

Enterprise perspectives on large language models



Businesses are eagerly adapting Large Language Models (LLMs) to streamline operations, improve resilience and build competitive advantage through task automation and extension. However, navigating the complexities of LLM implementation at scale poses challenges with evolving technology and many variables in enterprise implementations. To maximize LLM benefits, enterprises need not only technical expertise to construct and implement the models, but to get value from them quickly, they must adopt a realistic outlook on where these models can perform at a consistently high level against potential use cases.

Our team of data scientists recently conducted a series of in-depth experiments with LLMs aimed at deriving mature, actionable and practical insights to assist large organizations in setting and surpassing accurate expectations regarding LLMs. The insights illuminate key strengths, limitations and main resource demands of prominent models offering a guidance on their suitability for various enterprise use cases.

This whitepaper begins by establishing the context necessary for developing a working understanding of LLMs, followed by an exploration of the tools, techniques and methodologies employed in our experiments and applicable in enterprise-grade use cases. Finally, it presents the results and draws longer-term conclusions from them.

Historical context

To understand the current AI boom, we need trace back to 2017 and the debut of Transformers by Google researchers. These models marked a breakthrough by processing sentences, rather than relying on sequential understanding like previous models. This parallel processing capability, enabled by attention mechanism which provides information about the relationships between the words of the text, not only boosts efficiency but also allows models to 'remember' and consider earlier sections of the text while generating output.

Following the popularity of Transformers and attention mechanisms, OpenAI launched their GPT series, which prioritized advanced weighing of words and were specifically trained to be best at predicting the next token only.

Some researchers suggest that the latest iteration in the series, GPT-4, released in 2023, displays sparks of Artificial General Intelligence (AGI) due to its broad knowledge across extensive range of topics, human-like cognitive capabilities and ability to handle not only text but visual input as well.

Defining LLMs

LLMs, like GPT-3.5 and GPT-4 boast an exceptionally high number of parameters, trained on petabytes of textual data from the internet. Parameters are like neurons in the brain, essentially enabling GPT models to simulate human-like cognition. For reference, an average human brain has approximately 80-100 billion neurons, while ChatGPT built on GPT-3.5, features around 175 billion parameters, while GPT-4 might reach 1 trillion. This illustrates why LLMs excels at generative tasks such as summarizing documents, writing code and synthesizing knowledge as increasingly adaptive assistants that leverage conversational interfaces, albeit at a high resource cost of training, inference and maintenance, notably in terms of computational power.

The leading LLM service is currently offered by OpenAI, offering multiple models through an API with diverse possibilities such as embeddings, completions, moderations and more alongside Azure integration. Promising alternatives to OpenAI's offerings are constantly emerging, many of which are open source, greatly expanding customization opportunities.

See the following table for a high-level comparison of closed-dev and open-source LLMs:

	API calls - closed-dev (e.g. GPT-3.5/4)	Local hosting - open-source (e.g. Dolly, LLaMa, Alpaca)
Advantage	<ul style="list-style-type: none">• Easy setup• Cutting-edge models are available• Low latency	<ul style="list-style-type: none">• Greater selection of models• Lower inference cost• Independent
Disadvantage	<ul style="list-style-type: none">• Higher long-term costs• Dependent on third party API	<ul style="list-style-type: none">• Complex and costly setup• Weaker models

The main advantages and drawbacks of closed-dev and open-source LLMs

Note -the difference between a model from an application: GPT3.5, GPT-4 and LLaMa (LLama is a family of autoregressive large language models, released by Meta AI starting in February 2023.) are models, while ChatGPT is a chatbot application built on top of GPT3.5 and GPT-4. Other prominent applications include GitHub Copilot, a code assistant and Microsoft Bing, both of which use a version of GPT.

The experiments

The experiments conducted for this paper aimed to reveal insights regarding key performance considerations, such as identifying most suitable model or service for specific tasks and understanding associated costs and resource needs. To provide a full picture, the discussion will delineate various considerations across different experiments, showcasing and comparing the capabilities and limitations of LLMs across different tasks.

Database assistants

In our first experiment, we focused on an information retrieval use case centered on searching directly inside databases instead of focusing on embeddings. We primarily utilized OpenAI models but had also auditioned open-source alternatives. Beyond examining the querying capabilities, this experiment highlighted the key differences between the models.

Our database based on PostgreSQL comprised multiple tables. SQL being a widely used language in most companies, have these types of databases, we found it intriguing to explore how an LLM could answer questions about such a data source without the need for manually writing SQL queries.

Knowledge graphs with LLMs

The second use case combines knowledge graphs with LLMs to unlock multiple benefits, such as greatly mitigating the hallucinations and propensity for factual incorrectness that have limited the enterprise-grade adaptability of these models. While LLMs' possess implicit knowledge within their parameters, inaccessible and unexplainable to humans, their training on a general corpus limits their ability to fully adapt to specific domains unassisted and have a knowledge cutoff, renders them unable to answer questions beyond their training data. Their output can also contain hallucinations and other kinds of factually incorrect answers. Knowledge graphs hold promise in enhancing the effectiveness of LLMs by providing structured knowledge representation. However, building an adequate knowledge graph is complex and lacks established best practices.

Therefore, the experiment also involved leveraging LLMs to assemble lists of nodes and edges and, ultimately, creating a knowledge graph from raw textual data. With the newly established graph, we delved into information retrieval methodology and corner cases in depth. Given embeddings are commonly used to tackle most of the above-mentioned problems, we compared the graph-based and embedding-based solutions across various cases to see how they stack up and whether a graph can really solve the limited information spanning that's typical of embeddings.

LLMs with tools

The third use case entails a smart gardener assistant designed to showcase the power of chaining using LangChain. For this experiment, OpenAI's GPT-3.5 model serves as the brain and equipped with tools like image recognition, internet search and database querying. The goal is to assess the health of plants and offer care suggestions if any issues arise, focusing on developing a complex workflow with chaining and employing 'chain-of-thought' thought-action-observation cycles.

Models and other tools

This section discusses in detail the toolset and methodologies that we auditioned for the abovementioned enterprise-grade LLM experiments.

OpenAI services as the state of the art

When embarking on serious experimentation with LLMs, the logical starting point is to utilize a tool from the developers of ChatGPT. OpenAI boasts the most mature product currently available, renowned for its response quality and hallucination reduction, establishing itself a primary innovator in the field. Naturally, we sought to ascertain if these characteristics translate to generally superior suitability for business use cases.

Given that ChatGPT is already a comprehensive chatbot application, the underlying model serves as the core building block for the experiment. This was made feasible through the OpenAI API, offering access to various versions of the models, including GPT-4, the latest and most powerful iteration and GPT-3.5, the model underpinning first iteration of ChatGPT. One benefit of using the API is its provision of access to model configuration parameters. While chatGPT only pass dialogue text, API enables manipulation of parameters such as temperature, governing the randomness of the model output and role, allowing contextual setting for the agent's operate. These configuration options cater to the needs of the enterprise environment and the intended use cases for deploying the models.

Price

When evaluating pricing options, two primary main factors come into play: context window size and the underlying model. The context window determines the maximum number of tokens that can be processed by the models at once, essentially defining the maximum amount of information they can handle. OpenAI's largest context window size currently available for GPT-4 is 32k tokens, roughly equivalent to the content of 40 A4-size pages. Leveraging the API with this configuration costs \$0.12 per 1000 tokens, which aligns with the content of the A4 page. While seemingly affordable initially, these costs can accumulates rapidly, particularly for enterprises dealing with a large user base.

Older models such as GPT-3.5, boasting a 16k context window size, come at a mere \$0.004 per 1000 tokens, representing a significant cost reduction. However, this reduction in cost accompanies a trade off in power; earlier models are more susceptible to errors in reasoning and exhibit higher levels of bias. Nevertheless, these characteristics may still render them suitable for applications with narrower scope and functionality.

Open-source alternatives

A variety of open-source LLMs, including LLaMa, Alpaca, Vicuna, Falcon and Dolly, present alternatives to OpenAI's offerings, differing mainly in model size. While GPT-4 boasts around one trillion parameters, leading HuggingFace's LLaMa-2 adaptations features 70 billion parameters. Despite their smaller size, these models are capable of human-like conversation, albeit less refinement compared to ChatGPT. However, their smaller context window, such as LLaMa's 4K compared to GPT-3.5 Turbo's 16K, limits their ability to handle large amounts of information at once.

One advantage of open-source models is the ability to host them locally, eliminating reliance in third-party servers and APIs. However, managing the infrastructure falls on the user, incurring additional setup costs. Cloud services can help mitigate this to a degree by leveraging a cloud service, but defining the architecture will still be the user's task.

Furthermore, open-source models empowers users to train them on a custom dataset, unlike OpenAI offerings that mainly support fine-tuning. However, their models are highly generic, so these options are only ever relevant for specific tasks. The downside of training your own LLMs is that it necessitates significantly more resources, for instance, training a model is much more challenging than using it for inferencing. Our suggestion would be staying away from this option for business use cases and leaving it to research teams unless the use case necessitates specialized data management, for example due to the sensitivity of the data.

Moreover, open-source models may pose a significant limitation in business application: many are only available under non-commercial licenses. It's essential to examine these license terms before adopting these models. For example, the LLaMa model, published by Facebook Research, restricts commercial use, affecting all derivatives based on it. However, the new LLaMa-2 model operates a commercial-friendly license. In contrast, other prominent fully open models like Falcon, Dolly and OpenAssistant are all suitable for commercial purposes.

Model chaining as a new way of orchestration

Language models are, to a degree, single-task-specific. This task can have a narrow focus, such as code generation or broader utility such as maintaining a humanlike dialogue with the user. To enable models to work together in solving complex tasks, we use so-called "model chaining" frameworks.

In these frameworks, one language model acts as an agent: it breaks down a complex task into simple subtasks and delegates them to tools – which may also be language models – and continues this in an iterative loop until the main task is solved. This helps to create agents that have humanlike problem-solving capabilities, extended or limited by the tools with which they are equipped. Such tools typically include web search, SQL database connections, code generators, etc.

LangChain

LangChain is a versatile LLM chaining tool supports several open-source models alongside OpenAI's GPT models. It comes with a comprehensive set of integrated tools, including the Google API for web search, an SQL wrapper for database queries and various classes for streamlined API calls that handle errors efficiently. It even facilitates the creation of custom tools and agents boasting extensive documentation tutorials and a supportive community to help users at all levels of expertise.

HayStack

HayStack specializes in creating both general conversational agents and those tailored for answering questions about specific documents. With HayStack creating memory-equipped chatbots requires minimal code, eliminating the need for defining custom classes. However, one major downside of Haystack is that its less flexible tooling compared to LangChain and currently exclusive supports models from OpenAI.

JARVIS

JARVIS serves as a model chaining agent that seamlessly integrates language models with HuggingFace machine learning models. Users can effortlessly define task and provide a corresponding inputs allowing JARVIS to automatically invoke appropriate model. For example, JARVIS can identify the breed of dog is in an image, without requiring explicit coding for each task. Unfortunately, JARVIS currently has no integration with other tools such as web search.

Experiment processes and results

This section delve into the key takeaways from the three experiments and what they might mean for businesses. For each experiment, we will provide an overview of the setup and the various phases before evaluating the results.

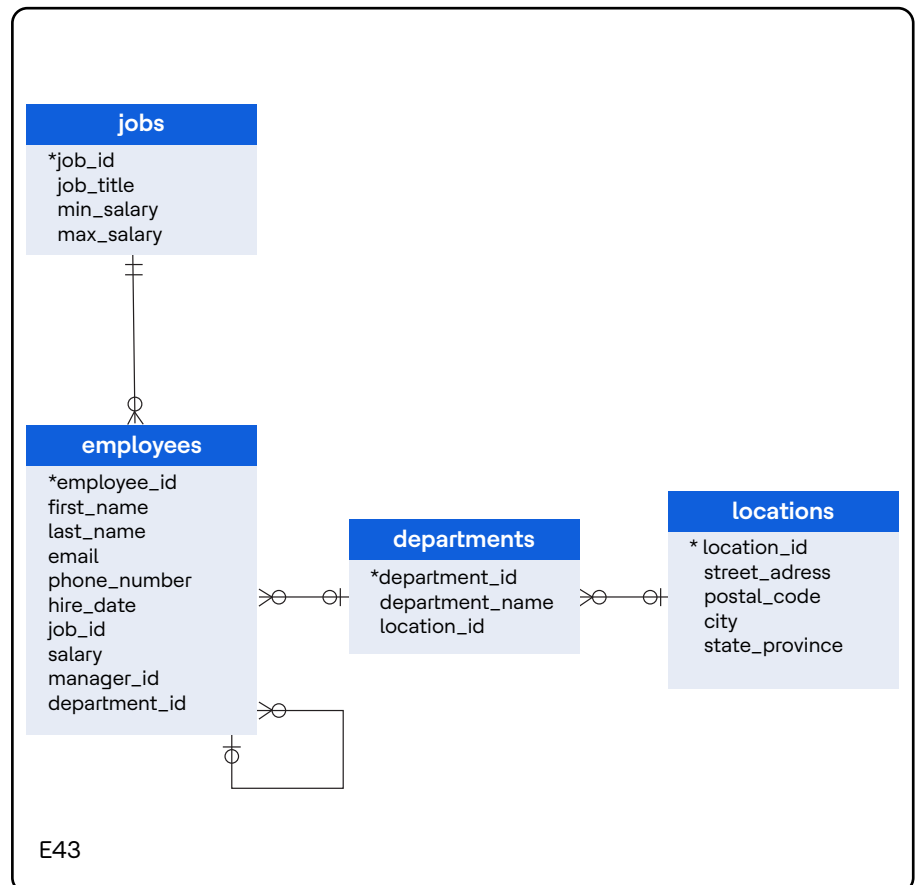
Database assistant

To review, this experiment had two main goals:

- Create an information retrieval system that shows the SQL querying capabilities of LLMs.
- See how various LLMs — open-source and leading OpenAI models — stack up in various aspects.

The setup

To make measurements that would enable us to get usable insights from the experiments, we creating an SQL database was imperative. Opting a publicly available dataset featuring multiple tables of dummy HR data for a fictional small business, we included information such as employees' personal data, their job titles and more. This below image depicts the database schema with the connections:



The database schema used in the database assistant experiment

Once we initialized this database in a PostgreSQL environment, we ensured its accessibility through secure API facilitating seamless interactions from code.

our subsequent task involved scouting for candidate LLMs. We aimed to identify models with distinct advantages while adhering to two constraints: they needed to be relatively small and/or easily integrable. Our rationale behind this was that, if this use case were to go into production, the domain would be small enough that it would have to run in a limited-resource environment and avoid a high setup cost. The next table lists the models used:

Name	Owner	Parameters	Context size	License	Open/closed
Dolly-v2-3B	Databricks	2.8 billion	2K	MIT (commercial use)	Open-source
GPT4All-13B-SNOOZY (GGML)	Nomic	13 billion	2K	Apache-2 (commercial use)	Closed
LLaMa-2-7B	Meta	7 billion	4K	Commercial use	Closed
LLaMa-2-13B	Meta	13 billion	4K	Commercial use	Closed
GPT-3.5	OpenAI	175 billion	8K	Commercial use	Closed
GPT-4	OpenAI	Unknown (estimated 1.7 trillion)	12K	Commercial use	Closed

The models used in the database assistant experiment

The subsequent task involved constructing questions or prompts for the model, gradually increasing in difficulty. Initially, we formulated straightforward questions that did not necessitate joining tables for answering. Subsequently, we designed questions still recognized as easy but incorporated synonyms. To tackle the most difficult questions, the model had to decipher not only the rephrasing but also, for example, joining and other advanced SQL techniques as well.

To facilitate the testing environment, we crafted a versatile testing script capable of adapting to various model interfaces. When using OpenAI's models proved uncomplicated for the open-source models, HuggingFace provided a coding framework ensuring unified and painless integration. When required, we used a GPU, typically opting for a smaller one like a Tesla T4.

The prompt and context

Every model received identical structured prompts and context. They were tasked with assuming the identity of a helpful assistant that transforms questions into SQL queries. Additionally, we furnished them with the database schema to acquaint them with the available fields for answering the user's questions.

However, a glance at the context sizes foreshadowed a major difference among the models: Dolly and GPT4All models were constrained to a 2K context length, limiting their ability to handle larger datasets, while LLaMa-2 boasted a 4K context length and OpenAI comes out on top with its sizes in the 8K-32K range. Our example database is considered tiny but, for reference, our prompts cost an average of 200 tokens, not including the question.

OpenAI takes the lead with ease

The OpenAI models excelled in every test, effortlessly executing basic SQL statements and adeptly rephrasing verbs and nouns from the question to the SQL query based on the schema. They demonstrated exceptional proficiency in handling keys and joining statements. While their SQL results might not always have produced the best-performance, they consistently provided a reliable answer every time.

Interestingly, there was no significant performance gap between GPT-3.5 and GPT-4, indicating that the older GPT is also comfortably capable of solving these kinds of problems. Not only did the models produce good results, but they were also stable, reproducible and very fast using OpenAI APIs.

Lightweight, with performance to match: Dolly

Dolly-V2, the smallest model participating in the experiment, struggled the most, failing most of the tests. The basic knowledge of SQL present in the model facilitated it to generate a correct functional query for questions such as, "How many employees are there?" or "How many employees have the last name King?". It could even list the names when asked about the employees who joined before 1990, demonstrating decent functionality for its size.

However, the hallucinations and heavily 'generative attitude,' repeatedly erring on the side of linguistic, rather than substantive appropriateness resulted in a mixed picture overall. For example, when asked "How long has Alexander Hunold been working at the company?" or "What is the role of Alexander Hunold?", it simply generated a random number for the answer. So, for every complex query in the result was either a hallucination or syntax errors which stemmed from the misuse of the keys for joining and misunderstanding the questions.

Interestingly, Dolly results were sometimes hard to reproduce and the model has a strangely oscillating performance, which was highlighted when asked "List each department with the number of employees working for that department. Return the department name and the headcount." Being a relatively complex statement, we expected the model to fail to provide an adequate output, only to be provided the following SQL query:

- ```
SELECT d.department_name, COUNT(e.employee_id)
FROM employees e
JOIN jobs j ON j.job_id = e.job_id
JOIN departments d ON d.department_id = j.department_id
JOIN locations l ON l.location_id = d.department_id
GROUP BY d.department_name
```

This syntax was very close to the ideal answer since it only contains two mistakes – the key is wrong in the second JOIN clause since only the employees (e in the query) table has department\_id and the jobs table does not. Furthermore, there is a similar issue with the third JOIN, as it should map l.location\_id with d.location\_id and not department\_id. If corrected, this query would produce the desired output.

## Well-optimized but inaccurate: GPT4All

Beginning with the positive, despite its larger size (13 billion parameters vs. Dolly's 2.8) GPT4All's resource demand was comparable to Dolly's. However, despite being able to perform basic tasks like counting rows with small WHERE clauses or filtering employees based on their hire dates, it kept making syntactical mistakes. It was unable to comprehend the parameter we provided in the prompt. For instance, when asked to count employees who have the letter t in their name, the model searched for the capital letter T and returned only a subset of the correct set. Furthermore, it was unable to utilize the right schema and employed fields from the overall schema with the wrong table. It also used clauses which were unnecessary frequently multiple times, adding a "WHERE employee\_id = '1234567890'" for no reason at all or tried to work with the manager\_id field even if it wasn't mentioned at all.

## Meta's Midranger: LLaMa-2

In case of LLaMa-2, we emphasized on the smallest version, LLaMa-2-7B. LLaMa-2 is a new competitor in the market building on the original LLaMa by Meta. The key improvements in this tool are a bigger context size, with 4K tokens and some performance gains. It has three versions, differentiated by the number of parameters: the 7-billion "7B", 13-billion "13B" and 70-billion "70B."

The performance of LLaMa-2-7B was great, when medium level questions or prompts were asked. It was able to create basic SQL statements with WHERE, COUNT, LIKE, GROUP BY and DISTINCT clauses. However, it had some persistent problems, including misuse of the keys at joining and creating syntactically correct but functionally incorrect statements. For example, for the task of "Who has the max salary in IT and how much bigger is their salary compared to the minimum IT salary? Return the first name, last name and the difference as a number", it encountered the following recurring errors:

- Misuse of keys, joining by departments.job\_id (non-existent)
- IT is in WHERE clause with LIKE searching in jobs.job\_title (not using the "departments" table)

- IT filtering is completely ignored
- Forgot GROUP BY clause
- Wrong calculation of differences (calculated against each employee's current salary, so the results were always zero)
- Random semicolons in the middle of statements
- Using Oracle SQL statements, not PostgreSQL

If we extended the conversation into a chat and provided feedback on the error, LLaMa successfully resolved some issues, substituting the Oracle statement with another function that works in Postgre as well, including the overlooked GROUP BY and filtering for the IT department. However, in some cases like the above-mentioned example, it struggled to figure out the solution despite these little nudges.

We evaluated most of the questions against the bigger version, LLaMa-2-13B which unfortunately did not produce much improvement. It handle cases similarly to LLaMa-2-7B and encountered the same errors in the identical scenarios. Furthermore, the reasons were similar, with occasional failures in creating the appropriate JOIN clause and overlooking specific department filters, for example. Interestingly, in certain cases, the 13-billion-parameter version generated specific IDs that had nothing to do with the questions.

Both LLaMa models were accessed via HuggingFace services. Responses remained consistent and relatively prompt, although the bigger model usually took twice as long to produce an answer as the seven billion-parameter version. Additionally, the performance of the model enhanced when we kept the chat history limited, ideally maintaining one query at a time.

## Knowledge graph

As a reminder, this experiment aimed to gauge whether knowledge graphs could offer a solution to overcome lingering inherent limitations of LLMs, such as lack of explainability, knowledge cutoffs, training and fine-tuning problems due to immense resource requirements, questionable reasoning capabilities, hallucinations and factual inconsistency.

While knowledge graphs show promise and are subject of intense ongoing research, they are not widely deployed at businesses. Building a versatile knowledge graph is an intricate task and best practices are yet to be established. Consequently, our first notable takeaway on the subject is that since knowledge graphs are uncommon, LLMs have limited knowledge of them. Although the models understand the concept, relevant tools and use cases, Their performance may not match that more established domains like SQL use cases. Moreover, the challenge of fine-tuning with knowledge graphs arises from the disparity between LLMs are based on prompt-completion pairs the nodes and edges structure of a graph, which makes fine-tuning an alignment between these both essential and challenging.

## Preparations

As the first step, we endeavoured to construct a knowledge graph with an LLM. Given raw textual data and some prompt engineering, the model successfully managed to build a usable graph based on the provided information. Subsequently, we examined how well it could use the graph to extend its prompts with relevant information. For this experiment, we used Neo4j as the chosen graph database management system. Neo4j offers hosted databases with secure API connections. It employs Cypher for querying, a recent addition to the LangChain framework, which encompasses graph-querying modules with syntax-checking and automated direct connection to sources. Throughout this experiment, we extensively utilized OpenAI, since their models had consistently displayed the best performance. Given the complexity inherent in building a knowledge graph, we deemed it unnecessary to subject other models to additional hurdles.

### Can an LLM build a knowledge graph?

The OpenAI models demonstrated the ability to generate knowledge graphs once the right prompt was given. Achieving a uniform structure for information extraction was imperative in converting it into nodes and edges within the graph. To accomplish this, we tested multiple prompts and iterated them until we got acceptable results. By framing the task as that of an investigator tasked with constructing knowledge graphs from narratives, we guided the model's approach. Employing one-shot prompting involved presenting the model with a short story alongside the expected output, explicitly stipulating a strict format: first, listing the entities it mentioned in the text, followed by detailing the connections between them.

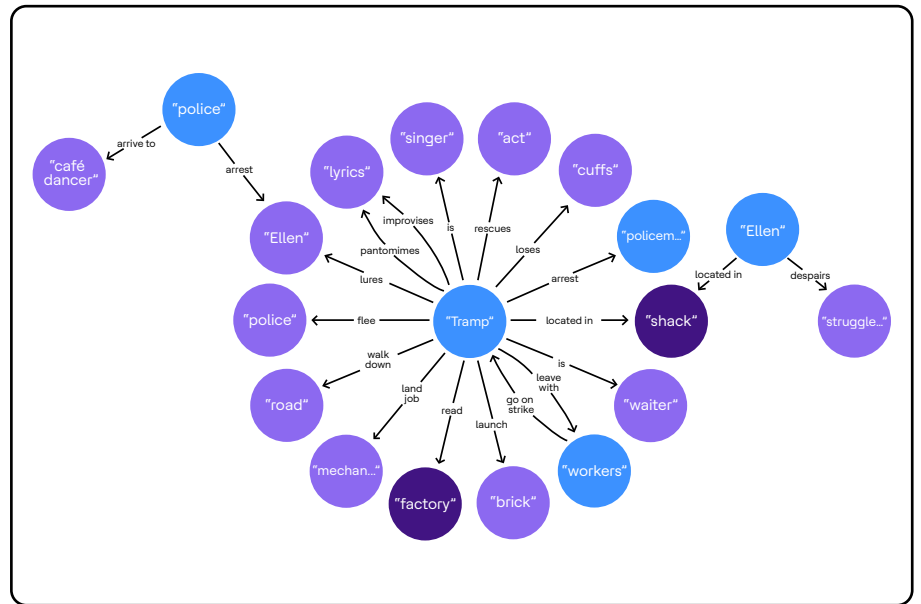
Once the entities and the edges between them we established, transferring this into specific queries to create the knowledge graph became imperative. To this end, we constructed a prompt wherein the model was tasked with generating Cypher statements based on the previous output. We also instructed the model not to create duplicate nodes or edges. This output was supervised via LangChain's GraphCypherQChain class, which was responsible for basic checks and then executing the queries on our Neo4j graph instance.

For experimentation, we provided a two-paragraph excerpt from the Wikipedia plot summary of Charlie Chaplin's Modern Times as input. We tasked both GPT-3.5 and GPT-4 with constructing the graph, at which point both models encountered a problem when trying to classify certain nodes, which could lead to issues with Neo4j's labeling, causing it to misuse the name, type and label parameters of a node. Despite this, the generated graphs covered the story very precisely.

### The knowledge graph according to GPT-3.5

The knowledge graph generated by GPT-3.5 model illustrates the below knowledge graph of the given content. Personas in the story are denoted by blue nodes, object by purple ones and locations by green ones. All the important connections from the story are captured to some extent.

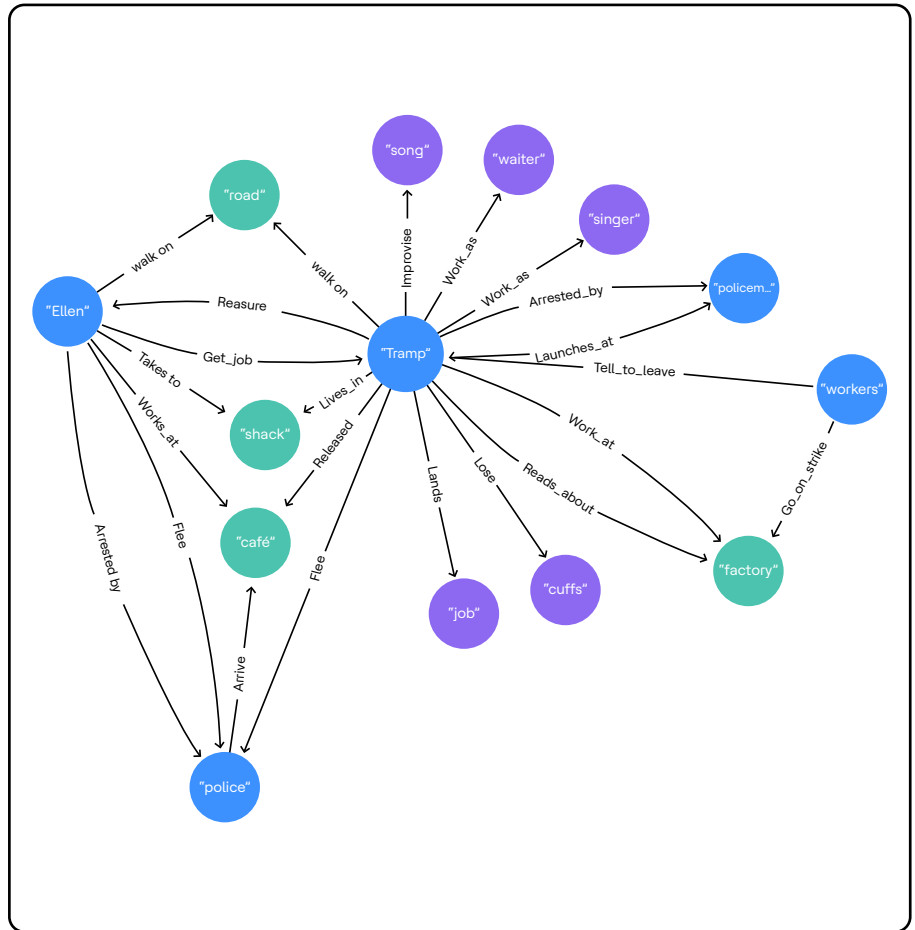
However, some nodes are duplicated due to the previously mentioned type misconception: for instance, 'police' is shown both as an object and a character, as is 'Ellen'. Additionally, minor hiccups arise with synonyms, 'policeman' and 'police' are represented as distinct nodes, although this could still fit the story logically.



Knowledge graph created by GPT-3.5

### The knowledge graph according to GPT-4

GPT-4 created a more streamlined graph compared to its predecessor. It successfully eliminated duplicate nodes featuring only one instance each of 'police' and 'Ellen', but still produced an overlapping 'policeman'. Notably, GPT-4 generated fewer nodes, offering a condensed representation of the story with less detail, notably omitting the brick-throwing incident and the protagonist's job with the mechanic. However, node classifications were more accurate, with 'café' categorized as a location and each persona correctly identified. When creating the edges, the GPT-4 model uses richer descriptions like `WORK_AS` instead of `IS` and the more precise `LIVES_IN` instead of `LOCATED_IN`.



Knowledge graph created by GPT-4

## Can an LLM actually use a knowledge graph?

Unfortunately, the models we tested struggled to effectively leverage the knowledge graphs. The key issue stemmed from their inability to comprehend the returned information from graph queries. Despite being provided with the schema and existing relations of the graph, both models faced difficulties in grasping the data, proving to be an insurmountable hurdle for LLMs lacking context.

Basically, while GPT-3.5 could generate functional Cypher queries, it encountered difficulty in understanding them. During our experiment, the syntax and clauses of its queries were consistently correct and functional, instilling optimism. However, the model ultimately faltered in generalizing properly.

For instance, when prompted with "Tramp worked as what?", it translated the relation to "WORK\_AS", whereas the graph represented this connection as "IS", leading the model to find no relevant information. At times, GPT-3 required explicit instructions regarding which nodes or relationships to use, resulting in a cumbersome process. On the bright side, it could use the basic graph algorithms and, for example, find the shortest path between nodes:

- MATCH (p:person {name: 'workers'}), (o:object {name: 'café'}),  
path = shortestPath((p)-[\*]-(o))  
RETURN path

GPT-4 demonstrated notable improvement in understanding the query results and grasping the underlying logic of the relations within the knowledge graph. It adeptly aligned the verbs and nouns in the question with the existing nodes and edges with minimal mistakes in types and labels. However, occasional inaccuracies persisted, partially in cases where the direction of relationship was confused. Nevertheless, GPT-4 displayed great talent in creating Cypher queries, even complicated ones. Challenges arose with multiple aggregations or utilizing different grouping hierarchies, which can be attributed to the model's lack of information on the topic.

A reliable and promising application for knowledge graphs, in terms of enterprise applicability, lies in leveraging them as "information pointers" in a retrieval use case. Constructing the knowledge graph to encompass entities and connections from the raw text and linking text chunks to the nodes enables the models to utilize the graph's architecture and reference relations between raw text and graph in novel ways, such as through embedding vector-based search. This approach eliminates the spanning information problems, allowing models to link text parts that were distant with respect to their embedding vector but have a deep connection that is needed for answering.

Overall, despite the current limitations, LLMs leveraging knowledge graphs are a promising path for development. There is a lot of untapped potential here that, in a few years, could lead to significant performance and reliability gains.

## Smart gardener

With our final experiment, we aimed to bridge the gap between the digital and physical world by creating a solution capable of interacting with environment via sensors and actuators. Thus, the Smart Gardener project was conceived – a conversational assistant designed to care for office plants. The results detailed below concern to the first phase of the experiment, which centred on software development and included the entire LLM-based workflow.

Our target requirements were for the application to:

- have access to camera recordings of office plants
- recognize the type of plant based on camera recordings
- perform health checks on plants
- identify plants' condition by referencing them with a database and suggest adjustments if needed
- display historical variations in sensor data



- have actuators to water the plants
- incorporate voice communication and chat capabilities

## Core architecture

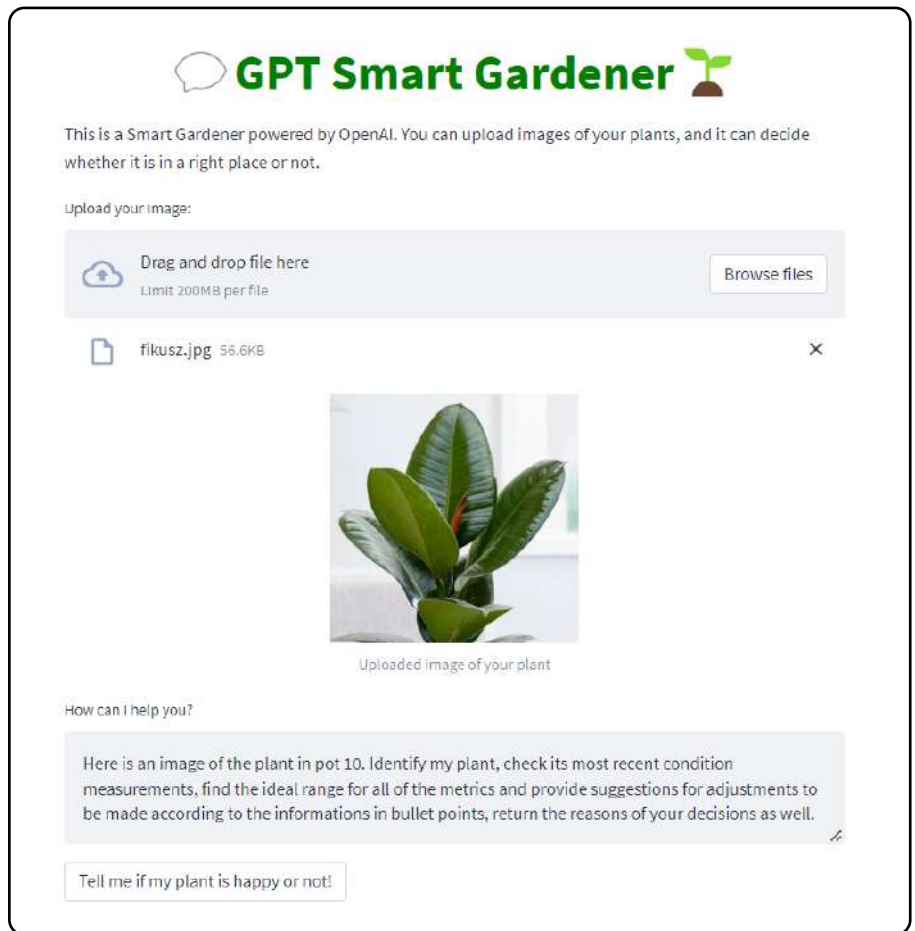
The architecture of the experiment relied heavily on the chaining technique, with additional tools for extended functionality. We specially emphasized on defining the interfaces of certain tools to make the system easy to adapt to use cases other than smart gardening. LangChain once again proved essential in creating the chaining flows on which this entire use case is built.

We equipped the GPT-3.5 “brain” with the following tools for the experiment:

- **Plant identifier:** A small neural network hosted on HuggingFace capable of recognizing a predefined set of plants from images. It provides classification results, indicating the plant type based on the input image.
- **Internet search:** Utilizing DuckDuckGo, this connector enables the LLM to search for specific topics, particularly concerning ideal plant conditions. To mitigate hallucinations, the LLM restricted from using the search tool without knowing the plant type and is directed to ask specific questions. Repeated searches are permitted to refine information.
- **Database search:** Connected to a SQLite database containing historical sensor information of plants, including temperature, humidity, light conditions, PH level and soil moisture. Synthetic data is used during this experimental phase until hardware construction is completed.
- **Condition comparison:** Another GPT-3.5 model, tasked with comparing actual plant conditions of against their ideal counterparts. Based on this comparison, the model make decision and offers recommendations.
- **Plot chart:** A third GPT-3.5 model, equipped with a specific prompt to create a Python chart for the Streamlit application. It identifies the data type required for the chart and leverages the database search tool to query the information for the desired plot.

The whole application was bundled up in a containerized Streamlit application to make it easy to set up anywhere and it also features a web-based interface.

## User interface



The Smart Gardener UI (work in progress)

The web UI provides two fields for input for uploading a photo of a plant and raise questions. Since the hardware components are yet to be installed, the plants are identified by their pot number, hence it is imperative to provide pot ID so the right set of data from the database can be fetched. We added a default prompt to show an example of a precise question.

## Can an LLM make a plant happier?

The initial run by the Smart Gardener's was a remarkable experience, demonstrating sparks of AGI. Every step taken by the model can be tracked with the help of the code that logs the entire chain-of-thought process of the LLM. The LLM model perfectly sequenced the exact steps it needed to perform the process.

It took errors in stride: when it made a mistake in its own syntax, it could easily rephrase its internal prompts and try again. If a tool didn't provide the required information, the model moved on without inserting hallucinations into the process and simply stated that a piece was missing. Below is a sample from the LLM's log:

- **Thought:** I need to identify the plant in pot 10 and then check its most recent condition measurements. I can use the `plant_identifier` tool to identify the plant and the `DatabaseSearch` tool to get the most recent measurements
- **Action:** `plant_identifier`
- **Observation:** Philodendron 'Imperial Green'
- **Thought:** I have identified the plant in pot 10 as Philodendron 'Imperial Green'. Now I need to use the `DatabaseSearch` tool to get the most recent condition measurements for this plant
- **Action:** `DatabaseSearch`
- **Action Input:** Pot number 10

From the log, we could see the first two cycles. The model realizes that it first needs to identify the plant from the photo, then get the corresponding data. It uses the right tool with the expected input formats and handles the output as well by understanding the results.

- **Thought:** I have the most recent condition measurements for the Philodendron 'Imperial Green' in pot 10. Now I need to search the internet for the ideal range for each metric and provide suggestions for adjustments based on the recorded values
- **Action:** `InternetSearch`
- **Action Input:** Ideal temperature range for Philodendron 'Imperial Green'
- **Observation:** Your Philodendron Imperial Green will thrive in temperatures between 75°F and 85°F (23°C to 29°C) ...
- **Thought:** I have found the ideal temperature range for Philodendron 'Imperial Green' to be between 65°F and 85°F (18°C to 29°C). The recorded temperature for pot 10 is 12.094217640511111, which is significantly lower than the ideal range. I suggest adjusting the temperature to be within the ideal range by providing additional heat or moving the plant to a warmer location.

After finding information from the database, the model starts to search for the ideal conditions – the above example shows the first instance, with the model checking the internet for the temperature information. As you can see, it was able to retrieve the relevant details – without any unnecessary noise – from the internet and already provided a basic recommendation.

The model takes each measurement – retrieved from the SQL database – and checks if it can find the right information. Then, it constructs the final answer and creates a chart of the temperature changes of the last few records of the plant. All of these go back to the UI displaying the results.

## My suggestions:

- The current temperature of the philodendron 'Imperial Green' is 19.72°C, which falls within the ideal range of 16-30°C. Therefore, the temperature is okay.
- The light condition is given as a numerical value of 1723.28, while the ideal condition is described qualitatively as "bright, indirect". Without a clear way to convert the numerical value to a qualitative description, it's not possible to definitively say whether the light condition is okay or not. However, if the numerical value represents light intensity in lux, then it might be too high as "bright, indirect" light typically corresponds to a lower lux value. It is suggested to adjust the light source or use a shade to reduce the light intensity.
- The current humidity level is 0.49 (assuming this is a relative humidity value in decimal form, i.e., 49%), which is considered low for a "moderate" humidity level typically required by philodendron 'Imperial Green'. It is suggested to increase the humidity level, perhaps by using a humidifier or placing the plant on a tray of water.
- The soil condition is currently "dry", while the ideal condition is "well-draining, water when top half feels dry". This suggests that the plant may need watering. It is suggested to water the plant until the top half of the soil feels moist.

Example output from Smart Gardener

During this process, various issues emerged including the reproducibility, handling of dates and resolving differences in categorical or numerical values. Despite efforts to ensure model parameters are consistent, the results vary for the same output. Sometimes, the model can find information for more features, while in other cases, for reasons currently unknown, it provides a final output based on limited observations without utilizing all the data available. When querying the database, it occasionally uses the wrong dates and fails to retrieve the most recent measurements, even if specifically asked and having the parameters set accordingly.

Another hurdle arises when comparing different scaled measurements, such as soil moisture information is currently stored in a categorical variable (dry, moist and wet) meanwhile, the information found on the internet (and scraped by the model) suggests different frequencies for watering. This discrepancy hinders LLM to accurately assess watering needs. Integrating the hardware components could resolve this issue, enhancing the model's accuracy and enabling it to create informed recommendations and take necessary action.

Despite the challenges listed above, our experiment demonstrates the significant potential of chaining technology to extend the utility of LLMs.

With increased adaptability, it can streamline various workflows to replace manual tasks, including choosing the right neural networks for predictive tasks, classification, regression, searching the web, connecting to an internal database to retrieve knowledge and more.

This high level of adaptability for LLMs opens a new perspective for application development, making it possible to, for example, automate the dashboard creation process. Tools such as anomaly detection modules, packages to create charts, database connections to retrieve the data for the charts and internet connectivity for competitor analysis will enable users to define their needs in natural language and leave it to the LLM to orchestrate the entire process, from creating the underlying logic to connecting the individual pieces and, finally, provide quick and effective output.

## Final thoughts

In the near future, the trajectory of LLM will likely shape their role as primary source of competitive advantage for enterprises. The technology, along with the systems built around it, is poised to reach new heights. As the above experiments demonstrated, most of the building blocks are already firmly in place and with the current pace of development and adoption, the early growing pains will likely prove fleeting. Below, we outline key takeaways, emphasizing that large language models have immense potential to simplify and elevate enterprise use cases, with even the tasks that are currently beyond leading models' abilities appearing well within reach in the foreseeable future.

### OpenAI's models are the ones to beat

OpenAI's models are currently the most powerful LLMs, demonstrating unmatched power and consistency when configured specific use cases. Notable, they excel in delivering precise, relevant responses while minimizing unnecessary noise. Their remarkable understanding of user intents also sets them apart from open-source models. Beyond their reduced frequency of hallucinations, their effectiveness in high-stakes communication is even underscored by their improved capability to express uncertainty by simply saying "I don't know" or "I am not sure".

As a result, when integrating LLMs into a general use case, our current recommendation is to go with one of OpenAI's models, while the open-source route only becomes preferable when there is a highly specific task at hand that necessitates a feature or characteristic found only in an open-source model.

On a more critical note, it also became apparent during our experiments that maintaining an OpenAI-based end-to-end solution may prove prohibitively expensive and their fine-tuning system for customization also has considerable limitations.

## The cutting edge is costly

Extending on the previous point, cutting edge technology comes at a cost. LLMs are inherently resource-intensive. Even for inference alone, substantial GPU RAM (often exceeding 100 gigabyte) is necessary, making hosting or using them locally on GPU cluster expensive and complex. While open-source models themselves may be free, the associated infrastructure costs for hosting locally or in the cloud must be considered.

On the bright side, the ability to use models in overall less resource-heavy systems is on the horizon, with intensive research efforts directed at fitting the inference of the big models into a simple environment using quantization techniques to control costs and other resource demands. However, there are already great hosting options, notably from Azure and HuggingFace services.

## The future is now – and more so tomorrow

Technologically, LLMs have reached a level of maturity that positions them favorably for demanding, sensitive enterprise use cases with ongoing developments reinforcing this suitability. Their general knowledge has been extended to where there are barely any topics about which they don't have at least some usable information. These should also get covered soon, with the development into novel methods to improve LLMs' knowledge, like the plugin system or defining built-in functions.

LLMs display an exceptional aptitude for sequential thinking and creating chain-of-thoughts cycles, enabling them to navigate complex scenarios effectively, despite lacking real reasoning capabilities and interpretability for humans due to the multitude of problems and solutions they handle simultaneously.

Regarding open-source models, context window size remains a significant limitation, with GPT-4's 32K context size dwarfs that of the LLaMa-2-based HuggingFace benchmark leaders, like the Platypus2, whose context length is 4096 tokens – but there are reassuring developments in this field. Innovations like Anthropic's Claude model, achieving a context length of 100K tokens, show promise for greatly expanding these boundaries.

# Author information



## **Bálint Kovács**

Balázs has a background in software development and has worked in a diverse range of roles, including as a research fellow and assistant lecturer at a top Hungarian university, a deep learning developer at a major multinational company and, currently, as a consultant data scientist. He enjoys diving deep into user data to uncover hidden insights and leverage them to create effective solution prototypes.



## **Szilvia Hodvogner**

Szilvia has extensive experience working for research-oriented companies, where she has developed a deep understanding of predictive models and NLP. At HCLTech, she currently works on NLP projects with a focus on LLM solutions. She holds a master's degree in computer science, specializing in artificial intelligence and computer vision.



## **Balázs Zempléni**

Balázs holds a degree in Engineering and specializes in digital image and signal processing. He has worked for multiple banks in various data engineering and business intelligence roles. In recent years, he has focused on developing an NLP solution to improve internal business processes based on textual data. In addition to his work, Balázs is an avid presenter at meetups and conferences.

# HCLTech | Supercharging Progress™

HCLTech is a global technology company, home to 222,000+ people across 60 countries, delivering industry-leading capabilities centered around Digital, Engineering and Cloud powered by a broad portfolio of technology services and software. The company generated consolidated revenues of \$12.3 billion over the 12 months ended December 2022. To learn how we can supercharge progress for you, visit [hcltech.com](https://hcltech.com).

[hcltech.com](https://hcltech.com)

