes in a transformer flow. The input sequence of words is first split into predefined sequences of characters called tokens. Tokens

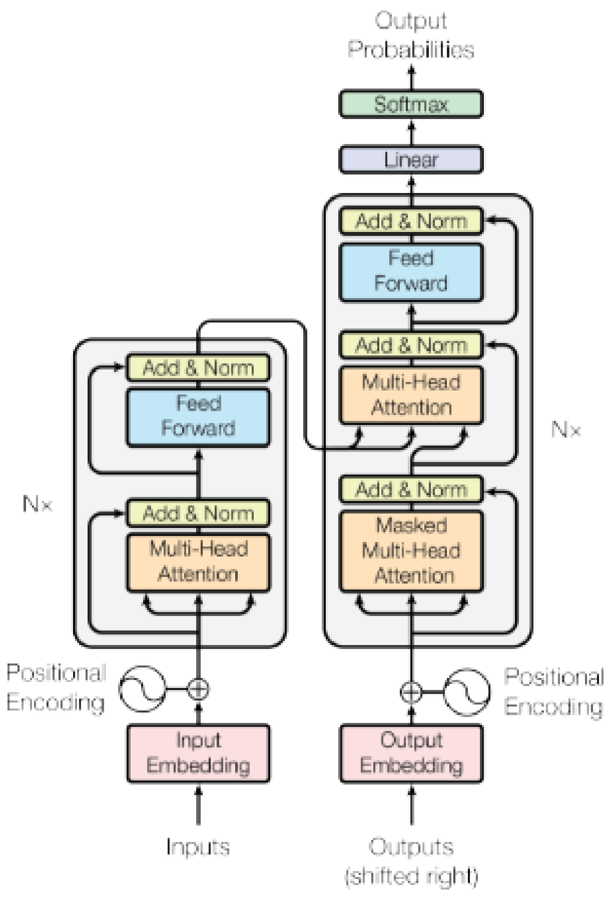

Figure 3.1: Architecture of the simulator

can be whole words or only subwords of the original words. Tokenization of sequence is usually done by a tokenizer that can be implemented in specific, more or less complicated ways.

After the tokenization process, the tokens are converted to multi-dimensional vectors using embedding tables or different methods to calculate embeddings for tokens. Embedding dimensions represent different features of the tokens, which are given based on trained parameters. Embeddings are a powerful way how to represent words and are used for different NLP tasks outside of transformers and LLMs, for example, for storing text in vector databases.

Transformer models can also utilize so-called positional encoding for tokens, which also takes their order into account when calculating the embeddings. This can help identify the right context for specific words because the classical embeddings do not consider that. Positional encodings are generated through the encoding function or can also be done using learned positional encodings.

Furthermore, transformer models and other NLP algorithms and structures might use the softmax function. Softmax function is crucial for obtaining results from calculating weights between different embeddings. For example, it calculates probabilities that match the standard distribution for the output of the decoder of transformer models that are used for determining the probability for the next word of a sequence. The softmax function is not only used for the final result of the transformer process but is also used to normalize the functions when they are being inputted to the next component, as seen in the diagram as `Add & Norm` in Figure 3.1.

The main component of the whole transformer architecture is the Attention mechanism. Transformers introduce three different types of attention. Standard attention can be calculated using the following formula from the original paper.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Attention allows the vectors in the embeddings to communicate with each other and update their values to better express the context of the whole sequence. The mechanism is a sum of vectors representing Query, Keys, and their respective Values. The scaled Dot-Product Attention is a product of Values with a result of SoftMax normalization of the dot product of Query and Key. The query is essentially the values that the specific vector is searching for in the other parts of the sequence. The key represents the current value of the vector, and the Value is a vector that represents the weight is the current token

Attention can be calculated upon each token simultaneously, so this architecture can be parallelized. The Transformer architecture in the paper used multi-headed attention, attention computed on multiple layers. The layers are concatenated into a single linear output. Attention can help identify connections between tokens in a sequence, so models based on transformer architecture can be much more powerful in tasks such as translation or text generation. Models used for translation usually use Encoder-Decoder attention to track connections between the input and the output to maintain the semantics of the translated sequence.

The original paper's authors also introduced masked attention, which is used by the decoder. It works the same way as standard attention, but it masks the values of tokens that are yet to be predicted, so the model cannot know the exact value of the predicted token.

Transformer architecture revolutionized the field of Deep Learning. Transformers outperformed traditional models like RNNs in translation and text generation tasks thanks to the attention mechanism. The second greatest feature of Transformers was the scalability. The result makes this architecture state of the art to this day, pushing the ability of deep learning models beyond boundaries. [30]

3.1.2 History of Transformer models

The number of new deep-learning models based on Transformer architecture has skyrocketed since 2017 [3]. This subsection will discuss examples of models that have been significant in the recent history of Transformer models. The variety of Transformer models is currently greater than ever. It is problematic to compare certain models as they have different purposes and functions. The section will mention and briefly explain each significant model to provide perspective into the history and recent development of Transformer models. The timeline of transformer models can be seen in figure 3.2 [3].

GPT

The GPT model was the first model that came out after introducing transformers. It was trained on a corpus of 7000 unpublished books [36]. This dataset helped GPT perform tasks such as answering questions and generating text. The model was trained by an

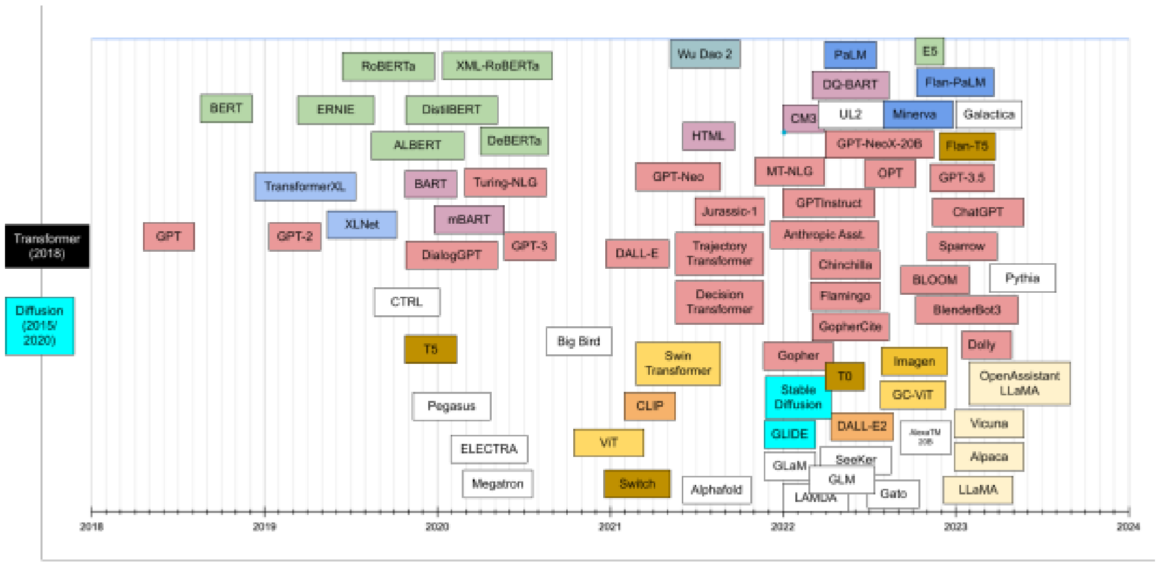

Figure 3.2: Timeline of the Transformer models.

unsupervised learning method and by supervised learning for fine-tuning. This model is a Decoder only, which means it was pre-trained for the next token prediction. This model produced reasonable responses to some extent, but it often hallucinated. Hallucination in terms of the LLM model is when the model outputs nonsensical or not factually correct output. It happens when LLM now has constraints that limit what kind of output it should have or when the prompt or input is ambiguous. However, there might be different reasons, such as lack of training or context misinterpretation.

BERT

BERT stands for Bidirectional Encoder Representation from Transformers. The model was pre-trained with the masked language model (MLM), which enables the attention mechanism to follow both directions. For example, the GPT model was pre-trained in a left-to-right direction. It is an encoder-only model that can perform some specific task. BERT researchers made the approach of pre-training and fine-tuning the model on NLP tasks famous. This approach was then used for different models, leading to the domination of Transformer models in the NLP leaderboards. BERT itself cannot output anything, but thanks to fine-tuning, which consists of adding one layer and retraining the model. [8]

GPT-2

The GPT-2 model, with 48 decoders and 1.5 billion parameters, was bigger than its predecessors [23]. The model is decoder-only, and it does token prediction as GPT-1. It was trained on a corpus of web pages consisting of 500B tokens [2]. It was developed by OpenAI. The main improvement compared to the GPT models was the context window size. It increased from 512 to 1024. The model was able to perform well in a variety of different tasks.

RoBERTa

RoBERTa model is an optimized BERT model with better results. Researchers found that the BERT model was undertrained. Their proposed model achieved, at that time, the best results on GLUE, RACE, and SQuAD benchmarks [18]. These benchmarks test LLMs in several NLP tasks. The main reason for this improvement was the different approach to learning the model. The training relies on randomly masking and predicting tokens. Instead of masking the tokens in ten different ways, the new model uses dynamic masking, greatly improving overall performance. Dynamic masking generates new masks for every sequence injected into the model.

GPT-3

GPT-3 model was a great improvement for the LLM world. The main difference between GPT-2 and GPT-3 is the number of parameters. GPT-3, with over 175 billion parameters, beat all its predecessors [6]. It was trained on five different corpora, including books, crawled websites, Wikipedia, web text, etc. It has better accuracy for the next word prediction. The model's strengths include sentiment analysis, semantic search, code generation, and overall user experience.

LLaMA and open source models

LLama base model was developed and released by Meta [28]. LLaMA was the answer for OpenAI's GPT models. Meta presented it as a commitment to open science: „As part of Meta's commitment to open science, today we are publicly releasing LLaMA (Large Language Model Meta AI), a state-of-the-art foundational large language model designed to help researchers advance their work in this subfield of AI" [12]. The model was released in February 2023. However, unlike other LLM models, LLaMA was released as the first model anyone could sign for and download locally. The model was accessible through a Google form [10]; however, the model was leaked on Torrent in March 2023 [4]. The LLaMA model and their fine-tuned versions are now available on Hugging Face [19]. The LLaMA 13B parameter version outperforms GPT-3 (175B) on most of the benchmarks [28]. LLaMA started a new page of LLM models with its OpenSource nature. Researchers and interested developers developed and optimized the models to increase performance and add new features by fine-tuning them. These models compete with the proprietary models from Google and OpenAI. The goal of meta-researchers was to provide the models to the public to explore the safety of LLM technology further. In February of 2023, there were a lot of questions about the state of security of LLMs and the potential impacts. LLaMA was not only a competitor to the other models, but it was also a response to the development of the models.

GPT-3.5

GPT-3.5 was the model that popularized the LLM world. With the release of ChatGPT, everybody with access to the internet could potentially use and experiment with the largest LLM model in the world. [35] The main improvement from the older models was the use of Reinforcement learning from human feedback (RLHF) [24]. The model's training consisted of pretraining, creating a reward model to increase success over training time, and fine-tuning the process with RLHF.

Mistral 7B

Mistral 7B is a 7 billion parameter model trained for performance for multiple kinds of tasks, including commonsense reasoning, word knowledge, reading comprehension, math, code, and other benchmarks [13]. The results made the model compete with LLamMA models on various benchmarks. Similarly to the LLaMA model, Mistral 7B is an open-source model that allows the community to quantize the model to use less data in cost of performance to fit the model into smaller and more accessible graphic cards, which allows to fine-tune the model with lower costs and also to run it locally. The team behind Mistral also released a fine-tuned version capable of generating text for chat called Mistral 7B instruct. The mistral model utilizes sliding window attention, which allows the handling of longer sequences at lower cost and at higher speed, which can allow potential usage of mistral for real-time applications. The example of mistral shows how the evolution of models makes the models less expensive to run and more accessible for fine-tuning and integrating into different kinds of systems and applications.

3.2 Large Language Models overview

Large Language models have proven their capabilities for NLP tasks and beyond. This section will provide a broader view of this type of model. The rapid growth of development and advancements of LLM brings new capabilities and opportunities for more sophisticated applications of models in software applications for different systems that require complex problem-solving and human-like interaction [20]. However, it is also important to accurately highlight this new technology's limitations and security concerns. Consequently, a comprehensive understanding of their strengths and weaknesses is essential for creating secure and robust applications.

Despite the success and strength of LLM models for dealing with certain problems. LLMs have certain limitations. However, it is important to note that with the rapid development of this technology, many of these issues and limitations might be solved or mitigated.

The main limitation is the computational [20]. Because LLM models require a lot of GPUs to be trained, the training process is expensive. For example, training of the GPT.3-5 cost based on OpenAI's statement above 100 million dollars [2]. This cost can be paid only by the biggest companies in the world, which makes the LLM technology valuable. This was also the reason for the democratization of the LLaMA model.

Due to the training of LLM on human-generated data, the LLM model can inherit societal biases, potentially leading to ethical concerns [20]. This significant limitation can be partially mitigated through the fine-tuning process. This process can improve the overall safety of the responses. However, it presumes the absence of bias among the individuals involved in the fine-tuning process. Therefore, the mitigation strategy, while beneficial, cannot be foolproof and requires careful implementation and consideration to minimize the potential bias in LLM models.

In specific instances, LLMs demonstrate a phenomenon referred to as „hallucinations" [20]. This term defines scenarios where the responses generated by the LLms are semantically incorrect with the context of the query, exhibit self-contradictions, or deviate from established world knowledge [20]. Such behaviour can be attributed to several factors, including inadequate fine-tuning in certain domains, the presence of ambiguous prompts, or prompts including information outside of the model's training corpus. This limitation is the critical area of focus in the development and refinement of LLMs and is usually improved by human feedback.

The rapid rise and usage of LLM models create questions about ethical considerations concerning these models. The generated content can contain harmful content or content that can be used negatively [20]. This calls for ethical frameworks and policies to govern the outputs of the models.

Information acquired by the LLM during training is limited and can become obsolete after some time. It is expensive to retrain the model frequently. [20] Therefore, models without the ability to search on the web can have limitations of limited knowledge they can provide in their prompts.

Another limit LLMs have is the limited context window they can provide. Because of that, LLMs tend to forget certain facts after several prompts because they go beyond their context. There are certain breakthroughs like StreamingLLM [32]. However, most of the current models have a limited context window. Even with the endless context window, there is still a problem with keeping track of the most important facts and information. Another approach for up-to-date knowledge is to use retrieval augmentation generation (RAG). RAG is a method that combines model generation with information retrieval, which enables the model to access external up-to-date knowledge [21].

Using LLMs comes with a certain risk of generating harmful, misleading or inappropriate content by accident or by giving specific prompts. This behaviour can be replicated using so-called „jailbreak“ prompts to trick the LLM into not following safety guidelines [29]. Furthermore, LLM can be tricked by „prompt injection“ [15]. Newer LLM models can scrape websites to gather factual information, and this process can accidentally trigger an action given in the scraped website. Therefore, LLM might accidentally leak data or save private user data based on some action given on the attacker's website.

3.3 Web Interaction Technologies and LLM Integration

Nowadays, there are a lot of technologies surrounding the world of large language models. Since the boom of ChatGPT, many developers and companies have tried to use LLms to their advantage. Multiple frameworks enable the integration of LLMs into applications. Technologies such as Semantic Kernel or Lang Chain enable developers to harvest the power of LLMs and use them to analyze data or do difficult tasks in their apps. Moreover, the technology used for web testing and creating apps that interact with the web is crucial for the simulator. Both of the technologies used for LLM integration and web interaction will be discussed in this section.

3.3.1 Web Automation and Interaction

In web development, web automation tools are essential for web testing. These tools allow control of the website using WebDriver [31], which created a remote interface for controlling a virtual web browser with instructions. Selenium and Puppeteer, two major web interaction libraries, will be discussed. To create a simulator integrating LLM models, it is crucial to use a library that can aid with such integration, Lang chain [16] and Semantic Kernel [26]. To create an effective simulator capable of mimicking human web browsing, it is essential to leverage capabilities from both web driver libraries that can control the web and libraries responsible for integrating LLMs into software applications.

DOM

Document Object Model (DOM) [17] is an API for HTML and XML documents. DOM can represent these documents and portray their parts as individual objects. DOM allows interaction and modifications to the documents and access to individual objects. The objects in the DOM have a logical structure similar to a tree-like structure. However, the implementation of the DOM does not specify a tree-like structure.

DOM is essential for WebDrivers [31] as it allows interaction with the web through a computer program. This is essential for creating a web-based simulator.

Selenium

Selenium is an ecosystem that allows interaction with a virtual web using browser drivers with support for several different browsers, such as GeckoDriver [11] for Mozilla Firefox, Google Chrome driver [9] for Chrome browser, etc. The WebDriver is a component that can drive the browser natively or on a remote server. Selenium can be used for viewing the contents of a webpage, initiating actions such as clicks on buttons and references, and inserting text into forms, as well as advanced features like managing cookies together with navigations as a human agent with a browser could. Selenium also supports multiple programming languages like Python, JavaScript, Java, and Csharp. [25]

Selenium allows a fairly straightforward approach to creating web testing scripts. However, it can also be used as a simulator to initiate different actions as a human agent would. This includes dynamic elements such as pop-ups and advanced web page structures. The main advantages of the Selenium environment are its open-source nature, multi-language support, and the variety of interactions it can perform.

Puppeteer

Puppeteer citepuppeteer24 is a library providing an API for Chrome over the Dev Tools protocol [7]. Puppeteer was made to test the performance and functionality of web pages or for web automation. It can perform interactions users could do manually. The library can also work with single-page and Server-side rendering web applications. Puppeteer is officially developed for JavaScript language. However, some unofficial libraries support Puppeteer usage within different languages, such as Python.

Compared to Selenium, Puppeteer brings a functional alternative that can outperform selenium in certain cases as it focuses mainly on chromium drivers. However, the only official language support can be a disadvantage when creating a simulator that has to be robust, especially when JavaScript language is not suitable for such use cases. Puppeteer could be used for specific scenarios where Selenium cannot perform a given task.

3.3.2 Large Language Model Integration

LLMs can dramatically influence the way applications are structured and developed. With the ability to integrate LLMs into software applications, developers can use LLMs to perform complicated tasks that are hard for algorithmic approaches and can solve the ambiguity of certain use cases. Semantic Kernel [26] and Langchain [16] are the two main ecosystems used for this process. Both have different integration approaches and allow different solutions. Both Langchain and Semantic Kernel will be discussed.

Semantic Kernel

Semantic Kernel [26] is an open-source SDK designed to integrate AI with software applications. It can integrate models from platforms like OpenAI, Azure OpenAI or Hugging Face. This SDK can be used in Csharp, Python and Java applications. Semantic Kernel allows for creating AI agents that can take on various tasks. The tasks include answering questions or automating different processes. The main feature of Semantic Kernel is the orchestration layer, which allows for seamless connection of AI models and plugins that add extensions for existing LLM models, such as the math plugin that allows the LLM model to calculate expressions to get accurate results.

LangChain

Langchain [16] is a Python library designed for building applications that leverage LLM models like GPT-3. It provides a framework for combining different components of LLM models in a modular and flexible way. Langchain uses LangChain Expression Language (LCEL), which is short for creating pipelines called „chains“ with certain logic.

One of the key features of LangChain is the ability to create agent applications or applications where an LLM model is used for reasoning. LangChain also implements the ReAct [34] method to help the model reason and solve problems using the available tools.

LangChain can be particularly useful in simulating human behaviour in browsers. It can help create systems that mimic human browsing patterns by making decisions based on current webpage information given to the LLM model. This can include clicking on buttons, filling in search and input forms, understanding the text, and taking actions based on the current context.

Chapter 4

Design and Architecture

This chapter will investigate the components and scope crucial for a functional and robust simulator architecture that replicates human behavior with the power of LLM. The objective is to create a simulator that enables replicating human behavior in web browsing.

4.1 Objectives and Components

There are several objectives for the simulator. The most important is the accuracy of navigation through a website compared with human behavior. The main objective of the simulator is to perform actions based on a given goal. The goal is a text representing what will be accomplished in the simulation. Because the simulator will not necessarily have prior information about the website and its structure, each page throughout the simulation has to be processed individually. The main component of the simulator is the decision-making model that decides the next interaction on the website. The input to the simulator is a specification of a goal and a URL to an existing website, together with additional parameters influencing the simulation behavior that is later described in the section 5.1. The simulator outputs are the logs representing the navigation path throughout the simulation, the actions done on the website, and information about requests made between the browser and the website. Logging and what data is stored is specified in a later section 5.6. The simulation ends when the decision goal of the simulation has been achieved.

4.2 Architecture

This section introduces the key components of the simulator and how they work together. Furthermore, it explains the high-level flow of the application during a simulation. The simulator is composed of three main components:

- Virtual Web Browser,
- Web Controller,
- Decision Making Model,

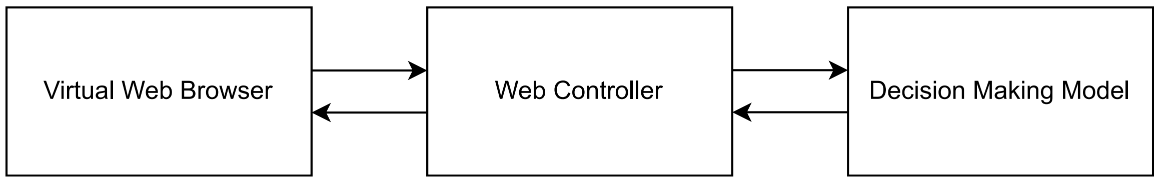

Figure 4.1: Architecture of the simulator

The simulation flow is visible in Figure 4.1. Firstly, the Web Controller gets the key information, the URL of the website, and the goal that should be accomplished. It sends a command to the Virtual Web Browser to load the website of a given URL and then sends the raw data of the website back to the Web Controller. Then, the Web Controller parses the raw data from the website into a structured form containing links and contents and passes it to the Decision Making Model. The Decision Making Model returns the most relevant interaction to the Web Controller, passing it to the Virtual Web Browser that updates the page. This loop repeats until the goal is achieved, then the simulation terminates.

4.2.1 Virtual Web Browser

The Virtual Web Browser has two main functionalities. Firstly, to download and display the content of a given web page, which is then passed in a raw format to the Web Controller. Secondly, to execute commands representing users' behaviors on the website, such as:

- input to a search bar,
- click on a button,
- click on a link.

4.2.2 Web Controller

The Web Controller is the middle layer between the Decision Making Model and Virtual Web Browser. It has four main functionalities:

- gets web data from the Virtual Web Browser,
- parses the data and sends it to the Decision Making Model,
- receives a selected interaction from the Decision Making model,
- sends the interaction to the Virtual Web Browser in the correct format.

The information obtained from the Virtual Web Browser is in HTML format and has to be parsed. The Web Controller contains a web parser that allows the raw HTML to be parsed in a structured format. The interactions received from a Decision Making Model have to be parsed again from a general format to a format that the Virtual Web Browser understands.

4.2.3 Decision Making Model

The Decision Making Model is the core part of the simulator. It receives information from the current page in a structured format, selects the most optimal interaction, and propagates it to the Virtual Web Browser with the help of the Web Controller. To make better predictions, the Decision Making Model uses internal memory of past actions.

Interactions

The interactions are modeling human behavior that is simulated in the simulator. The decisions a human agent would make while browsing are abstracted into a fixed list of four types of actions. The actions were selected to enable the most common interactions with a website. The types of actions are:

1. click on a link or button,
2. use a search bar,
3. retrieve information from the current page,
4. exit the simulation.

Action that handles clicks on a website was created to enable navigation on the website during the simulation. The action to use a search bar is there because most current websites have a search bar, which enables fast navigation and searching for detailed information.

Even though the last two actions are not part of the physical actions a human would make on a website, they are crucial for the simulator. Information retrieval is used mainly to allow the Decision Model more context because the standard content of the webpage obtained from the Web Controller might be limited. The exit action is used for the Decision Making Model to exit the simulation upon the goal being accomplished.

Output Format

The Decision Making Model is powered by the LLMs. To make the LLMs understand the context and create better outputs, the following output has a given form of three attributes:

1. Thought,
2. Action,
3. Action Context.

Each of these attributes has special functionality. The Thought component represents the reason for a specified action. It was created to help the LLM structure the output with an initial piece of information that adds context to the decision and reason for the decision.

Secondly, the Action holds the name of the action selected from the list of actions. It only contains information about the action type.

Lastly, The Action Context serves as additional information used by the Web Controller. For example, to click on a given link, the Web Controller must receive the link's name; this name must be contained in the Action Context.

The given format of the output that the Decision Making Model is used to reduce hallucinations made by the LLMs and create an abstraction of human behavior.

4.3 Scope

The simulator proposed in this thesis is designed to simulate human behavior on a website. The extent to which the simulator can represent human behavior is given by the architecture and the list of actions that are defined in subsection 4.2.3. The scope of the simulation defined the goals and websites that could be used for the simulation. The website must not contain a CAPTCHA challenge or other human verification methods. Secondly, the website must be functional without malicious content that could negatively influence the simulation. The goal specification must be clear and accomplishable on a given website. The architecture does not model real-time decision-making. Therefore, the timing of the action is not influenced by the real frequency of human decisions but rather by configurable timeouts discussed in section 5.1. The fixed list of actions defined in subsection 4.2.3 defines what goals can be accomplished by the simulator.

In summary, both the design and implementation are tailored to specific use cases, including web navigation and the extraction of facts and information from websites, where the extraction of facts is mainly on granting the simulator information needed for determining the goal completion.

Chapter 5

Implementation

This chapter explores and explains the simulator implementation defined by the design architecture explained in chapter 4. The simulator is created as a command-line interface application. It is implemented in Python with the two main libraries, Langchain and Selenium.

Firstly, the Selenium library is used for a Selenium driver that creates a headless browser that implements the Virtual Browser Component 4.2.1, and it can handle the interactions while receiving updated information from the browser.

Secondly, The Langchain library integrates LLMs and the program itself. Langchain handles the proper format of actions sent to Selenium and provides means for creating an interface that handles the core decision-making throughout the simulation. Furthermore, the implementation contains supporting components that handle configuration, the application's core, logging mechanisms, and parsers of the data obtained from the webpage or the LLM.

The implementation is logically separated into parts to improve maintainability and add features in future work. Each component of the simulator will be described in each section with information about it to understand the communication of other parts of the architecture.

5.1 Configuration

Configuration in any application enables a simple and fast way how to change the behavior of the applications for different use cases that increase flexibility and also maintainability. In this case, the configuration is implemented by a simple JSON parser component that propagates the values to the application. The JSON configuration is defined by the following JSON schema [1]:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Generated schema for Root",
  "type": "object",
  "properties": {
    "website": {
      "type": "string"},
```

[1]JSON schema converted available at: https://transform.tools/json-to-json-schema

```
    "goal": {
      "type": "string"},
    "request_rate": {
      "type": "number"},
    "persona": {
      "type": "string"},
    "llm_provider": {
      "type": "string"},
    "model_name": {
      "type": "string"},
    "temperature": {
      "type": "number"},
    "verbose": {
      "type": "boolean"},
    "initial_timout": {
      "type": "number"},
    "timeout_per_action": {
      "type": "number"},
    "web_data": { "type": "object",
      "properties": {
        "type": {"type": "string"},
        "name": {"type": "string"}},
      "required": ["type","name"]},
    "cookie_config": {
      "type": "object",
      "properties": {
        "buttons": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "type": {
                "type": "string"},
              "name": {
                "type": "string"}},
    "required": ["type","name"]}}},
    "required": ["buttons"]}},
  "required": ["website","goal","request_rate"
  ,"persona","llm_provider","model_name",
  "temperature","initial_timout","timeout_per_action",
  ]
}
```

An example of a functional configuration is the following JSON file:

```
{
    "website": "https://www.python.org/",
    "goal": "Find upcoming events happening in USA",
    "request_rate": 30,
    "persona": "Generic",
    "llm_provider": "openai",
    "model_name": "gpt-4-turbo",
    "temperature": 0.45,
    "verbose": true,
    "initial_timout": 3,
    "timeout_per_action": 0,
    "web_data": {
        "type": "class",
        "name": "search-text"
    },
    "cookie_config": {
        "buttons": [
            {
                "type": "id",
                "name": "onetrust-reject-all-handler"
            }
        ]
    }
}
```

Firstly, the configuration allows the user to specify a website using a URL and goal in plaintext in human-like language, specifying where the simulation will occur. The configuration allows the selection of an LLM provider, as explained in the section 5.4. The model allows selecting a particular LLM model from models accessible throughout the selected LLM provider. The persona option enables the selection of a predefined persona that is defined in the LLM provider 5.4. The user has the option to set timeouts used for a time the application takes between actions or before the simulation used for proper loading of the website before it load its contents. Next, the web data component allows for specifying the information used to access the search bar on websites where the search bar id is not found using the standard web parser. Similarly, the cookie configuration allows the user to resolve the cookie popup that appears on certain websites.

5.2 Web parsing

In each iteration of the main application loop, the LLM needs information from the website to understand the next optimal action for achieving the predetermined goal. The application implements two ways of gathering and preparing data for the LLM to access.

The web is parsed to collect all available `<a> href` attributes, buttons, and input fields. The raw data are obtained from the selenium method `get_source` and are parsed and enriched to more information by selenium methods that allow searching for IDs or text for particular elements. In each iteration, the web parser looks for search bar input by looking for an ID with a substring that contains the keyword *search*,*query*, or other familiar keywords, which works for most websites. For websites where the ID for the search bar is different, the user can configure the ID for the element so the decision chain can use the search bar action as discussed in section 5.1.

The second type of web data is a webpage summary generated for every decision. This summary helps the decision chain understand what type of content is on a given URL. In certain cases, it can help the LLM understand if the desired information it needs to retrieve is present in the current state of the simulation. Lastly, it helps the decision chat to reduce hallucinations where the llm does not know if the webpage's content has changed.

5.3 LLM Chains

The application implements two Langchain chains: a decision chain and actions chains made to parse the input for a selected action in a given format. These two chains are in the core part of the program and handle the decision of actions by LLM and subsequent interpretation of the output together with action invocation by the action chains.

5.3.1 Decision chain

The core of the application is the decision chain. It is implemented by a chat prompt and output parser in JSON format. The prompt template introduces the LLM to the simulation of human behavior and instructs it to decide what action should be chosen from a specific list of actions on a website URL address. The LLM is informed about the goal that should be accomplished. With each interaction of the application loop, the chat gets information about the current URL, goal, list of hrefs, website summary, and past actions. The chat then outputs a decision in three components, as shown below.

```
{
"Thought" : <thought of the llm>,
"Action" : <action chosen from the list of actions>,
"Action context" : <the detailed information of the action>,
}
```

The thought is used to help the LLM formulate the output, starting with a thought of the action that will be chosen. The action then includes the name of the selected action from the list of actions. The action context includes the data for the action that has been selected.

The available list of actions:

- Use a search on a current URL
- Retrieve information from current URL
- Click on a button or a link
- Exit simulation

The LLM model can use the list of actions to navigate the website and retrieve information to accomplish a current goal. Each action is then handled by its specific action chain.

The output from the decision chain includes a JSON parser implemented inside the Langchain library. However, as the JSON parser can fail in certain scenarios, it has to be checked with an additional parser that checks the structure based on a given schema.If the output schema is not met or there is an error with the standard parser. The output from the main chain is given to a chain called `fix chain` that is used for fixing the incorrect structure of the output. When all parsers fail, or there is an error with the invocation of a current action, the LLM is informed about the result in the next iteration throughout the memory component and can react to the situation.

5.3.2 Action chains

When the decision is made by the decision chain, the LLM's output is given to one of the four chains responsible for the subsequent actions. Each chain is specialized for handling a given action. By dividing the task of a decision and action into smaller tasks, a chain reduces hallucinations by giving the LLM less 'work.'

The chain responsible for clicking on an element outputs the name of the button or link in a given format by an output parser to JSON. This simplifies the parsing of the decision chat because the content or the button's name might not be consistent in the provided output of the decision chain. The parsed output from the action chain is then directly used to call a Selenium driver method to interact with a given element.

For the search action, the chain output is a keyword inputted into a search bar on the browsed website. In certain situations, the decision chain outputs a sentence with the content of what it wants to search for, and the search chain selects a keyword or set of keywords that best defines the searched content. The output is then used again to handle the selenium driver's actions.

Retrieving data from the website does not need a chain to better structure the output used by selenium, but a different approach handles the process. The action extracts data from a current URL using a web loader method implemented in the long chain library that enables reading data from a website and parsing it only to textual content. LLM retrieves information relevant to the context of the given retrieved action. When the current URL contains a large amount of data, the retrieve action utilizes embeddings of the content that can be generated, and the LLM can query for a chunk relevant for the action context.

The exit action is not implemented using an LLM chain, so this section does not need to be discussed.

5.4 LLM provider

To create a robust and maintainable component for handling a connection to a LLM. The simulator implements a separate component of an LLM provider that is used in the chains. The LLM provider is Factory, which instantiates objects of specific LLM providers defined by a class. Each class of LLM providers must implement the connection to the model and return the chains used in the application. The methods are polymorphic and are called in the main application loop. For different LLM providers, the exact implementation of the chain might differ. The implemented providers in the simulator include OpenAI and Mistral LLM providers. This structure allows for a possible extension of the available providers for other LLM models of all kinds of forms or possibly different models that follow the IO of the whole application schema. The LLM provider also handles different personas configured in the configuration and defines what personas are available for selection. For the purpose of this work, the Generic persona type was used.

5.5 Selenium controller

The selenium driver is initialized at the beginning of the simulation. The application uses a Firefox browser as it has fewer prerequisites for installation. The website is loaded by a method `.get()` method provided by the driver object.

The interaction with the website consists mainly of finding elements the application wants to interact with by ID, link name, or other attributes. For clickable links or buttons, the selenium driver uses `.click()` method to click the given element; for inputting data into input fields, selenium uses `.send_keys()` with the content that should be inputted.

For handling search bar information, the input element is firstly clear using `.clear()` method, and after the keys are sent, the enter key is used to submit the query.

At the end of the simulation, the driver object is also used to retrieve the requests made between the browser and the website. It contains requests made by and to advertising websites. The requests are stored in a log file. The list of requests can be retrieved using `driver.requests` iterable.

5.6 Logger

The application utilizes logging, a standard Python library, to handle and manage logs throughout the runtime. The implementation uses two different logs. The first log provides information about the states of the application and warnings or errors that might occur during the runtime. This log type is displayed in the terminal and written to a log file `log.txt` where it is stored for a single simulation run. The second logger is used for logging information about the chosen actions and their context stored in a different file `log_actions.txt` and is used for analysis made by the benchmarks /refbench.

The logs of requests caught during the simulation and obtained by the Selenium driver are parsed in a simple method and saved to a file called
`simulation_requests.ndjson` with a structure.

```
{"method": <method>,
"url": <URL>,
"headers": <headers>,
"body": <body>,}
```

optimal for the input for a classification used later in the 6 section

The configuration can specify file names and the path of the logs, or optionally. It can turn off the file or standard output options.

5.7 Challenges

The main challenge that had to be tackled is the LLM context window is limited to a certain number of tokens; therefore, the data needs to have certain preprocessing, mainly the web page summary and the data retrieval action. Therefore, the llm is only a portion of the data because the summary or parsing omits a portion humans could perceive. The memory component then has to be limited and only contain a short portion of data and has a certain length.

Secondly, the application is limited to the actions it can perform. These actions are used in most human traffic where humans need to get information on the web or navigate a website.

Furthermore, web summary, parsing, and decision-making take several seconds to complete. In a context where the goal is to simulate human behavior, the delays prove advantageous, mirroring the natural pace of human decision-making.

The work was mainly focused on exploring the ability of the LLM to simulate human behavior in web navigating and data-retrieving tasks. The application could be overhauled for more complex web tasks in future work.

Chapter 6

Experiments and Testing

This chapter will discuss the three types of experimentation and tests used to test and grade the application's usability and similarity to human behavior or web traffic. The system testing consisted of three types of tests aimed at different aspects. The application was first tested using automated benchmarks to determine the simulation's success rate and grade its performance together by retrieving valuable metrics rating its performance on a given website and goal. The second experiment compares the simulator and humans using calculated metrics. Lastly, the final experiment compares differences and similarities in web navigation between the simulator and human data.

6.1 Website selection

Both the human data and data obtained by the simulator contain selected websites with specific and different goals for each website. The websites with their goal are:

- Python.org with the goal 'Find information about Python events in the Netherlands,
- Dictionary.cambridge.com with the goal 'Find the definition of the word Fungus',
- Wikipedia.org with the goal 'Find when Václav Havel completed his secondary education',
- u.gg/champions with the goal 'Find counter for champion Ahri',
- rottentomatoes.com with the goal 'Find the name of the director who made the film Monty Python and The Holy Grail.

The websites were chosen to represent unique website types with different use cases. Each goal can be achieved with a different combination of actions. To successfully complete the goal in a given simulation, the Model has to navigate to a webpage containing the searched information and retrieve the information either using the web summary or by performing the retrieve action. The success rate of different simulation runs was measured using the benchmarks.

6.2 Benchmarks

The benchmarks were implemented in a separate Jupyternotebook to test the simulator and record metrics used for evaluation. They come with the standard simulator configuration with additional parameters, like the number of repetitions of a simulation run. Additionally, benchmarks allow the creation of a list of different simulations with unique configuration settings. The results of the benchmarks are used to calculate metrics used for experiments.

6.2.1 Configuration

To understand how the benchmark functions. The following example of a benchmark configuration shows its functionality.

```
{
    "benchmark_name": "experiment_data",
    "simulations": [
        {
            "provider":"openai",
            "model":"gpt-4-turbo",
            "request_rate":30,
            "verbose": "true",
            "initial_timout": 3,
            "website_name_short" : "python-org",
            "website": "https://www.python.org/",
            "goal": "Find info about Python events in the Netherlands",
            "target_urls": [
                "https://www.python.org/events/python-events/",
                "https://www.python.org/events/"
            ],
            "final_action": "Retrieve",
            "runs": 3
        },
        ...
    ]}
```

The benchmark configuration is a list of configurations for the goal, website, and target URL and action. Target URL and action can help determine whether the simulation was successful based on the final URL and action before the action Exit.

6.2.2 Output

The benchmarks are recording several metrics to allow for metric calculation. The metrics are stored in a specific path as a JSON file or logs and raw data. An example of a simulation benchmark:

```
    {
    "exit": false,
    "goal_complete": false,
    "action_count": 7,
    "action_per_type": {
        "Search": 1,
        "Click": 6,
        "Exit": 0,
        "Retrieve": 0
    },
    "simulation_length": 78.07544684410095,
    "average_time_per_action": 11.153635263442993,
    "raw_data": [...]
```

The key `exit` and `goal_complete` is used to determine whether the simulation was successful. The key `action_count` is used for measuring the number of actions. Furthermore, `action_per_type` displays the distribution of actions based on their type. The keys also contain information about the length of the simulation and average time per action in fields `simulation_length` and `average_time_per_action`. The JSON file also contains a list of parsed `actions.txt` log.

The JSON files are stored in a specific path in a directory `experimend_data` with a specific path. The subdirectories contain the model's name specified in the benchmark configuration, e.g., `gpt-4-turbo`. The subdirectory of a model directory contains the name of the `website_name_short` field from the benchmark configuration. Finally, the website directory contains JSON files of each run of one specific benchmark. The JSON files containing the simulation data are created to be unique with a specific pattern of their name: `D_HH_MM_SS_runnumber_metrics.json` to avoid duplications. Example path to a specific JSON file can be as follows:

```
\experiment_data\gpt-4-turbo\python-org\05_23_40_32_0_metrics.json
```

The results from the directories are parsed and aggregated to produce metrics for each model and specific sites. The aggregated metrics include the following:

- average number of actions per simulation,
- average time for action in simulations,
- average length of simulations,
- and success rate of simulations.

The results are used in a specialized Jupyter notebook to create graph representations of the data to compare the results. The benchmark helps create an automated pipeline for recording and storing data from multiple simulation runs with different metrics and automatically creating graphs.

6.3 Human Data collection

The human data were collected using a Google form [1]. The form contains instructions and the same set of websites and predefined goals as used for the benchmarks. To effectively collect the user interaction the users used Selenium IDE[2] chrome extension and screen recording extension[3]

[1]form available at the link: https://forms.gle/DNrDNpKr7VzPV7h98 with a requirement of a Google account because it requires upload of files to Google Drive

[2]chrome extension: Selenium IDE available at https://chromewebstore.google.com/

[3]Screen Recorder, available at: https://chromewebstore.google.com/

The Selenium IDE extension was used to record user interactions with the website in a JSON-like format. The data from the video recording was used because the Selenium IDE extension does not store timestamps and current URLs of actions. The data was annotated using video recordings.

6.4 Experiments

The experiments compare performance between LLMs GPT 3.5 Turbo and GPT 4 Turbo based on success rate. Furthermore, the experiments show the differences and similarities based on calculated metrics between the simulator and the human data. The last experiment shows differences between web navigation on the five websites using a Sankey diagram.

6.4.1 Success rate

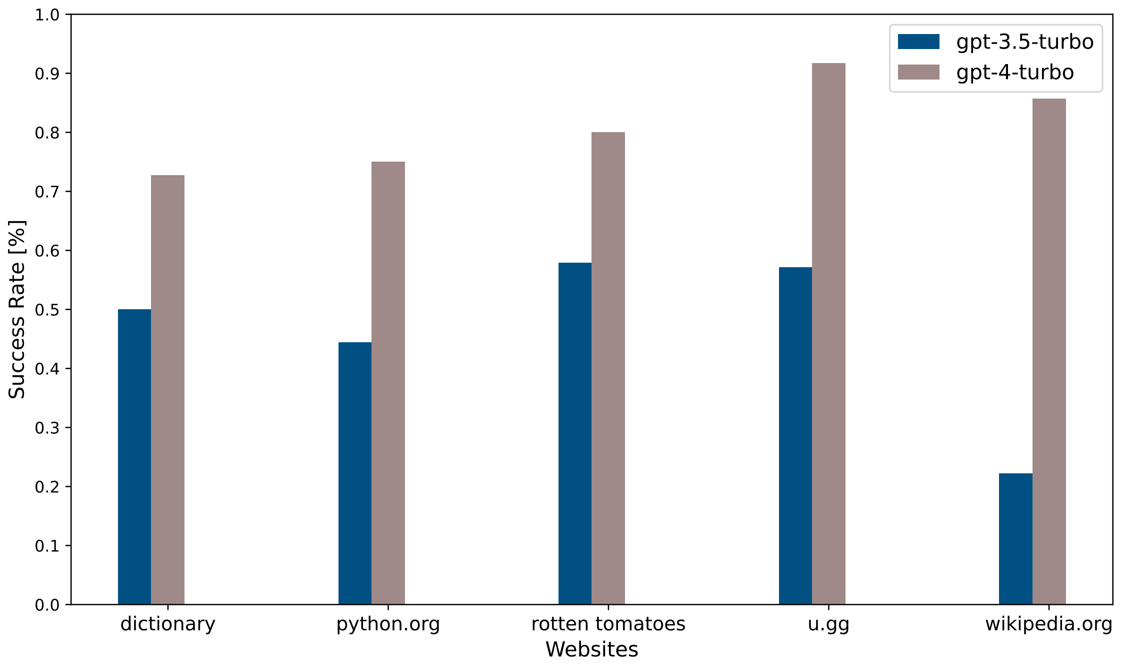

Figure 6.1: Success Rate by Website and Model

Figure 6.1 shows the success rate of models GPT 3.5 Turbo and GPT 4 Turbo on the five websites. The plot shows success rates around 50% for model GPT 3.5 Turbo on four websites and a lower success rate for website Wikipedia.org. On the other hand, GPT 4 Turbo shows an increase in the success rate across all of the websites, averaging on 82.1% success rate. For the model, GPT 3.5 Turbo, the simulation failed mainly due to the model's hallucination. Meanwhile, GPT 4 Turbo failed in situations caused by a selenium driver or timeouts caused by the website. Finally, the results show the usability of the system and a great improvement in success rate with model GPT 4 Turbo.

6.4.2 Comparison on metrics with human data

The following experiments show metrics results compared between humans and the simulator with two models, GPT 3.5 Turbo and GPT 4 Turbo. The metrics include:

- average simulation time,
- average number of actions per simulation,
- average time for action,

Simulation time comparison

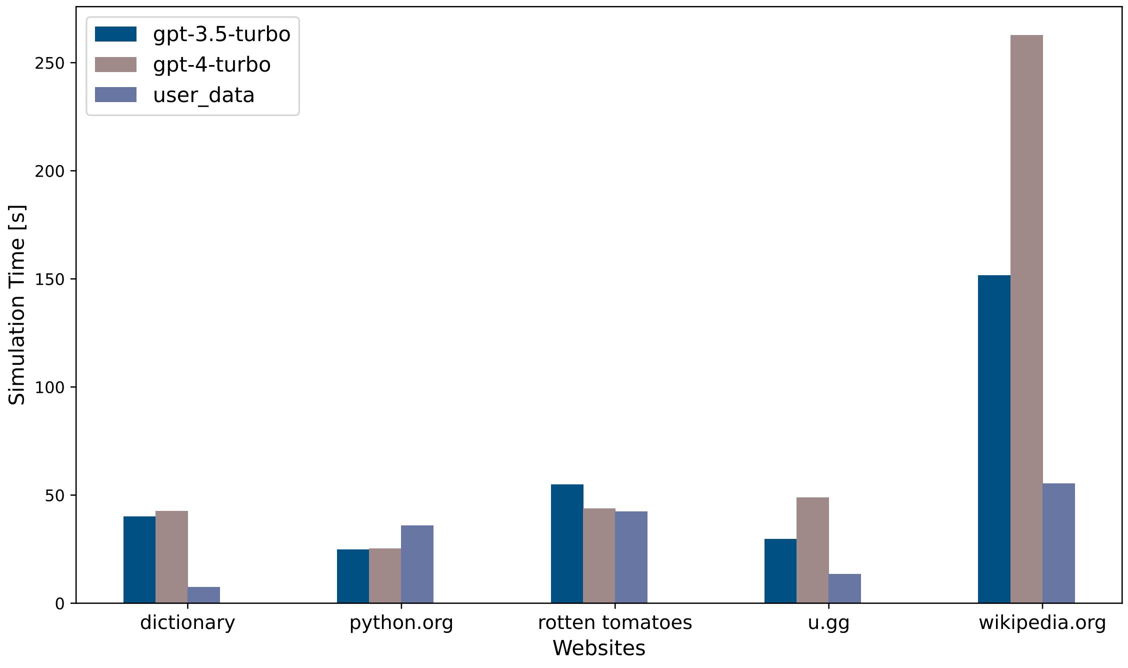

Figure 6.2: Average Simulation comparing Humans and LLM models

The average simulation time between LLMs and Humans shows different results for each site seen in Figure 6.2. On average, humans have shorter simulation time than LLMs. The biggest difference is on the Wikipedia website.

The high difference is caused by the large amount of content on the page that has to be processed by the simulator, which takes a few seconds per LLM query result per simulation loop. The difference between the models and humans is smaller for other websites because they have less content. For the website Python.org, the LLMs have a lower simulation time. This shows one of the differences between the simulator and human data. However, except on the Wikipedia page, the differences are not that dramatic.

Action count

Figure 6.3 The average count of actions per site shows a high number of actions for the human respondents and a lower number for LLM models. The LLMs during the simulation

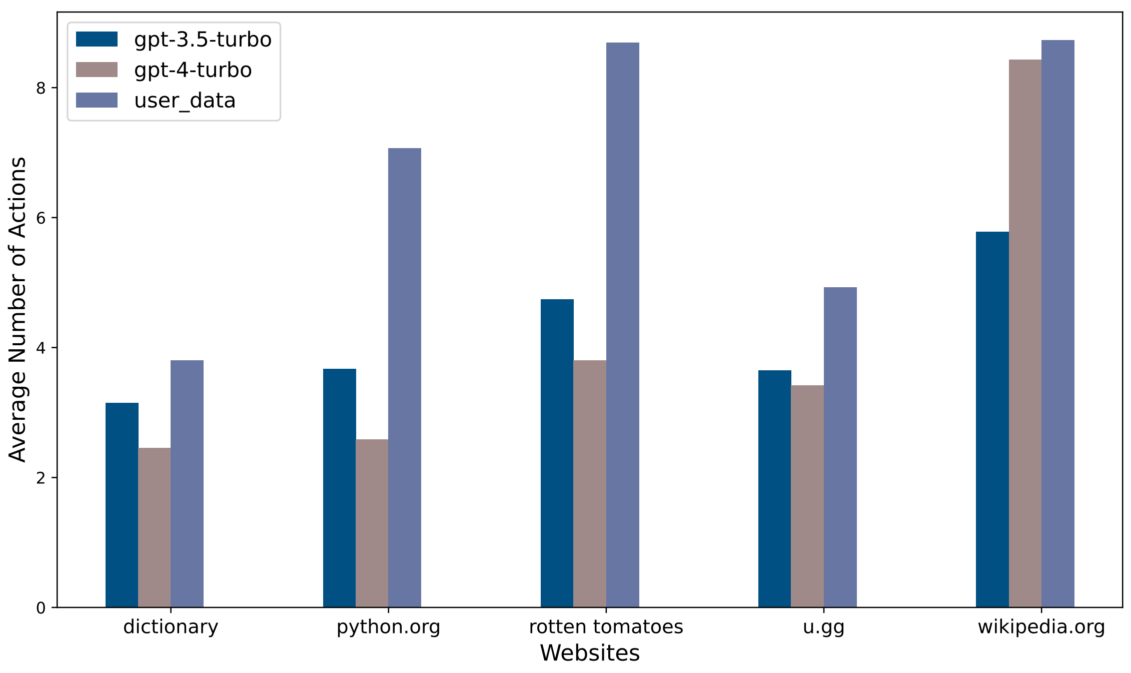

Figure 6.3: Actions per simulation

tend to select accurate results because of the architecture of the simulator that provides detailed information on each webpage, which helps the LLM to make the best actions in a given state. This shows that the average number of actions for LLMs tends to be lower than for humans, showing yet another difference between humans and the simulator. However, the simulator was designed to complete the goals the most effectively.

Time for action

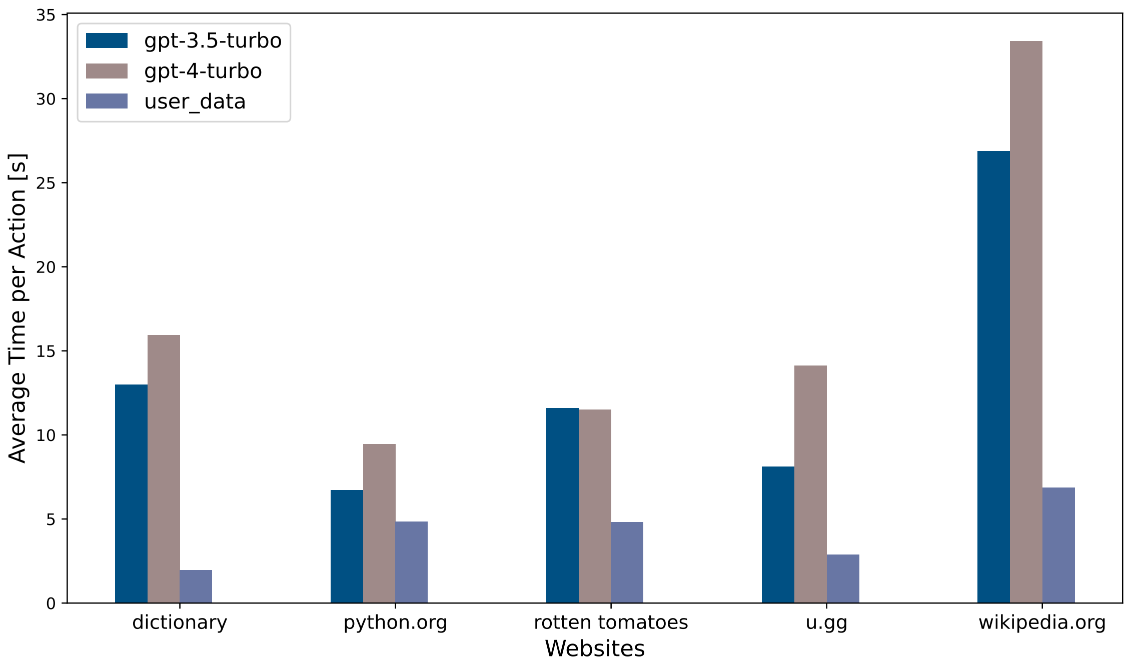

Figure 6.4: average time per action

Figure 6.4 shows large differences between the LLM models and human data. Because the simulator is designed not to model a frequency of actions in a human-like sense and the LLM models are slow, the time to do an action is higher.

6.4.3 Evaluation of Web navigation

The last experiment explores the differences and similarities between humans' web navigation and the simulator. The Sankey diagram was created from URLs in a given action using a Jupiter notebook. The Diagram shows transitions between different pages for each of the simulations or human recordings.

Wikipedia.org

Figure 6.5 shows the navigation of the simulator throughout the simulation. The results show that the decisions of the LLM tend to be the same, And all of the simulations end on a page containing information about Václav Havel. The diagram shows four different paths towards the final page.

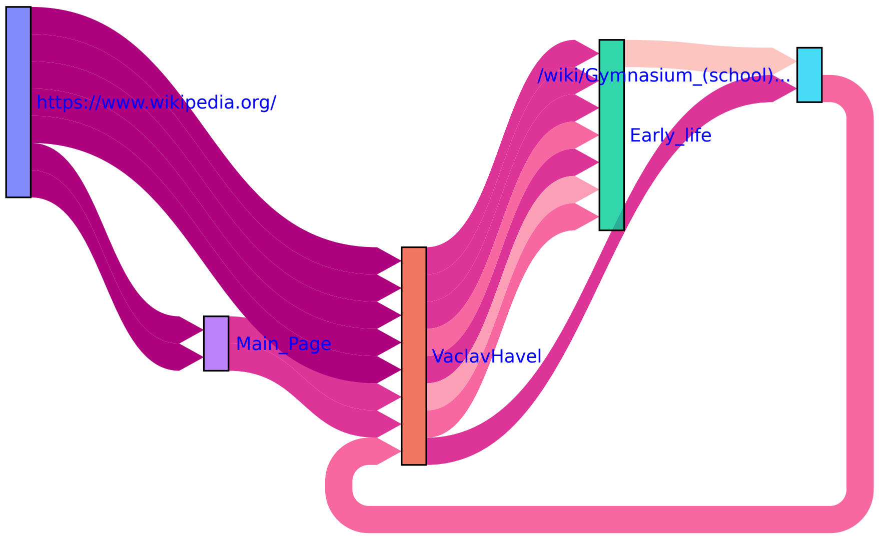

Figure 6.5: Sankey diagram of LLM navigation on Wikipedia.org

Figure 6.6 displays the path of humans throughout the webpage. Most of the Human respondents followed a similar path. The diagram shows that most of the paths have a direct transition between the main page and the final page containing the information for the goal accomplishment.

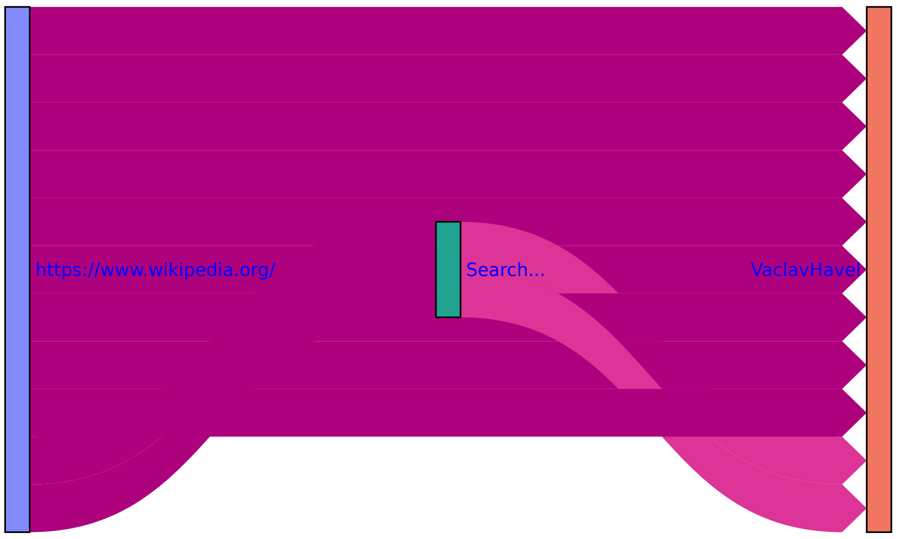

Figure 6.6: Sankey diagram of human navigation on Wikipedia.org

Completing the goal on the Wikipedia page does not provide many different possibilities. Therefore, the graphs do not have that many nodes; however, there are still certain differences. The differences are in the number of unique pages visited before the goal is accomplished. Furthermore, it shows that the human paths are more consistent on the Wikipedia page, whereas the LLM models differ for certain runs.

U.gg

Because the u.gg website is made to find information about characters in a video game in a short amount of time, the path to accomplishing the goal is straightforward. Most of the successful paths contain only two or three unique URLs.

Figure 6.7 shows the paths of the simulations. As seen in the diagram, the URLs between the simulations are the same. The portion of the simulations only needed to make a transition to one URL to be able to complete the goal. The results show that the LLM chose similar actions for the simulation.

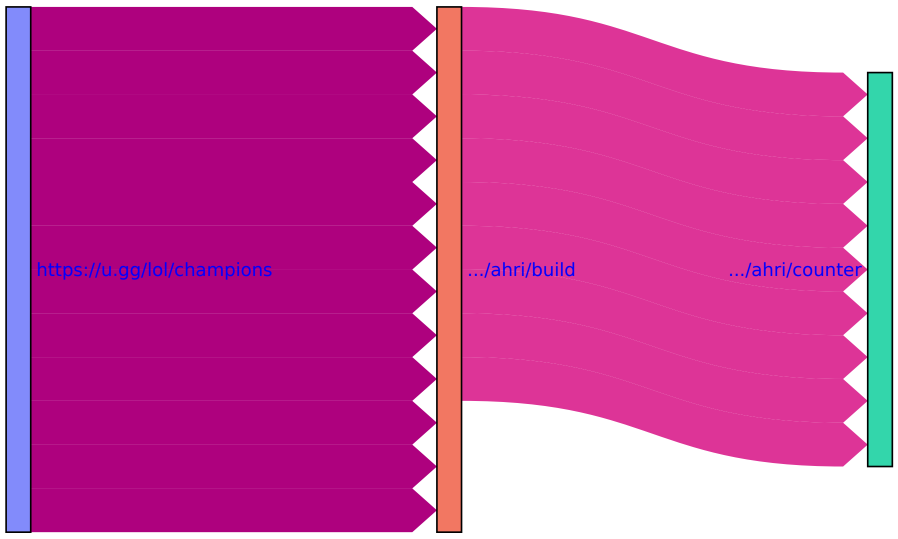

Figure 6.7: Sankey diagram of LLM navigation on u.gg

Figure 6.8 shows a similar number of different paths with certain cases where the human respondents visited additional websites. Most of the humans chose the same approach as the simulator.

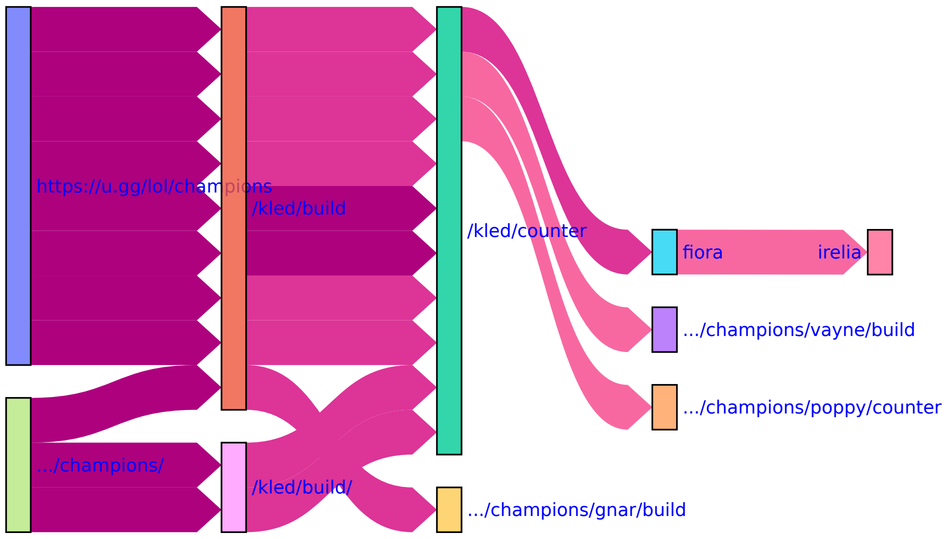

Figure 6.8: Sankey diagram of human navigation on u.gg

Both diagrams show similar results for the website u.gg. The human diagram shows four cases where humans visited an additional page with a unique URL that was not necessarily needed to complete the goal.

Python.org

Figure 6.9 shows small variations in the different simulator runs. The LLMs were able to make accurate decisions in a consistent way. However, the lack of variation shows that the simulator does not simulate all kinds of possible human behaviors.

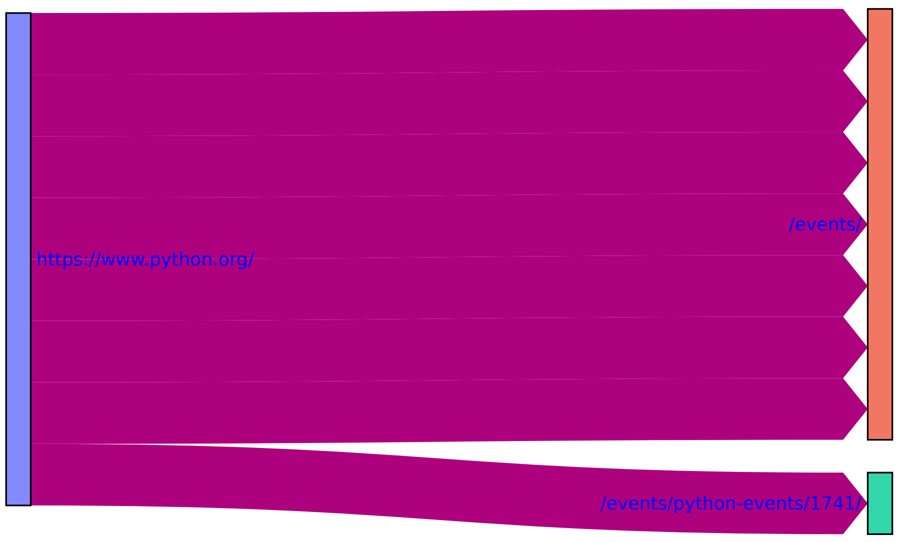

Figure 6.9: Sankey diagram of LLM navigation on python.org

Figure 6.10 displays human navigation on the website. The Sankey diagram shows very chaotic web navigation. There are a lot of humans who choose different approaches to accomplishing their goals.

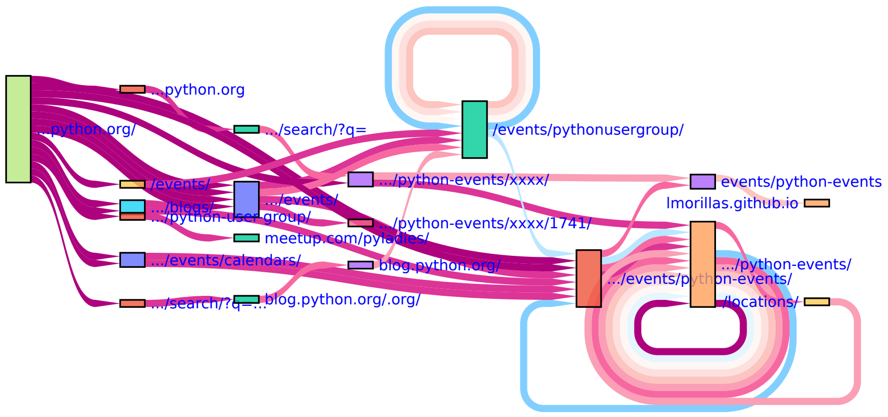

Figure 6.10: Sankey diagram of human navigation on Python.org

The results on the Python.org website show a large difference between the number of variants of humans and the simulator. This result can be attributed to the fact that the Python.org website allows for multiple of ways how to get to a page that contains information about the upcoming events and has multiple pages with different URLs that contain the information. Furthermore, the example with Python.org shows that the simulator is incapable of automatic variations between the runs. It usually finds the most optimal way how to accomplish the goal.

Dictionary.cambridge.org

The dictionary.cambridge.org website uses a search bar to find definitions. Therefore, its navigation is simple. Figure 6.11 shows that all simulation runs were the same regarding the transitions between the different URLs. Additionally, Figure 6.12 shows the same results for the human respondents. The Dictionary site shows its simplicity and no evident differences in the navigation between the LLMs and humans.

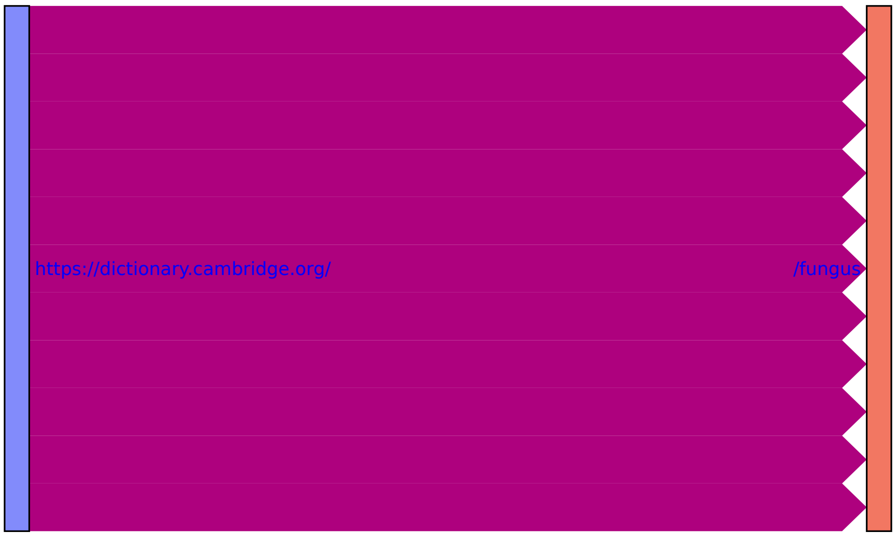

Figure 6.11: Sankey diagram of LLM navigation on Cambridge.Dictionary.org

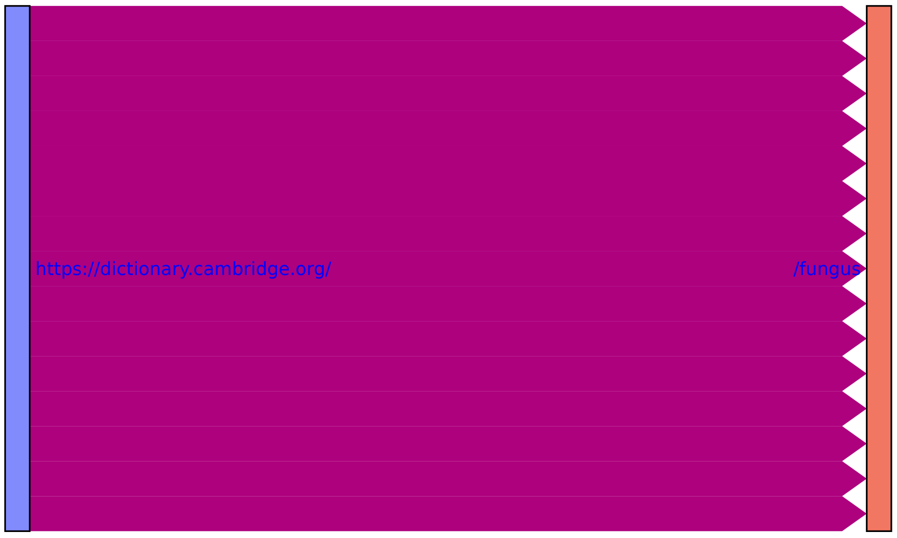

Figure 6.12: Sankey diagram of Human navigation on Cambridge.Dictionary.org

Rottentomatoes.com

Navigation on rottentomatoes.org shows navigation for bot the LLMs and the human respondents. The website has a searchbar that is the only way to retrieve information about a specific film.

Figure 6.13 displays the navigation of the simulator with a consistent selection of transitions between the URLs. Because of the design of the website, there are not many other ways how to navigate the website to accomplish the goal.

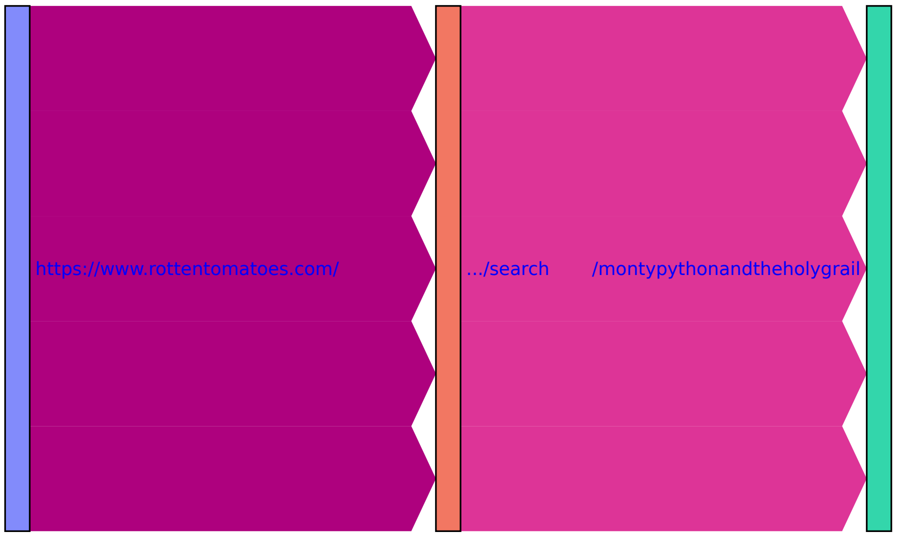

Figure 6.13: Sankey diagram of LLM navigation on rottentomatoes.org

Figure 6.14 shows the navigation on the website by humans. The results show that some human respondents were redirected to a website when they misspelled the movie's name. Furthermore, some respondents continued to provide specific details of the director to find their name even though the information could be read from the website.

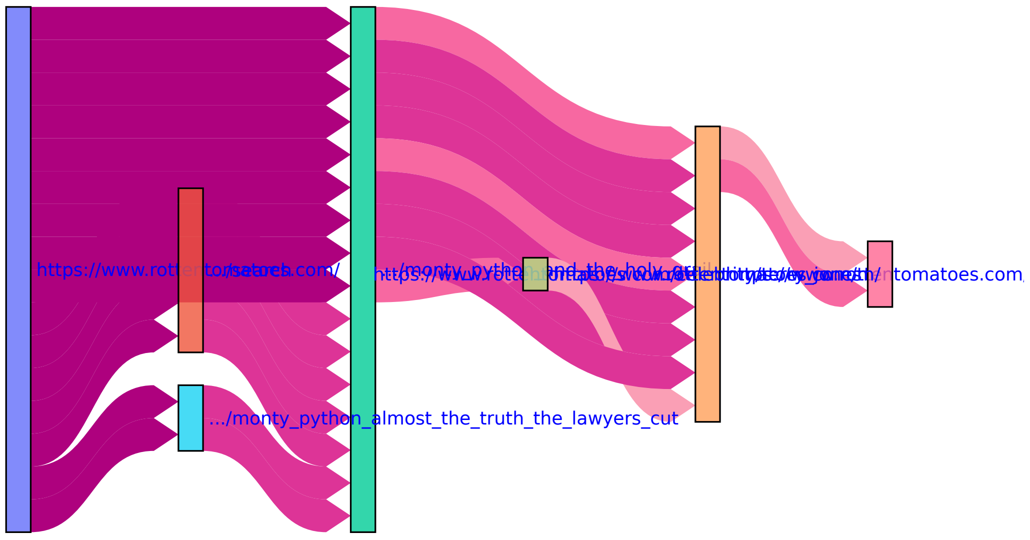

Figure 6.14: Sankey diagram of Human navigation on rottentomatoes.org

The comparison between LLMs and humans on the website rottentomatoes.com shows that the types of transitions are similar. The difference is mainly caused by some humans misspelling the movie name or by not finishing the goal on the page containing the information needed for the goal completion.

Chapter 7

Conclusion

To conclude, simulating human web browsing is far from trivial. LLMs represent a powerful technological advancement capable of solving problems previously thought unsolvable. Utilizing LLMs for the simulation of human behavior presents a particularly promising avenue.

This work's result is a promising and capable tool for automatic web navigation and goal completion based on the general specification of a goal and website. The proposed architecture and implementation allow for the simulation to work on a large number of websites. The result shows the usability of LLM models for human simulation and the creation of decision-making applications. The work defines challenges to be tackled when developing LLM-powered tools and shows how to resolve them.

The results of the experiments show an over 80% success rate for the GPT 4 Turbo model and a decent success rate for the other model. Together, they show a high accuracy in comparing Human behavior, showing the proposed tool's capability and similarity to organic traffic. The comparison with human data shows limitations of the proposed tool that can be further improved and mitigated. The results suggest the tool can be helpful in real use cases.

Future research should aim to test the proposed tool on a broader set of websites and compare it to a better dataset of human data. Also, the simulator could be improved to use real-time timeouts between the chosen actions. The tool could be further enhanced to let the LLM model create its own goals and better simulate Human web traffic. With a dataset of different human personas, the tool could be tested to see if the predefined personas match the actual behavior of a given persona. Furthermore, the predefined actions can be extended, providing more capabilities. Additionally, the tool can be extended to a different use case that involves automatic web navigation and interaction.

Ultimately, this path towards creating an advanced web browsing simulator pushes the boundaries of LLM capabilities and opens new pathways for understanding and replicating human-computer interactions that can be used for many use cases.

Bibliography

[1] Adams, T. *AI-Powered Social Bots.* 2017.

[2] Aher, G., Arriaga, R. I. and Kalai, A. T. *Using Large Language Models to Simulate Multiple Humans and Replicate Human Subject Studies.* 2023.

[3] Amatriain, X., Sankar, A., Bing, J., Bodigutla, P. K., Hazen, T. J. et al. *Transformer models: an introduction and catalog.* 2023.

[4] Andrew. *A brief history of LLaMA models - AGI Sphere.* Apr 2023. Available at: `https://agi-sphere.com/llama-models/`.

[5] Awad, M. and Khalil, I. *Prediction of User's Web-Browsing Behavior: Application of Markov Model.* March 2012. DOI: 10.1109/TSMCB.2012.2187441.

[6] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. et al. *Language Models are Few-Shot Learners.* 2020.

[7] *Chrome DevTools Protocol.* 2024. Available at: `https://chromedevtools.github.io/devtools-protocol/`.

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

[9] for, W. *ChromeDriver - WebDriver for Chrome.* 2024. Available at: `https://chromedriver.chromium.org/`.

[10] Form, R. *Request Form.* 2019. Available at: `https://docs.google.com/forms/d/e/1FAIpQLSfqNECQnMkycAp2jP4Z9TFX0cGR4uf7b_fBxjY_OjhJILlKGA/viewform`.

[11] *Geckodriver.* Jan 2024. Available at: `https://github.com/mozilla/geckodriver/`.

[12] Meta. *Introducing LLaMA.* 2023. Available at: `https://ai.meta.com/blog/large-language-model-llama-meta-ai/`.

[13] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S. et al. *Mistral 7B.* 2023.

[14] Jin, J., Offutt, J., Zheng, N., Mao, F., Koehl, A. et al. *Evasive bots masquerading as human beings on the web.* IEEE Computer Society, 2013. DOI: 10.1109/DSN.2013.6575366. Available at: `https://doi.org/10.1109/DSN.2013.6575366`.

[15] Keary, T. *Prompt Injection Attack.* Sep 2023. Available at: `https://www.techopedia.com/definition/prompt-injection-attack`.

[16] LANGCHAIN. *Https://www.langchain.com/.* 2023. Available at: `https://www.langchain.com/`.

[17] *Level 1 Document Object Model Specification.* 2024. Available at: `https://www.w3.org/TR/WD-DOM/`.

[18] LIU, Y., OTT, M., GOYAL, N., DU, J., JOSHI, M. et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* 2019.

[19] *LLaMA*. 2014. Available at: `https://huggingface.co/docs/transformers/main/en/model_doc/llama`.

[20] NAVEED, H., KHAN, A. U., QIU, S., SAQIB, M., ANWAR, S. et al. *A Comprehensive Overview of Large Language Models.* 2023.

[21] PROMPTINGGUIDE.AI. *Retrieval augmented generation.* 2021. Available at: `https://www.promptingguide.ai/techniques/rag`.

[22] QIAN, C., CONG, X., LIU, W., YANG, C., CHEN, W. et al. *Communicative Agents for Software Development.* 2023.

[23] RADFORD, A., WU, J., CHILD, R., LUAN, D., AMODEI, D. et al. *Language Models are Unsupervised Multitask Learners.* 2019. Available at: `https://api.semanticscholar.org/CorpusID:160025533`.

[24] LAMBERT, N. *Reinforcement learning from human feedback.* 2022. Available at: `https://huggingface.co/blog/rlhf`.

[25] SELENIUM. *Selenium.* 2023. Available at: `https://www.selenium.dev/`.

[26] *Semantic-kernel.* Jan 2024. Available at: `https://github.com/microsoft/semantic-kernel`.

[27] SZYMANSKI, B. and CHUNG, M.-S. *A method for indexing Web pages using Web bots.* 2001. DOI: 10.1109/ICII.2001.983028.

[28] TOUVRON, H., LAVRIL, T., IZACARD, G., MARTINET, X., LACHAUX, M.-A. et al. *LLaMA: Open and Efficient Foundation Language Models.* 2023.

[29] VARKEY, B. *Large language models: techniques, examples, prevention methods | lakera – protecting ai teams that disrupt the world.* 2023. Available at: `https://www.lakera.ai/blog/jailbreaking-large-language-models-guide`.

[30] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L. et al. *Attention Is All You Need.* 2023.

[31] *WebDriver.* Jun 2018. Available at: `https://www.w3.org/TR/webdriver1/`.

[32] XIAO, G., TIAN, Y., CHEN, B., HAN, S. and LEWIS, M. *Efficient Streaming Language Models with Attention Sinks.* 2023.

[33] YANG, Y., VLAJIC, N. and NGUYEN, U. *Next Generation of Impersonator Bots: Mimicking Human Browsing on Previously Unvisited Sites.* November 2015. DOI: 10.1109/CSCloud.2015.93.

[34] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I. et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023.

[35] Ye, J., Chen, X., Xu, N., Zu, C., Shao, Z. et al. *A Comprehensive Capability Analysis of GPT-3 and GPT-3.5 Series Models*. 2023.

[36] Yenduri, G., M, R., G, C. S., Y, S., Srivastava, G. et al. *Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*. 2023.

[37] Zhou, T., Han, X. and Wang, B. *Towards the understanding of human dynamics*. 2008.

Appendix A

Contents of the included storage media

```
SD_card/
 ├─ data_humans/
 ├─ experiment_data/
 ├─ experiment_data_requests/
 ├─ figures/
 ├─ src/
 │   ├─ __init__.py
 │   ├─ chains.py
 │   ├─ llm_provider.py
 │   ├─ logger.py
 │   ├─ simulation.py
 │   └─ web.py
 ├─ __init__.py
 ├─ app.py
 ├─ benchmark.ipynb
 ├─ benchmark.py
 ├─ config.json
 ├─ RADME.md
 └─ doc
     ├─ BP.pdf
     └─ src/
```