

AI TOOL FOR SCIENTIFIC LITERATURE DATA EXTRACTION

Justiina Ronkainen
Bachelor's Thesis
Spring 2025
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology
Option of Software Development

Author: Justiina Ronkainen

Title of thesis: AI tool for scientific literature data extraction

Thesis supervisor: Raija Westerlund

Term and year of completion: Spring 2025

Pages: 49

The aim of this project was to explore artificial intelligence (AI) by developing a tool that leverages large language models (LLMs) to extract structured information from scientific articles. Systematic literature reviews and meta-analyses are based on accurate data extraction, a process that is traditionally manual, time-consuming and prone to mistakes. This project investigated the potential of LLMs to automate and streamline this task. Focus was to extract the key study elements such as article metadata, study design, statistical methods and results.

The project was implemented using Python's LangChain framework and OpenAI API. Key techniques included prompt engineering, text processing and chunking to adapt the content suitable for LLM. GPT-3.5 and GPT-4.1 models were tested and evaluated against each other and human-extracted gold standard to assess performance. The model demonstrated potential in extracting some information, such as article metadata and study design; however, it struggled to reliably extract all relevant results from the articles.

Despite current limitations, LLMs hold promise for automating aspects of scientific data extraction. Improvements such as section-specific extraction and potential fine-tuning may enhance the performance and offer a scalable solution to one of the most labour-intensive steps in systematic literature review.

CONTENTS

ABSTRACT	2
CONTENTS.....	3
1 INTRODUCTION	5
2 THEORETICAL FOUNDATION.....	6
2.1 Systematic literature reviews and meta-analysis in health sciences ..	6
2.1.1 Overview	6
2.1.2 Data extraction from scientific journal articles	7
2.2 Generative AI	8
2.2.1 Traditional vs. generative AI	8
2.2.2 Large language models, transformer architecture and attention mechanism	10
2.2.3 Key concepts	11
2.3 Prompt engineering	14
2.3.1 What is prompt engineering?.....	14
2.3.2 Naïve vs. structured prompts.....	15
2.3.3 Five principles of prompting.....	17
3 MATERIAL AND METHODS	23
3.1 OpenAI platform and GPT models	23
3.2 Python programming language	24
3.2.1 Reasons for selecting Python	24
3.2.2 Used frameworks and libraries	24
3.3 System workflow	25
3.3.1 Input data selection	25
3.3.2 Preprocessing.....	26
3.3.3 Model integration	26
3.3.4 Defining the information to be extracted	27
3.3.5 Prompt engineering	28
3.3.6 Data extraction	30
3.3.7 Output formatting.....	32
3.4 Testing and validation	32
4 RESULTS	34

4.1	Pre-processing the input and the prompt	34
4.1.1	Text conversion and chunking	34
4.1.2	Prompt engineering	34
4.1.3	Technical and practical challenges	35
4.2	AI tool performance evaluation	35
4.2.1	Extraction of the article metadata	35
4.2.2	Extraction of the study design and statistical methods	37
4.2.3	Extraction of the results	39
4.3	Comparison with manual (human) extraction	41
5	DISCUSSION	42
6	CONCLUSIONS	44
	REFERENCES	45

1 INTRODUCTION

Systematic literature reviews and meta-analyses are critical components of evidence-based research. These methods depend on the accurate and consistent extraction of structured information from large number of scientific publications. Traditionally, this data extraction is performed manually, and it is a time-consuming and labour-intensive task and very prone to human error. The need for scalable and efficient alternative for manual data extraction is urgent as the volume of scientific literature continues to grow.

Recent achievements in artificial intelligence (AI), particularly generative AI models and large language models (LLMs) offer new opportunities to automate complex tasks involving understanding of natural language. Generative AI models such as OpenAI's generative pre-trained transformer (GPT) models are trained on enormous collection of texts enabling them to interpret and generate human-like language with contextual understanding. These capabilities are well-suited for tasks such as extracting metadata, study characteristics and results from scientific articles.

The aim of this project was to gain practical experience with AI programming and assess the feasibility of LLMs in extraction of structured data from scientific articles. The project involved implementing a prototype tool that used GPT models via the OpenAI API using techniques such as prompt engineering, preprocessing the articles and experimenting with structured output formats. Accuracy and completeness of extracted data was evaluated against human-extracted gold standard. This project provides insight into the capabilities and limitations of LLMs in scientific data extraction and offers recommendations for future development.

2 THEORETICAL FOUNDATION

2.1 Systematic literature reviews and meta-analysis in health sciences

2.1.1 Overview

In health and other natural sciences, research is conducted to provide information on the relationship between exposures and outcomes of interest. However, the results of a single study often cannot provide definitive universal answer to the research question. (Gopalakrishnan and Ganeshkumar, 2013). Systematic review is a way to synthesize up-to-date summary of published knowledge obtained from multiple primary studies and give a wider understanding of the phenomenon of interest. Systematic reviews usually involve a detailed plan and search strategy of published research articles with an aim to use explicit and systematic methods to provide unbiased, high-quality summarization of the topic. (Lasserson, Thomas and Higgins, 2024).

Often, systematic reviews include a meta-analysis component which uses statistical techniques to quantify the overall effect from multiple individual study results (Uman, 2011). Before a meta-analysis can be conducted, researchers must extract the results and other relevant information from the original publications. This process, known as data extraction, is most time-consuming phase of a systematic review and meta-analysis. If not carried out properly, data extraction may lead to low-quality inputs, ultimately resulting in poor-quality outputs with unreliable conclusions. (Taylor, Mahtani and Aronson, 2021). Systematic reviews can utilize different sources of data including journal articles, conference abstracts, errata and letters, trial registers and clinical study reports, regulatory reviews and individual participant data. Journal articles are the most used source which is why the tool developed during this project was specifically designed to extract data from scientific journal articles. (Li, Higgins and Deeks, 2024).

2.1.2 Data extraction from scientific journal articles

Scientific writing is different from literary writing as it emphasizes clarity, precision and objectivity over artistic expression. A typical structure of scientific article follows the IMRAD structure, i.e. introduction, methods, results and discussion. Despite variations in headings and subheadings based on individual journal guidelines, generally all journals follow the IMRAD structure. (Heßler, Rottmann and Ziegler, 2020).

While this standardized format facilitates easier data extraction, results that appear only in tables or figures can complicate automated extraction process. Another challenge arises from the wide variation in the article formats. Older articles are often available only as scanned images, whereas modern articles can be in a range of digital formats. Automated data extraction is easier if the article is available in markup languages like hypertext markup language (HTML) or extensible markup language (XML) as these formats use specific, computer-friendly tags to define different structures. Most articles, however, are published in portable document format (PDF), which does not record structural information. In addition, varying journal layouts, such as differences in the number of columns, further confuse the automated extraction methods. (Hong *et al.*, 2021).

One article usually contains results on multiple different exposures and outcomes. Furthermore, a variety of different statistical methods and models may be employed. For the systematic review, and especially meta-analysis, it is essential to extract not only the results themselves, but also the method and units used to obtain them. As indicated by the IMRAD structure, methods and results are typically reported in separate sections of an article. Automated data extraction thus requires the ability to accurately link methodological details with their corresponding results across different parts of the text. Recent emergence of natural language processing (NLP), language models (LMs) and especially large language models (LLMs) has enabled greater ability to extract complex data accurately. (Polak and Morgan, 2024). These approaches are discussed in more detail later in the following chapters.

2.2 Generative AI

2.2.1 Traditional vs. generative AI

Artificial intelligence (AI) has been a hot topic for the last two decades. We have used the traditional (also known as narrow or weak) AI for the most part of those decades, but with the emergence of generative AI, it is becoming more accessible to the general population. Key difference between traditional and generative AI is that traditional AI focuses on analysing the data and making predictions, while generative AI creates new data that mirrors original, so-called training data. In other words, traditional AI is great at pattern recognition, whereas generative AI is designed for pattern creation (Marr, 2023). Some additional differences are listed in TABLE 1 below.

TABLE 1. Comparison of traditional and generative AI.

	Traditional AI	Generative AI
Focus and output	Limited scope of operations. Uses data analysis to identify patterns, make predictions and perform specific predetermined tasks from the data. Highly efficient problem solver.	Creates original data based on human input and data analysis. Can be viewed as creative assistant.
Transparency	Operates on predefined rules, so its decision-making process is more transparent and interpretable.	Can be less transparent due to complex nature of its learning algorithms. Challenging to understand how it arrives at specific outputs.
Example applications	Spam filtering, fraud detection and recommendation systems.	Content creation, like writing, music composition, and image generation.

In fact, generative AI has been around for several decades already. The overview of the development of generative AI methods and applications is illustrated in the FIGURE 1. First systems, *rule-based generative AI systems* were developed in the 1950's. These programs generate data by following rules determined by

human experts. One of the first AI chatbots, ELIZA, was introduced in 1966 to act as a psychotherapist. It processes text inputs and then seeks predesigned responses based on the rules. The scope of discussion topics was limited, but the rules for those topics were quite effective for simulating human conversations. (He, Cao and Tan, 2025). Other rule-based system, SYSTRAN, was developed in 1968 and served as a translation tool for web browsers until 2007 (SYSTRAN, 2025). *Model-based generative algorithms* were designed based on statistical or physical models. These include methods such as machine learning, neural networks, computer graphics and computer vision. *Deep generative methods* like deep neural networks have greatly benefited from the growth in computational power and data resources enabling them to demonstrate remarkable capabilities in content generation. (He, Cao and Tan, 2025).

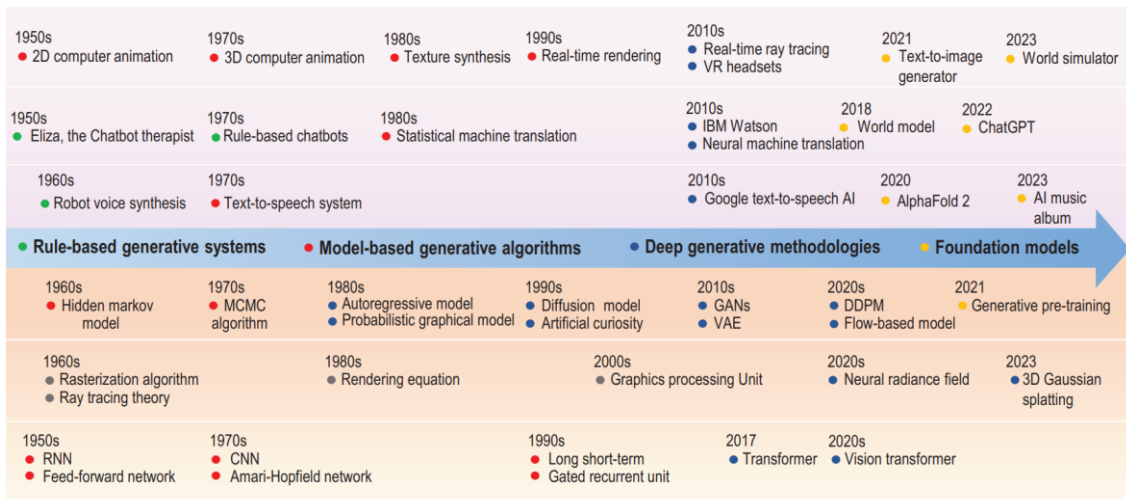


FIGURE 1. Timeline of the development generative AI methods and applications. Obtained from (He, Cao and Tan, 2025).

Finally, the emergence of *foundation models*, such as generative pre-trained transformers (GPTs) marks a major revolution in generative AI. These models leverage deep learning techniques and stand out due to their massive scale in terms of model size and the training data used (TABLE 2).

TABLE 2. Development of GPT models. Obtained from (Alammar and Grootendorst, 2024).

Model	Released	Parameter count	Training data
GPT-1	11.6.2018	117 million	BookCorpus: 4.5 GB of text from 7000 unpublished books
GPT-2	14.2.2019 (initial) 5.11.2019 (full)	1.5 billion	WebText: 40 GB of text and 8 million documents from 45 million web pages
GPT-3	28.5.2020	175 billion	499 billion tokens consisting of Common Crawl (570 GB), WebText, English Wikipedia and two books corpora
GPT-3.5	15.3.2022	175 billion	Undisclosed
GPT-4	14.3.2023	1.8 trillion (rumoured)	Undisclosed, but rumoured to be 13 trillion tokens

2.2.2 Large language models, transformer architecture and attention mechanism

Large language models (LLMs) use deep neural networks that have been developed over the past few years. Before LLMs, traditional methods excelled in categorisation tasks such as email spam filtering and pattern recognition, but they underperformed in language tasks that required complex understanding and ability to create coherent and contextually correct original text. Previous generations of language models could not write an email from a list of keywords – a task that is trivial for LLMs today. (Raschka, 2024). Transformer models such as GPTs offered in OpenAI’s ChatGPT are LLMs that were first developed to solve any task that transforms an input sequence to an output sequence. These sequence-to sequence tasks include translation, text summarisation question answering and story or content generation. (Ferrer, 2024)

Transformers use a mathematical technique called attention to analyse relationships between words and comprehend context and meaning. Transformers often use an encoder-decoder architecture. The encoder maps the input symbols to numeric representations and decoder then generates an output sequence of symbols. (Vaswani *et al.*, 2017). The original transformer architecture had six encoder, and six decoder layers stacked sequentially, but modern transformer models can use as many layers as needed, depending on the task and computational resources (FIGURE 2). (Ferrer, 2024). Model is auto regressive, meaning that it uses previously generated symbols as additional input when generating the next. (Vaswani *et al.*, 2017).

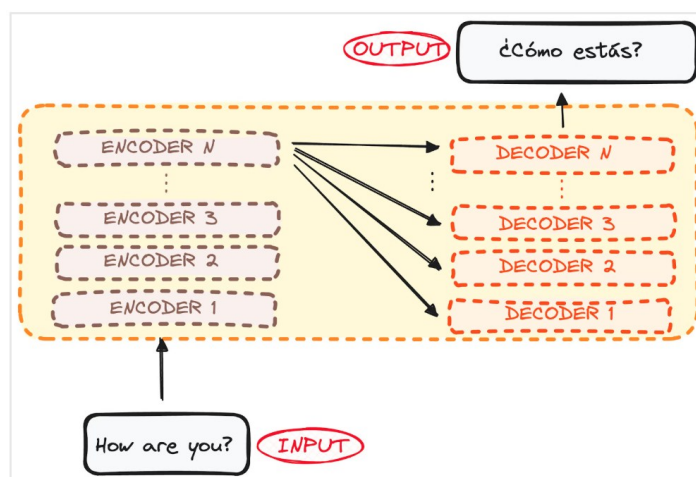


FIGURE 2. Illustration of transformer model performing sequence-to-sequence translation with multiple encoder and decoder layers. Obtained from (Ferrer, 2024).

2.2.3 Key concepts

Tokenization

Tokenization connects raw text and language models by translating human-readable text into tokens that can be used by statistical models (Schmidt *et al.*, 2024). Tokens can be words or punctuation marks, and they are transformed into vectors, which are numerical representations of the words included in the tokens. Tokenization is often performed at the sub word level, where rare words are split into smaller units, while more frequent words are kept as distinct entities. For example, sentence “Tokenization is an essential part of LLM utilization” can be

tokenized to: ["token", "##ization", "is", "an", "essential", "part", "of", "llm", "utilization"], where "##" indicates that the corresponding token should be attached to the previous one. Tokens are compared against a set of unique tokens, known as the *vocabulary*, which was created when the *corpus*, i.e. the collection of texts used to train LLM, was tokenized. (Metzger, 2022).

Embeddings

Embeddings are tokens mapped into a vector space. Vectors are multidimensional arrays of numbers that capture semantic properties and relationships of the tokens. Machine learning models and neural networks can learn the relationships between the vectors automatically rather than having to specify them manually. (Metzger, 2022). Models are trained to identify and learn these relationships ensuring that words with similar meanings are mapped close to one another (FIGURE 3). This enables LLMs to understand context, semantics and the web of relationships that form language (Phoenix and Taylor, 2024).

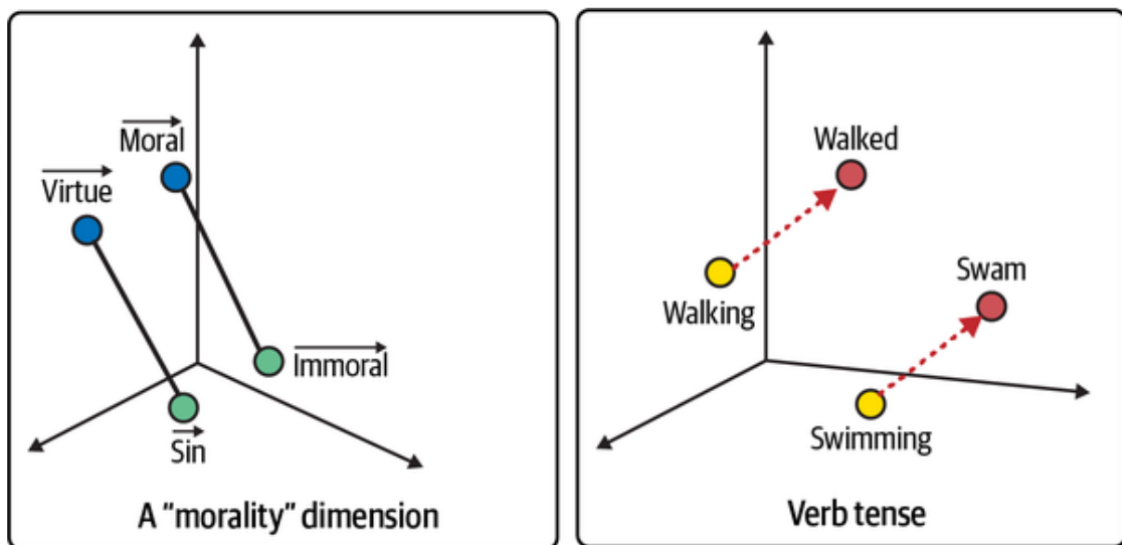


FIGURE 3. Semantic proximity of word vectors within embedding space. Obtained from (Phoenix and Taylor, 2024).

Tokenization and embeddings are a way to make the human language numeric and understandable for machines. FIGURE 4 shows a simplified representation of the process.

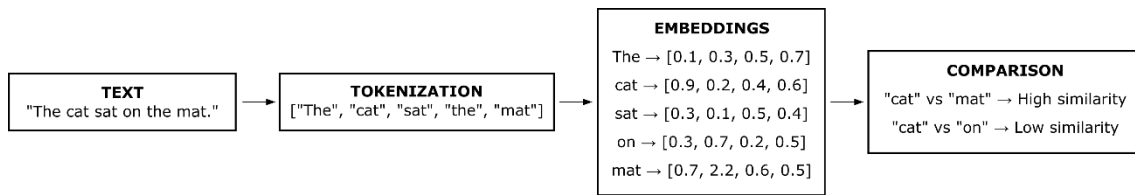


FIGURE 4. Flowchart of the text tokenization, embeddings and semantic comparison of an example sentence. Generated by ChatGPT.

Context window and token limits

When working with LLMs, one of the key considerations is the token limit. The context window refers to the maximum number of tokens that the model can process in a single request before reaching the token limit. The size of the context window is specific to each model, and it includes both input (the prompt) and output tokens (model’s response). In addition to the total context window, some models have their own maximum output token limit, which defines how many tokens they can generate in response to a given prompt. With more complex tasks, the context window often forces a trade-off between the input prompt and the output. If a longer response is required, the prompt may need to be shortened, and conversely, a longer prompt may limit the length of the model’s response. (Bajaj, 2025). Token limits may change over time, some of the current limits and other information on different GPT models are listed in the TABLE 3.

TABLE 3. Context capacity, output limits, knowledge cutoff, and pricing of selected OpenAI language models (as of May 2025).

Model	Context window (tokens)	Maximum output tokens	Knowledge cutoff	Pricing (per 1M tokens)	
				Input	Output
GPT-4.1	1,047,576	32,768	1.1.2024	\$2.00	\$8.00
o4-mini	200,000	100,000	1.1.2024	\$1.10	\$4.40
o3	200,000	100,000	1.1.2024	\$10.00	\$40
o3-mini	200,000	100,000	1.10.2023	\$1.10	\$4.40
gpt-3.5-turbo	16,384	4,096	1.10.2021	\$0.50	\$1.50

Fine-tuning vs. prompt engineering

Fine-tuning is a technique for adapting a pre-trained model (such as GPT-4) to new data or specific tasks. It allows a general-purpose model to be specialised for specific needs. During fine-tuning, the pre-trained model processes examples from a task-specific dataset, learns by comparing its predictions to expected outputs and adjusting its parameters to minimise errors, repeating this process until performance stabilises. A high-quality, representative dataset is essential, and fine-tuning large models typically requires hundreds to thousands of examples. (Srivastava, 2023).

Prompt engineering is described in more detail in the next section. Unlike fine-tuning, prompt engineering does not change the model's internal parameters. Instead, it focuses on crafting and refining the prompt that guide the model's output. Main differences between fine-tuning and prompt engineering are presented in TABLE 4. (Nexla, 2025).

TABLE 4. Key differences between fine-tuning and prompt engineering. Modified from (Nexla, 2025).

Difference	Fine-tuning	Prompt engineering
Definition	Adjusting model parameters using task-specific data.	Modifying input prompts without changing model weights.
Process	Involves data preparation, training and optimisation.	Involves crafting and refining prompts.
Accuracy	Typically achieves higher task-specific accuracy.	Depends on prompt quality, generally less precise.
Flexibility	Less flexible across domains.	More flexible across domains.
Resource investment	More	Less

2.3 Prompt engineering

2.3.1 What is prompt engineering?

A prompt is a piece of text or set of instructions given to a LLM, such as ChatGPT. It triggers LLM to generate a response, which can be for example text completion,

translation, summary of information, creative text piece, or even code generation. Prompt engineering, as the name suggests, is the process of developing effective prompts to guide LLM in producing the desired responses that are both structured and reproducible. (Ahmed, 2024). In 2023, Robin Li, cofounder and CEO of Chinese AI giant Baidu boldly claimed that “In ten years, half of the world’s jobs will be in prompt engineering, and those who cannot write prompts will be obsolete”. While the prediction is likely overstated, it highlights how mastering the interaction with AI systems could become a core competency across industries. (Duranton, 2023). On the other hand, some believe prompt engineering will be a basic skill required of many jobs, like proficiency in Microsoft Excel, rather than a job on its own (Phoenix and Taylor, 2024). Other future perspective is that prompt engineering might eventually become obsolete, because AI could soon understand human intent well enough to reduce the need for precise prompting (Anderson, 2024). Indeed, prompt engineering was shown to be best done by the AI model itself (Battle and Gollapudi, 2024). Despite these differing projections about the future of prompt engineering, for now, it continues to be a valuable tool to improve the quality and relevance of LLM outputs.

2.3.2 Naïve vs. structured prompts

Information and examples in the following two chapters are mainly from Phoenix and Taylor (2024).

Naïve prompt is the unstructured prompt such as `Can I have a list of product names for a pair of shoes that can fit any foot size?` that we usually give to ChatGPT or other generative AI chatbot to get fast response. The response could be something like:

```
Certainly! Here's a list of product names for a pair of shoes that can fit any foot size:  
UniFit SoleStrides  
FlexiSize All-Fit Shoes  
OmniFit FootFlex  
...  
AnyStep Adaptiwalkers  
FlexiRange UniversalSteps  
PerfectGlide MultiFit Shoes
```

Naïve prompts usually produce good enough results when you don’t plan to do the same task again. If you plan to put the prompt into production, naïve prompts

may not be enough. Usually in the production of for example an application that generates product names, we would like to output the information from the LLM response in a certain form. As we see from the above response, suggested shoe names vary in respect of the length and style. In our product name generator, we might want to get only certain types of names (a single word or a concatenation) or perhaps emulate someone famous to create great product names in our opinion. Furthermore, the response is a list of names separated by line and the number of names is unspecified. When we run the prompt multiple times, the suggestions can sometimes be a numbered list, sometimes a bulleted list or they might come in one line separated by comma. Often, the response has some text at the beginning which also makes it hard to be parsed programmatically. (Phoenix and Taylor, 2024). Examples of low and higher quality prompts are listed in TABLE 5.

TABLE 5. Examples of lower and higher quality prompts. Collected from Udemmy online course 'The Complete Prompt Engineering for AI Bootcamp' (Taylor and Phoenix, 2025).

Lower quality prompt	Higher quality prompt
Can you give me book recommendations?	Can you recommend any recent science fiction novels that explore themes of artificial intelligence and humanity's future?
What's the best computer?	What's the best computer for video editing tasks in 2023, considering a budget of around 2000 €?
Tell me about Shakespeare.	Can you provide an analysis of the thematic elements in William Shakespeare's play "Macbeth", focusing on the role of ambition and power?
How to make a cake?	What are the steps to baking a classic French Genoise sponge cake, and what common mistakes should I avoid?
What's the weather?	What's the forecast for Oulu, Finland on June 5, 2025, specifically focusing on humidity levels and the chance of rain?

In the structured prompt, we can ask LLM to provide the response in precise structure and even give examples of the style we wish the product names to be in. Following structured prompt asks the shoe names in the style of Steve Jobs and gives instructions and examples on how to output the results:

```
Brainstorm a list of product names for a shoe that fits any foot size, in the style of Steve Jobs.
```

```
Return the results as a comma-separated list, in this format:  
Product description: A shoe that fits any foot size  
Product names: [list of 3 product names]
```

```
## Examples
```

```
Product description: A refrigerator that dispenses beer  
Product names: iBarFridge, iFridgeBeer, iDrinkBeerFridge
```

```
Product description: A watch that can tell accurate time in space  
Product names: iNaut, iSpace, iTime
```

```
Product description: A home milkshake maker  
Product names: iShake, iSmoothie, iShake Mini
```

The response would be something like this:

```
Product description: A shoe that fits any foot size  
Product names: iFitFoot, iPerfectFit, iShoeSize
```

Above prompt is optimized to deliver solid product names in the right format. The user of our product name generator can use any other person or style of names they like, change the response format and the output of this prompt can be the input of another. We can even update the examples used in the prompt based on user feedback, which makes our system smarter over time. (Phoenix and Taylor, 2024).

2.3.3 Five principles of prompting

In their book *Prompt Engineering for Generative AI*, James Phoenix and Mike Taylor describe the process of prompt optimization to follow Five principles of prompting. These address five common issues often related to naïve prompts – direction, format, examples, quality and labour division – helping to engineer more effective and better performing structured prompts. (Phoenix and Taylor, 2024).

Give direction

Whether the style and type of the LLM response are good is often a subjective matter. AI can generate an endless number of product names, but without an idea of the names we like, its chances of guessing the best ones are low. In the example prompt of the previous chapter, we asked for product names emulating the style of Steve Jobs and received responses similar to Apple products, such as iFitFoot, iPerfectFit and iShoeSize. If we change the style to some else, e.g.

IKEA, the response changes completely, yielding names like FÖTSKOR, STRYDA, PASSFORM. This powerful prompt engineering technique is called *role-playing*. This and some other common techniques are listed in the TABLE 6. Giving direction makes prompting very effective, but it is important not to overdo it. Too much direction can lead the model to quickly get to conflicting combinations that it can't resolve. (Phoenix and Taylor, 2024).

TABLE 6. Overview of techniques commonly used in prompt engineering. Collected from (Phoenix and Taylor, 2024).

Technique	Description	Example
Role-playing	Direct the model to adopt a specific role, job, function or style.	Brainstorm a list of product names for a shoe that fits any foot size, in the style of Steve Jobs.
Prewarming/ internal retrieval	Start the conversation asking for best practice advice and then asking the model to follow its own advice.	Please give me 5 tips for naming products based on expert industry advice. Wait for response, then ask Using this advice, can I have a list of product names for a pair of shoes that can fit any foot size?
Least to most	Start with the least amount of information and expand it stepwise to collect larger and larger amount of information.	Create list of Disney characters. Wait for response, then ask For each Disney character, generate me a bio and tell me more about the character.
Explain it like I'm five (a.k.a. ELI5)	Ask the model to explain a phenomenon or a given text as if it was talking to a five-year-old child. Great way to convert technical documents into simple summaries that anyone can understand. Other ages or age groups can be used, e.g. 'explain it like I'm college student'.	Explain the following text as if I was 5 years old. Explain it like I'm 5. Text: Photosynthesis is the process by which green plants use sunlight to make their own food. They take in carbon dioxide from the air and water from the soil. Using sunlight, they turn these into sugar and oxygen.
Ask for context	Encourage the model to ask for additional context if needed.	Should I use MongoDB or PostgreSQL for my project? Provide a list of requirements that would help you to make an informed and high-quality decision about which database solution I should choose? If you need more context, please specify what would help you to make a better decision.
Meta prompting	Ask the model to create the prompt for you.	write a prompt that helps extract key data points (like sample size, methodology, results, and conclusions) from a scientific research article in health sciences.

Technique	Description	Example
Avoiding hallucinations with reference	Instruct the model to only answer using the reference text. Can be useful for data extraction task.	Refer to the articles enclosed within triple quotes to respond to queries. In cases where the answer isn't found within these articles, simply return "I could not find an answer".
Self-evaluation	Ask the model to evaluate itself.	Generate a "Hello world" Python function. Wait for response, then ask Given the previous function, can you improve it again with anything else? Perhaps defaults or type hints?

Specify format

AI models can translate not only text between languages (e.g. from French to English or from natural language to Python code) but also information between data structures (e.g. from JavaScript object notation, JSON, to XML). We can increase efficiency and make later parsing of the LLM response easier by specifying the expected format upfront. For example, it can be helpful to output text generation models in JSON format instead of an ordered list, as that is the format used in API responses. The JSON can then be parsed, error-checked and used to render the front-end of an application. We can modify the earlier prompt by specifying the format and giving examples of the desired output:

Return a comma-separated list of product names in JSON for "A pair of shoes that can fit any foot size.". Return only JSON.

Examples:

```
[{
  "Product description": "A home milkshake maker.",
  "Product names": ["HomeShaker", "Fit Shaker",
    "QuickShake", "Shake Maker"]
},
{
  "Product description": "A watch that can tell accurate time in
space.",
  "Product names": ["AstroTime", "SpaceGuard", "Orbit-Accurate",
    "ElipTime"]}
]
```

Broken JSON results a parsing error, which can act as a trigger to retry the prompt or investigate before continuing. It can also help to specify at the beginning or end of the prompt, or in the system message, that `you are a helpful assistant that only responds in JSON` to ensure the output is in the correct structure. (Phoenix and Taylor, 2024).

Provide examples

A prompt without examples is called *zero-shot* and its response tends to be an approximation based on the average information available on the internet. This usually is not enough for the model to accurately determine our preferences. Even a *one-shot* prompt, where we provide just one example, can help a lot. In practice, it is common to test the model performance with multiple examples, known as *few-shot* prompting. (Phoenix and Taylor, 2024). Indeed, the model accuracy on certain tasks increased from 10% to nearly 50% with one example, and to over 60% with a few examples (FIGURE 5) (Brown *et al.*, 2020).

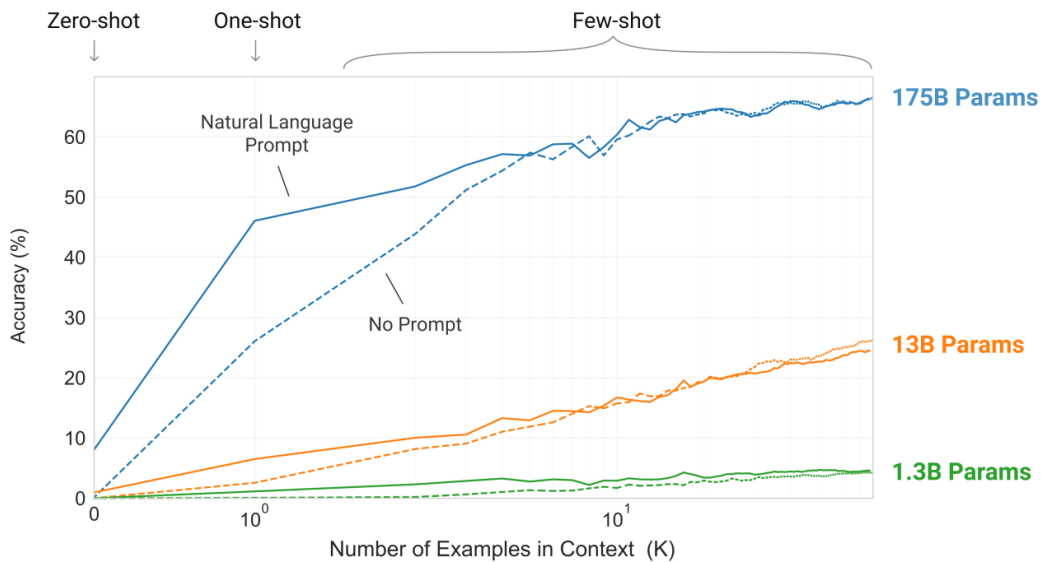


FIGURE 5. Effect of the number of examples to model accuracy in different GPT-3 models. Solid lines mean the model is given an instruction in plain English (or another natural language), dashed lines correspond the models where instructions are given as input-output examples only. Colours indicate the size (number of parameters) of the GPT-3 model. Obtained from (Brown *et al.*, 2020).

Just like when explaining a task to a person, providing examples can sometimes be easier than describing what we exactly want. However, since the amount of text that can fit in a prompt is limited, selecting diverse and instructive examples becomes crucial part of prompt engineering. Furthermore, while adding many examples can improve the reliability of the results, it may also reduce creativity by constraining the model to closely follow the examples. In general, providing 3 to 5 examples is usually sufficient to give good guidance without sacrificing creativity. (Phoenix and Taylor, 2024).

Evaluate quality

Up to this point, we have evaluated the LLM response by basic trial and error of running the prompt and observing the results. This approach is called *blind prompting*, and it can work well for prompts that are used only once or rarely. If a prompt is intended for repeated use or as part of an application, it is important to evaluate its performance and results more systematically. (Phoenix and Taylor, 2024).

A simple way to evaluate model performance is by running the same prompt several times and manually grading the responses. Human evaluation is generally considered the most accurate form of feedback, but it can be time-consuming and costly to rate many samples by hand. In some cases, we can establish reference answers to programmatically rate the results, which scales up the testing and monitoring efforts considerably. By rating the responses, we can update the examples in the prompt with the good responses and continuously make the system smarter over time. (Phoenix and Taylor, 2024)

There are several metrics available for evaluating the LLM performance. Three key metrics include Elo rating, Bilingual Evaluation Understudy (BLEU) and cross entropy (Ray, 2024). Elo rating was originally created for rating chess players, but in the context of LLMs it can be used by comparing two responses side-by-side and choosing the better one. After multiple evaluations, best rated response has the highest score and can be considered the best one. (Restack, 2025). BLEU was originally developed for evaluating machine translation systems. This method compares machine-generated text to human-written reference by calculating overlap of short word sequences between the generated text and the reference. Resulting score ranges from 0 to 1 with higher score indicating higher similarity with the reference. (Ray, 2024). Cross entropy is a measure from information theory that quantifies the difference between two probability distributions. When LLM generates text, it is essentially predicting the next word in a sequence. Cross entropy measures how close those predictions are to the actual next words. Low cross entropy means that models predictions are close to the actual words and the model is performing well. Cross entropy can also serve

as a loss function guiding the model to adjust its parameters to produce higher quality predictions by minimizing cross entropy. (Chi, 2024).

Divide labour

One of the core principles in all engineering is to decompose complex tasks into smaller sub-tasks. The same applies to prompting, breaking the AI tasks into multiple calls that are chained together helps accomplishing complex tasks and provide visibility into what part of the chain is failing. Even with a single prompt, the process can be broken down into a series of iterative steps, as simply as adding `Let's think step by step` to the prompt. This can lead to increase in reasoning and proficiency and provide an audit trail for quality assurance and debugging. In our product name generator, we could add another step to evaluate the generated names to help us decide which are the best one and then generate a description for that.

```
Let's think step by step. Rate this list of product names for a pair of shoes that can fit any foot size. The rating should be out of 10, inline next to the product name:
```

```
UniFit SoleStrides  
FlexiSize All-Fit Shoes  
OmniFit FootFlex  
...  
AnyStep Adaptiwalkers  
FlexiRange UniversalSteps  
PerfectGlide MultiFit Shoes
```

The response could be something like:

```
Here's a rating out of 10 for each product name, considering factors like descriptiveness, catchiness, and overall appeal:
```

1. **UniFit SoleStrides**: 7/10 - Sounds unifying but may not immediately convey the "fit any size" idea.
2. **FlexiSize All-Fit Shoes**: 8.5/10 - Combines flexibility and the notion that it fits all sizes.
3. **OmniFit FootFlex**: 8/10 - Omnipresence (fitting all) and flexibility nicely conveyed.
- ...- 18. **AnyStep Adaptiwalkers**: 7.5/10 - "AnyStep" suggests versatility, but the overall name might be less direct.
- 19. **FlexiRange UniversalSteps**: 8/10 - Combines flexibility and universal fit.
- 20. **PerfectGlide MultiFit Shoes**: 8.5/10 - "MultiFit" is clear, and "PerfectGlide" suggests a comfortable experience.

```
Remember, the best name can also be influenced by your target audience, branding strategy, and the overall message you want to convey.
```

We can then choose the best name and ask LLM to generate a description for it: `Describe the new product "OneFit UltraStride Shoes," a shoe that can fit any foot size, in great detail.` Then, we can even ask for an image of the provided description: `Convert this description into a short prompt for an AI image generation model such as DALL-E.` (Phoenix and Taylor, 2024).

3 MATERIAL AND METHODS

3.1 OpenAI platform and GPT models

OpenAI was chosen for this project because of its state-of-the-art generative understanding of human language, which is essential for accurately handling complex data extraction tasks. Additionally, the extensive documentation and high-quality tutorials available for OpenAI GPT models facilitated the implementation and fine-tuning process, making them well-suited for this project. Other options were not initially investigated, as the topic was new to the developer and the priority was put to rapid prototyping using reliable and well-documented solution. Alternative tools and frameworks can be evaluated and potentially implemented in the future if a more suitable option is found.

Access to the OpenAI platform (<https://platform.openai.com/>) was set up for API requests. The usage of the OpenAI API is billed based on the number of tokens and varies depending on the model used, with newer models being more expensive than older ones (OpenAI, 2025). The pay-as-you-go tier was selected for the current project and a credit of 12.60 EUR (equivalent 10 USD plus 25.5% VAT) was deposited. This ensured that model usage would have automatically stopped if the 10 USD credit limit had been reached, preventing any unexpected or excessive charges. In general, older models (e.g. GPT-3.5) were used initially to test API connection and script functionality. Once these tests were successful, newer models (e.g. GPT-4.1) were used to evaluate if they improved the responses compared to the older models. In some cases, switching to a specific model was necessary because of functionality requirements of certain libraries.

3.2 Python programming language

3.2.1 Reasons for selecting Python

There are many languages suitable for AI programming such as Python, Java, JavaScript, Julia, Haskell and Lisp. In general, Python is most popular in AI and machine learning probably because it has easy-to-learn syntax and a wide range of libraries and frameworks (Upwork, 2024). A search of public repositories on GitHub that mentioned machine learning shows that Python is by far the most popular language, with approximately 11.6 times more repositories than the second most used language, JavaScript (TABLE 7) (Yadoshchuk, 2024).

TABLE 7. Number of public GitHub repositories mentioning machine learning by programming language. Obtained from (Yadoshchuk, 2024).

Language	Public repositories
Python	34,766 times
JavaScript	2,998 times
R	2,105 times
C++	1,926 times
MATLAB	1,881 times
Java	1,783 times
C#	1,783 times

Python has also a strong community support, which is important as it gives developers easy access to documentation, learning materials and help. Additionally, Python is platform-independent, meaning it runs on Linux, Windows and macOS. Python code can be used to create standalone executable programs, allowing applications to be distributed and used without requiring a Python interpreter. (Yadoshchuk, 2024).

3.2.2 Used frameworks and libraries

LangChain was used as the main framework for the current project. LangChain is an open-source framework specifically designed for developing LLM applications. It provides interface for numerous different models and related

technologies and integrates hundreds of providers. (LangChain, 2025c). LangChain consists of several open-source libraries, the main ones used in the current project are presented in the TABLE 8.

TABLE 8. Overview of LangChain libraries used in the project, describing their main functionalities and specific modules implemented.

Library	Description
<code>langchain_openai.chat_models</code>	Provides integration of OpenAI's chat models into LangChain ecosystem. In this project, <code>ChatOpenAI</code> module was used, allowing customisation of parameters such as the model type, temperature, token limit, etc.
<code>langchain.prompts</code>	The <code>ChatPromptTemplate</code> module was used to create a well-structured prompts that translates user input and parameters into instructions for the LLM.
<code>langchain.output_parsers.json</code>	Parses the output of the LLM call to a JSON object for easier data handling and manipulation. In this project, <code>SimpleJsonOutputParser</code> was used for parsing the JSON.
<code>langchain_text_splitters</code>	The <code>CharacterTextSplitter</code> module was used to divide long text (i.e. the scientific articles) into smaller chunks to manage token limits effectively.

3.3 System workflow

3.3.1 Input data selection

This project is part of an EU-funded initiative named STAGE, which investigates healthy ageing and multimorbidity using a life-course approach (<https://stage-healthyageing.eu/>). One of the goals of this initiative is to conduct a systematic review of published literature on biological markers of ageing. A screening tool for identifying articles with a specific research focus was developed in a parallel sub-project.

For the current prototype, only articles available in PDF and accessible to the developers were included as inputs. Future improvements include incorporating

additional sources of articles, such as URLs, DOIs and direct API access to databases like PubMed, Scopus and Web of Science.

3.3.2 Preprocessing

The selected articles were downloaded from their sources as PDF, read using the PyMuPDF library (PyMuPDF, 2025) and the text from all pages was then combined into a single string:

```
doc = pymupdf.open("path/to/article.pdf")
text = "\n".join(page.get_text("text") for page in doc).
```

Before using the text in a prompt, it had to be cut into smaller pieces with the `text_splitter` function shown below to prevent hitting the token limit of the model. This function used `CharacterTextSplitter` function with `tiktoken` tokenizer from `langchain_text_splitters` library (LangChain, 2025b):

```
text_splitter = CharacterTextSplitter.from_tiktoken_encoder(
    encoding_name="cl100k_base", chunk_size=10000, chunk_overlap=5)
```

The `encoding_name` argument specifies the tokenizer used, with `cl100k_base` being compatible with models like GPT-4 and GPT-3.5. The `chunk_size` argument sets the maximum number of tokens allowed in each chunk and `chunk_overlap` defines how many tokens overlap between consecutive chunks. This overlap helps to maintain continuity and context ensuring the text is not split at critical points.

3.3.3 Model integration

OpenAI was integrated to the model by:

```
chat = ChatOpenAI(model="gpt-4.1-2025-04-14",
    model_kwargs={'response_format': {"type": "json_object"}})
```

Several different models were tested and the `gpt-4.1-2025-04-14` was chosen in the end because it has high token limits, and it supports the `json_object` as response format. The `model_kwargs` parameter (short from model keyword arguments) allowed passing additional options directly to the API call, even if a high-level function, such as LangChain's `ChatOpenAI` does not support them. The `response_format` kwarg ensured that the output was returned as a valid JSON object,

rather than just a JSON-like string. This structured format simplified downstream processing of the output.

3.3.4 Defining the information to be extracted

The initial aim of this project was to develop an AI tool to capable of extracting study information and results from articles focusing on biological markers of aging. Information defined to be extracted are listed in TABLE 9.

TABLE 9. Information extracted from the articles.

Category	Field
Article information	<ul style="list-style-type: none"> - First author - All authors - Publication year - Title - Full citation - DOI
Study population information	<ul style="list-style-type: none"> - Cohort name - Mean age - Age SD - Sex distribution - Sample size - Subpopulation?
Information on the method(s) of biological age estimate	<ul style="list-style-type: none"> - Source (DNA methylation, metabolomics, transcriptomics, inflammation or other) - Tissue (blood or other)
Statistical methods and their corresponding results	<ul style="list-style-type: none"> - Statistical method - Biological age estimate name - Biological age estimate reference (citation if published or new if published first time) - Outcome name - Parameter name (effect estimate, odds ratio, hazard ratio or other) - Parameter value - Error name (standard error, confidence interval or other) - Error value - P value

3.3.5 Prompt engineering

Following prompt was developed with the assistance of ChatGPT. It uses LangChain's `ChatPromptTemplate` to create optimal structure for downstream processes and includes an example of the desired JSON output. The curly brackets in the JSON example had to be replaced with double curly brackets to escape them, since single curly brackets are used in the prompt template to indicate input variables, such as the article text in this case (`{input_article}` at the end of the prompt).

```
chat_prompt = ChatPromptTemplate.from_messages([
    ("system", """
    You are a scientific research assistant.
    Extract structured metadata and results from the following scientific article.
    Always provide comprehensive and reproducible information from scientific articles,
    even if the data appears sparse or ambiguously presented.
    If certain elements cannot be found, indicate them as 'NA' instead of omitting them
    entirely.
    Ensure all results are parsed, regardless of the format in the article.
    Ensure that all results mentioned in the article are captured.
    If necessary information (like standard error or subgroup analyses) is missing, mark
    them as 'NA' to maintain a consistent structure.
    Extract the following structured information from a scientific article, summarizing
    key sections clearly and concisely. Output the data in JSON format. Do not add any
    additional fields to the JSON.
    Refer only to the provided article. In cases where the answer isn't found within
    these articles, simply return 'NA'.

    1. Article Information
    - First author
    - Year of publication
    - Title of the article
    - List of all authors
    - Journal name
    - Volume, issue, and page numbers (if available)
    - DOI (if available)
    - Does the article report original biological age estimate or use already published
    one?

    2. Study Design
    - Type of study (e.g., cohort, case-control, cross-sectional).
    - Population details including:
      - Total sample size
      - Definition criteria for cases and controls, including diagnostic or clinical
      criteria.
      - Sample size for cases
      - Sample size for controls
    - Demographic characteristics
      - Geographic location
      - Ethnicity
      - Sample size for males
      - Sample size for females

    3. Statistical Methods
    - Primary statistical method used for analysis.
    - Software used for the analysis (if specified).
    - Assumptions made during the statistical analysis.

    4. Results
    - Provide a separate entry for each unique outcome variable, even if multiple
    outcomes are reported together in the same sentence or table. Do not group
    multiple outcomes into a single result.
    - For example, if the article reports associations of a single exposure with
    multiple outcomes (e.g., Horvath Age, Hannum Age, Grim Age), create separate
    result entries for each outcome, each with its own effect size, p-value, and other
    metadata.
    - If only a range of values (e.g., "0.06 to 0.11 years") is reported for multiple
    outcomes, assign the value "NA" for individual effect sizes but still create
    individual entries for each outcome to preserve result granularity.
    - Do not output result entries if Effect size (effect_size field) is "NA".
    - Each entry should correspond to one exposure-outcome pair and contain following:
      - Unique result ID
```

- Name of the exposure variable.
- Method of measurement for the exposure.
- Units used for the exposure variable (if applicable).
- Name of the outcome variable.
- Method of measurement for the outcome.
- Units used for the outcome variable (if applicable).
- List of covariates used in the analysis.
- Justification for their inclusion.
- Effect size (including regression coefficients or odds ratios)
- Confidence interval (lower and upper bound)
- Standard error
- P-value
- Test statistics (if available)
- Subgroup analyses (if applicable)
- Notes (e.g., limitations or observations on result robustness)

5. Conclusion Summary

- A brief summary of the conclusions.
- Implications of the findings (including potential practical or research impacts).

Output Example:

Ensure the response follows this format:

```

{{
  "article_info": {{
    "first_author": "Smith J.",
    "year": 2022,
    "title": "Association between Air Pollution and Cardiovascular Disease: A
    Longitudinal Study",
    "authors": ["Smith J.", "Doe A.", "Brown L.", "Chen Y.", "Patel K."],
    "journal": "Journal of Environmental Health",
    "volume": "58",
    "issue": "4",
    "pages": "123-135",
    "doi": "10.1234/jeh.2022.00456",
    "is_original_method": "yes",
  }},
  "study_design": {{
    "type": "Cohort",
    "population": {{
      "total_sample_size": 5000,
      "case_control_criteria": "Diagnosed with cardiovascular disease based on ICD-
      10 codes I20-I25",
      "control_definition": "Participants without any cardiovascular diagnosis",
      "cases_sample_size": 1000,
      "controls_sample_size": 4000
    }},
    "demographics": {{
      "geographic_location": "Finland",
      "ethnicity": "Predominantly Caucasian",
      "males_sample_size": 2400,
      "females_sample_size": 2600
    }},
    "cohorts": [
      {{
        "name": "NCPTSD",
        "sample_size": 647,
        "percent_male": 62.9,
        "mean_age": 51.85,
        "sd_age": 10.6
      }},
      {{
        "name": "TRACTS",
        "sample_size": 434,
        "percent_male": 90.3,
        "mean_age": 32.44,
        "sd_age": 8.63
      }}
    ]
  }},
  "statistical_methods": {{
    "primary_method": "Logistic regression",
    "software": "R (version 4.2.0)",
    "assumptions": "Linearity of log odds, No multicollinearity among covariates,
    Independence of observations"
  }},
  "results": [
    {{
      "result_id": 1,
      "sample_size": 5000,
      "subgroup": "No",
      "exposure_variable": "PM2.5 concentration",
      "exposure_measurement_method": "Air quality monitoring stations",
      "exposure_units": "ug/m³",
      "outcome_variable": "Cardiovascular disease incidence",
      "outcome_measurement_method": "Hospital discharge records",
      "outcome_units": "NA",
    }}
  ]
}}

```

```

    "covariates": "Age, Sex, Smoking status, BMI, Socioeconomic status",
    "covariate_justification": "Known confounders of cardiovascular disease risk",
    "effect_size": 1.45,
    "confidence_interval": "1.20, 1.75",
    "se": 0.14,
    "p_value": 0.002,
    "test_statistics": "Z = 3.12",
    "notes": "Effect remained significant after adjusting for all covariates"
  }},
  {{
    "result_id": 2,
    "sample_size": 2600,
    "subgroup": "Females only",
    "exposure_variable": "NO2 concentration",
    "exposure_measurement_method": "Satellite remote sensing",
    "exposure_units": "ppb",
    "outcome_variable": "Hypertension diagnosis",
    "outcome_measurement_method": "Clinical blood pressure measurements",
    "outcome_units": "mmHg",
    "covariates": "Age, Sex, Smoking status, BMI, Socioeconomic status",
    "covariate_justification": "Important lifestyle-related confounders in hypertension",
    "effect_size": 1.20,
    "confidence_interval": "1.05, 1.38",
    "se": 0.08,
    "p_value": 0.015,
    "test_statistics": "Z = 2.43",
    "notes": "Association stronger among elderly participants; borderline significant in females"
  }}
],
"conclusion_summary": {{
  "summary": "Exposure to both PM2.5 and NO2 was associated with increased risk of cardiovascular-related outcomes.",
  "implications": "These results support policies aiming to reduce air pollutant exposure as a strategy to lower cardiovascular and hypertension burden in the population."
}}
}}
{input_article}
""")
])

```

3.3.6 Data extraction

Chain for invocation

LangChain enables easy chaining of tasks (referred to as runnables in LangChain) where the output of previous task is automatically passed as input to the next (LangChain, 2025a). This process is known as chain for invocation. In this project, following chain was used:

```
chain = chat_prompt | chat | SimpleJsonOutputParser()
```

It first calls the `chat_prompt` defined in Chapter 3.3.5 (p. 28), then integrates the `chat` model as described in Chapter 3.3.3 (p. 26), and finally parses the output with LangChain's JSON parser. The chain for invocation was then invoked separately for each text chunks containing parts of the original article (see 3.3.2, p. 26).

```

for chunk_number, chunk in enumerate(chunks, start=1):
    print(f"Processing {article} - Chunk {chunk_number}/{total_chunks}")
    # Invoke the pipeline with article chunk input

```

```
results = chain.invoke({"input_article": chunk})
```

Extracting the data

One article can contain multiple results for different exposures, outcomes and models, all of which the tool should be able to collect. However, since the text is split into chunks and the same information may appear multiple chunks, a function `merge_new_fields` was implemented to check that newly extracted data is not already captured. It uses `frozenset()` to convert the extracted data into an immutable and hashable form. This allows the data to be uniquely identified and used as a key in sets or dictionaries. As a result, the function can check and remove the duplicates.

```
def merge_new_fields(existing_data, new_data):
    # Merges only new fields from new_data into existing_data.
    if not existing_data: # If no previous data, return new_data as-is
        return new_data

    updated_data = existing_data.copy()

    for key, value in new_data.items():

        if key not in existing_data:
            updated_data[key] = value # Add new key-value pairs

        elif isinstance(value, list) and isinstance(existing_data[key], list):
            # If lists contain dictionaries, merge them uniquely
            if all(isinstance(item, dict) for item in value):
                existing_dicts = {frozenset(d.items()): d for d in existing_data[key]}
                for item in value:
                    frozen_item = frozenset(item.items())
                    if frozen_item not in existing_dicts:
                        existing_dicts[frozen_item] = item
                updated_data[key] = list(existing_dicts.values()) # Convert back to list
            else:
                updated_data[key] = list(set(existing_data[key] + value)) # Merge uniques

        elif isinstance(value, dict) and isinstance(existing_data[key], dict):
            # Recursively merge dictionaries
            updated_data[key] = merge_new_fields(existing_data[key], value)

    return updated_data
```

The `merge_new_fields` function was then used to combine the results from each chunk into complete article-level results. These were then saved to the `combined_results` JSON, with each article stored under its own key.

```
# Ensure the results are not empty
if results and results != {} and results != "null":

    # Merge only new fields into article_results
    article_results = merge_new_fields(article_results, results)

    # Store the combined results for the article
    if article_results: # Only store if there's meaningful data
        combined_results[article] = article_results
```

3.3.7 Output formatting

The output format was initially set to JSON to facilitate easier downstream processing. However, since JSON may not be the most user-friendly format for reading, it was converted to CSV after extraction and saved. The below `flatten_dict` function was created for the conversion and save.

```
import json
import pandas as pd

def flatten_dict(d, parent_key='', sep='.'):
    items = []
    for k, v in d.items():
        new_key = f"{parent_key}{sep}{k}" if parent_key else k
        if isinstance(v, dict):
            items.extend(flatten_dict(v, new_key, sep=sep).items())
        else:
            items.append((new_key, v))
    return dict(items)

rows = []

for article_filename, article_data in combined_results.items():
    # Extract article-level metadata (excluding 'results')
    metadata = {k: v for k, v in article_data.items() if k != 'results'}
    flat_meta = flatten_dict(metadata)
    flat_meta['article_file'] = article_filename

    # Loop through results and combine with metadata
    for result in article_data.get('results', []):
        flat_result = flatten_dict(result)
        row = {**flat_meta, **flat_result}
        rows.append(row)

# Convert to DataFrame
df = pd.DataFrame(rows)

# Save to CSV
df.to_csv("data_extraction_results.csv", sep=";", index=False)
print("Data saved as rows per result.")
```

3.4 Testing and validation

To evaluate the precision and accuracy of the tool, results from 8 scientific articles were extracted both manually and using the tool. The outputs were then compared side by side and standard evaluation metrics – precision, recall and F1-score – were computed using the `sklearn` library .

```
import json
from sklearn.metrics import precision_score, recall_score, f1_score

def binary_match(pred, gold):
    return int(pred == gold)

def evaluate_dicts(pred_dict, gold_dict):
    common_keys = set(pred_dict) & set(gold_dict)
    y_true = [gold_dict[k] for k in common_keys]
    y_pred = [pred_dict[k] for k in common_keys]

    binary_true = [1] * len(y_true)
    binary_pred = [binary_match(p, g) for p, g in zip(y_pred, y_true)]

    precision = precision_score(binary_true, binary_pred)
    recall = recall_score(binary_true, binary_pred)
```

```

    f1 = f1_score(binary_true, binary_pred)
    return precision, recall, f1
# Load JSONs
with open("res_gold.json") as f:
    gold = json.load(f)
with open("res_llm.json") as f:
    pred = json.load(f)
# Run evaluation
precision, recall, f1 = evaluate_dicts(pred, gold)
print(f"Precision={precision:.2f}, Recall={recall:.2f}, F1={f1:.2f}")

```

Precision (P) is defined as the number of true positives (Tp) divided by the number of true positives plus number of false positives (Fp):

$$P = \frac{Tp}{Tp + Fp}$$

Recall (R) is defined as number of Tp divided by the number of Tp plus the number of false negatives (Fn):

$$R = \frac{Tp}{Tp + Fn}$$

F1 score (F1) is the harmonic mean of the precision and recall and is calculated using the function:

$$F1 = \frac{2 * Tp}{2 * Tp + Fp + Fn}$$

4 RESULTS

4.1 Pre-processing the input and the prompt

4.1.1 Text conversion and chunking

Articles were read using PyMuPDF library (PyMuPDF, 2025), which creates a document object with each page in separate page objects. For LLM, the content from all pages was combined into a single string. Since the combined text along with the prompt could exceed the model's token limits, the text was split into manageable chunks using LangChain's `CharacterTextSplitter.from_tiktoken_encoder`. This method also tokenized the input with the tokenizer optimised for the GPT-model in use.

4.1.2 Prompt engineering

The model was initially tested using a straightforward prompt:

```
I want you to extract the biological age measure name, name of the outcome or exposure it is associated with, result, confidence interval lower limit, confidence interval upper limit, of standard error, p value and statistical method from the following text.

Here is the JSON schema:
  "biol_age_measure": string
  "outcome_or_exposure": string
  "result": float
  "ci_lower": float
  "ci_upper": float
  "se": float
  "p_value": float
  "stat_method": string

{article_to_extract}

---
If there are multiple measures, then put them in a 'measures' key, which is a list of the above schema.
```

This approach worked well for a single articles with relatively simple analysis plan. However, when processing multiple articles, it became necessary to also extract article metadata and details about the statistical analysis models used. To address this, a more detailed prompt was crafted with the assistance of ChatGPT. The revised prompt as presented in the Material and Methods section (p. 28),

clearly specified the required information and included examples of the desired JSON output format.

4.1.3 Technical and practical challenges

Originally, the tool was intended to be able to process multiple different article formats. Structured formats such as HTML or XML, commonly used in the online versions of the articles, would likely have been easier to prepare for extraction. However, not all articles were available in these formats, and in some cases, access required user authentication. As a result, PDF was chosen as primary input format for this stage. Support for other formats can be added later to enhance accessibility.

One of the main challenges at this point was the token limits. Both the prompt and the article text were already quite lengthy, which made token limits a recurring issue. This was evident with the earlier models used in the beginning. Initially, a GPT-3 model gpt-3.5-turbo-1106 was used, but its total token limit of 16,384 and output limit of 4,096 tokens meant chunk sizes had to be reduced. During the project, a general-purpose GPT-4 model gpt-4.1-2025-04-14 was released, offering token limits of 200,000 for total and 100,000 for output tokens, which effectively solved the token constraint problems.

4.2 AI tool performance evaluation

Information was extracted using the same prompt with both the gpt-3.5-turbo-1106 and gpt-4.1-2025-04-14 models. The chunk size was set to 3,000 for GPT-3.5 and increased to 10,000 tokens for GPT-4.1.

4.2.1 Extraction of the article metadata

The article metadata included details such as the authors, publication year, title, reference, DOI and whether the article presented an original method for

estimating biological age. Model output JSONs of one article (Chen *et al.*, 2024) are compared in FIGURE 6.

GPT-3.5	GPT-4.1
<pre> "chen-2024.pdf": { "article_info": { "first_author": "Chen J.", "year": 2024, "title": "Associations of childhood, adolescence, and midlife cognitive function with DNA methylation age acceleration in midlife", "authors": ["Cirrillo P.", "Prada-Ortega D", "Suglia SF", "Simpson E.", "Factor-Litvak P.", "Clausing E. S.", "Cirillo P", "Baccarelli A.", "O'Shea DM", "Suglia S. F.", "Chen J.", "Harding A.", "Conneely K", "Moubadder L.", "Salford S.", "Clausing ES", "Levine K.", "Link BG", "Coleman K.", "Bennet-King L.", "Conneely K. N.", "Kezios K. L.", "Hüls A.", "Factor-Litvak P", "DeVivo I", "Shelton RC", "DunedinPACE", "Maytland L.", "Campbell S.", "Shaw B.", "Larsson K.", "Cohn B", "Link B. G.", "Cohen J.", "Maynard T", "Shelton R. C.", "Clark P.", "Stevens R.", "Baccarelli AA", "Blackwell G.", "Lee P.", "Tremont G."], "journal": "AGING", "volume": "16", "issue": "11", "pages": "9350-9352", "doi": "NA", "is_original_method": "NA" }, } </pre>	<pre> "chen-2024.pdf": { "article_info": { "first_author": "Chen J.", "year": 2024, "title": "Associations of childhood, adolescence, and midlife cognitive function with DNA methylation age acceleration in midlife", "authors": ["Junyu Chen", "Leah Moubadder", "Elizabeth S. Clausing", "Katrina L. Kezios", "Karen N. Conneely", "Anke Hüls", "Andrea Baccarelli", "Pam Factor-Litvak", "Piera Cirrillo", "Rachel C. Shelton", "Bruce G. Link", "Shakira F. Suglia"], "journal": "Aging", "volume": "16", "issue": "Advance (No. 11 as reference for p9350-9358)", "pages": "9350-9358", "doi": "NA", "is_original_method": "no" }, } </pre>

FIGURE 6. Comparison of article metadata JSON extracted by GPT-3.5 and GPT-4.1 models.

Both models extracted the first author, year, title and reference correctly. However, the author lists differed notably between the two models. GPT-4.1 gave the correct list of authors. While all the correct authors were also present also in GPT-3.5 output, some names appeared multiple times, and a few completely unrelated names were included. One name (DunedinPACE) was the name of one of a biological age estimate used in the study, indicating that the extraction had failed to properly distinguish between author names and other entities.

Both models failed to extract the DOI for this article, although DOIs were successfully extracted from other articles. This suggests that the DOI may not have been included in every articles' PDF version. Out of the eight articles, GPT-4.1 correctly identified two as reporting original method for estimating biological age. In contrast, GPT-3.5 labelled six articles as presenting original methods and marked two as NA.

4.2.2 Extraction of the study design and statistical methods

Study design field included the information about type of the study (e.g. cross-sectional, longitudinal or case control), population sample sizes, demographic information (geographic location, ethnicity, number of males and females) and cohorts. Statistical methods contained primary statistical method, software and assumptions related to the selected method. FIGURE 7 shows an example of information extracted from one article (Hawn *et al.*, 2022). The study type was set to "cohort" in GPT-3.5 and "cross-sectional" in GPT-4.1 output. Both models identified case-control criteria, which indicates that the study type could also be case-control. A single study can be classified simultaneously as a case-control, cohort and cross-sectional study. Used prompt had only one field for this, so if more detailed information is to be extracted, it should be indicated more clearly in the prompt. GPT-3.5 was unable to extract the total sample size and incorrectly identified the sample sizes of two cohorts as representing cases and controls. GPT-4.1 extracted the total sample size correctly. Both models successfully extracted the geographic location, but only GPT-4.1 was able to accurately extract information on ethnicity and the number of males and females. Cohorts' information was extracted correctly by both models; however, they appeared

multiple times, indicating that a validation step should be added to the downstream JSON processing to prevent duplicate entries.

GPT-3.5	GPT-4.1
<pre> "study_design": { "type": "Cohort", "population": { "total_sample_size": "NA", "case_control_criteria": "Trauma- exposed military cohorts", "control_definition": "NA", "cases_sample_size": 647, "controls_sample_size": 434 }, "demographics": { "geographic_location": "USA", "ethnicity": "NA", "males_sample_size": 62.9, "females_sample_size": "NA" }, "cohorts": [{ "name": "Cohort 1", "sample_size": 647, "percent_male": 62.9, "mean_age": 52, "sd_age": "NA" }, { "name": "Cohort 2", "sample_size": 434, "percent_male": 90, "mean_age": 32, "sd_age": "NA" }, { "name": "NCPTSD", "sample_size": 647, "percent_male": 62.9, "mean_age": 52, "sd_age": 10.6 }, { "name": "TRACTS", "sample_size": 434, "percent_male": 90, "mean_age": 32.4, "sd_age": 8.6 }, { "name": "NCPTSD", "sample_size": 647, "percent_male": 62.9, "mean_age": 51.85, "sd_age": 10.6 }, { "name": "TRACTS", "sample_size": 434, "percent_male": 90.3, "mean_age": 32.44, "sd_age": 8.63 }] }, "statistical_methods": { "primary_method": "Linear regression models", "software": "NA", "assumptions": "NA" } </pre>	<pre> "study_design": { "type": "Cross-sectional (two independent trauma-exposed military cohorts)", "population": { "total_sample_size": 1081, "case_control_criteria": "Trauma- exposed veterans (and a subset of spouses in NCPTSD); PTSD and other psychiatric diagnosis made by CAPS and SCID-IV.", "control_definition": "NA", "cases_sample_size": "NA", "controls_sample_size": "NA" }, "demographics": { "geographic_location": "Boston, MA, USA", "ethnicity": "Majority White (NCPTSD: 81.1%, TRACTS: 74.2%), with Black/African American, Asian, American Indian/Alaska Native, and other groups also represented.", "males_sample_size": 799, "females_sample_size": 282 }, "cohorts": [{ "name": "NCPTSD", "sample_size": 647, "percent_male": 62.9, "mean_age": 51.85, "sd_age": 10.6 }, { "name": "TRACTS", "sample_size": 434, "percent_male": 90.3, "mean_age": 32.44, "sd_age": 8.63 }, { "name": "NCPTSD", "sample_size": "NA", "percent_male": "primarily male", "mean_age": "NA", "sd_age": "NA" }, { "name": "TRACTS", "sample_size": "NA", "percent_male": "primarily male", "mean_age": "NA", "sd_age": "NA" }] }, "statistical_methods": { "primary_method": "Linear regression (hierarchical and multiple regression), correlation analysis, confirmatory factor analysis", "software": "NA", "assumptions": "Linear relationships, inclusion of ancestry principal components and estimated blood cell type proportions to control for technical confounding, FDR correction for multiple testing, Monte Carlo permutation for imaging multiple comparison correction." } </pre>

FIGURE 7. Comparison of study design and statistical methods JSON extracted by GPT-3.5 and GPT-4.1 models.

4.2.3 Extraction of the results

One article can have multiple results that differ by the exposure, outcome, model or sample size. The prompt was crafted so that it can extract as many results as it identifies from the text and a validation step was added to check if the result already is in the list or contains the required information. Extracted information for results contained result ID, sample size, whether it is a subgroup analysis, exposure name, measurement method and unit, outcome name, measurement, method and unit, covariates, justification of covariates, effect size (this is the result), confidence interval (if reported), standard error (SE, if reported), P value, test statistics and notes. Quantitative analysis and comparison to human-extracted data of all extracted results is described later in chapter 4.3 (p. 41).

Two selected results entries from (Hawn *et al.*, 2022) are illustrated in FIGURE 8. The same results are shown side by side for easier comparison, although the order of entries differed between the models. GPT-3.5 extracted 13 distinct results, while GPT-4.1 extracted 28. The result ID was not unique in either of the model outputs even though this was instructed in the prompt. GPT-3.5 included sample size for only 1 out of 13 results while GPT-4.1 provided sample sizes for 18 out of 28. GPT-3.5 did not extract information on the exposure measurement method or covariates, while GPT-4.1 captured both correctly. Outcome variables were identified accurately by both models. GPT-3.5 failed to extract SEs, which – along with confidence intervals – are essential for any subsequent meta-analysis. This limitation would need to be addressed if GPT-3.5 were to be used. Additionally, some entries in the GPT-4.1 output included unclear references such as “see above”, making it difficult to interpret the full context.

GPT-3.5	GPT-4.1
<pre> "results": [{ "result_id": 1, "sample_size": "NA", "subgroup": "NA", "exposure_variable": "Externalizing Factor Scores", "exposure_measurement_method": "NA", "exposure_units": "NA", "outcome_variable": "GrimAge Residuals", "outcome_measurement_method": "NA", "outcome_units": "NA", "covariates": "NA", "covariate_justification": "NA", "effect_size": 0.35, "confidence_interval": "NA", "se": "NA", "p_value": "< 0.001", "test_statistics": "NA", "notes": "Follow-up analyses examining each component of the externalizing spectrum reveal SUD and ASPD predicted GrimAge residuals" }, </pre>	<pre> "results": [{ "result_id": 1, "sample_size": 647, "subgroup": "NCPTSD full sample", "exposure_variable": "Externalizing factor score", "exposure_measurement_method": "SCID-based dimensional score (confirmatory factor analysis)", "exposure_units": "factor score (unitless)", "outcome_variable": "GrimAge residuals", "outcome_measurement_method": "DNA methylation-based GrimAge minus regression on chronological age", "outcome_units": "residual (unitless, higher is higher mortality risk)", "covariates": "Top 3 ancestry principal components, sex, estimated CD8-T cells, CD4-T cells, NK cells, B-cells, monocytes", "covariate_justification": "Control for population stratification, technical confounding, and cellular heterogeneity", "effect_size": 0.35, "confidence_interval": "NA", "se": 0.044, "p_value": "<0.001", "test_statistics": "standardized β = 0.35, unstandardized β = 0.351", "notes": "Effect remained significant in sensitivity analyses" }, </pre>
<pre> { "result_id": 2, "sample_size": "NA", "subgroup": "NA", "exposure_variable": "Lifetime Non- Alcohol SUD", "exposure_measurement_method": "NA", "exposure_units": "NA", "outcome_variable": "GrimAge Residuals", "outcome_measurement_method": "NA", "outcome_units": "NA", "covariates": "NA", "covariate_justification": "NA", "effect_size": 0.75, "confidence_interval": "NA", "se": "NA", "p_value": "0.028", "test_statistics": "NA", "notes": "Follow-up analyses confirm the association with SUD" }], </pre>	<pre> { "result_id": 4, "sample_size": 434, "subgroup": "TRACTS full sample", "exposure_variable": "Lifetime non- alcohol use substance use disorder (SUD)", "exposure_measurement_method": "SCID- based diagnosis", "exposure_units": "diagnosis presence", "outcome_variable": "GrimAge residuals", "outcome_measurement_method": "DNA methylation-based GrimAge minus chronological age", "outcome_units": "residual (unitless)", "covariates": "Top 3 ancestry principal components, sex, estimated blood cell types (see above)", "covariate_justification": "see above", "effect_size": 0.1, "confidence_interval": "NA", "se": "NA", "p_value": 0.028, "test_statistics": "std β = 0.10, unstandardized β = 0.75", "notes": "Replication of NCPTSD externalizing effect" }], </pre>

FIGURE 8. Comparison of two results entries from JSON extracted by GPT-3.5 and GPT-4.1 models.

4.3 Comparison with manual (human) extraction

Precision, recall and F1 scores were calculated using `sklearn` library. Three datasets were used: 1) Common included results identified by both GPT-4.1 and the gold standard (human-extracted results), 2) Gold only included results found only in the gold standard, and 3) GPT-4.1 only covered results extracted exclusively by GPT-4.1. Performance metrics are shown in TABLE 10.

TABLE 10. Performance metrics for result extraction across different datasets. “Common” refers to results identified by both GPT-4.1 and the gold standard, “Gold only” includes results found only in the gold standard, and “GPT-4.1 only” covers results extracted exclusively by GPT-4.1. The “N# of results” column indicates the number of results in each dataset.

Dataset	Recall	F1	N# of results
Common	0.66	0.79	41
Gold only	0.20	0.34	122
GPT-4.1 only	0.25	0.40	101

5 DISCUSSION

The aim of this project was to get familiar with AI programming by developing a tool that utilises large language models to extract structured information from scientific articles. While scientific articles follow a relatively consistent structure, significant variability exists across journals and academic disciplines. Differences in discipline-specific terminology and formatting conventions may introduce challenges that complicate the extraction process. Large language models such as GPTs used in this project provide powerful capabilities in understanding and interpreting natural language without the need for large task-specific training datasets. These models demonstrate strong contextual understanding and are particularly effective at handling tasks such as data extraction. Several techniques were implemented during the project, such as chunking articles into manageable pieces, prompt engineering and experimenting with structured output formats.

The tool was able to reliably extract the metadata (e.g. authors, title, reference) and the general information about the study design and population. However, the extraction of results presented significant challenges. While the model successfully retrieved several results that matched manually extracted content, it also introduced a substantial number of irrelevant results. This was shown in the recall and F1 scores of the models. The low recall scores indicate that the model missed large proportion of relevant results. Even the highest recall of 0.66 suggest that one-third of the relevant results were missed. This is critical in scientific data extraction tasks and must be accounted for when developing the tool further.

Chunking of the data was based on the token count, allowing the model to process the entire article within input limits. However, it might have been more effective to divide the article according to the sections and extract the information from the section in which it most commonly appears. For instance, article metadata, study details and methodological information could have been extracted from the introduction and materials and methods sections, whereas

results could have been extracted from the results section only. This section-based approach was not used initially, because the goal was to ensure that all potentially relevant information would be accessible. In hindsight, however, this strategy likely introduced some of the irrelevant results, particularly because the introduction and discussion sections often summarise findings from previous publications. The model was likely unable to distinguish between newly reported and previously published results. In addition to section-based chunking, the model could be fine-tuned by providing examples that clearly define what should and should not be extracted from each section.

OpenAI's GPT-3.5 and GPT-4.1 models were utilised in this project with the newer GPT-4.1 demonstrating better performance in almost all tasks. This highlights the rapid development of these models and suggests that ongoing improvements may help address some of the current limitations. OpenAI was selected primarily due to its extensive documentation and wide range of tutorials; however, other options should also be tested in future work. While testing and running the models during this project was relatively inexpensive, processing a large number of articles may cause significant costs, and this should be considered in any future scaling.

6 CONCLUSIONS

In conclusion, this project demonstrated both the potential and limitations of using large language models for extracting structured information from scientific articles. The tool handled certain types of information successfully; however, particularly result extraction remains a complex challenge requiring further refinement. More targeted extraction strategies, such as section-specific chunking and fine-tuning should be considered in the future development. There is strong potential for developing highly effective and scalable tools for scientific information extraction. Manual extraction is an extremely time-consuming and labour-intensive, especially when dealing with large numbers of articles, highlighting the value of automating this task.

REFERENCES

Ahmed, S. (2024) *LLM Prompt Engineering for Beginners: What It Is and How to Get Started*, *The Deep Hub*, *Medium*. Available at: <https://medium.com/thedeephub/llm-prompt-engineering-for-beginners-what-it-is-and-how-to-get-started-0c1b483d5d4f> (Accessed: 22 April 2025).

Alammar, J. and Grootendorst, M. (2024) *Hands-On Large Language Models*, <https://learning.oreilly.com/library/view/hands-on-large-language/9781098150952/>. O'Reilly Media.

Anderson, D. (2024) *A short journey through LLM prompting*, *MPB Tech*, *Medium*. Available at: <https://medium.com/mpb-tech/a-short-journey-through-llm-prompting-b6ce9dd8c8b3> (Accessed: 22 April 2025).

Bajaj, G. (2025) *How Context Window in LLMs Refers Both Input and Output Tokens*, *Medium*. Available at: <https://medium.com/@ganeshrbajaj/how-context-window-in-llms-refers-both-input-and-output-tokens-0bda7b830784> (Accessed: 31 May 2025).

Battle, R. and Gollapudi, T. (2024) 'The Unreasonable Effectiveness of Eccentric Automatic Prompts', *arXiv* [Preprint], (2402.10949). Available at: <https://arxiv.org/abs/2402.10949v2> (Accessed: 23 April 2025).

Brown, T.B. *et al.* (2020) 'Language models are few-shot learners', *Advances in Neural Information Processing Systems* [Preprint], (2020.10447). Available at: <https://arxiv.org/pdf/2005.14165> (Accessed: 25 April 2025).

Chen, J. *et al.* (2024) 'Associations of childhood, adolescence, and midlife cognitive function with DNA methylation age acceleration in midlife', *Aging*, 16(11), pp. 9350–9368. Available at: <https://doi.org/10.18632/aging.205943>.

Chi, C. (2024) *Cross Entropy in Large Language Models (LLMs), AI: Assimilating Intelligence*, *Medium*. Available at: <https://medium.com/ai-assimilating-intelligence/cross-entropy-in-large-language-models-llms-4f1c842b5fca> (Accessed: 25 April 2025).

Duranton, S. (2023) *Prompting Progress: We'll All Speak AI In The Future Workforce*, *Forbes*. Available at: https://www.forbes.com/sites/sylvainduranton/2023/06/15/prompting-progress-well-all-speak-ai-in-the-future-workforce/?utm_source=chatgpt.com (Accessed: 22 April 2025).

Ferrer, J. (2024) *How Transformers Work: A Detailed Exploration of Transformer Architecture*, *DataCamp*. Available at: <https://www.datacamp.com/tutorial/how-transformers-work> (Accessed: 18 May 2025).

Gopalakrishnan, S. and Ganeshkumar, P. (2013) 'Systematic Reviews and Meta-analysis: Understanding the Best Evidence in Primary Healthcare', *Journal of Family Medicine and Primary Care*, 2(1), p. 9. Available at: <https://doi.org/10.4103/2249-4863.109934>.

Hawn, S.E. *et al.* (2022) 'For whom the bell tolls: psychopathological and neurobiological correlates of a DNA methylation index of time-to-death', *Translational Psychiatry*, 12(1). Available at: <https://doi.org/10.1038/s41398-022-02164-w>.

He, R., Cao, J. and Tan, T. (2025) 'Generative artificial intelligence: a historical perspective', *National Science Review*, 12(5). Available at: <https://doi.org/10.1093/nsr/nwaf050>.

Heßler, N., Rottmann, M. and Ziegler, A. (2020) 'Empirical analysis of the text structure of original research articles in medical journals', *PLOS ONE*, 15(10), p. e0240288. Available at: <https://doi.org/10.1371/JOURNAL.PONE.0240288>.

Hong, Z. *et al.* (2021) 'Challenges and Advances in Information Extraction from Scientific Literature: a Review', *JOM*, 73(11), pp. 3383–3400. Available at: <https://doi.org/10.1007/S11837-021-04902-9/FIGURES/6>.

LangChain (2025a) *How to chain runnables*, *LangChain*. Available at: https://python.langchain.com/docs/how_to/sequence/ (Accessed: 29 May 2025).

LangChain (2025b) *How to split text by tokens*, LangChain. Available at: https://python.langchain.com/docs/how_to/split_by_token/ (Accessed: 25 May 2025).

LangChain (2025c) *Introduction*, LangChain. Available at: <https://python.langchain.com/docs/introduction/> (Accessed: 24 May 2025).

Lasserson, T.J., Thomas, J. and Higgins, J.P. (2024) *Chapter 1: Starting a review*. In: *Cochrane Handbook for Systematic Reviews of Interventions version 6.5*. Edited by Higgins JPT et al. Cochrane. Available at: <https://training.cochrane.org/handbook>.

Li, T., Higgins, J.P. and Deeks, J.J. (2024) *Chapter 5: Collecting data*. In: *Cochrane Handbook for Systematic Reviews of Interventions version 6.5*. Edited by Higgins JPT et al. Cochrane. Available at: <https://training.cochrane.org/handbook>.

Marr, B. (2023) *The Difference Between Generative AI And Traditional AI: An Easy Explanation For Anyone*, Forbes. Available at: <https://www.forbes.com/sites/bernardmarr/2023/07/24/the-difference-between-generative-ai-and-traditional-ai-an-easy-explanation-for-anyone/> (Accessed: 26 April 2025).

Metzger, S. (2022) *A Beginner's Guide to Tokens, Vectors, and Embeddings in NLP*, Medium. Available at: <https://medium.com/@saschametzger/what-are-tokens-vectors-and-embeddings-how-do-you-create-them-e2a3e698e037> (Accessed: 27 April 2025).

Nexla (2025) *Prompt Engineering vs. Fine-Tuning—Key Considerations and Best Practices*, Nexla. Available at: <https://nexla.com/ai-infrastructure/prompt-engineering-vs-fine-tuning/> (Accessed: 31 May 2025).

OpenAI (2025) *Overview - OpenAI API*, OpenAI. Available at: <https://platform.openai.com/docs/overview> (Accessed: 24 May 2025).

Phoenix, J. and Taylor, M. (2024) *Prompt Engineering For Generative AI*. 1st edn. O'Reilly Media.

Polak, M.P. and Morgan, D. (2024) 'Extracting accurate materials data from research papers with conversational language models and prompt engineering', *Nature Communications* 2024 15:1, 15(1), pp. 1–11. Available at: <https://doi.org/10.1038/s41467-024-45914-8>.

PyMuPDF (2025) *Welcome to PyMuPDF*, <https://pymupdf.readthedocs.io/en/latest/>. Available at: <https://pymupdf.readthedocs.io/> (Accessed: 25 May 2025).

Raschka, S. (2024) *Build a large language model (from scratch)*. Manning Publications. Available at: <https://www.oreilly.com/library/view/build-a-large/9781633437166/> (Accessed: 26 April 2025).

Ray, J. (2024) *Metrics for evaluating LLMs, Better ML, Medium*. Available at: <https://medium.com/better-ml/metrics-for-evaluating-llms-2c9d86acdbf6> (Accessed: 25 April 2025).

Restack (2025) *Llm Elo Rating Insights*. Available at: <https://www.restack.io/p/llm-evaluation-answer-elo-rating-cat-ai> (Accessed: 25 April 2025).

Schmidt, C.W. et al. (2024) 'Tokenization Is More Than Compression', *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 678–702. Available at: <https://doi.org/10.18653/v1/2024.emnlp-main.40>.

Srivastava, P. (2023) *Fine-Tuning AI Models: A Guide, Medium*. Available at: <https://medium.com/@prabhuss73/fine-tuning-ai-models-a-guide-c515bcd4b580> (Accessed: 31 May 2025).

SYSTRAN (2025) *Translation tool for professionals*. Available at: <https://www.systransoft.com/> (Accessed: 26 April 2025).

Taylor, K.S., Mahtani, K.R. and Aronson, J.K. (2021) 'Summarising good practice guidelines for data extraction for systematic reviews and meta-analysis', *BMJ Evidence-Based Medicine*, 26(3), pp. 88–90. Available at: <https://doi.org/10.1136/BMJEBM-2020-111651>.

Taylor, M. and Phoenix, J. (2025) *The Complete Prompt Engineering for AI Bootcamp*, Udemy. Available at: <https://www.udemy.com/course/prompt-engineering-for-ai/> (Accessed: 25 April 2025).

Uman, L.S. (2011) 'Systematic Reviews and Meta-Analyses', *Journal of the Canadian Academy of Child and Adolescent Psychiatry*, 20(1), p. 57. Available at: <https://doi.org/10.1016/j.revmed.2014.05.011>.

Upwork (2024) *10 Best AI Programming Languages To Know in 2025*, Upwork. Available at: <https://www.upwork.com/resources/best-ai-programming-language> (Accessed: 24 May 2025).

Vaswani, A. *et al.* (2017) 'Attention Is All You Need', *arXiv* [Preprint]. Available at: <https://doi.org/https://doi.org/10.48550/arXiv.1706.03762>.

Yadoshchuk, V. (2024) *Why Use Python for AI and Machine Learning*, Waverley. Available at: <https://waverleysoftware.com/blog/python-for-ai-and-ml/> (Accessed: 24 May 2025).