Can Language Models Enable In-Context Database?

Yu Pan
University of Nebraska-Lincoln
Lincoln, USA
yu.pan@unl.edu

Hongfeng Yu
University of Nebraska-Lincoln
Lincoln, USA
hfyu@unl.edu

Tianjiao Zhao
East Carolina University
Greenville, USA
zhaot22@ecu.edu

Jianxin Sun
University of Nebraska-Lincoln
Lincoln, USA
s-jsun5@unl.edu

***Abstract*—Large language models (LLMs) are emerging as in-context learners capable of handling a variety of tasks, including comprehension, planning, reasoning, question answering, arithmetic calculations, and more. At the core of these capabilities is LLMs' proficiency in representing and understanding structural or semi-structural data, such as tables and graphs. Numerous studies have demonstrated that reasoning on tabular data or graphs is not only feasible for LLMs but also gives a promising research direction which encodes these types of data as in-context data. The lightweight and human readable characteristics of in-context database can potentially make it an alternative for the traditional database in typical RAG (Retrieval Augmented Generation) settings. However, almost all current work focuses on static in-context data, which does not allow dynamic update. In this paper, to enable dynamic database update, delta encoding of database is proposed. We explore how data stored in traditional RDBMS can be encoded as in-context text and evaluate LLMs' proficiency for CRUD (Create, Read, Update and Delete) operations on in-context databases. A benchmark named InConDB is presented, and extensive experiments with model fine-tuning are conducted to show the performance of different language models in enabling in-context database by varying the database encoding method, prompting method, operation type and input data distribution, revealing both the proficiency and limitations.**

***Index Terms*—In-Context Database, Large Language Model**

I. Introduction

The recent surge in enthusiasm for LLMs (Large Language Models) stems from their growing ability to handle and interpret textual data, as demonstrated by notable studies from [1]–[8]. Originally designed to process sequential text, these models have now been adapted to performing a wide range of tasks across different modalities, such as voices, images and videos [7], [9], [10]. Modern LLMs are evolving as few-shot (or zero-shot) learners [5] capable of handling a variety of tasks, ranging from question answering, semantic comprehension, planning, reasoning, arithmetic calculations and more [11]–[18]. Collectively, these developments have reinforced the belief that LLMs are critical milestones on the journey toward achieving artificial general intelligence (AGI) [19]. Researchers believe LLMs' capability rely on their representations and comprehension of various structures from outside world, which can be collected and expressed as structural data such as graphs and tables.

Recently numerous studies have shown that LLMs are capable of doing in-context reasoning on graphs [20]–[24] and tables [25]–[30]. Given the structural data and the description of a task, possibly with extra examples and instructions, LLMs need to reply with the correct result of the task on the structural data. By serializing the structural data using various encoding scheme and adopting different prompting technologies, these work try to find out the optimal protocol through which the input can maximize LLMs' capability of reasoning on structural data. However, almost all of the exiting work does not take into account of the dynamic feature of the structural data, which may be updated frequently in the actual application scenarios. For instance, the nodes and edges may be added/removed to/from the graph, and new knowledge will be inserted into the knowledge graph while outdated knowledge will be removed. Similarly, tabular data may also be edited to reflect the updated situation. So the question arises naturally: can LLMs enable in-context update of structural data? In this case, not only a querying task is allowed as input to LLMs, but an updating task will also be allowed. If the update is acceptable and does not violate any constraint of the structural data, LLMs need to register the update, otherwise LLMs need to reply with an error message. In this setting, LLMs should not only be capable of comprehending the structures of the data, but also understanding the updates along the temporal dimension.

As the context length of LLMs increases rapidly with some technologies being proposed to even scale the context to infinitely long [31], it is promising that LLMs' context window can accommodate way larger dataset in the near future. Combined with LLMs' in-context reasoning capability on dynamic structural data, we believe it is possible for LLMs to enable in-context database, which is a lightweight alternative of traditional database in which CRUD (Create, Read, Update and Delete) operations are handled by LLMs, rather than pre-programmed procedures as in traditional DBMS. In the prevalent settings of RAG (Retrival Augmented Generation), where the whole dataset is stored on external database and sampled as required by the specific task, in-context database provide an alternative solution. Also the adoption of in-context database and traditional database in RAG is not necessarily exclusive, they can complement with each other. For instance, external database can store the full dataset which has lower rate of updates whereas in-context database can act as a partial cache which is also more fresh. In a typical multi-agent [32], [33] configurations, there can be an agent which is mainly responsible for data management, which act as a database, but can understand and execute queries more intelligently.

In this work, we perform the first comprehensive evalu-

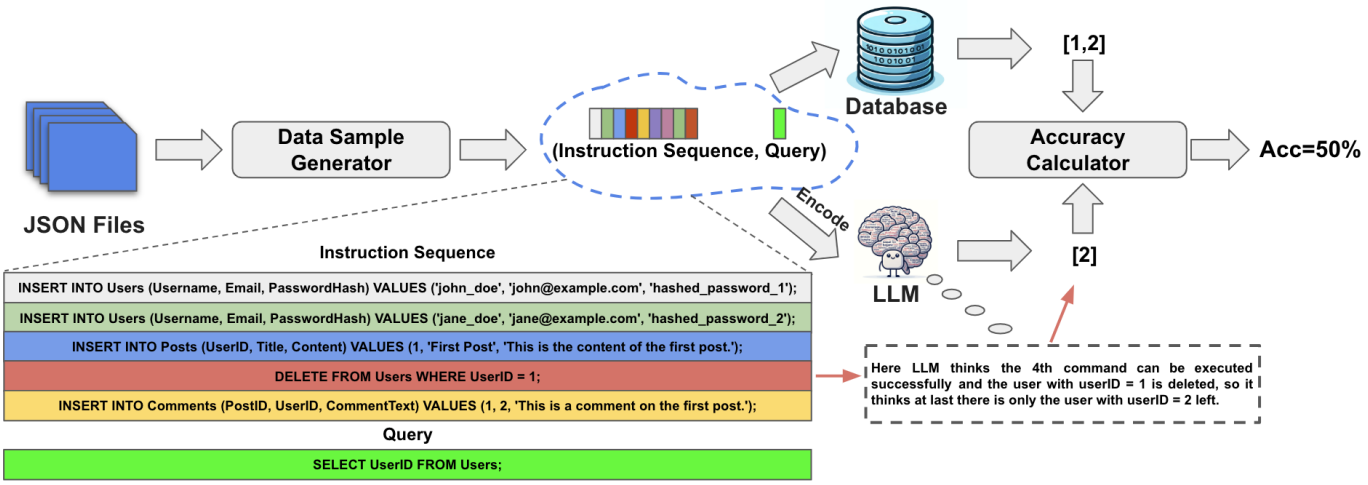


Fig. 1: Overview of our evaluation framework. First we compose several JSON files, each containing a bunch of CRUD operations for one database schema. Then we use a data sampler to generate samples of (instruction sequence, query) pair. The instruction sequence is used to imitate the daily database operations which constantly come in and change the status of the database. We can also consider the instruction sequence as a representation of the current status of the database. Then a query is used get the result on the current status of the database. We use real database such as MySQL to execute the instruction sequence followed by the query to get the ground truth query result. As another branch, we send the prompting-decorated encoding of the (instruction sequence, query) pair to the large language model, and ask the LLM to imitate a database to execute all the instructions and the query to get a result. Finally we use an accuracy calculator to get the accuracy score measuring the discrepancy between ground truth and LLM-generated query result. In this illustration, the LLM thinks the 4th command (delete operation) can be executed successfully, without noting it violates the foreign key constraint.

ation about how the data stored in the traditional RDBMS database can be encoded as text and LLMs' proficiency for CRUD operations on in-context database. A benchmark named InConDB is proposed which includes dataset and queries for RDBMS databases. We conduct extensive experiments to demonstrate the performance of language models in enabling in-context database depends on various factors such as database encoding method, prompt engineering, operation type and data distribution. Both the proficiency and limitations of current LLMs are revealed. Figure 1 illustrates the evaluation framework.

Our contribution can be summarized as follows:

- For the first time, we propose the concept of LLM based in-context database, which can be a lightweight alternative for the traditional database in a RAG setting.
- To our knowledge, our research is the first to evaluate LLMs as a full-fledged relational DBMS, which not only evaluates their capability of reasoning on structures but also on temporal updates.
- We create a benchmark named InConDB to evaluate the performance of different LLM models in enabling in-context database, by varying different input factors such as database encoding method, prompting method, operation type and input data distribution.
- The experiment results are not only valuable in evaluating language models' performance as in-context database, but also valuable in evaluating large language model's capability of understanding the accumulating effects of historical events in a more general perspective.

This paper is organized as following sections. Section II presents the framework of in-context database. Section III shows the experimental results.Section IV introduces the closely related topics to our research. Section V concludes the paper.

II. Framework

A. Problem Formalization

Let f represents the function of a language model, in which $f : W \mapsto W$ maps the input tokens to output tokens, both from the set of all series of tokens W. A relational database $D = \{T_i | i = [1, N]\}$ contains N tables T_i, where $i \in [1, N]$. Q represents a CRUD (Create, Read, Update and Delete) operations on database D. And $S = s(D, Q)$ denotes the ground truth solution when performing query Q on database D, where s represents a real world database management system (DBMS). For select operations, S will be the query result of Q and for modification operations such as insert, update or delete, S will be either "Succeed" when the query succeeds, or "Fail" when the query fails.

To enable the in-context database, we present both the database D and the query Q to the language model. To optimize the performance of language model, we introduce encoding method d of database D and query Q, where d can be either SQL or natural language (NL), which means the database D and query Q can be described in either SQL commands or in natural language. Also we introduce prompting method p, which could

be one of the three methods: zero-shot, zero-shot-COT (zero-COT) and few-shot introduced in IV-B. Then the solution generated by the language model f is $f(p(d(D), d(Q)))$.

Given a evaluation set of tuples $\mathcal{D} = \{(D, Q)\}$, now our problem turns to maximize the expected accuracy score of the solution generated by f, w.r.t the encoding methods d and the prompting method p, which can be written as follows:

$$\max_{p,d} \mathbb{E}_{D,Q \in \mathcal{D}} \text{Acc}[f\ (p(d(D), d(Q)), s(D, Q))] \quad (1)$$

B. Delta Encoding of Database

Different from existing works which use plain tables to represent tabular data or relational databases, here in our paper we use "delta" encoding. That is, we use a sequence of commands to represent the current status of the database, in which each command encodes a minor modification of the status of the database in previous step. Then $d(D)$ is a command sequence $[C_1, ..., C_i, ..., C_k]$, and $d(Q) = C_j$, where $C_i, C_j \in \mathcal{I}$ and $\mathcal{I}$ is the set of all database operations. Formally, we can get the following equation:

$$D = d^{-1}([C_1, ..., C_i, ..., C_k]) = C_k(...(C_i(...C_1(\emptyset)))) \quad (2)$$

where $\emptyset$ represent the initial status of all databases containing no data. The current status of database D is equivalent to the composition function of C_i for i from 1 to k, starting from empty database.

The benefit of using delta representation is obvious: new database operation can be appended to the end of historical command sequence immediately. It is very convenient to represent the current status of a database in such accumulative fashion, without any real database operations. So the insert, update or delete will always be an $O(1)$ operation. The cost for the delta encoding is that it requires high reasoning capability of language models, to infer the current status of the database from the command sequence D and conduct the query Q on it accordingly. This process not only requires large language model to understand the structure of the database, but decipher the changes along the time axis as well. So generally speaking, our work also evaluate large language model's capability to understand the accumulating effects of historical events.

C. Evaluation Framework

To evaluate the performance of language models as in-context database, for various combinations of encoding methods and prompting methods, we propose a two-branch evaluation framework illustrated in Figure 1. First we prepare a bunch of JSON files, each containing a bunch of CRUD operations for one database schema, then a data sample generator is utilized to sample a tuple: (instruction sequence D, query Q). The sampled tuple (D,Q) is subject to a distribution $\mathcal{D}$:

$$(D, Q) \sim \mathcal{D}(l, b, o) \quad (3)$$

in which the distribution $\mathcal{D}$ has parameters l, b, o. l represents the number of commands in the command sequence D, b represents the ratio of insert operations in the whole command sequence, and o represents the overlap ratio between insert operations and non-insert operations in the command sequence. Figure 2 illustrates the concepts of these three parameters. In our experiment, we'll evaluate the performance of language models as in-context database, by varying these three parameters. Intuitively, by increasing the length of the command sequence, the difficulty of reasoning on the input sequence will also increase. We are also interested in the impact of varying the ratio of the insert operations in the whole command sequence, on the performance of the in-context database. Here the intuition is a command sequence containing pure insert operations may be easier than a command sequence containing some non-insert operations. Similarly, increasing the overlap between insert and non-insert operations in the command sequence may pose extra burden for language models' reasoning performance. After all, the interleaving of insert and non-insert operations may take the language model more efforts to figure out what is happening along the way.

After we get the tuple (D,Q), the evaluation will split into two branches. In the upper branch, we feed the original tuple (D,Q) into a real world database s, such as MySQL, to get the query result $s(D, Q)$; in the lower branch, we feed the the encoded tuple $(d(D), d(Q))$, decorated with prompting method p, into the large language model f, and ask it to mimic the behavior of an RDMS to get the query result $f(p(d(D), d(Q)))$. Finally an accuracy calculator will compare and calculate the accuracy score, reflecting the discrepancy between the true query result $s(D, Q)$ and language model's result $f(p(d(D), d(Q)))$.

For insert, update or delete operations, since the query result is in ["Succeed","Fail"], the accuracy is calculated as follows:

$$\text{Acc}(f, s) = \alpha Acc(f, "Succeed") + (1 - \alpha) Acc(f, "Fail") \quad (4)$$

Here we use f and s are abbreviated for $f(p(d(D), d(Q)))$ and $s(D, Q)$ respectively. Since the cases of "Succeed" and "Fail" are unbalanced, the weight α is introduced to re-balance the accuracy score.

For select operations, the query result is a set of integers, strings or objects, the accuracy is calculated as follows:

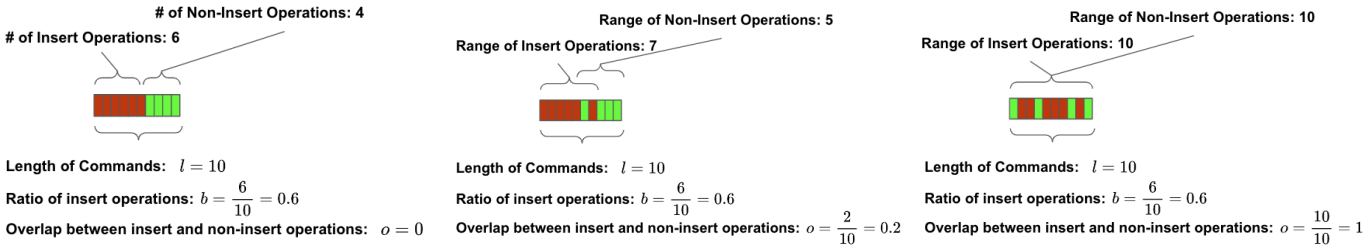
$$\text{Acc}(f, s) = \beta J(f, s \neq \emptyset) + (1 - \beta) J(f, s = \emptyset) \quad (5)$$

where J is the Jaccard coefficient between two sets, measuring the discrepancy of f and s. Again a weight β is introduced to re-balance the accuracy score between empty query results and non-empty query results.

D. Prompting and Encoding

As introduced in II-A, we evaluate the performance of different languages models by varying the combination of encoding and prompting methods. There are 2 encoding methods: SQL and NL (natural language) and 3 prompting methods: zero-shot, zero-COT and few-shot.

Figure 3 illustrates a case of the model input and output for encoding method of SQL and prompting method of zero-shot. All the commands will be given in SQL syntax. In the system prompt, we instruct the language model to imitate a



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 2: Illustrations of length of command sequence l, the ratio of insert operations b and the overlap o between insert and non-insert operations. In Case 1, the insert (in red) and non-insert operations (in green) have no overlap, so o=0. In Case 2, the range of insert and non-insert operations have overlap of 2, thus we calculate o as 2 over the union of their range: 10, so $o = \frac{2}{10} = 0.2$. Similarly, in Case 3, the overlap of insert and non-insert operations is 10, so $o = \frac{10}{10} = 1$.

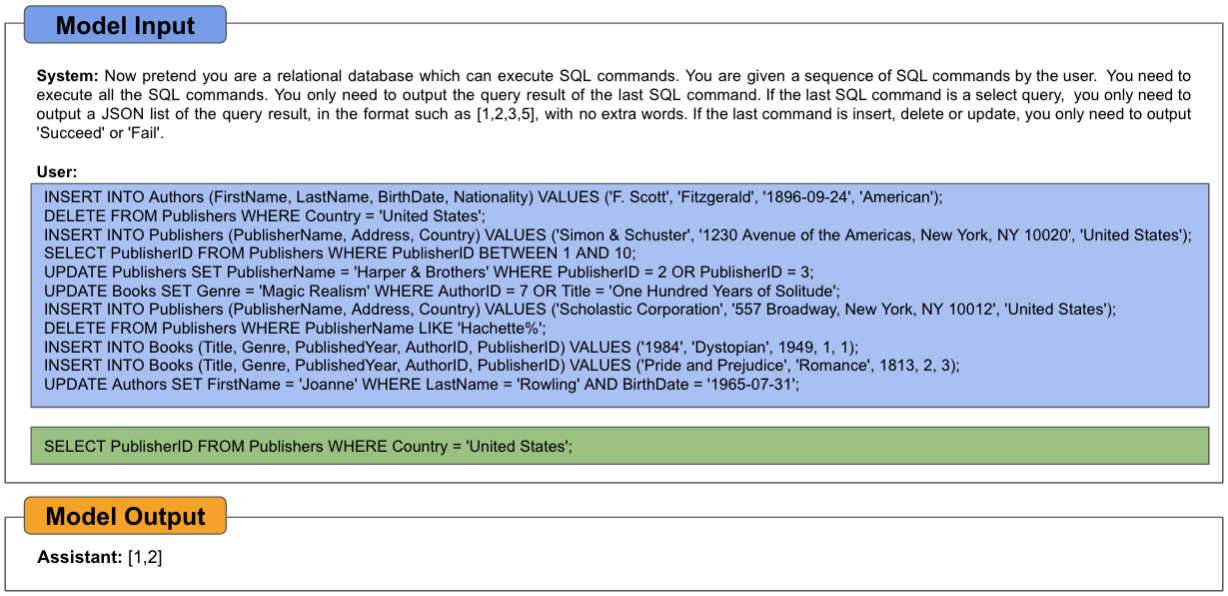


Fig. 3: An example of model input and output for encoding method: SQL and prompting method: zero-shot

relational database and we also specify the format of user input and model output. Since this case is zero-shot, the user is expected to feed in the SQL instruction sequence (in shallow blue) and query (in shallow green) at once, without any model output in between each command. We can see the instruction sequence contains insert, delete, select and update queries. The language model is required to synthesize the execution of all the commands and directly reply with the correct query result. In the illustration, the query is a "SELECT" query, and the model outputs the IDs of publishers in "United States".

Figure 4 illustrates a case for encoding method of natural language and prompting method of zero-shot. Instead of following rigorous SQL syntax, All the commands from the user will be described in natural language. Similar to the case illustrated in Figure **??**, in the model input, the corresponding output from the model (in orange) will be immediately after each user's command (in shallow blue). The language model is also only required to output the correct query result.

In our evaluation framework, we explore different combinations of prompting and encoding methods and check the corresponding performance. Next section will show more about the experiment results and we'll discuss about the implications of the results.

III. Experiments

A. Experiments Configuration

1) Benchmark: As introduced in II-C, We create 20 JSON files, each contains the schema and CRUD operations for a relational database, which contains 3-5 tables, some of which have foreign keys referencing other tables. Each database file contains several insert, delete, update and select operations, and the select operations are further divided into several categories such as select queries with different number of filtering conditions, joint queries with different number of tables, range queries, queries with ranking, queries with count. In the following experiments, we evaluate the performance of language models by these different types of database operations.

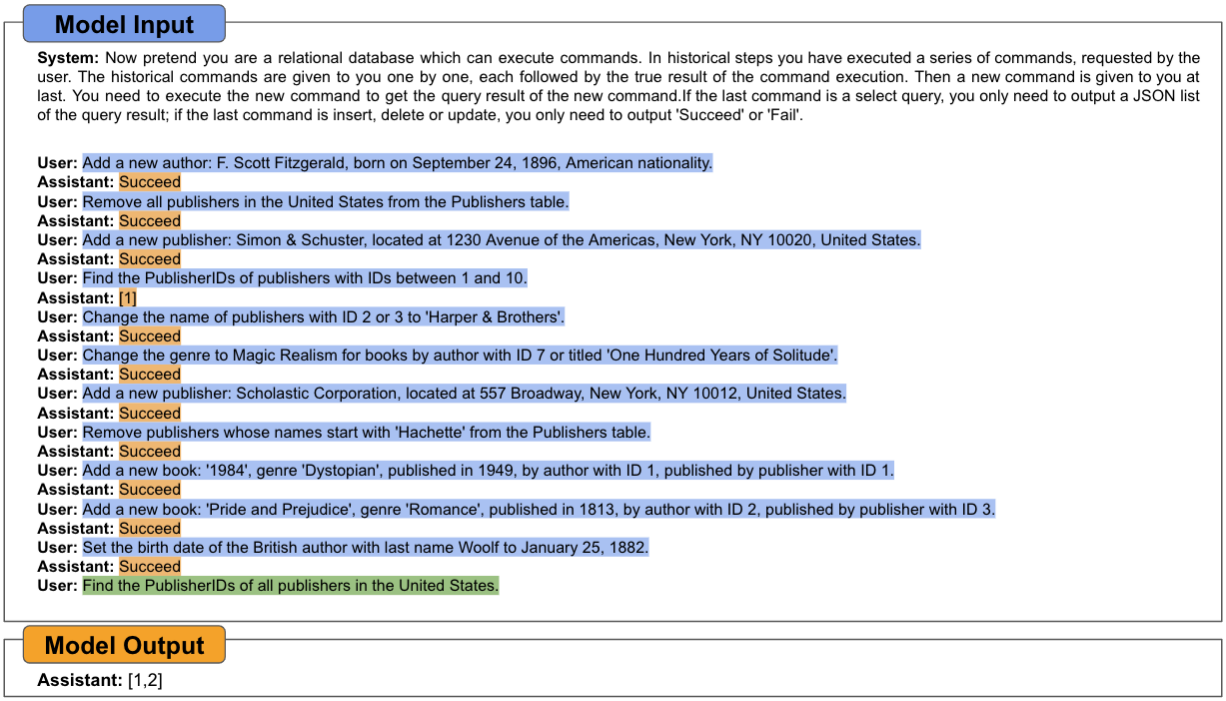


Fig. 4: An example of model input and output for encoding method: natural language (NL) and prompting method: few-shot

The data sample generator will sample the tuples of database and query (D,Q) from the 20 databases, subject to the distribution $\mathcal{D}(l, b, o)$, where l is the length of the command sequence, b is the ratio of insert operations, and o is the overlap between insert and non-insert operations. In the following experiments, we'll vary these three parameters and evaluate the performance of language models under the corresponding data distribution. The 20 databases, together with the evaluation framework constitute the benchmark called InConDB, which can be used to evaluate the performance of the large language model's performance as in-context database. The link for the repository of InConDB can be found in this link [34].

2) Models: For evaluation, we choose 5 large language models: GPT-4o, LLama3.1-8B, Mistral, Gemma2-9B and LLAma3.2-3B. We also fine-tune LLama3.1-8B using the data sampled from the InConDB benchmark. For GPT-4o [35], we call its official API. For other open source models, we use the model server ollama [36] for evaluati on. When we run our experiments, we use temperature = 0.5 for all the models.

3) Fine-Tuning: LLama-Factory [37] is utilized for supervised fine-tuning LLama3.1-8B. The fine-tuning method is 4-bit QLoRA. Other parameters are listed as follows: learning rate is $5e^{-5}$, the number of training epochs is 4 and the number of training samples is 600. All the training data is sampled subject to the same distribution $\mathcal{D}$ introduced in II-C, with the number of input commands l in $[10, 100]$, the ratio of insert operations b set to 0.5 and the overlap between insert and non-insert operations o set to 0.5.

4) Hyper Parameters: In our experiments, when calculating the accuracy, we set the hyper parameter α and β introduced in II-C as 0.5 and 0.9 respectively.

5) Platform: We run the all the experiments on a workstation with AMD Ryzen Threadripper PRO 3995WX 64-Cores with 2.7GHz frequency, 512GB RAM, NVIDIA RTX A5500 GPU with 24G memory. The host OS is Windows 11 and we use the virtual machine WSL ubuntu to run all the experiments.

B. Experiment 1. Comparing Language Models

In this experiment, we evaluate and compare the performance of different language models for CRUD database operations: update, delete, insert and select, by varying different combinations of prompting and encoding methods. We evaluate 3 types of prompting methods: zero-shot, zero-COT and few-shot, and 2 types of encoding methods: SQL and NL. The exact meaning of the prompting and encoding methods are already introduced in II-D. We choose l, b and o defined in II-C as 100, 0.5 and 0.5 respectively. For each parameter setting, we sample 300 tuples of (D,Q), calculate the accuracy defined in II-C, and get the average accuracy for all 300 tuples. Since we only fine-tune LLama3.1-8B for the prompting method of few-shot, so we merely show the result for this case. We render the background color intensity based on the accuracy.

Figure 5 illustrates the comparison of different models for CRUD database operations, for prompting method of few-shot and encoding method of SQL. We group the data by each operation. For insert and select operations, GPT-4o and the fine-tuned LLama3.1-8B are the top 2 models; for update, the situations are different that Gemma2 outperforms other models and fine-tuned LLama3.1-8B is the second; for delete operation, all the performance of all the models, except Mistral, is close

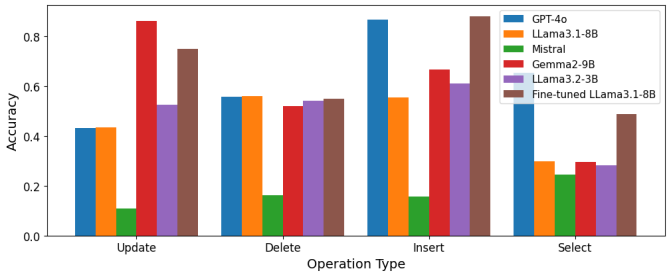


Fig. 5: Model Performance Comparison by Operation Type

to each other. The performance of Mistral is the lowest for all the CRUD operations.

The comprehensive evaluation results are listed in Table I. The best case for each operation is enclosed in yellow box. Generally speaking, GPT-4o has best performance and our fine-tuned LLama3.1-8B also has great performance by using few-shot prompting. Gemma2 only performs well when few-shot is adopted. LLama3.2-3B performs a little better than LLama3.1-8B. Mistral performs worst among all the models.

We can also see few-shot is the best prompting method for all the models. To our surprise, zero-shot outperforms zero-COT for all the models, we think the reason is that in COT, we asked the model to output the correct answer in the last line, which may be too rigorous (see **??**). So even the thinking process of a model is correct, it may still fail to output the correct result in the last line.

As for the encoding methods, for zero-shot or zero-COT, natural language (NL) performs similarly or even better than SQL encoding when the query is update, delete and insert. It looks when there's no example of results given, natural language can describe the query more clearly for update, select and insert. But for select query, SQL outperforms natural language because SQL can define the output format of a select query more easily. For few-shot, natural language generally performs worse than SQL. We think this is because by giving historical query-answer pairs, SQL has more expressive capability than natural language.

C. Experiment 2. Comparing Data Retrieval Methods

In this experiment, we evaluate and compare the performance of different language models for 10 categories of select operations such as select queries with 0-3 filtering conditions in the "where" statement, join queries with 1-3 tables, range queries, select queries with "order by" predicates, and queries with "count", by varying different combinations of prompting and encoding methods. We evaluate 3 types of prompting methods: zero-shot, zero-COT and few-shot, and 2 types of encoding methods: SQL and NL. We choose l, b and o defined in II-C as 100, 0.5 and 0.5 respectively. For each parameter setting, we sample 300 tuples of (D,Q), calculate the accuracy defined in II-C, and get the average accuracy for all 300 tuples. Since we only fine-tune LLama3.1-8B for the prompting method of few-shot, so we only show the result for this case. We render the background color intensity based on the accuracy.

TABLE I: Comparison of the performance of different language models as in-context database on 4 CRUD operations, 3 prompting methods and 2 encoding methods.

Prompting	Encoding	Update	Delete	Insert	Select
GPT-4o					
ZERO-SHOT	SQL	22.73%	23.06%	21.33%	42.01%
	NL	49.83%	28.54%	23.45%	20.64%
ZERO-COT	SQL	19.35%	16.41%	18.38%	35.84%
	NL	30.77%	24.33%	20.9%	17.77%
FEW-SHOT	SQL	43.14%	55.77%	86.89%	65.39%
	NL	87.96%	45.17%	82.88%	52.23%
LLama3.1-8B					
ZERO-SHOT	SQL	4.31%	8.65%	6.55%	3.28%
	NL	5.26%	7.99%	7.70%	1.66%
ZERO-COT	SQL	2.35%	2.81%	2.19%	3.17%
	NL	2.49%	2.77%	2.84%	1.58%
FEW-SHOT	SQL	43.48%	55.97%	55.57%	29.96%
	NL	39.46%	47.29%	50.36%	28.85%
Mistral					
ZERO-SHOT	SQL	1.78%	1.97%	1.94%	1.40%
	NL	1.94%	2.25%	2.19%	0.72%
ZERO-COT	SQL	0.86%	1.13%	0.9%	1.28%
	NL	0.91%	1.65%	1.33%	0.54%
FEW-SHOT	SQL	11.0%	16.29%	15.66%	24.52%
	NL	14.67%	39.75%	24.51%	24.13%
Gemma2-9B					
ZERO-SHOT	SQL	6.96%	14.28%	13.02%	4.82%
	NL	8.49%	20.74%	15.4%	2.34%
ZERO-COT	SQL	2.69%	2.4%	4.01%	4.90%
	NL	2.59%	3.74%	3.35%	2.39%
FEW-SHOT	SQL	86.32%	52.12%	66.72%	29.65%
	NL	46.17%	49.12%	77.15%	29.79%
LLama3.2-3B					
ZERO-SHOT	SQL	6.88%	17.25%	10.20%	4.11%
	NL	7.32%	19.36%	12.90%	2.15%
ZERO-COT	SQL	2.68%	2.12%	3.12%	3.53%
	NL	3.02%	3.07%	2.89%	1.79%
FEW-SHOT	SQL	52.52%	54.13%	61.21%	28.25%
	NL	56.91%	56.06%	63.10%	17.20%
Finetuned LLama3.1-8B					
FEW-SHOT	SQL	75.0%	55.0%	88.20%	48.82%
	NL	49.83%	56.21%	90.24%	46.41%

Table II contains the detailed results for each model. The best performer for each category of select operation in each column is enclosed in yellow box.

GPT-4o dominates almost all the categories, except fine-tuned LLama3.1-8B performs best for count operation. All the best performance happens when few-shot prompting and SQL encoding are adopted. GPT-4o also outperforms other models when zero-shot and zero-cot are adopted. The fine-tuned LLama3.1-8B is second to GPT-4o, which is impressive given its model scale is way smaller than GPT-4o. The performance

of Gemma2-9B and LLama3.2-3B are close to each other. Still Mistral performs worst among all the models.

As the number of filtering conditions increases, the performance generally decreases. This agrees with our intuition, because more filtering conditions generally means more complicated query. The only exception is that the performance for 3-Filter is better than 2-Filter, 1-Filter or even 0-Filter. This is because select queries with 3-Filter normally return empty set of results, which reduces the prediction complexity.

Similarly, when the number of joined tables increases, the performance generally decreases, for more tables means more complicated queries. 3-Table query has some exception, in which its performance outperforms 2-Table or 1-Table. We think the reason is still the same as the filtering case: 3-Table joining frequently returns empty set of results, which is easier for the models to predict.

Range queries perform similar to filtering operations, and rank queries perform worse than range queries. Count queries perform the worst. Even GPT-4o can not count right for most of the count queries. Surprisingly, fine-tuned LLama3.1-8B performs pretty good for count queries.

When comparing encoding methods, natural language (NL) performs worse than SQL, for all the prompting methods and across all the models. We believe this is because SQL can define the select query, together with the expected output format more precisely than natural language.

D. Experiment 3. Varying Input Scale

In this experiment, we evaluate the performance of language models by varying the input scale. Figure 6 illustrates the performance of different models as a function of input scale. We change the input scale l from 10 commands to 400 commands, and fix the encoding method to SQL, prompting method to few-shot and query method to no-filtering select query. We choose b and o as 0.5 and 0.5 respectively. For each input scale and model, we sample 300 (D,Q) pairs and calculate the average accuracy for all the pairs.

When the input scale increases, the performance of all models drops for all models. The trend of dropping is slower when the number of commands exceeds 250, or the accuracy approaches 20%. GPT-4o and fine-tuned LLama3.1-8B is the best performer among all the models, except when the input scale exceeds 250, the performance of latter drops drastically. We think this is because we fine tune LLama3.1-8B with training data from l in $[10, 100]$, so when the input scale exceeds certain bound, the data distribution becomes too different than the distribution of training data and thus can be considered as OOD data. But still, we can observe for $l \in [100, 200]$, fine-tuned LLama3.1-8B performs as expected, which proves the fine-tuning can be generalized when l exceeds 2x the upper bound of l in the training data.

E. Experiment 4. Varying the Ratio of Insert Operations

In this experiment, we evaluate the performance of language models by varying the ratio of insert operations. Figure 7 illustrates the performance as a function of the ratio of insert

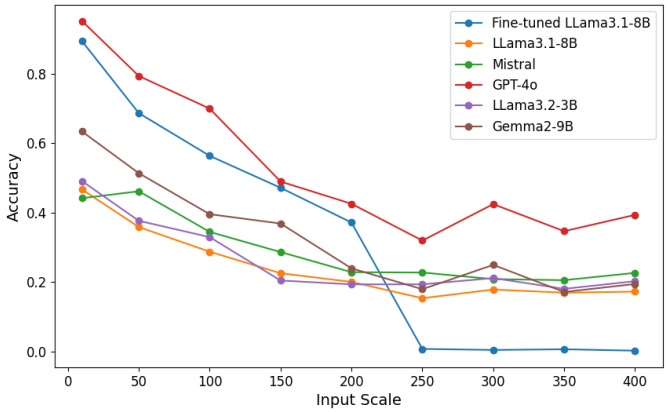


Fig. 6: Model Performance vs Input Scale

operations b. We change the ratio of insert operations b from 0 to 1, and fix the encoding method to SQL, prompting method to few-shot and query method to no-filtering select query. We choose b and o as 0.5 and 0.5 respectively. For each parameter setting, we sample 300 (D,Q) pairs and calculate the average accuracy for all the pairs.

We can see when the ratio of insert operations increases, the performance of all models increases, which agrees the intuition, since larger ratio of insert operations means less data complexity. After all, the language model can figure out which data is in the database when there are less update, delete operations adulterated in the command sequence.

Still GPT-4o and fine-tuned LLama3.1-8B is the best performer among all the models and Mistral performs worst. When the ratio of insert operations approaches 1, the accuracy of GPT-4o can almost approach 1 and fine-tuned LLama3.1-8B can approach 0.8.

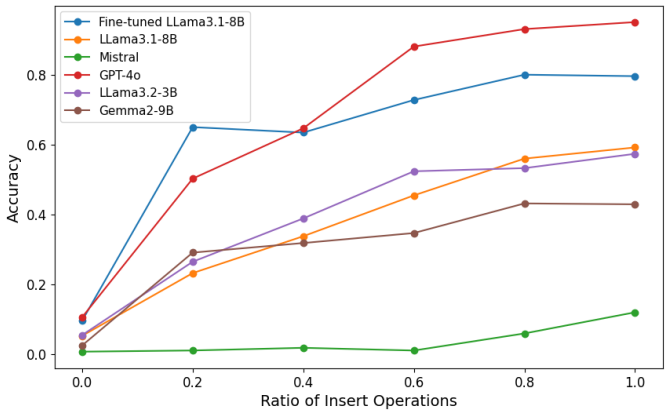


Fig. 7: Model Performance vs Ratio of Insert Operations

F. Experiment 5. Varying the Overlap between Insert and Non-Insert Operations

In this experiment, we evaluate the performance of language models by varying the overlap between insert and non-insert operations. Figure 8 illustrates the performance as a function

TABLE II: Comparison of the performance of different language models as in-context database on various categories of select operations, 3 prompting methods and 2 encoding methods.

Prompting	Encoding	0-Filter	1-Filter	2-Filter	3-Filter	Range	Rank	Count	1-Table	2-Table	3-Table
GPT-4o											
ZERO-SHOT	SQL	53.31%	52.15%	45.28%	43.56%	49.55%	27.95%	25.20%	53.06%	37.37%	32.65%
	NL	36.40%	27.28%	23.81%	18.34%	29.49%	20.02%	0.2%	27.82%	13.08%	10.0%
ZERO-COT	SQL	45.35%	44.63%	38.33%	38.80%	40.58%	23.64%	21.76%	44.75%	33.26%	27.34%
	NL	32.13%	22.80%	21.37%	15.42%	25.06%	16.25%	0.15%	24.53%	11.54%	8.49%
FEW-SHOT	SQL	78.52%	67.87%	64.42%	82.73%	70.55%	47.40%	37.5%	71.56%	69.97%	63.33%
	NL	74.14%	63.09%	46.42%	68.79%	58.54%	66.47%	24.30%	69.17%	27.01%	27.01%
LLama3.1-8B											
ZERO-SHOT	SQL	5.19%	4.86%	1.51%	3.84%	2.78%	1.72%	1.4%	3.1%	3.02%	5.35%
	NL	3.93%	2.33%	0.75%	2.6%	2.04%	1.05%	0.13%	1.57%	1.21%	1.0%
ZERO-COT	SQL	5.0%	4.41%	1.48%	4.49%	2.22%	1.7%	1.56%	3.88%	2.91%	4.09%
	NL	3.56%	1.7%	0.82%	2.73%	1.45%	1.11%	0.11%	2.3%	1.17%	0.87%
FEW-SHOT	SQL	38.73%	38.55%	26.15%	34.37%	35.37%	17.23%	11.24%	39.80%	29.04%	29.15%
	NL	43.79%	31.0%	29.84%	32.06%	38.29%	31.94%	6.61%	37.25%	22.79%	14.97%
Mistral											
ZERO-SHOT	SQL	1.75%	2.23%	0.65%	1.81%	0.93%	1.29%	0.41%	2.14%	0.92%	1.83%
	NL	0.95%	1.14%	0.65%	0.79%	0.67%	0.66%	0.04%	1.29%	0.38%	0.58%
ZERO-COT	SQL	2.42%	1.12%	0.6%	1.25%	1.1%	0.78%	0.23%	1.81%	1.19%	2.3%
	NL	1.28%	0.83%	0.16%	1.4%	0.34%	0.47%	0.05%	0.4%	0.24%	0.26%
FEW-SHOT	SQL	43.55%	22.21%	20.39%	14.98%	33.21%	16.53%	0.78%	33.67%	32.78%	27.07%
	NL	46.27%	20.81%	15.97%	19.29%	35.71%	36.41%	2.64%	38.97%	16.03%	9.16%
Gemma2-9B											
ZERO-SHOT	SQL	6.16%	5.37%	2.89%	5.6%	4.27%	3.16%	2.22%	6.76%	5.07%	6.74%
	NL	4.04%	2.76%	1.67%	4.35%	2.28%	1.52%	0.06%	3.21%	1.71%	1.72%
ZERO-COT	SQL	5.43%	6.11%	2.68%	4.19%	3.25%	2.88%	1.77%	8.49%	7.39%	6.81%
	NL	3.39%	2.05%	1.04%	3.73%	2.51%	1.55%	0%	7.01%	1.56%	1.1%
FEW-SHOT	SQL	39.56%	38.06%	32.84%	25.73%	36.44%	15.83%	16.50%	38.36%	36.36%	26.84%
	NL	44.32%	46.01%	35.26%	34.68%	33.28%	30.43%	12.56%	41.16%	13.19%	7.04%
LLama3.2-3B											
ZERO-SHOT	SQL	5.90%	5.04%	2.61%	5.58%	3.35%	2.74%	1.91%	4.68%	3.96%	5.32%
	NL	4.33%	2.32%	1.46%	3.84%	2.45%	1.42%	0.14%	2.67%	1.46%	1.39%
ZERO-COT	SQL	5.17%	4.7%	2.06%	4.87%	2.89%	2.32%	1.52%	3.64%	3.13%	5.04%
	NL	3.72%	2.18%	1.06%	3.21%	1.99%	1.12%	0.13%	2.18%	1.13%	1.19%
FEW-SHOT	SQL	38.01%	25.52%	32.40%	33.42%	31.68%	19.63%	7.01%	33.51%	32.18%	29.04%
	NL	20.91%	17.11%	23.9%	17.08%	21.22%	16.55%	2.45%	23.62%	13.99%	15.21%
Fined-tuned LLama3.1-8B											
FEW-SHOT	SQL	61.03%	42.09%	37.21%	57.51%	40.25%	37.5%	70.71%	32.43%	15.59%	33.9%
	NL	64.52%	57.70%	36.49%	36.99%	48.36%	63.64%	41.22%	62.55%	30.61%	21.99%

of overlap between insert and non-insert operations. We change the overlap o between insert and non-insert operations from 0 to 1, and fix the encoding method to SQL, prompting method to few-shot and query method to no-filtering select query. We choose b and o as 0.5 and 0.5 respectively. For each parameter setting, we sample 300 (D,Q) pairs and calculate the average accuracy for all the pairs.

When the overlap between insert and non-insert operations increases, the performance of all models changes not much, varying between 0.4 and 0.8, which means even we interleave the non-insert operation with insert ones, it does not change the complexity of the database much.

GPT-4o and fine-tuned LLama3.1-8B is the best performer among all the models. The performance of other four models keeps almost constant, around 0.4.

IV. Related Work

A. In Context Learning

A lot of studies have shown LLMs are capable of acting as few-shot learners [5], [11], [38], in which LLMs exhibit the capability to learn a novel task given the in context examples. These examples presented to LLMs is analogy to the

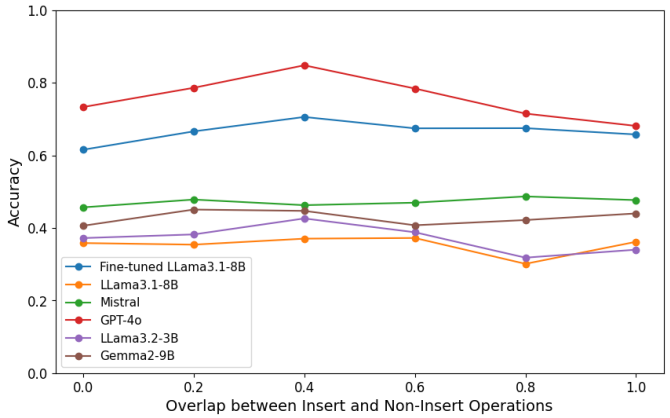


Fig. 8: Model Performance vs Overlap between Insert and Non-Insert Operations

program given to a traditional computer, but in an abstract and declarative way, instead of detailed and imperative way. Though still limited, it is nevertheless the first time a deep model can achieve such a high-order ability. Researchers believe such an ability roots from LLMs' capability to express and comprehend structures contained within in-context examples.

B. Prompt Engineering

The goal of prompt engineering is to optimize the the output of LLMs for a specified task. There has been several methods being proposed: zero-shot in which the model is provided with a task description without any further examples, few-shot [5] in which a small number of examples along with their corresponding outputs are presented to the model and the model can conduct in-context learning to generate output on new inputs, chain-of-thought(CoT) [39] where the model is provided with a number of examples, each showing how to solve the corresponding task step by step, zero-shot COT [40] which is similar to CoT except the model is not presented with any examples but simply with a simple prompt like: "let us think step by step". In this paper, we'll evaluate LLMs' ability as in-context database by adopting the above prompting methods.

C. In Context Graph Reasoning

Recently there are emerging studies evaluating LLMs' capability of doing in-context reasoning on general graphs or knowledge graphs [20]–[24], [41]. By presenting a description of a graph and a query on the graph, these studies are curious about how well LLMs can answer the query. Majority of the work conclude that LLMs can demonstrate the capability of in-context reasoning on graphs (though limited) while giving feasible suggestions and methods on how to optimize such capability. All the existing work focus on in-context learning on static graphs, which does not allow any updates.

D. In Context Tabular Data Reasoning

Similar to graph reasoning, there are also studies evaluating LLMs' capability of in-context reasoning on tabular data [25]–[30], by presenting a serialized description of a table and various tasks to language models. These studies have shown LLMs perform well on some of the tasks while still have limited capability on other tasks. Various prompting and encoding frameworks are developed to optimize the performace of LLMs. Also all of the existing work focus on static tabular data, while in our paper we evaluate LLMs' capability of in-context reasoning on dynamic tabular data.

E. Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) [41]–[44] is proposed to solve the hallucination problems by augmenting LLMs with external knowledge which is more fresh and domain-specific. It is more lightweight compared with fine-tuning, because no model parameters need to be updated and the external knowledge can be appended along with the query sent to the language model. RAG often relies on external databases which have already been populated with domain knowledge and can be sampled in queried time. That is, only the knowledge pertinent to current query will be retrieved and presented to the language model. Our work, on the other hand, tries to put all the data in the context of LLMs and utilized LLMs' own capability to extract necessary information.

V. Conclusion

In this paper, we propose dynamic in-context database. A benchmark named InConDB is presented and we investigate how data stored in traditional RDBMS databases can be represented as text and evaluate the capability of large language models (LLMs) to perform CRUD (Create, Read, Update, Delete) operations on in-context databases. We introduce a benchmark called InConDB and conduct extensive experiments to evaluate the performance of LLMs in enabling in-context database interactions. Our study highlights how performance varies depending on factors such as database encoding techniques, query encoding strategies, prompt engineering, operation type, and data distribution, uncovering both strengths and limitations of LLMs in this context. We find few-shot and SQL is the best combination of prompting and encoding method. Different types of queries also have effect on the accuracy of model prediction. GPT-4o outperforms all other evaluated models, and fine-tuned LLama3.1-8B achieves competitive performance. The number of input commands has a negative impact on the performance of language models. We also find that larger ratio of insert operations result in better performance and overlap between insert and non-insert operation has no obvious impact on the performance of models.

In current stage, in-context database is still a challenge for SOTA language models. But we believe as the size of context window and the reasoning capability of language models increases, in-context database will be enabled and as a result, for some light-weighted application scenarios, in-context database can replace traditional dataase in the near future.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[4] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[6] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[7] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[8] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[9] X. Chen, X. Wang, S. Changpinyo, A. Piergiovanni, P. Padlewski, D. Salz, S. Goodman, A. Grycner, B. Mustafa, L. Beyer *et al.*, "Pali: A jointly-scaled multilingual language-image model," *arXiv preprint arXiv:2209.06794*, 2022.

[10] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "Vivit: A video vision transformer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6836–6846.

[11] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[12] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International conference on machine learning*. PMLR, 2022, pp. 9118–9147.

[13] J. Andreas, "Language models as agent models," *arXiv preprint arXiv:2212.01681*, 2022.

[14] A. Adhikari, X. Yuan, M.-A. Côté, M. Zelinka, M.-A. Rondeau, R. Laroche, P. Poupart, J. Tang, A. Trischler, and W. Hamilton, "Learning dynamic belief graphs to generalize on text-based games," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3045–3057, 2020.

[15] P. Ammanabrolu and M. Riedl, "Learning knowledge graph-based world models of textual environments," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3720–3731, 2021.

[16] N. Tandon, B. D. Mishra, K. Sakaguchi, A. Bosselut, and P. Clark, "Wiqa: A dataset for" what if..." reasoning over procedural text," *arXiv preprint arXiv:1909.04739*, 2019.

[17] A. Madaan, S. Zhou, U. Alon, Y. Yang, and G. Neubig, "Language models of code are few-shot commonsense learners," *arXiv preprint arXiv:2210.07128*, 2022.

[18] A. Creswell, M. Shanahan, and I. Higgins, "Selection-inference: Exploiting large language models for interpretable logical reasoning," *arXiv preprint arXiv:2205.09712*, 2022.

[19] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg *et al.*, "Sparks of artificial general intelligence: Early experiments with gpt-4," *arXiv preprint arXiv:2303.12712*, 2023.

[20] B. Fatemi, J. Halcrow, and B. Perozzi, "Talk like a graph: Encoding graphs for large language models," *arXiv preprint arXiv:2310.04560*, 2023.

[21] J. Guo, L. Du, H. Liu, M. Zhou, X. He, and S. Han, "Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking," *arXiv preprint arXiv:2305.15066*, 2023.

[22] B. Perozzi, B. Fatemi, D. Zelle, A. Tsitsulin, M. Kazemi, R. Al-Rfou, and J. Halcrow, "Let your graph do the talking: Encoding structured data for llms," *arXiv preprint arXiv:2402.05862*, 2024.

[23] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov, "Can language models solve graph problems in natural language?" *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[24] R. Ye, C. Zhang, R. Wang, S. Xu, and Y. Zhang, "Natural language is all a graph needs," *arXiv preprint arXiv:2308.07134*, 2023.

[25] Y. Sui, M. Zhou, M. Zhou, S. Han, and D. Zhang, "Evaluating and enhancing structural understanding capabilities of large language models on tables via input designs," *arXiv preprint arXiv:2305.13062*, 2023.

[26] J. Jiang, K. Zhou, Z. Dong, K. Ye, W. X. Zhao, and J.-R. Wen, "Structgpt: A general framework for large language model to reason over structured data," *arXiv preprint arXiv:2305.09645*, 2023.

[27] H. Gong, Y. Sun, X. Feng, B. Qin, W. Bi, X. Liu, and T. Liu, "Tablegpt: Few-shot table-to-text generation with table structure reconstruction and content matching," in *Proceedings of the 28th International Conference on Computational Linguistics*, 2020, pp. 1978–1988.

[28] A. Liu, H. Dong, N. Okazaki, S. Han, and D. Zhang, "Plog: Table-to-logic pretraining for logical table-to-text generation," *arXiv preprint arXiv:2205.12697*, 2022.

[29] Y. Sui, M. Zhou, M. Zhou, S. Han, and D. Zhang, "Table meets llm: Can large language models understand structured table data? a benchmark and empirical study," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024, pp. 645–654.

[30] X. Zhang, D. Wang, L. Dou, Q. Zhu, and W. Che, "A survey of table reasoning with large language models," *arXiv preprint arXiv:2402.08259*, 2024.

[31] T. Munkhdalai, M. Faruqui, and S. Gopal, "Leave no context behind: Efficient infinite context transformers with infini-attention," *arXiv preprint arXiv:2404.07143*, 2024.

[32] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," *arXiv preprint arXiv:2402.01680*, 2024.

[33] T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, Z. Tu, and S. Shi, "Encouraging divergent thinking in large language models through multi-agent debate," *arXiv preprint arXiv:2305.19118*, 2023.

[34] "In-context database llm," https://github.com/impanyu/in_context_db_llm, 2024, gitHub repository. [Online]. Available: https://github.com/impanyu/in_context_db_llm

[35] OpenAI, "Gpt-4 overview," https://platform.openai.com/docs/models/gpt-4o, 2024. [Online]. Available: https://platform.openai.com/docs/models/gpt-4o

[36] Ollama and contributors, "Ollama: Local large language models and apis," https://github.com/ollama/ollama, 2024, gitHub repository. [Online]. Available: https://github.com/ollama/ollama

[37] Hiyouga and contributors, "Llama-factory: Fine-tuning and inference for llama models," https://github.com/hiyouga/LLaMA-Factory, 2024, gitHub repository. [Online]. Available: https://github.com/hiyouga/LLaMA-Factory

[38] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[39] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[40] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[41] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[42] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[43] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *International conference on machine learning*. PMLR, 2020, pp. 3929–3938.

[44] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Rozière, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz *et al.*, "Augmented language models: a survey," *arXiv preprint arXiv:2302.07842*, 2023.