arXiv:2306.03901v2 [cs.AI] 7 Jun 2023

CHATDB: AUGMENTING LLMS WITH DATABASES AS THEIR SYMBOLIC MEMORY

Chenxu Hu[1]* **Jie Fu**[2]*† **Chenzhuang Du**[1] **Simian Luo**[1] **Junbo Zhao**[3] **Hang Zhao**[1]†
[1]Tsinghua University [2]Beijing Academy of Artificial Intelligence [3]Zhejiang University
fujie@baai.ac.cn hangzhao@mail.tsinghua.edu.cn

ABSTRACT

Large language models (LLMs) with memory are computationally universal (Schuurmans, 2023). However, mainstream LLMs are not taking full advantage of memory, and the designs are heavily influenced by biological brains. Due to their approximate nature and proneness to the accumulation of errors, conventional neural memory mechanisms cannot support LLMs to simulate complex reasoning. In this paper, we seek inspiration from modern computer architectures to augment LLMs with symbolic memory for complex multi-hop reasoning. Such a symbolic memory framework is instantiated as an LLM and a set of SQL databases, where the LLM generates SQL instructions to manipulate the SQL databases. We validate the effectiveness of the proposed memory framework on a synthetic dataset requiring complex reasoning. The project website is available at https://chatdatabase.github.io/.

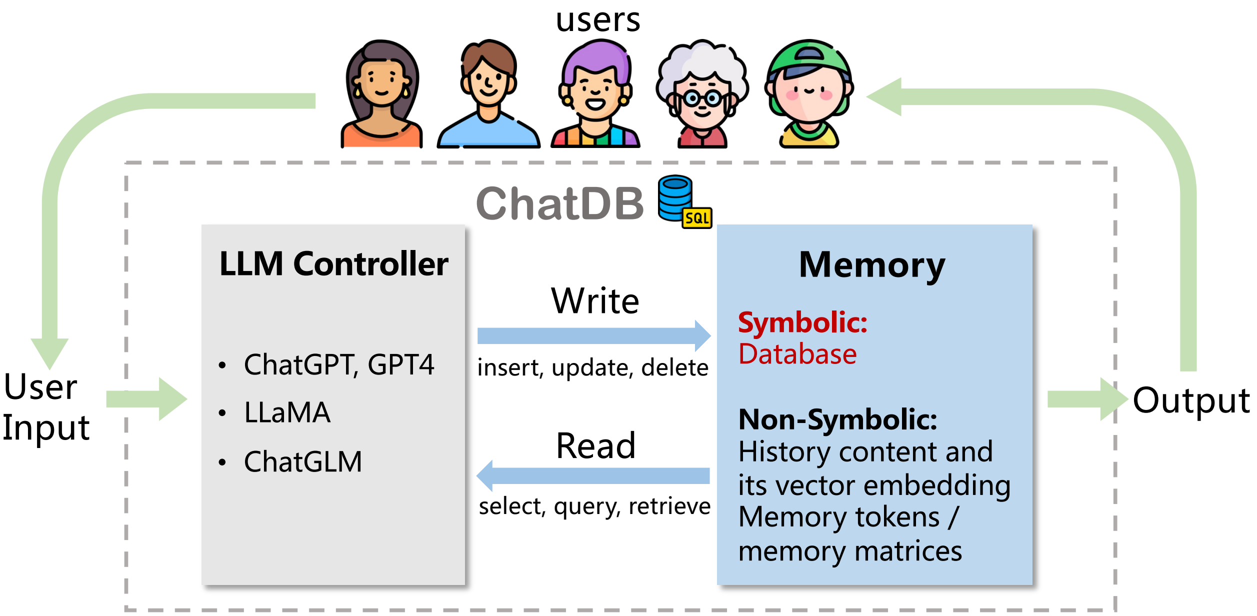


Figure 1: Overall workflow of ChatDB. The LLM controller controls the read and write operations to the memory. The memory stores historical information and provides relevant historical information to assist in responding to user input. In ChatDB, we focus on augmenting LLMs with databases as their symbolic memory.

---

*Equal technical contribution.
†Corresponding authors.

1 Introduction

Large language models (LLMs), such as GPT-4 (OpenAI, 2023) and PaLM 2 (Anil et al., 2023), have increasingly become an essential component of modern artificial intelligence (AI) systems, revolutionizing our understanding of natural language processing (NLP) and transforming various industries (Hao et al., 2023; Wang et al., 2023). While LLMs have made significant strides in understanding and generating contextually relevant responses, they have limitations (Chen et al., 2023). One of the main challenges is that multi-turn interactions with language models generate a large number of tokens, which can easily exceed the input token limit of LLMs. For example, GPT-4 (32K) can only handle 32,000 tokens. As the interaction progresses, the LLMs must maintain contextual information (*e.g.*, user inputs, and previous responses) and generate responses based on the accumulated data. Simply concatenating all contextual information and cramming it into LLMs, however, can easily exceed the processing capabilities of LLMs and accumulate errors, causing the model to lose track of the conversation and produce less accurate responses.

Some neural memory mechanisms have been explored (Wu et al., 2022a; Khattab et al., 2022; Zhong et al., 2022) to overcome the limited token input issue of LLMs. The memory components serve as a storage and retrieval system for relevant information from previous interactions. However, augmenting LLMs with conventional neural memory usually leads to difficulties in storing, retrieving, and manipulating historical information in memory, especially for tasks requiring complex multi-hop reasoning. Two main causes are (a) They do not store historical information in a structured form; (b) Their manipulation of the information stored in memory is not symbolic, as they all rely on some vector similarity calculations, which can be inaccurate, thus leading to the accumulation of errors.

To address the aforementioned issues, we propose using databases as novel symbolic memory for LLMs. The whole framework is named **ChatDB**. As shown in Figure 1, ChatDB consists of two components: an LLM controller and its memory. The LLM controller can be any commonly used LLM (OpenAI, 2023; Touvron et al., 2023; Du et al., 2022; Zeng et al., 2022) and is responsible for controlling the read and write operations to the memory. The memory of LLMs, which can be symbolic or non-symbolic, or a combination of both, is responsible for storing historical information and providing information when needed to assist the LLM in responding to user input. In ChatDB, we focus on using databases as symbolic memory, which allows for the structured storage of historical information through the execution of a symbolic language, namely SQL statements. These SQL statements are generated by an LLM. Incorporating a database as symbolic memory is particularly useful in scenarios requiring precise recording, modification, querying, deletion, and analysis of historical data. For example, a store manager needs to maintain daily sales records, where using plain text or matrices as memory is unsuitable (Chen et al., 2023). However, using a database as external symbolic memory is highly suitable. The database enables accurate operations, including data insertion, deletion, update, and selection, using SQL statements. Thus, employing databases as external symbolic memory ensures precision and efficiency in managing and manipulating historical data, significantly enhancing the performance of LLMs in scenarios that require high accuracy and long-term data recording and processing.

In the ChatDB framework, we propose the *chain-of-memory* (CoM) approach to manipulate the external symbolic memory more effectively, thereby further enhancing the reasoning capabilities of LLMs. The chain-of-memory approach transforms user input into a series of intermediate memory operation steps that lead to final results. Through the chain-of-memory approach, a complex problem is decomposed into multiple steps of memory operations, significantly reducing the complexity of problem-solving. In ChatDB, each intermediate step involves one or more SQL statements.

Our ChatDB makes several contributions to the field of LLMs. Firstly, we propose augmenting LLMs with databases as their external symbolic memory, allowing for structured storage of historical data and enabling symbolic and complex data operations using SQL statements. Secondly, our chain-of-memory approach enables effective memory manipulation by converting user input into multi-step intermediate memory operations, which enhance the performance of ChatDB, enabling it to handle complex, multi-table database interactions with improved accuracy and stability. Finally, our experiments demonstrate that augmenting LLMs with symbolic memory improves multi-hop reasoning capabilities and prevents error accumulation, thereby enabling ChatDB to significantly outperform ChatGPT on a synthetic dataset.

2 Related Work

Memory-Augmented Large Language Models (LLMs). LLMs, such as GPT-4 (OpenAI, 2023) and PaLM 2 (Anil et al., 2023), have demonstrated powerful reasoning and decision-making abilities. However, LLMs are often hindered by their limited context window sizes (*e.g.*, GPT-4 can only handle 32K tokens). Memory-augmented LLMs (Wu et al., 2022a,b; Zhong et al., 2022; Lewis et al., 2020; Guu et al., 2020; Park et al., 2023; Khattab et al., 2022; Izacard et al., 2022) incorporate a memory module that prevents the model from forgetting crucial information and allows it to handle long text inputs that exceed the context window size. Retrieval-augmented in-context learning (Khattab et al., 2022) uses retrieval models (RM) to retrieve relevant information that can be inserted into the LLM as a prompt. For example, Auto-GPT [3] and Generative Agents (Park et al., 2023) utilize a memory module to store the text prompt directly, allowing the agent to keep track of its history. The past and current prompts are then input into the LLM for processing. Neural Turing Machines (NMT) (Graves et al., 2014), which incorporate the recurrent neural network (RNN) with external trainable memory resources and learn to interact with the memory module with gradient descent. Gated Graph Sequence Neural Network (GGS-NN) (Johnson, 2017) constructs and modifies graphs and utilizes the graphs to produce reasonable outputs. Recurrent Memory Transformer (RMT) (Bulatov et al., 2022) introduces additional memory tokens to the input and output sequences to store, process and exchange local and global information between segments of long sequences, and then train the model to control both memory operation and sequence representations processing.

Reasoning with LLMs. LLMs are known to struggle in complex reasoning tasks. Previous methods focus on incorporating specially designed supervisory signals or fine-tuning to enhance the reasoning ability of language models (Piękos et al., 2021; Ran et al., 2019; Andor et al., 2019; Cobbe et al., 2021; Chen et al., 2022). Recent methods mainly improve the reasoning ability of language models through In-Context Learning (Brown et al., 2020; Lester et al., 2021; Wei et al., 2021, 2022; Wang et al., 2022). The most representative of these is Chain-of-Thought (CoT) (Wei et al., 2022), which presents the intermediate reasoning process of solving sample problems to the language model, greatly enhancing its reasoning capabilities.

LLMs with DBs. LLMs have demonstrated an impressive capability in generating code, including Python code, execution commands for Excel, and Structured Query Language (SQL) for databases (OpenAI, 2023). ChatExcel [4] uses LLMs to generate the Excel execution command, simplifying the user interaction process. BINDER (Cheng et al., 2022) proposes a framework that maps task inputs to executable programs in a programming language (*e.g.*, Python code) bound with an API to call LLMs to perform a wide range of functionalities. SQL-PALM (Sun et al., 2023) proposes an LLM-based Text-to-SQL model, using the execution-based self-consistent prompting approach, and outperforms previous Text-2-SQL methods by a large margin. While previous works involve databases to some extent, our proposed ChatDB system differs significantly from these methods. In specific, ChatDB views the databases as the external symbolic memory module for the LLM, and then leverages the database for reading and writing essential data information to enhance the reasoning process via *chain-of-memory*, leading to more accurate reasoning results.

Tool-using LLMs. From the tool-using perspective, ChatDB can also be seen as an LLM utilizing DBs as a tool (Schick et al., 2023; Shen et al., 2023; Surís et al., 2023; Paranjape et al., 2023). Toolformer (Schick et al., 2023), through a series of demonstrations, instructs the language model that it can invoke some APIs to utilize external tools to solve the current problem. Another representative work is Auto-GPT [5], which enables the language models to complete a series of impressive tasks using a search engine. The advantage of ChatDB, which uses a database as an external tool, is that it allows the language model to maintain more accurate records and use historical data, thereby addressing more complex problems, especially those that require accurate historical data for reasoning.

3 ChatDB

In this section, we first briefly introduce the task definition and setting. We then describe the overall framework of our proposed ChatDB. Finally, we delve into the details of the chain-of-memory approach, which is the main component of ChatDB.

3.1 Task Definition

Given a user input in natural language and details of existing tables in the database (not required if there are no existing tables), the goal is to manipulate the symbolic memory, *i.e.*, the external database, to fulfill the user's request. For instance, if the user (*e.g.*, a store manager) command is to record, modify, query, and delete specific data, the corresponding SQL operation should be to insert, update, select, and delete the relevant data in the appropriate tables, respectively. These operations often involve multiple tables in the database.

3.2 Framework Overview

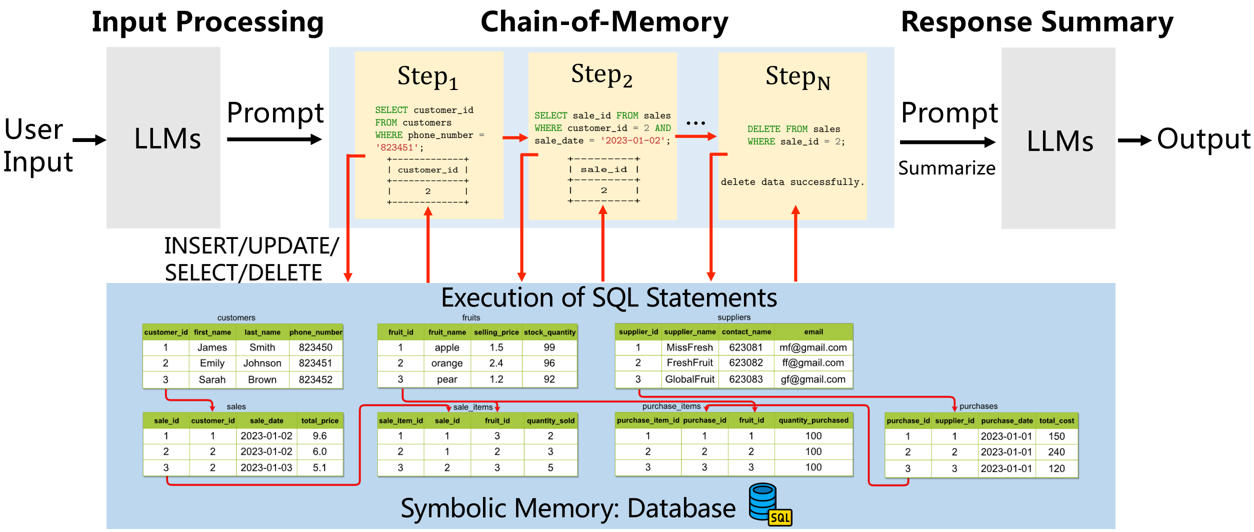
ChatDB framework consists of three main stages: input processing, chain-of-memory, and response summary, as illustrated in Figure 2. Algorithm 1 provides a detailed illustration of the entire algorithmic procedure of ChatDB to respond to the user input.

---

[3] `https://github.com/Significant-Gravitas/Auto-GPT`

[4] `https://chatexcel.com/`

[5] https://github.com/Significant-Gravitas/Auto-GPT



```
SELECT customer_id
FROM customers
WHERE phone_number =
'823451';
  +-------------+
  | customer_id |
  +-------------+
  |      2      |
  +-------------+
```

```
SELECT sale_id FROM sales
WHERE customer_id = 2 AND
sale_date = '2023-01-02';
     +---------+
     | sale_id |
     +---------+
     |    2    |
     +---------+
```

```
DELETE FROM sales
WHERE sale_id = 2;

delete data successfully.
```

customers

customer_id	first_name	last_name	phone_number
1	James	Smith	823450
2	Emily	Johnson	823451
3	Sarah	Brown	823452

fruits

fruit_id	fruit_name	selling_price	stock_quantity
1	apple	1.5	99
2	orange	2.4	96
3	pear	1.2	92

suppliers

supplier_id	supplier_name	contact_name	email
1	MissFresh	623081	mf@gmail.com
2	FreshFruit	623082	ff@gmail.com
3	GlobalFruit	623083	gf@gmail.com

sales

sale_id	customer_id	sale_date	total_price
1	1	2023-01-02	9.6
2	2	2023-01-02	6.0
3	2	2023-01-03	5.1

sale_items

sale_item_id	sale_id	fruit_id	quantity_sold
1	1	3	2
2	1	2	3
3	2	3	5

purchase_items

purchase_item_id	purchase_id	fruit_id	quantity_purchased
1	1	1	100
2	2	2	100
3	3	3	100

purchases

purchase_id	supplier_id	purchase_date	total_cost
1	1	2023-01-01	150
2	2	2023-01-01	240
3	3	2023-01-01	120

Figure 2: ChatDB framework. The red arrow lines represent the process flow of chain-of-memory, indicating the connection between multiple memory operations. The red arrow lines between database tables represent the reference relationships between primary keys and foreign keys, which start from primary keys to foreign keys. Only the first four columns of each table are shown for brevity. This example showcases the process of returning goods purchased on 2023-01-02 by a customer with the phone number 823451.

Input Processing. ChatDB generates a series of intermediate steps to manipulate the symbolic memory by utilizing LLMs if responding to the user input requires the use of memory. Otherwise, we use LLMs directly to generate a reply.

Chain-of-Memory. ChatDB executes a series of intermediate memory operation steps to interact with symbolic memory. ChatDB manipulates the symbolic memory in sequence according to a series of previously generated SQL statements, including operations such as insert, update, select, delete, etc. The external database executes the corresponding SQL statements, updates the database, and returns the results. It is worth noting that ChatDB decides whether to update the memory operation step based on the results of previous SQL statements before executing this operation. ChatDB executes the next step following the same procedure until all operations on the memory are completed.

Response Summary. ChatDB summarizes the final response to the user based on the results of a series of chain-of-memory steps.

3.3 Chain-of-Memory

Chain-of-thought (Wei et al., 2022) emphasizes decomposing complex reasoning into a series of intermediate steps. Chain-of-memory (CoM) can be seen as a way to augment chain-of-thought by providing a symbolic memory mechanism to support the storage associated with those intermediate steps.

The purpose of chain-of-memory is to enhance the reasoning capabilities and robustness of LLMs when manipulating symbolic memory. The approach involves converting the user input into a sequence of intermediate memory operations, enabling LLMs to more accurately and effectively manipulate the memory in a symbolic way. The ability to manipulate symbolic memory is particularly valuable for real-world applications that involve complex and accurate interactions with historical data, such as record-keeping and data analysis in a management setting.

To enhance the performance and stability of our approach, we employ in-context learning (Brown et al., 2020), providing prompt exemplars of several sequences of chain-of-memory steps, and chain-of-thought prompting. A robust and accurate chain-of-memory process enables LLMs to better reason over symbolic memory and handle more complex scenarios.

The advantages of chain-of-memory are twofold. Firstly, it enables LLMs to perform complex database manipulations with greater accuracy, enhancing their multi-hop reasoning ability over symbolic memory. Secondly, by decomposing complex operations into a sequence of intermediate memory operations, the chain-of-memory approach enhances the capability of LLMs when handling complex, multi-table interactions. This approach enables LLMs to handle edge cases and unexpected scenarios better, making it a promising method for real-world applications.

Algorithm 1 The algorithm of ChatDB

```
Input: userInput, dataBase
Output: reply
                                                        ▷ Begin of Input Processing
if need manipulate memory to respond to userInput then
    memOps = $\mathrm{LLM}_{\mathrm{getSteps}}$(userInput)     ▷ Generate intermediate steps using LLMs
else
    reply = LLM(userInput)                              ▷ Generate reply using LLMs directly
    return reply
end if
                                                        ▷ Begin of Chain-of-Memory
sqlResults = []
newMemOps = []
for each memOp in memOps do
    if need update memOp based on sqlResults then
        newMemOp = $\mathrm{LLM}_{\mathrm{updateOperation}}$(memOp, sqlResults)
    else
        newMemOp = memOp
    end if
    sqlResult = executeOperation(newMemOp, dataBase)    ▷ Execute operation on database
    sqlResults.append(sqlResult)
    newMemOps.append(newMemOp)
end for
                                                        ▷ Begin of Response Summary
reply = $\mathrm{LLM}_{\mathrm{summary}}$(userInput, newMemOps, sqlResults)   ▷ Summarize the final reply
return reply
```

3.4 Comparison with previous memory-augmented LLMs

Table 1: Comparison with prompt-based memory and matrix-based memory.

Type	Model	Memory format	Supported operations	Memory storage	Memory execution	Interpretability	State tracking
Symbolic	ChatDB	Symbolic storage (*e.g.*, database)	Insert, Delete Update, Select	Structured	Symbolic	High	Yes
Prompt-based	Auto-GPT	Content and its vector embeddings	Insert, Select	Semi-Structured	Non-Symbolic	Normal	No
Matrix-based	RMT	Memory tokens/ memory matrices	Read, Write	Semi-Structured	Non-Symbolic	Low	Yes

In this subsection, we provide a comprehensive comparison between ChatDB and recent approaches that augment Transformer-based language models with memory module. The memory modules of language models proposed in previous work can be broadly categorized into two types. The first type of memory stores context and uses a retrieval model to find content from past interactions that's most relevant to the current conversation, and then uses it as a prompt for the language model (Khattab et al., 2022). We refer to this type of memory as *prompt-based memory*. The second type of approach utilizes additional memory tokens or memory matrices as memory (Bulatov et al., 2022), which we refer to as *matrix-based memory*. We compare ChatDB with these approaches based on the following aspects:

1. Memory Format. This aspect relates to the format used for storing memory. ChatDB utilizes databases as its memory. *Prompt-based memory* (Park et al., 2023) stores relevant interaction content and/or their corresponding vector embeddings. *Matrix-based memory* employs additional trainable memory tokens (Bulatov et al., 2022, 2023) or trainable memory matrices (Graves et al., 2014).

2. Supported Operations. This aspect refers to the operations supported for manipulating memory. ChatDB supports operations such as insertion, deletion, update, and selection of data in the database memory. The *prompt-based memory* primarily supports insertion and selection operations but lacks complete support for updates and deletions. *Matrix-based memory* supports reading (selection) and writing (insertion, updating, deletion) operations. However, the exact operations performed by the neural network are not explicitly known.

3. Memory Storage. This aspect refers to the format in which data is stored in memory, specifically whether it is structured or not. ChatDB stores memory in a structured format using databases, while both the *prompt-based memory* and *matrix-based memory* are viewed as semi-structured. The reason is that vector embeddings and memory matrices have specific dimensions and sizes, but each individual dimension does not carry a specific and explicit meaning.

4. Memory Execution. This aspect focuses on how memory operations are executed, specifically whether they are symbolic or not. ChatDB executes operations on its database memory using SQL, which is a symbolic language, thus making it inherently symbolic. *Prompt-based memory* performs selections based on a similarity metric using vector embeddings and employs a language encoder to obtain vector embeddings for insertions. Both of these are considered non-symbolic executions. In *matrix-based memory* augmentated LLMs, memory operations are entirely controlled by the neural network, resulting in non-symbolic executions as well.

5. Interpretability. This aspect refers to the degree of interpretability of memory. In ChatDB, memory is stored in a structured and explicit format, and its operations are symbolic, leading to a high level of interpretability. In the *prompt-based memory*, interpretability is generally limited due to inherent challenges in interpreting vector embeddings. For *matrix-based memory* approach, the degree of interpretability is low since memory is completely controlled by the neural network implicitly.

6. State Tracking. This aspect refers to whether the memory effectively tracks the current state of the LLMs. In the case of ChatDB, its memory accurately tracks the current state of the LLMs. The fruit shop experiment serves as a demonstration, where after processing each record, ChatDB's database memory is updated to reflect the latest state of the fruit shop. This showcases how ChatDB's memory effectively tracks its current state. Thanks to the symbolic memory execution, ChatDB's memory allows for easy rollback to any desired timestamp, providing greater flexibility and controllability. In *matrix-based memory* approach, memory is continuously updated and altered by the model itself, enabling it to track the current state of the LLMs. However, the *prompt-based memory* approach simply stores the historical context and only knows what has happened in the past without a clear understanding of the current state.

By examining these aspects, we observe the distinctive features and capabilities of ChatDB in comparison to existing approaches. The superiority of ChatDB highlights the advantages of using symbolic memory to augment LLMs.

4 Evaluation

In this section, we conduct experiments to evaluate the effectiveness of augmenting LLMs with databases as their symbolic memory. Our experiment results demonstrate that ChatDB significantly outperforms the baseline model ChatGPT, highlighting the advantages of symbolic memory integration.

4.1 Experimental Setup

As previously mentioned, using a database as symbolic memory is particularly suitable for scenarios that require precise recording and processing of historical information, such as various data management scenarios. To adapt to the use case of ChatDB and enable quantitative comparisons with other models, we constructed a synthetic dataset simulating the management of a fruit shop.

Furthermore, in order to evaluate the performance of the models, we collected a set of 50 questions with annotated standard answers. These questions varied in difficulty, ranging from hard ones that required multi-hop reasoning to easy ones that only required retrieving information from historical data. There are 15 easy questions and 35 hard questions. Each question is independently answered by the models.

4.1.1 Model Configuration

ChatDB. The LLM used in ChatDB is ChatGPT (GPT-3.5 Turbo), and the hyperparameter *temperature* is set to 0. We used the MySQL database as the external symbolic memory.

Baseline. We use ChatGPT (GPT-3.5 Turbo) as the baseline model with a maximum token length of 4096. Similar to ChatDB, we set the *temperature* to 0.

4.1.2 Dataset

We synthesize a dataset of fruit shop management records, referred to as the "Fruit Shop Dataset". This dataset simulates four common operations in a shop: purchasing, selling, changing prices, and goods returns. We ensure that all historical records are valid and do not encounter issues such as negative inventory. We generate 70 records arranged in chronological order, totaling approximately 3.3k tokens, which is within the maximum token length limit of ChatGPT (4096 tokens).

(a) Purchasing goods: the shop purchases goods from suppliers.

Record: **We restocked our store on 2023-01-01 with a new supply of fruits from 'ABC' (abc_sup@example.com, 10080). The purchased quantities include 24 kg cherry, at unit prices of 0.8. Our intended selling price of cherry is 1.3 dollars per unit.**
ChatDB:
Begin to interact with the external database (symbolic memory).
Step1: Insert supplier 'ABC' if not exists

```
INSERT INTO suppliers (supplier_name, contact_number, email)
SELECT 'ABC', '10080', 'abc_sup@example.com'
WHERE NOT EXISTS (SELECT 1 FROM suppliers WHERE supplier_name
↪   = 'ABC');
```

Step2: Insert fruit (set the selling price to NULL and stock quantity to 0) if not exists

```
INSERT INTO fruits (fruit_name, selling_price, stock_quantity,
↪   fruit_type, shelf_life)
SELECT 'cherry', NULL, 0, NULL, NULL
WHERE NOT EXISTS (SELECT 1 FROM fruits WHERE fruit_name =
↪   'cherry');
```

Step3: Insert purchase

```
INSERT INTO purchases (supplier_id, purchase_date,
↪   total_cost)
VALUES ((SELECT supplier_id FROM suppliers WHERE
↪   supplier_name = 'ABC'), '2023-01-01', 24 * 0.8);
```

Step4: Insert purchase item

```
INSERT INTO purchase_items (purchase_id, fruit_id,
↪   quantity_purchased, cost_per_item, item_total_cost)
VALUES ((SELECT MAX(purchase_id) FROM purchases), (SELECT
↪   fruit_id FROM fruits WHERE fruit_name = 'cherry'), 24,
↪   0.8, 24 * 0.8);
```

Step5: Update the stock quantity of cherry

```
UPDATE fruits
SET stock_quantity = stock_quantity + 24
WHERE fruit_name = 'cherry';
```

Step6: Update the selling price of cherry if given new selling price

```
UPDATE fruits
SET selling_price = 1.3
WHERE fruit_name = 'cherry';
```

(b) Selling goods: customers buy goods from the shop

Record: **A sale was made on 2023-01-02 to 'Bob Smith' (contact details: 123-456-7893, bob.smith@example.com). The items purchased were 9 kg apple, 4 kg cherry.**
ChatDB:
Begin to interact with the external database (symbolic memory).
Step1: Insert customer 'Bob Smith' if not exists

```
INSERT INTO customers (first_name, last_name, phone_number,
↪   email)
SELECT 'Bob', 'Smith', '123-456-7893',
↪   'bob.smith@example.com'
WHERE NOT EXISTS (SELECT 1 FROM customers WHERE phone_number
↪   = '123-456-7893');
```

Step2: Insert sale

```
INSERT INTO sales (customer_id, sale_date, total_price)
VALUES ((SELECT customer_id FROM customers WHERE phone_number
↪   = '123-456-7893'), '2023-01-02', (SELECT selling_price
↪   FROM fruits WHERE fruit_name = 'apple') * 9 + (SELECT
↪   selling_price FROM fruits WHERE fruit_name = 'cherry') *
↪   4);
```

Step3: Insert sale item

```
INSERT INTO sale_items (sale_id, fruit_id, quantity_sold,
↪   price_per_item, item_total_price)
VALUES ((SELECT MAX(sale_id) FROM sales), (SELECT fruit_id
↪   FROM fruits WHERE fruit_name = 'apple'), 9, (SELECT
↪   selling_price FROM fruits WHERE fruit_name = 'apple'),
↪   (SELECT selling_price FROM fruits WHERE fruit_name =
↪   'apple') * 9),
((SELECT MAX(sale_id) FROM sales), (SELECT fruit_id FROM
↪   fruits WHERE fruit_name = 'cherry'), 4, (SELECT
↪   selling_price FROM fruits WHERE fruit_name = 'cherry'),
↪   (SELECT selling_price FROM fruits WHERE fruit_name =
↪   'cherry') * 4);
```

Step4: Update the stock quantity of apple and cherry

```
UPDATE fruits
SET stock_quantity = CASE
    WHEN fruit_name = 'apple' THEN stock_quantity - 9
    WHEN fruit_name = 'cherry' THEN stock_quantity - 4
    ELSE stock_quantity
END
WHERE fruit_name IN ('apple', 'cherry');
```

(c) Return of goods: customers bring back purchased items to the shop.

Record: **Because the customer returned their purchase, we are required to undo the sales transaction made by customer 'John Doe' (phone: 123-456-7890, email: john.doe@example.com) on 2023-01-05.**
ChatDB:
Begin to interact with the external database (symbolic memory).
Step1: Find the sale_id for this customer on this date

```
SELECT sale_id FROM sales
WHERE customer_id = (SELECT customer_id FROM customers WHERE
↪   phone_number = '123-456-7890' AND email =
↪   'john.doe@example.com') AND sale_date = '2023-01-05';
```

```
Database response:
+---------+
| sale_id |
+---------+
|    9    |
+---------+
```

Step2: Get all the fruit_id and quantity_sold for this sale, replace <sale_id> with the results from the previous queries

```
SELECT fruit_id, quantity_sold FROM sale_items
WHERE sale_id = 9 [<sale_id>];
```

```
Database response:
+----------+---------------+
| fruit_id | quantity_sold |
+----------+---------------+
|    7     |       10      |
|    6     |        9      |
+----------+---------------+
```

Step3: Increase the stock_quantity for each fruit sold in this sale, replace <quantity_sold> <fruit_id> with the results from the previous queries

```
UPDATE fruits
SET stock_quantity = stock_quantity + 10 [<quantity_sold>]
WHERE fruit_id = 7 [<fruit_id>];

UPDATE fruits
SET stock_quantity = stock_quantity + 9 [<quantity_sold>]
WHERE fruit_id = 6 [<fruit_id>];
```

Step4: Delete the sale items for this sale, replace <sale_id> with the results from the previous queries

```
DELETE FROM sale_items WHERE sale_id = 9 [<sale_id>];
```

Step5: Delete the sale record, replace <sale_id> with the results from the previous queries

```
DELETE FROM sales WHERE sale_id = 9 [<sale_id>];
```

(d) Changing prices: modify the price of the goods.

Record: **On 2023-01-05, the sale price of pear in the store was changed to 1.6 dollar per unit.**
ChatDB:
Begin to interact with the external database (symbolic memory).
Step1: Update the selling price of pear

```
UPDATE fruits
SET selling_price = 1.6
WHERE fruit_name = 'pear';
```

Figure 3: Examples showcasing ChatDB's response to the four common operations in the records of the Fruit Shop Dataset. "9 [<sale_id>]" represents that <sale_id> is replaced with 9 in the updated SQL statement. Due to page size limitations, some unnecessary details and ChatDB's summary are omitted.

~~~
Suppose you are a fruit shop manager and good at analyzing history records.
The fruit shop newly opened on January 1, 2023. Given the history records for the fruit shop in January 2023, which include customer names, transaction dates, fruit prices, quantities purchased, and whether the items were returned, you need to answer some questions.
By default, exclude the sales transactions that have been returned when performing calculations.
Here are the historical records of the fruit shop, which are arranged in chronological order based on the occurrence time, surrounded by triple backticks:
```
{records}
```
Based on the history records, answer the question about the fruit shop:
{question}
~~~

Figure 4: Prompt for ChatGPT to answer questions in the Fruit Shop Dataset. The placeholders "records" and "question" will be replaced by specific details when it is actually used.

Why do we limit the token length of the dataset? If the token length of the dataset exceeds the maximum token length of ChatGPT, memory becomes necessary. However, the mainstream memory retrieval methods based on vector embedding are prone to errors. This inevitably leads to a decline in the performance of ChatGPT, which is not desired. Therefore, we deliberately design the token length of the dataset to be within the maximum token length of ChatGPT to avoid using memory and maximize the model's performance. Note that ChatDB's performance is generally unaffected by the token length of the dataset. Thus, if ChatDB outperforms ChatGPT when the dataset is small, it indicates that ChatDB is also superior to memory-augmented ChatGPT when the dataset is large.

4.1.3 Processing records

For ChatDB, the first step is to initialize the database. We need to generate a reasonable database schema for the specific task scenario and create tables in the database. The generation of the database schema can be done manually or using LLMs. Next, for each record in the dataset, ChatDB processes them one by one. Using the LLM controller, ChatDB manipulates the external database (*i.e.*, symbolic memory) following Algorithm 1. We provide examples of ChatDB's response to the four common operations in the Fruit Shop Dataset, namely purchasing, selling, changing prices, and goods returns, as shown in Figure 3. It is worth emphasizing that ChatDB processes record one by one, so it is not sensitive to the total number of records. Furthermore, each step of the database operation in ChatDB is symbolic and free from errors. Therefore, in theory, ChatDB can handle an infinite number of historical records without sacrificing performance. However, for ChatGPT or existing memory-augmented LLMs, excessively long historical records can significantly degrade performance. In this experiment, as for the ChatGPT baseline, since the records are not long, we simply treat them as part of the prompt.

4.1.4 Answering questions

When answering questions, ChatDB no longer requires records to be part of the prompt. After processing the records, the information is stored in symbolic memory. Following Algorithm 1, ChatDB utilizes SQL statements to perform a series of database queries (including calculations) in order to answer the question. On the other hand, ChatGPT includes records as part of the prompt and directly asks the question. The prompt template is shown in Figure 4.

4.2 Results

The experimental results are presented in Table 2, which clearly demonstrate that ChatDB outperforms ChatGPT with significantly higher accuracy. While ChatGPT is capable of answering easy questions, it falls short in handling hard questions that necessitate multi-hop reasoning and precise calculations. Consequently, ChatGPT exhibits a low accuracy rate for these hard questions. In contrast, ChatDB exhibits a notably high accuracy rate, underscoring the advantages of utilizing a database as symbolic memory. This approach not only prevents error accumulation but also enhances the multi-hop reasoning and precise calculation capabilities of LLMs.

We present several examples of the two models answering questions in Figure 5 for comparison. In all these examples, ChatDB correctly answers the questions while ChatGPT fails. ChatGPT often exhibits errors in calculating the total price of each sale transaction, as observed in Figure 5(a). Sometimes, the formulas are correct but the calculations are wrong, while other times, even the formulas are incorrect. In addition, ChatGPT struggles to find all valid sale

(a) **Question:** **What was the total revenue for January 2023?**

ChatGPT:
To calculate the total revenue for January 2023, we need to sum up the revenue generated from each sales transaction. We can use the following formula to calculate the revenue for each transaction: revenue = quantity * selling price
We also need to exclude the sales transactions that have been returned. Based on the history records, the following sales transactions have been returned:
- Alice Smith on 2023-01-17 - Bob Smith on 2023-01-05 [...]

Therefore, we need to exclude these transactions when calculating the total revenue. Here is the calculation for each sales transaction:
- Bob Smith on 2023-01-02: (9 kg * 3.8 USD/kg) + (4 kg * 1.3 USD/kg) = 39.1 USD [WRONG]: calculation error, should be 39.4
- Sue Davis on 2023-01-02: 1 kg * 3.8 USD/kg = 3.8 USD
- Bob Smith on 2023-01-05: (5 kg * 1.8 USD/kg) + (2 kg * 3.0 USD/kg) + (6 kg * 2.8 USD/kg) = 47.6 USD [WRONG]: calculation error and this transaction has been returned.

[...] [WRONG]: most of the sale transactions are calculated incorrectly.
[STOP]: Because it exceeds the maximum token length.

ChatDB:
Begin to interact with the external database (symbolic memory).
Step1: Calculate the total revenue for January 2023

```
SELECT SUM(total_price) AS total_revenue
FROM sales
WHERE sale_date >= '2023-01-01' AND sale_date < '2023-02-01';
```

```
Database response:
+---------------+
| total_revenue |
+---------------+
|     707.0     |
+---------------+
```

(b) **Question:** **Which day in January 2023 had the highest revenue?**

ChatGPT:
To calculate the revenue for each day, we need to sum up the total sales for that day (excluding any returned items) and multiply by the selling price. We can then compare the revenue for each day to determine which day had the highest revenue.
Here is the breakdown of sales for each day:

- January 2: 9 kg apple at 3.8 = 34.2, 4 kg cherry at 1.3 = 5.2, 3 kg cherry at 1.3 = 3.9, 3 kg orange at 2.1 = 6.3, total sales = 49.6 [WRONG]: should be 43.2
- January 4: 2 kg apple at 1.5 = 3, 4 kg pear at 1.6 = 6.4, 7 kg cherry at 1.3 = 9.1, total sales = 18.5 [WRONG]: should be 27.9
- January 5: 5 kg cherry at 1.9 = 9.5, 10 kg orange at 1.8 = 18, 1 kg cherry at 1.9 = 1.9, 1 kg grape at 3.2 = 3.2, total sales = 32.6 [WRONG]: should be 83
- January 6: 3 kg apple at 1.5 = 4.5, 10 kg grape at 3.2 = 32, total sales = 36.5 [WRONG]: should be 29.4
- January 7: 4 kg pear at 1.6 = 6.4, total sales = 6.4 [WRONG]: should be 37.2
- January 8: 6 kg peach at 3 = 18, total sales = 18 [WRONG]: should be 21.6

[...] [WRONG]: most of the daily revenues are calculated incorrectly.
[STOP]: Because it exceeds the maximum token length.

ChatDB:
Begin to interact with the external database (symbolic memory).
Step1: Calculate the total revenue for each day in January 2023

```
SELECT sale_date, SUM(total_price) AS revenue
FROM sales
WHERE sale_date BETWEEN '2023-01-01' AND '2023-01-31'
GROUP BY sale_date;
```

```
Database response:
+------------+---------+
| sale_date  | revenue |
+------------+---------+
| 2023-01-02 |   43.2  |
| 2023-01-04 |   27.9  |
|    ...     |   ...   |
+----------------------+
```

Step2: Find the day with the highest revenue

```
SELECT sale_date, SUM(total_price) AS revenue
FROM sales
WHERE sale_date BETWEEN '2023-01-01' AND '2023-01-31'
GROUP BY sale_date ORDER BY revenue DESC LIMIT 1;
```

```
Database response:
+------------+---------+
| sale_date  | revenue |
+------------+---------+
| 2023-01-30 |  119.7  |
+------------+---------+
```

(c) **Question:** **What was the average weight of fruit per sales transaction in January 2023?**

ChatGPT: To calculate the average weight of fruit per sales transaction, we need to first identify all the sales transactions in January 2023 and calculate the total weight of fruit sold in those transactions. Then, we can divide the total weight by the number of sales transactions to get the average weight per transaction.
Here are the sales transactions in January 2023:
- Bob Smith on 2023-01-02: 9 kg apple, 4 kg cherry
- Sue Davis on 2023-01-02: 1 kg apple
- Bob Smith on 2023-01-05: 5 kg orange, 2 kg peach, 6 kg pear [WRONG]: this transaction has been returned.
- Sue Davis on 2023-01-05: 5 kg pear, 10 kg orange, 1 kg cherry
- Alice Smith on 2023-01-05: 5 kg cherry

[...] [WRONG]: not find out all valid sale transactions.
To calculate the total weight of fruit sold, we can add up the quantities of each fruit sold in the above transactions:
- Apples: 16 kg - Cherries: 12 kg - Oranges: 10 kg - Pears: 18 kg - Kiwis: 12 kg - Grapes: 18 kg - Bananas: 27 kg - Strawberries: 7 kg - Peaches: 9 kg - Plums: 8 kg

The total weight of fruit sold in January 2023 is 127 kg. [WRONG]: It should be 278. Even according to the calculation above, the total should be 137, not 127.
To calculate the average weight per sales transaction, we need to divide the total weight by the number of sales transactions. There are 18 sales transactions in January 2023, [WRONG]: It should be 28 valid sale transactions in total. so the average weight [STOP]: Because it exceeds the maximum token length.

ChatDB:
Begin to interact with the external database (symbolic memory).
Step1: Calculate the total weight of fruit sold in January 2023

```
SELECT SUM(si.quantity_sold) as
↪  total_weight
FROM sale_items si
JOIN sales s ON si.sale_id = s.sale_id
WHERE s.sale_date BETWEEN '2023-01-01'
↪  AND '2023-01-31';
```

```
Database response:
+--------------+
| total_weight |
+--------------+
|     278      |
+--------------+
```

Step2: Count the number of sales transactions in January 2023

```
SELECT COUNT(DISTINCT s.sale_id) as
↪  num_sales
FROM sales s
WHERE s.sale_date BETWEEN '2023-01-01'
↪  AND '2023-01-31';
```

```
Database response:
+-----------+
| num_sales |
+-----------+
|    28     |
+-----------+
```

Step3: Calculate the average weight of fruit per sales transaction in January 2023, replace <total_weight> and <num_sales> with the results from the previous queries

```
SELECT 278 [<total_weight>] / 28
↪  [<num_sales>]
as avg_weight_per_sale;
```

```
Database response:
+---------------------+
| avg_weight_per_sale |
+---------------------+
|        9.93         |
+---------------------+
```

Figure 5: Examples of ChatDB and ChatGPT answering questions. While ChatGPT fails to answer any of the questions correctly, ChatDB successfully answers all of them. The issues with ChatGPT are highlighted in red. Some irrelevant details and ChatDB's summary are omitted for brevity. The gray ellipsis [...] indicates that the response is trimmed.

Table 2: The experimental results of answering questions in the Fruit Shop Dataset. There are 50 questions in total, with 15 being easy questions and 35 being hard questions.

Model	Easy	Hard	All	Accuracy
ChatGPT	10/15	1/35	11/50	22%
ChatDB (ours)	13/15	28/35	41/50	82%

transactions, leading to errors in its answering process. This issue is commonly encountered and evident in all these examples. Furthermore, ChatGPT tends to make sequential errors, resulting in significant error accumulation.

In contrast, ChatDB performs quite well in these examples. During the initial processing of records, symbolic operations (*i.e.*, SQL operations) are applied to manipulate the database (*i.e.*, symbolic memory), ensuring that all information is stored in a structured form within the database. When answering questions, ChatDB generates SQL statements to query the database. The three examples demonstrate the effectiveness of ChatDB in solving problems requiring one, two, and three chain-of-memory steps, respectively. We can observe that ChatDB accurately answers the questions, and the execution logic of the chain-of-memory is clear, with each step tightly connected and approaching the ultimate answer. From these examples, the advantages of ChatDB are evident in two aspects:

1. Through the chain-of-memory approach, complex problems are decomposed into multiple steps of memory operations, simplifying the problem's complexity. Each step's result is accurately stored as an intermediate outcome and used in subsequent steps, which greatly assists in complex reasoning.

2. Symbolic memory enables precise operations and calculations. ChatDB delegates many calculational tasks to the external database by executing SQL statements, ensuring the accuracy of each step and preventing error accumulation.

In summary, by leveraging external databases as symbolic memory, ChatDB significantly outperforms ChatGPT in this experiment.

5 Conclusion

In this paper, we introduce ChatDB, a framework that augments LLMs with symbolic memory in the form of databases. We demonstrate the advantages and capabilities of symbolic memory and chain-of-memory approach in enhancing complex reasoning and preventing error accumulation. By providing a precise storage mechanism for intermediate results, symbolic memory enables accurate and reliable operations. Moreover, the use of symbolic languages, such as SQL, allows symbolic computation and manipulation of stored information. Through experimental evaluation, we observe a significant improvement in performance with ChatDB compared to ChatGPT. The integration of symbolic memory in ChatDB substantially enhances the model's ability to handle various queries and reasoning tasks in management settings. This improvement highlights the benefits and effectiveness of leveraging symbolic memory in LLMs.

References

Andor, D., He, L., Lee, K., and Pitler, E. (2019). Giving bert a calculator: Finding operations and arguments with reading comprehension. *arXiv preprint arXiv:1909.00109*.

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. (2023). Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

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

Bulatov, A., Kuratov, Y., and Burtsev, M. (2022). Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091.

Bulatov, A., Kuratov, Y., and Burtsev, M. S. (2023). Scaling transformer to 1m tokens and beyond with rmt. *arXiv preprint arXiv:2304.11062*.

Chen, A., Phang, J., Parrish, A., Padmakumar, V., Zhao, C., Bowman, S. R., and Cho, K. (2023). Two failures of self-consistency in the multi-step reasoning of llms. *arXiv preprint arXiv:2305.14279*.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. (2022). Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Cheng, Z., Xie, T., Shi, P., Li, C., Nadkarni, R., Hu, Y., Xiong, C., Radev, D., Ostendorf, M., Zettlemoyer, L., et al. (2022). Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z., and Tang, J. (2022). Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M. (2020). Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Hao, S., Gu, Y., Ma, H., Hong, J. J., Wang, Z., Wang, D. Z., and Hu, Z. (2023). Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. (2022). Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*.

Johnson, D. D. (2017). Learning graphical state transitions. In *International conference on learning representations*.

Khattab, O., Santhanam, K., Li, X. L., Hall, D., Liang, P., Potts, C., and Zaharia, M. (2022). Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*.

Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

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

OpenAI (2023). Gpt-4 technical report.

Paranjape, B., Lundberg, S., Singh, S., Hajishirzi, H., Zettlemoyer, L., and Ribeiro, M. T. (2023). Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*.

Piękos, P., Michalewski, H., and Malinowski, M. (2021). Measuring and improving bert's mathematical abilities by predicting the order of reasoning. *arXiv preprint arXiv:2106.03921*.

Ran, Q., Lin, Y., Li, P., Zhou, J., and Liu, Z. (2019). Numnet: Machine reading comprehension with numerical reasoning. *arXiv preprint arXiv:1910.06701*.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Schuurmans, D. (2023). Memory augmented large language models are computationally universal. *arXiv preprint arXiv:2301.04589*.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. (2023). Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Sun, R., Arik, S. O., Nakhost, H., Dai, H., Sinha, R., Yin, P., and Pfister, T. (2023). Sql-palm: Improved large language modeladaptation for text-to-sql.

Surís, D., Menon, S., and Vondrick, C. (2023). Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Wang, W., Chen, Z., Chen, X., Wu, J., Zhu, X., Zeng, G., Luo, P., Lu, T., Zhou, J., Qiao, Y., et al. (2023). Visionllm: Large language model is also an open-ended decoder for vision-centric tasks. *arXiv preprint arXiv:2305.11175*.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., and Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2021). Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. (2022a). Memorizing transformers. *arXiv preprint arXiv:2203.08913*.

Wu, Y., Zhao, Y., Hu, B., Minervini, P., Stenetorp, P., and Riedel, S. (2022b). An efficient memory-augmented transformer for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2210.16773*.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al. (2022). Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

Zhong, Z., Lei, T., and Chen, D. (2022). Training language models with memory augmentation. *arXiv preprint arXiv:2205.12674*.