












# On the Construction of Database Interfaces Based on Large Language Models

João Pinheiro<sup>1</sup><sup>a</sup>, Wendy Victorio<sup>1,2</sup><sup>b</sup>, Eduardo Nascimento<sup>1,2</sup><sup>c</sup>, Antony Seabra<sup>1</sup><sup>d</sup>,  
Yenier Izquierdo<sup>2</sup><sup>e</sup>, Grettel García<sup>2</sup><sup>f</sup>, Gustavo Coelho<sup>2</sup><sup>g</sup>, Melissa Lemos<sup>1,2</sup><sup>h</sup>,  
Luiz Paes Leme<sup>3</sup><sup>i</sup>, Antonio Furtado<sup>1</sup><sup>j</sup> and Marco Casanova<sup>1,2</sup><sup>k</sup>

<sup>1</sup>Departamento de Informática, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil

<sup>2</sup>Instituto Tecgraf, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil

<sup>3</sup>Instituto de Computação, UFF, Niterói, 24210-310, RJ, Brazil

Keywords: Large Language Models, Database Interfaces, Conversational Interfaces.

Abstract: This paper argues that Large Language Models (LLMs) can be profitably used to construct natural language (NL) database interfaces, including conversational interfaces. Such interfaces will be simply called LLM-based database (conversational) interfaces. It discusses three problems: how to use an LLM to create an NL database interface; how to fine-tune an LLM to follow instructions over a particular database; and how to simplify the construction of LLM-based database (conversational) interfaces. The paper covers the first two problems with the help of examples based on two well-known LLM families, GPT and LLaMA, developed by OpenAI and Meta, respectively. Likewise, it discusses the third problem, with the help of examples based on two frameworks, LangChain and LlamaIndex.

## 1 INTRODUCTION


Natural language database interfaces allow users to access databases using queries formulated in natural language (NL) (Affolter et al., 2019). Going one step further, NL database conversational interfaces (software-based dialogue systems or chatbots) (Motger et al., 2022)(Caldarini et al., 2022) offer users the possibility to interact with databases using techniques similar to human conversation; they accept queries typically formulated in NL, return answers also commonly expressed in NL, and support persistent conversational contexts, thereby generating user-


computer dialogues.


This paper argues in favor of using Large Language Models (LLMs) (Manning, 2022) to construct NL (conversational) database interfaces. Such interfaces will be simply called *LLM-based database (conversational) interfaces*, leaving it implicit that the queries are formulated in NL. The adoption of LLMs would simplify the interpretation of NL queries, the generation of NL answers, and the control of user dialog, tasks that are addressed in LLM training.


A Large Language Model (LLM) is a language model based on a deep neural network architecture with a very large number of parameters, trained on enormous quantities of unlabeled text, using self-supervised or semi-supervised learning. One should single out LLMs that have been trained to follow instructions, which are most suited for constructing conversational interfaces. Such LLMs are sometimes called Chat Models. LLMs are a transformative technology, enabling developers to build novel applications. LLMs came to the foreground with the announcement of conversational interfaces that are able to generate high-quality textual answers.


Specifically, the paper addresses three problems:


<sup>a</sup> <https://orcid.org/0000-0002-0909-4432>


<sup>b</sup> <https://orcid.org/0009-0003-0545-2612>


<sup>c</sup> <https://orcid.org/0009-0005-3391-7813>


<sup>d</sup> <https://orcid.org/0009-0007-9459-8216>


<sup>e</sup> <https://orcid.org/0000-0003-0971-8572>


<sup>f</sup> <https://orcid.org/0000-0001-9713-300X>

<sup>g</sup> <https://orcid.org/0000-0003-2951-4972>

<sup>h</sup> <https://orcid.org/0000-0003-1723-9897>

<sup>i</sup> <https://orcid.org/0000-0001-6014-7256>

<sup>j</sup> <https://orcid.org/0000-0003-3710-624X>

<sup>k</sup> <https://orcid.org/0000-0003-0765-9636>

- P1.** How to use an LLM to create an NL database interface.
- P2.** How to fine-tune an LLM to follow instructions (i.e., to become a Chat Model) over a particular database.
- P3.** How to simplify the construction of LLM-based database (conversational) interfaces.

Sections 3 and 4 cover the first two problems with the help of examples based on two well-known LLM families, GPT and LLaMA, developed by OpenAI and Meta, respectively. Likewise, Section 5 discusses the third problem, with the help of examples based on two frameworks, LangChain and LlamaIndex.

The paper is organized as follows. Section 2 reviews LLMs and some popular LLM families. Section 3 comments on GPT-based database interfaces and Section 4, LLaMA-based database interfaces. Section 5 covers LangChain and LlamaIndex. Section 6 concludes the paper by summarizing the lessons learned.

## 2 BACKGROUND

Briefly, Affolters et al. (Affolter et al., 2019) offers a comparative survey of 24 NL database interfaces, classified as keyword-, pattern-, parsing- and grammar-based. As lessons learned, the authors indicate that, for simple questions, keyword-based systems are enough, whereas for complex questions, an NL database interface needs to apply some parsing to identify structural dependencies, and that grammar-based systems are the most powerful, but depend on manually designed rules.

Diefenbach et al. (Diefenbach et al., 2020) assumes that an NL query can be correctly interpreted considering only the semantics of the words. For example, the NL query “Give me actors born in Berlin” is reformulated as a keyword question “Berlin, actors, born in”, that is, the semantics of the words “Berlin”, “actors”, and “born” suffice to deduce the intention of the user. The authors proceed to map the NL query into one of four templates, which limits the usefulness of the process. Interestingly, however, they show that the process has good performance on several RDF datasets from the QALD benchmarks in different languages.

Recent surveys of conversational interfaces (software-based dialogue systems or chatbots) can be found in (Motger et al., 2022)(Caldarini et al., 2022). Such systems offer users the possibility to interact with databases using techniques similar to human conversation; they accept queries typically formu-

lated in NL, return answers also typically expressed in NL, and support persistent conversational contexts, thereby generating user-computer dialogues.

In another direction, the approaches involved in Natural Language Processing (NLP) tasks have undergone profound changes over the past decades. (Manning, 2022) classifies NLP approaches in roughly four eras. From 1950 to 1969, the first era was characterized by simple rule-based models, achieving very limited results due to the lack of computation, data, and knowledge of human language structure. The second era, from 1970 to 1992, benefited from the rapid development of linguistic theories, allowing the creation of a new generation of hand-built systems from knowledge-based artificial intelligence models. The third era, from 1993 to 2012, saw a shift from knowledge-based to machine-learning models, when digital text became widely available, providing enough data to allow some level of language understanding by using mainly supervised machine-learning methods. The last era, from 2013 to the present, extended the use of machine learning, allowing a more generalized language understanding by using self-supervised learning and deep learning-based models to represent words with dense vectors. Within this context, the first LLM was successfully trained in 2018 by exposing a model to an extremely large quantity of text, representing an enormous amount of language knowledge.

LLMs follow the *transformer* neural network architecture (Vaswani et al., 2017)(Tunstall et al., 2022)(Jurafsky and Martin, 2023). Manning (Manning, 2022) contains a readable introduction to transformers. The central intuition behind transformers is the use of the *attention* mechanism, where the representation of a given position is computed based on a weighted combination from others. The result is a much more complex setup when compared to simple neural networks. Based on this architecture, the self-supervised strategy for LLMs usually involves masking words in the text and training the model to predict the missing words from the outer context. After this process is extensively repeated, the model learns generalized syntactic structures of sentences.

As a consequence of their generalization capacity, LLMs perform well in a wide variety of NLP tasks, to the point of shifting the focus of NLP research away from the previous paradigm of training specialized supervised models for each task. An LLM can be redeployed to a particular NLP task with a small number of further instructions.

However, an LLM reflects the data it was trained with. In particular, an LLM suffers from the “temporal generalization problem” – capturing facts that change over time – and the “factual grounding prob-

lem” – capturing specific facts. To circumvent these limitations, the user may *fine-tune* the LLM, that is, retrain it with more examples, or he may adopt *few-shot learning*, that is, add a few examples in a dialog interaction so that the model can capture what the user is trying to do and generate a plausible completion.

Chat Models must in turn (Ouyang et al., 2022): avoid unacceptable answers, including avoiding incorrect or made-up facts, nonsensical answers, and biased or toxic text; seek useful user dialogues, that is, the language modeling objective must be aligned with the objective “follow the user’s instructions helpfully and safely”. Putting it in more explicit terms, a Chat Model must be (Askill et al., 2021): *helpful* (they should help the user solve their task), *honest* (they shouldn’t fabricate information or mislead the user), and *harmless* (they should not cause physical, psychological, or social harm to people or the environment).

Currently, there are three well-known LLM families:

**GPT** – Generative Pre-Trained Transformer: GPT 3.5<sup>1</sup> (announced on November 30th, 2022) and GTP 4<sup>2</sup> (announced on March 14th, 2023) respectively support ChatGPT and ChatGPT Plus.

**LLaMA** – Large Language Model Meta AI: LLaMA<sup>3</sup> (announced on February 23rd, 2023) and LLaMA 2<sup>4</sup> (announced on July 18th, 2023) were developed by Meta.

**LaMDA** – Language Model for Dialogue Applications and **PaLM** – Pathways Language Model: LaMDA<sup>5</sup> (announced on May 18th, 2021) and PaLM (announced in March 2023 and upgraded in May 2023) support Google’s Bard.

The examples in Sections 3 and 4 respectively use the GPT and the LLaMA families.

### 3 GPT-BASED INTERFACES

#### 3.1 The GPT Family and OpenAI API

The OpenAI Research API offers several models of the GPT family, with different capabilities and

<sup>1</sup><https://platform.openai.com/docs/model-index-for-researchers>

<sup>2</sup><https://openai.com/research/gpt-4>

<sup>3</sup><https://ai.facebook.com/blog/large-language-model-llama-meta-ai/>

<sup>4</sup><https://ai.facebook.com/blog/large-language-model-llama-meta-ai/>

<sup>5</sup><https://en.wikipedia.org/wiki/LaMDA>

prices<sup>6</sup>.

OpenAI models are non-deterministic, that is, identical inputs can yield different outputs.

Section 3.3 illustrates the limitations of the maximum number of tokens (column “**Max tokens**”). Note that the models were trained up to September 2021 or June 2021, which imposes limitations, as illustrated in Section 3.2.

The OpenAI API covers many tasks that involve understanding or generating natural language, code, or images.

The API has three key concepts:

**Prompts:** a model is “programmed” by providing some instructions or a few examples to the completions and chat completions endpoint; prompts can be used for any task, including content or code generation, summarization, expansion, conversation, creative writing, style transfer.

**Tokens:** text is processed by breaking it down into tokens; the number of tokens processed in a given API request depends on the length of both inputs and outputs.

**Models:** the API works with several models, as listed in the documentation.

As for training a model, the user can adopt few-shot learning, passing examples in the prompts, or fine-tune the model, as mentioned in Section 2. However, fine-tuning is currently only available for the GPT 3 base models.

#### 3.2 ChatGPT

ChatGPT from OpenAI was released on November 30, 2022, uses GPT 3.5, and is optimized for user dialogues. According to OpenAI<sup>7</sup>, “*ChatGPT provides articulated answers across several knowledge domains, but uneven factual accuracy has been identified as a significant drawback, and sometimes it writes plausible-sounding but incorrect or nonsensical answers*”. Training data include “Internet phenomena”, software documentation, and code. ChatGPT has limited knowledge of events that occurred after September 2021. The training process leveraged both supervised learning and reinforcement learning (Neelakantan et al., 2022). Both approaches use human trainers to improve the model’s performance. OpenAI used outsourced Kenyan workers earning less than USD 2.00 per hour to label “toxic content”.

<sup>6</sup>The relationship between the models the OpenAI API offers and those mentioned in the papers is described at <https://platform.openai.com/docs/model-index-for-researchers>

<sup>7</sup>ChatGPT: Optimizing Language Models for Dialogue. <https://openai.com/blog/chatgpt/>

The power and limitations of ChatGPT to generate dialogues are well-known. The reader is invited to test ChatGPT by running the following three generic knowledge queries, for which ChatGPT produces answers of varying quality: (*Correct answer*) “How Liz Taylor, Richard Burton, and John Hurt are related?”; (*Incorrect answer*) “Where did D. Pedro I of Brazil die?”; (*Failed answer due to lack of data*) “Tell me about the coronation of Charles III of England.”

ChatGPT Plus, the paid version of ChatGPT, was released on March 14th, 2023, uses GPT 4 (OpenAI, 2023), at a cost of USD 20.00 per month (on May 2023). According to the OpenAI announcement<sup>8</sup>, “GPT-4 is 82% less likely to respond to requests for disallowed content, is 40% more likely to produce factual responses than GPT-3.5, and can take images as input”.

### 3.3 Examples of GPT-Based Database Interfaces

Innumerable applications of ChatGPT were published in the few months after its announcement (Abdullah et al., 2022)(Fraïwan and Khasawneh, 2023), ranging from a simple application developed to help write English essays (Fitria, 2023) to more complex medical applications (Lecler et al., 2023), passing through tourism (Carvalho and Ivanov, 2023), digital entertainment [omitted], and analyzing social systems data (Wang et al., 2023).

Such applications can benefit from the fact that ChatGPT is context-aware and accepts feeding data before questions. One can append text directly to the prompt or provide a contextual set of documents using LlamaIndex (see Section 5.2). Passing data as prompts has one major limitation, though: the number of tokens may not exceed the LLM limits.

The rest of this section describes two simple examples that use ChatGPT (or the OpenAI API) to create NL database interfaces with literally no programming.

The simplest way to create a GPT-based database interface would be as follows:

1. Create a description of the relational schema and pass it as a context  $C$ .
2. Given an NL query  $N$ , ask to translate  $N$  into an SQL query  $Q$  under the context  $C$ .
3. Execute  $Q$ .

The same approach also applies to RDF datasets with RDF schemas to translate NL queries into SPARQL. GPT 3.5 (and GPT-4) has been trained for these translation tasks and performs remarkably well, as re-

ported in (Liu et al., 2023). The application could adopt LangChain (see Section 5.1) to force the execution of the SQL (or SPARQL) query.

However, this approach has three limitations:

1. The schema and the NL query must be written in the same terms. Hence, the relational schema has to be hidden with a layer of views that remaps the internal table and column names to terms that the user is familiar with.
2. The database must have a schema, which is always true for relational databases but not RDF datasets.
3. The database schema must generate fewer tokens than the maximum allowed

Furthermore, the NL query may refer to terms that denote entity names, attribute values, or data in general that occur in the database, and not just to terms that occur in the relational schema. For example, consider the NL query “Where is Khwarazm?”, which asks where the oasis region “Khwarazm” is located (the region where Muhammad ibn Musa al-Khwarizmi lived – the mathematician from whose name the term “algorithm” is derived), but omits the detail that “Khwarazm” is the name of a region (the answer is “Uzbekistan” and “Turkmenistan”). Interestingly, ChatGPT can infer the missing information and synthesize a correct SQL query. Note that passing the entire database as context would not be feasible due to prompt size limits.

To circumvent these limitations, an approach would be to use a database keyword search tool to mediate access to the database. ChatGPT (or the OpenAI API) would be used to extract a list  $K$  of keywords from an NL query  $N$  (a text block) and pass  $K$  to the database keyword search tool, which would then create an answer to  $N$ . This approach would be a simple way to provide an NL interface to the database keyword search tool.

The GPT-based natural language interface to the database keyword search tool would be as follows:

1. Create a description  $T$  of the database keyword search API and pass it as a context.
2. Given an NL query  $N$  (a text block), ask to extract a list  $K$  of keywords from  $N$ .
3. Pass  $K$  to the tool, under the context  $T$ , to generate an answer for  $N$ .

Lastly, the text-to-SQL problem, that is, the translation of NL questions to SQL queries, has been intensively studied. The Spider Web site<sup>9</sup>, a popular benchmark to assess text-to-SQL tools, maintains a

<sup>8</sup><https://openai.com/product/gpt-4>

<sup>9</sup><https://yale-lily.github.io/spider>

leaderboard of the best-performing tools. It should be noted that the best four tools use GPT-4, whereas the fifth best tool uses GPT-3.5-Turbo-0301, at significantly lower cost.

## 4 LLaMA-BASED INTERFACES

This section describes two examples that address the problem of fine-tuning an LLM to follow instructions (i.e., to become a Chat Model) over a particular database. The examples are Alpaca and GPT4ALL that create chat models by fine-tuning LLaMA.

### 4.1 The LLaMA Family

LLaMA (Touvron et al., 2023), which stands for Large Language Model Meta AI, is a collection of transformer-based large language models ranging from 7B to 65B parameters. Based on a recent work (Hoffmann et al., 2022), the authors decided to increase the number of tokens, while fixing the number of parameters. It was observed that a 7B model continues to improve even after 1T tokens, when the recommendation of (Hoffmann et al., 2022) was to train a 10B model on 200B tokens.

The results in (Touvron et al., 2023) indicate that the series of LLaMA LLMs were competitive with state-of-the-art models. In particular, LLaMA-13B outperformed GPT 3 despite being more than ten times smaller. The authors also pointed out that the accuracy notably increased by increasing the number of tokens.

The LLaMA 2 family<sup>10</sup>, announced on July 18th, 2023, changed the LLM open-source scenario. The family includes pretrained and fine-tuned language models (LLaMA Chat, Code LLaMA) — ranging from 7B to 70B parameters. Llama 2 was trained between January 2023 and July 2023, and the pre-training data has a cutoff of September 2022, but some tuning data is more recent, up to July 2023. Llama 2-Chat models are optimized for dialogue use cases and outperform open-source chat models on most benchmarks and human evaluations for helpfulness and safety. Code Llama comes in multiple flavors to cover a wide range of applications: foundation models (Code Llama), Python specializations (Code Llama - Python), and instruction-following models (Code Llama - Instruct) with 7B, 13B, and 34B parameters each.

<sup>10</sup><https://ai.meta.com/resources/models-and-libraries/llama/>

### 4.2 Examples of LLaMA-Based Database Conversational Interfaces

Alpaca (Taori et al., 2023), developed by the Stanford Center for Research on Foundation Models (CRFM), is an instruction-following model fine-tuned from the LLaMA-7B model. Alpaca adopted the self-instruct method (Wang et al., 2022) in an interesting way. The authors applied GPT 3.5 (text-davinci-003) to the 175 human-written instruction-output pairs defined in (Wang et al., 2022), generating 52K unique pairs of instructions and the corresponding outputs. The dataset generated was used to fine-tune the LLaMA-7B model using Hugging Face’s training framework. To evaluate Alpaca, the authors created a varied list of user-oriented instructions: email writing, social media, and productivity tools. The evaluation confirmed that Alpaca-7B achieved competitive performance when compared with GPT 3.5.

GPT4All (Anand et al., 2023) is an assistant-style chatbot from Nomic AI that features chat models fine-tuned from LLaMA. Nomic AI first gathered sample questions/prompts from three publicly available data sources, the unified chip2 subset of LAION OIG, coding questions with a random sub-sample of Stackoverflow Questions, and instruction-tuning with a sub-sample of Big-science/P3. These questions/prompts were handled over to GPT-3.5-Turbo to generate roughly 800,000 prompt-generation pairs. Then, the data was curated to remove low-diversity responses, ensuring that the data covered a wide range of topics, as in the Alpaca project. The final dataset contained approximately 440,000 prompt-generation pairs. Nomic AI then trained several models fine-tuned from LLaMA-7B, using LoRA (Hu et al., 2021) for four epochs.

Alpaca and GPT4ALL illustrate a much more elaborated process to create a database conversational interface that can be summarized as follows:

1. Let  $D_1, \dots, D_n$  be one or more databases and  $L$  be an LLM.
2. Create an initial set  $P$  of sample questions/prompts from the databases.
3. Create an expanded set  $P'$  of questions/prompts, perhaps using ChatGPT.
4. Create a curated set  $P''$  from  $P'$ .
5. Create a chat model  $C$  by fine-tuning  $L$  using  $P''$ .
6. The database conversational interface for  $D_1, \dots, D_n$  will be built using the chat model  $C$ .

Other examples that follow this strategy, but which are not based on LLaMA, are GPT4All-J (Anand et al., 2023) from Nomic AI, announced on

April 13, 2023, and Dolly 2.0 (Conover et al., 2023) from Databricks, announced on April 12, 2023.

Finally, following its announcement, there was a hectic series of models obtained by fine-tuning the LLaMA 2 models. The Open LLM Leaderboard<sup>11</sup>, which evaluates open LLMs and chatbots, lists a large number of models obtained by fine-tuning LLaMA 2 models, mostly obtained using LoRA–Low-Rank Adaptation (Hu et al., 2021), or QLoRA–Quantized Low-Rank Adaptation (Dettrmers et al., 2023).

## 5 GENERIC LLM FRAMEWORKS

This section introduces two frameworks, LangChain and LlamaIndex. LangChain helps isolate the development of an LLM-based database (conversational) interface from the underlying LLM, whereas the LlamaIndex is a simple, flexible interface between an LLM and external data.

### 5.1 LangChain

LangChain<sup>12</sup> is a framework to help develop LLM-powered applications. It is designed to enable two types of applications:

- **Data-aware:** connects an LLM to a data source;
- **Agentic:** allows an LLM to interact with its environment.

It also provides two main props:

- **Components** consist of modular abstractions necessary to work with language models;
- **Use-Case Specific Chains** consist of assembling components in particular ways to accomplish a particular use case. Its central idea is to “chai” components to create more advanced use cases around LLMs.

At its core, LangChain has six modules:

- **Schema** covers the basic data types and schemas used.
- **Models** represent the different types of models that are used in LangChain.
- **Prompt** refers to the input to the model and is the new way of programming models.
- **Index** refers to structuring documents so that LLMs can best interact with them. Indexes and retrieval types are currently centered around vector databases.

<sup>11</sup>[https://huggingface.co/spaces/HuggingFaceH4/open\\_llm\\_leaderboard](https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard)

<sup>12</sup><https://docs.langchain.com>

- **Memory** is the concept of storing and retrieving data in a conversation. There are two main methods: based on input, fetch any relevant pieces of data, and, based on the input and output, update the state accordingly.
- **Chain** is a generic concept that returns a sequence of modular components (or other chains) combined in a particular way to accomplish a common use case.
- **Agent** accesses a suite of tools and, depending on the user input, decides which, if any, of these tools to call.

LangChain supports application development using Python and JavaScript/TypeScript.

LangChain can help design applications such as personal assistants, chat-bots, question-and-answering (QA) over documents, summarizing long documents, extracting structured information from unstructured text, and querying tabular data. For example, (Kemper, 2023) illustrated how to use LangChain to extend GPT-based models for tasks such as QA from tabular data and QA from documents, navigating through more complex custom database schemas. For this task, a chain type called `SQLDatabaseChain`, whose main task is to provide GPT with information from custom databases, is used, as shown in the code snippet<sup>13</sup> below. Note that “gpt-3.5-turbo” is used as a parameter of the model class. A set of eight queries over the database schema was designed for testing. The queries address different challenging aspects of information retrieval. The results showed that one query had an obvious problem. The remaining seven were handled well, including queries requiring joins over three tables.

```
#creating the model object
as ChatOpenAI class
llm = ChatOpenAI(temperature=0,
    model_name="gpt-3.5-turbo")
#creating a Chain using SQLDatabaseChain
db_chain = SQLDatabaseChain(llm,
    database=db, verbose=True)
#running a query example
db_chain.run("Sum up the total revenue")
```

### 5.2 LlamaIndex

LlamaIndex<sup>14</sup>, also known as GPT Index, is a simple, flexible interface between LLMs and external data. It provides the following tools:

<sup>13</sup>Full code is available at <https://github.com/kemperd/langchain-sqlchain>

<sup>14</sup><https://gpt-index.readthedocs.io/en/latest/>

- Data connectors to existing data sources and data formats (APIs, PDFs, docs, SQL, etc.);
- Indices over unstructured and structured data for use with LLMs;
- A user interface to query the index (feed in an input prompt) and obtain a knowledge-augmented output; and
- A comprehensive toolset, trading off cost and performance.

The LlamaIndex indices store context in an easy-to-access format for prompt insertion. They help to deal with prompt limitations when the context is too big, and dealing with text splitting. Depending on the index structure adopted, LlamaIndex offers different methods of synthesizing a response (*response synthesis mode*) from the relevant context. The response synthesis modes are based on two concepts *node* and *response synthesis*. *Node* is a generic data container that contains a piece of data (e.g., a chunk of text, an image, a table, etc.) and additional meta-data about its relationship to other Node objects. *Response synthesis* synthesizes a response given the retrieved Nodes. LlamaIndex also supports synthesizing a response across heterogeneous data sources by composing a graph over the existing data.

The general workflow use of LlamaIndex is as follows:

1. Load in documents, either manually or through a data loader, using available data connectors.
2. Parse the documents into nodes.
3. Construct an index, from nodes or documents.
4. Optionally, build indices on top of other indices.
5. Query the index.

Querying an index involves three main components:

**Retriever:** retrieves a set of nodes from an index, given a query;

**Response Synthesizer:** takes a set of nodes and synthesizes an answer, given a query; and

**Query Engine:** takes a query and returns a response object.

LlamaIndex allows querying data for any downstream LLM use case, such as QA tasks, semantic search, summarization, or as a component in a chatbot. Indeed, it provides features powered by LLMs to analyze structured data through augmented text-to-SQL capabilities. The code snippet below shows a simple example of using the “text-to-SQL” conversion features. The toy table in this example, `city_stats`, contains `city/population/country` information.

```
from sqlalchemy
    import create_engine
from llama_index
    import SQLiteDatabase,
        GPTSQLStructStoreIndex
#use SQLAlchemy to setup a db connection
engine =
    create_engine("<database_information>")
. . .
# wrap the SQLAlchemy engine
# with the SQLiteDatabase wrapper
sql_database = SQLiteDatabase(engine,
    include_tables=["city_stats"])
# the db is already populated with data
# then, create the index from the table
# with a blank document list
index = GPTSQLStructStoreIndex([],
    sql_database=sql_database,
    table_name="city_stats")
query_engine = index.as_query_engine()
response =
    query_engine.query("Which city
        has the highest population?")
print(response)
```

The derived SQL query can be accessed by the code line

```
response.extra_info["sql_query"]
```

which shows something like

```
SELECT city_name, population
FROM city_stats
ORDER BY population DESC LIMIT 1
```

## 6 CONCLUSIONS

This paper argued that LLMs can be profitably used to construct natural language database (conversational) interfaces. The discussion focused on three problems: (1) How to use an LLM to create an NL database interface; (2) How to fine-tune an LLM to follow instructions (i.e., to become a Chat Model) over a particular database; and (3) How to simplify the construction of LLM-based database (conversational) interfaces. Sections 3, 4, and 5 briefly addressed these problems with the help of examples.

The paper left out a discussion on Retrieval Augmented Generation (RAG), for lack of space. RAG does not require model fine-tuning but combines a prompt engineering approach based on vector embeddings to encode the data, stored in a vector store.

The take-home lessons can be summarized as follows. The OpenAI API and ChatGPT offer interesting facilities to create NL database interfaces which

greatly simplifies the interpretation of NL queries, the generation of NL answers, and the control of user dialog. The fine-tuning of an LLM to follow instructions generated from a data source is far more challenging. LangChain offers a useful framework to isolate the development of the database interface from the underlying LLM, and the LlamaIndex provides facilities to connect an LLM and a data source.

Lastly, the leaderboards mentioned before provide references to the latest evolutions in LLM applications related to the topics of this paper.

## REFERENCES

- Abdullah, M., Madain, A., and Jararweh, Y. (2022). Chatgpt: Fundamentals, applications and social impacts. In *2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS'22)*, pages 1–8. doi: 10.1109/SNAMS58071.2022.10062688.
- Affolter, K., Stockinger, K., and Bernstein, A. (2019). A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28.
- Anand, Y. et al. (2023). Gpt4all-j: An apache-2 licensed assistant-style chatbot. [https://static.nomic.ai/gpt4all/2023\\_GPT4All-J\\_Technical\\_Report\\_2.pdf](https://static.nomic.ai/gpt4all/2023_GPT4All-J_Technical_Report_2.pdf).
- Askell, A. et al. (2021). A general language assistant as a laboratory for alignment. *arXiv*. doi: 10.48550/arXiv.2112.00861.
- Caldarini, G., Jaf, S., and McGarry, K. (2022). A literature survey of recent advances in chatbots. *Information*, 13(1).
- Carvalho, I. and Ivanov, S. (2023). Chatgpt for tourism: applications, benefits, and risks. *Tourism Review*. doi: 10.1108/TR-02-2023-0088.
- Conover, M. et al. (2023). Free dolly: Introducing the world's first truly open instruction-tuned llm. <https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm>. online: May 22, 2023.
- Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms.
- Diefenbach, D., Both, A., Singh, K., and Maret, P. (2020). Towards a question answering system over the semantic web. *Semant. Web*, 11(3):421–439.
- Fitria, T. N. (2023). Artificial intelligence (ai) technology in openai chatgpt application: A review of chatgpt in writing english essay. *ELT Forum: Journal of English Language Teaching*, 12(1):44–58. doi: 10.15294/elt.v12i1.64069.
- Fraivan, M. and Khasawneh, N. (2023). A review of chatgpt applications in education, marketing, software engineering, and healthcare: Benefits, drawbacks, and research directions. *arXiv*. doi: 10.48550/arXiv.2305.00237.
- Hoffmann, J. et al. (2022). Training compute-optimal large language models. *arXiv*. doi: 10.48550/arXiv.2203.15556.
- Hu, E. J. et al. (2021). Lora: Low-rank adaptation of large language models. *arXiv*. doi: 10.48550/arXiv.2106.09685.
- Jurafsky, D. and Martin, J. H. (2023). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall. Third Edition draft.
- Kemper, D. (2023). Querying complex database schemas with gpt and langchain. <https://analytix.nl/post/querying-complex-database-schemas-with-gpt-and-langchain>. online: May 22, 2023.
- Lecler, A., Duron, L., and Soyer, P. (2023). Revolutionizing radiology with gptbased models: Current applications, future possibilities and limitations of chatgpt. *Diagnostic and Interventional Imaging*. doi: 10.1016/j.diii.2023.02.003.
- Liu, A., Hu, X., Wen, L., and Yu, P. S. (2023). A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability.
- Manning, C. D. (2022). Human Language Understanding & Reasoning. *Daedalus*, 151(2):127–138. doi: 10.1162/daed\_a.01905.
- Motger, Q., Franch, X., and Marco, J. (2022). Software-based dialogue systems: Survey, taxonomy, and challenges. *ACM Comput. Surv.*, 55(5).
- Neelakantan, A. et al. (2022). Text and code embeddings by contrastive pre-training. doi: 10.48550/ARXIV.2201.10005.
- OpenAI (2023). Gpt-4 technical report. *arXiv*. doi: 10.48550/arXiv.2303.08774.
- Ouyang, L. et al. (2022). Training language models to follow instructions with human feedback. In *Proceedings of the Advances in Neural Information Processing Systems 35 (NeurIPS'22)*. online: May 25, 2023.
- Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. (2023). Stanford alpaca: An instruction-following llama model. [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca).
- Touvron, H. et al. (2023). Llama: Open and efficient foundation language models. *arXiv*. doi: 10.48550/arXiv.2302.13971.
- Tunstall, L., von Werra, L., and Wolf, T. (2022). *Natural Language Processing with Transformers, Revised Edition*. O'Reilly Media, Inc.
- Vaswani, A. et al. (2017). Attention is all you need. In *Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS'17)*, volume 30.
- Wang, F.-Y. et al. (2023). Chatgpt for computational social systems: From conversational applications to human-oriented operating systems. *IEEE Transactions on Computational Social Systems*, 10(2):414–425. doi: 10.1109/TCSS.2023.3252679.
- Wang, Y. et al. (2022). Self-instruct: Aligning language model with self generated instructions. doi:10.48550/arXiv.2212.10560.