Memory-enhanced Language Models as a Service

Edward Kusel

July 2023

Abstract

This paper explores the limitations of Language Models as-a-Service (LMaaS) like ChatGpt and Bard due to their lack of memory. Two examples, REM-X and BROKE-LEG demonstrate the failures of these models in remembering critical information after their memory is artificially exceeded. The paper proposes a solution to enhance statefulness of LMaaS through an external data structure and summarization techniques that add memory to the models' performance. Using a meta-language, the enhanced model can efficiently store, retrieve, and delete information in an external data structure, such as a hash table or prefix tree. The model implements the external memory system using a hash table paired with the Python library SymPy to transform expressions into symbols. The experimental results show that the memory-supported model successfully remembers simple mathematical information when the memory-unsupported gpt-3.5-turbo model's internal memory capacity is exceeded. The findings suggest that enhancing LMaaS with external memory systems can improve their performance and safety in critical applications.

1 Introduction

Language Models as-a-Service (LMaaS), such as ChatGPT and Bard [5, 9], have become popular tools that people around the world use every day as a complementary, when not a substitute, of search engine. Differently from search engines, users interact with LMaaS by engaging in a conversation. Though powerful, LMaaS can only access (and thus use) information that a user recently prompted in a conversation. In practice, these models do not have memory. In this project, we will research methods to enhance statefulness through summarization techniques and then add an interface that adds this capability to test it against the existing models. This project's first milestone is to provide an overview of the problem of (lack of) memory in LMaaS. We do that by showing how the problem occurs and why it is important to develop methods that enhance the statefulness of LMaaS. The second part of the project is devoted to the development of a memory system that guarantees that critical information is stored, retrieved, and fed to the LMaaS when needed.

2 Motivating Examples

We introduce 2 different kinds of motivating examples for 2 different types of LLMs. One example is REM-X, where the model is asked to remember a simple variable declaration like "Remember x equals 1" or "Remember that x plus y equals 10 times z". Then the memory of the model is artificially exhausted through a lengthy (3 paragraph) second prompt irrelevant to the first prompt (perhaps a few paragraphs from an unrelated fiction novel). After this, the model will not remember the value of x. The second example is BROKE-LEG, where the model is told that the user broke their leg today. Then, in a similar manner as in the first example, the memory is exhausted through a lengthy unrelated second prompt. When the model is then prompted that the user's friends are playing football outside on a nice day and the user asks for activity recommendations, the model will recommend that the user plays soccer. Both examples are clear faults of memory.

2.1 Attention-based Architectures Don't Have Memory

On ChatGPT-3.5-turbo (but this works on Bard as well), if we prompt the model with the text `‘Remember that x=1.’`, and after the model's response (which ensures that it will remember that information), we further query a sequence of words that are not related to the initial prompt (e.g., a chapter from a novel), yet they are long enough to *saturate* the model's attention, at the very moment we will ask the model again what was the value of x, it won't be able to reply. We report the REM-X example for ChatGPT in Figure 1 and the BROKE-LEG example for ChatGPT in Figure 3

2.2 Enhancing Attention-based Architectures with Text-based Systems

An example of an LLM interface which attempts to incorporate enhanced memory for additional user-generated context is MemoryGPT. Although, MemoryGPT fails in some REM-X cases. MemoryGPT implements a 1,000 character prompt limit whereas ChatGPT implements around a 4,096 character prompt limit. MemoryGPT implements enhanced memory through the use of summarization. Example techniques are entity recognition, semantic analysis, and syntactic parsing. Both REM-X and BROKE-LEG are examples of MemoryGPT failing due to having limited contextual power and memory. We report the REM-X example for MemoryGPT in Figure 2 and the BROKE-LEG example for MemoryGPT in Figure 4

3 Related Works

Equipping LMs with memory is a long-standing challenge [1, 4] that attracted attention from the deep learning community well before the introduction of the self-attention mechanism [10], the first Transformer-based models [3, 6] and the breakthrough of LMs as zero-shot learners [2].

With the rise of LMaaS [5, 9], whose architecture is still primarily based on Transformers (though their architecture and training data have massively scaled to an unprecedented level [7]), a renewed interest has aroused on memory-augmented LMaaS [11]. Their commercial success has pushed researchers to work towards memory systems that enhance the user's experience [12], yet researchers are still conducting theoretical work on the limit of Transformers-based architecture [8].

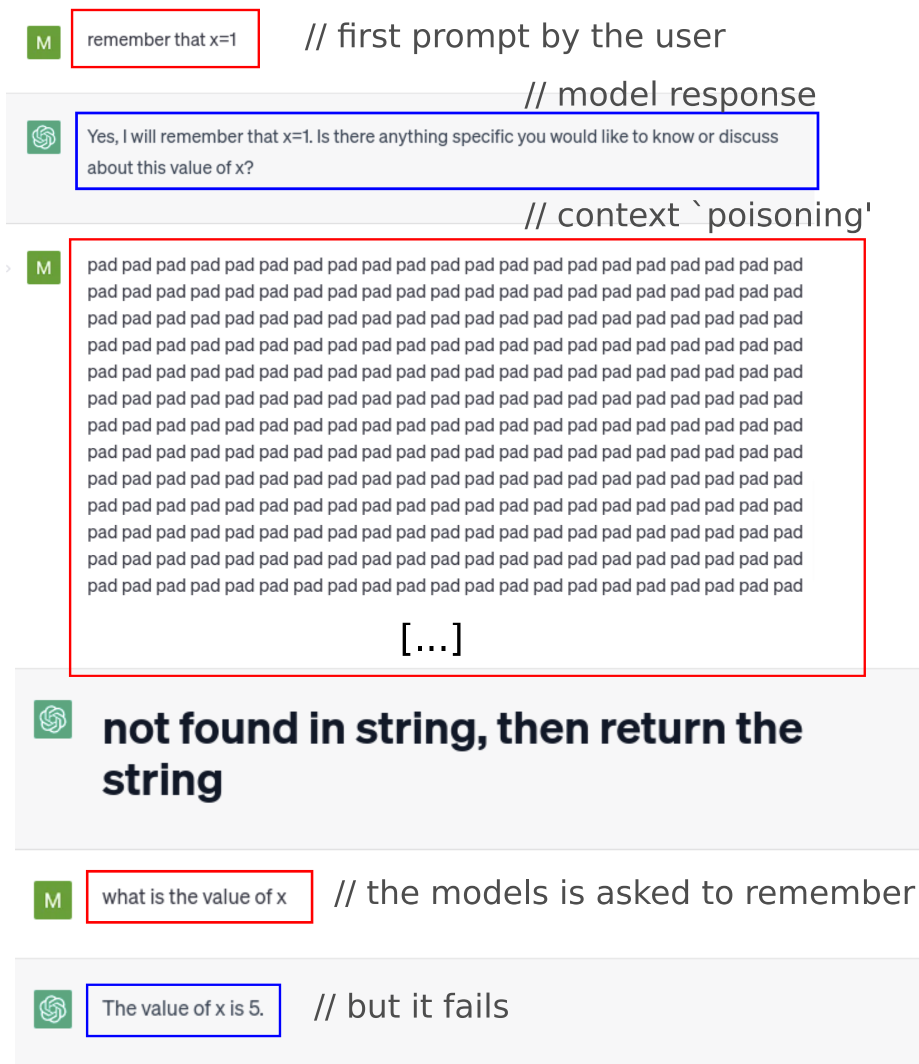

Figure 1: Example of 'context poisoning' that makes a model easily forget information.

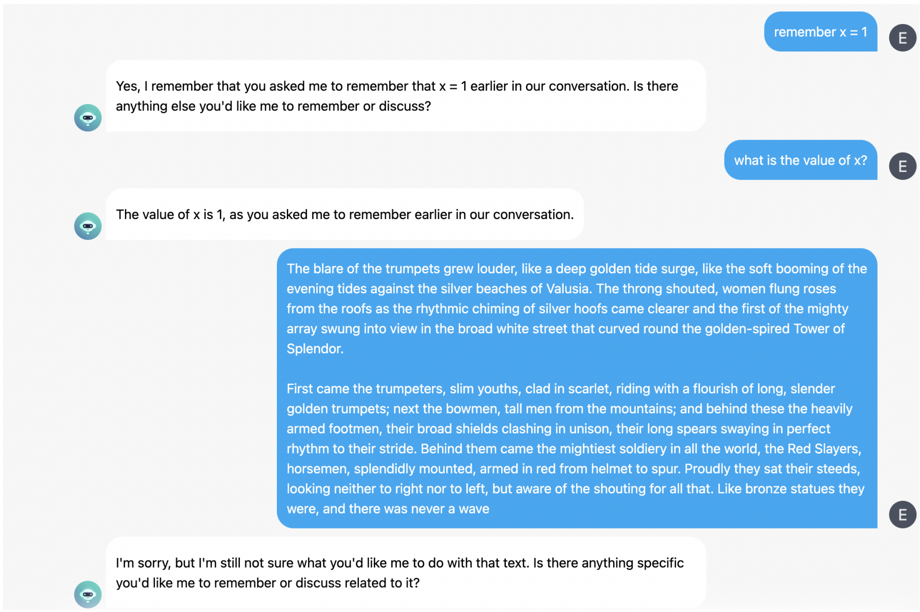

Figure 2: Example of MemoryGPT Failure on REM-X example

4 Methodology

We denote an LMaaS as a function that outputs a sequence of symbols in response to a textual input. Both the input and the output are expressed as a series of symbols and drawn from a finite vocabulary $\mathcal{V}$, namely

$$\psi : \mathcal{V}^+ \rightarrow \mathcal{V}^+. \tag{1}$$

LMaaS are probabilistic models, in the sense that they output a new symbol v_t in an auto-regressive fashion, namely

$$v_t \in argmax_{v \in \mathcal{V}} \mathbb{P}(v_t = v | v_{<t}). \tag{2}$$

This work focuses on auto-regressive LMaaS, yet models such as BERT work slightly differently. To distinguish when an LMaaS outputs the next symbol from when it outputs a sequence, we denote the former as

$$\psi_t : \mathcal{V}^+ \rightarrow \mathcal{V}. \tag{3}$$

4.1 Lack of Memory of LMaaS

LMaaS have finite memory, in the sense that when they generate the next symbol of a sequence, as per Eq.3, they are limited by the length of the context vector v_t as per Eq. 2. For example, models such as ChatGPT have a context length of around $32K$ tokens, which means that, without any compression technique

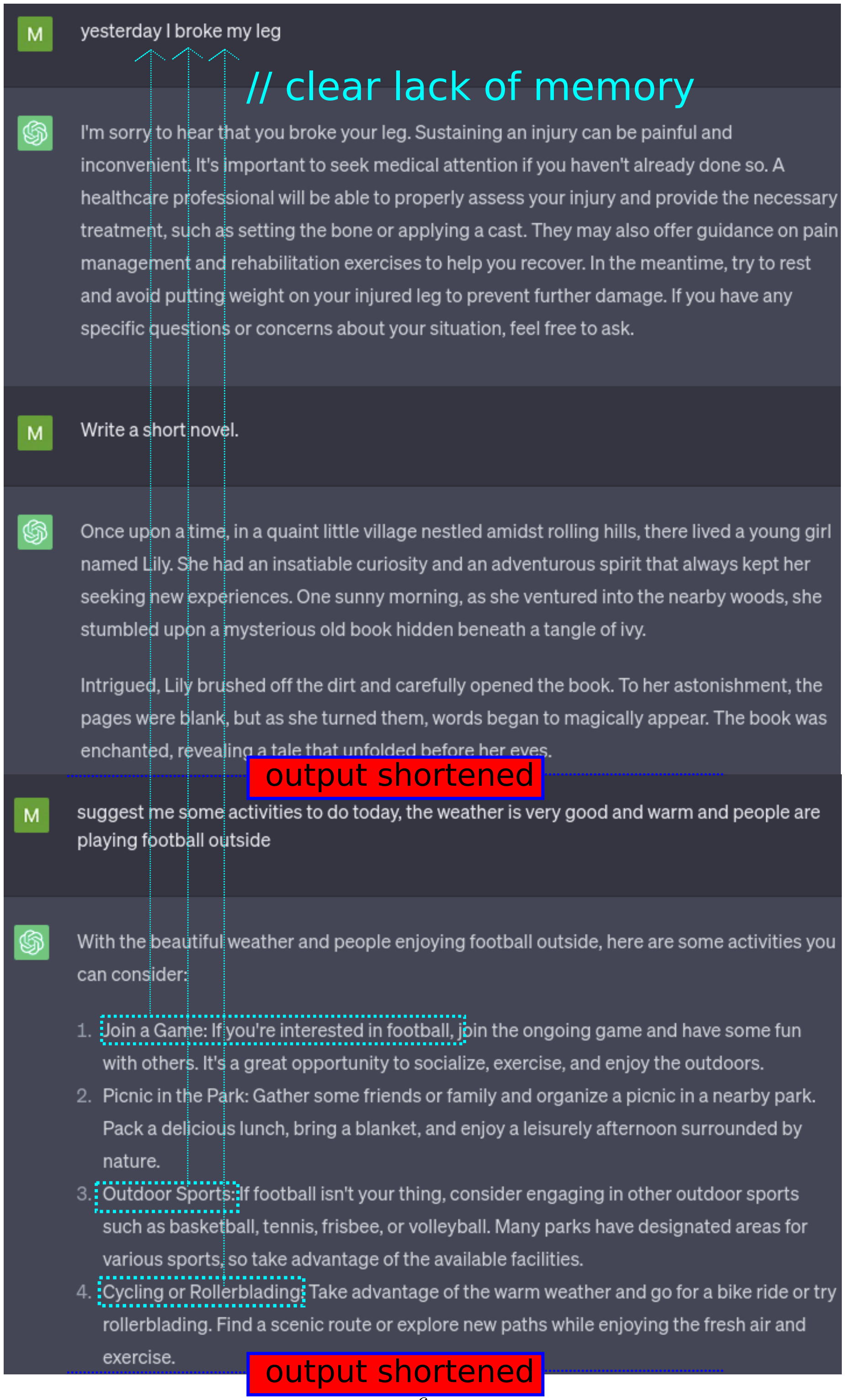

Figure 3: A case to justify the necessity of human-like memory. The model is not asked to remember that I broke my leg, but when it is asked for advice for activities, it replies with nonsense like playing football.

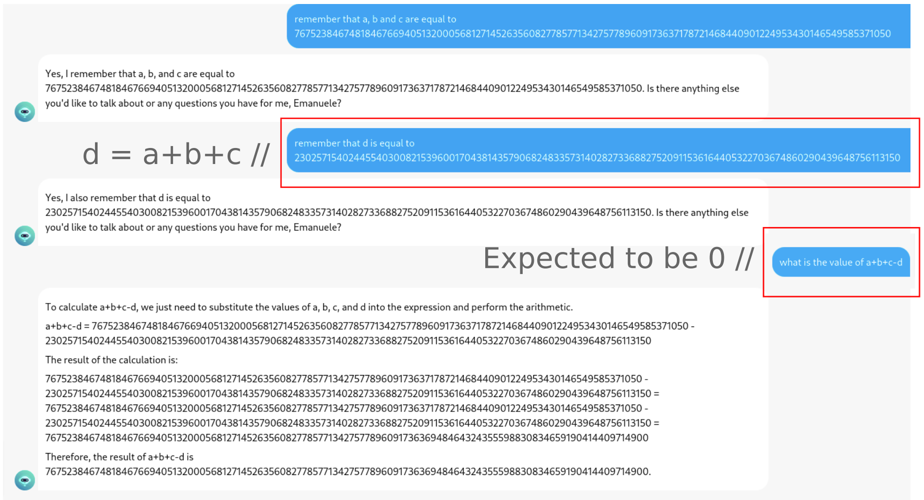

Figure 4: Example of 'context poisoning' that makes a model such as MemoryGPT, which is equipped with a memory that relies on text storage, easily forget information.

applied to v_t, such a model can memorize at most $24K$ words.[1] While this does not necessarily constitute a problem when querying an LMaaS for information, in safety-critical applications, this could lead to dangerous behaviors. We thus advocate for enhancing LMaaS with memory systems. In the next sections, we present the pros and cons of a straightforward technique based on looping over chunks of the context $v_{<t}$, followed by a structured approach to memorization that addresses the problem of storing safety-critical information that can then be fed to the LMaaS.

4.2 A Straightforward Memory System

A naive approach to incorporating memory into an LMaaS query prompt response would be looping over all past conversations with the LMaaS to retrieve some information relevant to the query. A full loop procedure could be:

Algorithm 1 takes as input an LMaaS ψ, a prompt p, which, for example, expresses a query that involves an expression that we want the LMaaS to remember, a context C that contains chunks c of the historical conversations with the LMaaS, and a measure of distance $dist(\cdot, \cdot)$ between sentences, that is used to retrieve the prompt that is most likely to contain the desired information. Without a key-based search mechanism, this procedure allows for scenarios where all historical prompts $C = \{c_1, .., c_m\}$ are reviewed before incorporating memory relevant to the query q (line 3). Furthermore, there is no guarantee that the information required, which can be expressed in natural language in

[1] `https://openai.com/pricing`

Algorithm 1 Basic Memory System.

Require: $\psi, p, C, dist(\cdot, \cdot)$

Ensure: Prompt an LMaaS with memory-enhanced context.

$p^*, d^* = (p, -\infty)$

for $c \in C$ **do** ▷ Loop over all the possible contexts

  $d \leftarrow dist(c, p)$ ▷ Distance between a chunk of memory and the query

  $p^*, d^* \leftarrow ((c.p, d)$ if $d < d^*$ else $(p^*, d^*))$ ▷ Keep the most relevant query and the relative distance

end for

$a^* = \psi(p*)$ ▷ Prompt the LMaaS with the best memory-enhanced prompt

return a^*

p, is actually contained in the returned query p^*, and consequently prompted to the model. A full historical loop is time and space inefficient, requiring all prompts, regardless of importance, to be stored in memory in addition to a full-loop procedure. Each prompt's relevance must also be determined and running an importance determining function on each stored prompt in memory causes further time and space complexities.

4.3 A Meta-language for Safety-critical Information

In this section, we describe a meta-language that can be used to retrieve information that can be fed efficiently to an LMaaS. We begin by specifying which expressions can be efficiently stored in a data structure such as a tree or a hash table. Despite its simplicity, our solution comes with the guarantee that a unit of information, for which we give an explicit definition in the next paragraph, will be memorized and fed to the LMaaS as part of the context v_t.

A unit of information is defined by the following regular expression

[R]expr=value[/R],

where expr represents an expression such as a variable name, and value its assignment. A valid unit of information is for example [R]x=2[/R], which instructs the LMaaS that a particular variable has been assigned the value of 2.

Similarly to the one previously defined, we define an expression to store information that we want is kept by the LMaaS. Formally, to retrieve a unit of information previously stored by an LMaaS we use the expression

[Q]expr[/Q].

Complex units of information. In our meta-language, one can specify more complex assignments such as [R]x+y=z-5[/R], which we resolve by isolating all the variables, in this case, x, y and z, and checking whether there is an assignment that satisfies them.

Another operation that we support is variables update, with expressions such as [R]x+=2[/R], which instructs our meta-language process to increase the value of x by 2.

As regards retrieving information previously stored, one can for example query an expression in the form [Q]x+y[/Q], for which our solution algorithm checks the presence in the data structure of the variables x and y, and then combines them to output the desired sum.

4.4 Storing, Retrieving and Updating Information Efficiently

Our implementation of the external memory system is through the use of a hash table. A hash table is a data structure used to store key-value pairs and uses a hash function to compute the index in an array by which an element can be inserted, searched, or deleted. The average complexity to search, insert, and delete data in a hash table is constant time and thus efficiently implements an external data structure. The dictionary element in python implements a hash table and is what will be used.

Our model requires a user to explicitly use "R" tags to tell our model that an expression should be remembered and "Q" tags to retrieve the value of remembered variables and expressions if they exist in memory. As such, for each user prompt, our model searches for R or Q tags before utilizing memory functions. If these tags are found, the expression is "cleaned" to strip these tags and then reformatted into symbolic terms to be used by the model.

For each expression passed to the model formatted by our meta language standard, the Python library SymPy is used to transform the expression into symbols and thus provides computer algebra capability. This is accomplished in the "infer assignment" function which parses an expression for mathematical symbol and solves for variable-value combinations. This transformation leads directly to key-value pairs as each variable can be considered a key and each variables assigned value can be considered a value.

In this way, the functions append, search, and delete are implemented. If an expression is passed to the model in the "Remember" ("R" tag) format, each symbolic variable term with an assigned value is updated in the hash table. If an expression is passed to the model in the "Query" ("Q" tag) format, the passed symbolic variable terms are searched for the hash table and if they exist, will be returned as the resulting computation of the complete passed expression. For example, if the value of x plus y is queried, and x equals 5 and y equals 10, the returned value will be the mathematical result of 5 plus 10, which is 15.

5 Experimental Evaluation

5.1 Prompt Engineering Experiments

In our experiments, code is run locally on a 2021 MacBook Pro using an Apple M1 Pro chip with 16GB of memory. Python 3.10.1 is used along with gpt-3.5-turbo and the Python libraries SymPy, re, OpenAi, and time.

To find the optimal gpt-3.5-turbo prompt to transform an expression in natural language into our meta language format, a prompt engineering test is conducted. Using four text files, 100 different 'Remember' prompts and 100 different 'Query' prompts are separately passed to the model to be transformed into our meta language format and compared to the expected format. In this way, x1.txt contains prompts like 'Bear in mind, y is 10' or 'Remember that y+x is z + 10' to be compared to the expected format in y1.txt. The expected format for these prompts would be [R]y=10[/R] and [R]y+x=z+10[/R]. The x2.txt file contains 'Query' prompts like 'What is x minus 3?" and "Tell me the value of z minus x' which are expected to be transformed to [Q]x-3[/Q] and [Q]z-x[/Q] The comparison of the produced transformation to the expected format also takes into account mathematically equivalent expressions by comparing every possible response combination to every possible expected combination, while considering mathematically important ordering that may change equivalence values.

To transform each of these natural language prompts into our meta language, gpt-3.5-turbo is passed a composition of each Remember and Query prompt with an auxiliary prompt like: 'Rewrite the following prompt in the format [Q]value[/Q] where value is expressed in terms of variables and operators'. Seven different 'Remember' auxiliary prompts are tested and six different 'Query' auxiliary prompts are tested. We are able to achieve 0.8469 accuracy for 'Remember' prompt transformations 5 and 0.78 accuracy for 'Query' prompt transformations 6.

5.2 External Data Structure Experiment

To test the performance of our external memory system, we attempt to succeed with the REM-X example. A series of prompts are passed to our model to display how an external memory-unsupported gpt-3.5-turbo model fails to remember the value of x after its internal memory is exceeded by the passing of more than five thousand characters of text. Then, using 'Remember' and 'Query' prompts in addition to our external memory system, the memory-supported gpt-3.5-turbo model can remember the value of x after the same long text is passed 7. Prompt 1 is 'Remember x equals 10. What is x?', to which the memory-unsupported model successfully responds that 'x equals 10'. Prompt 2 is 'Remember that x equals 10. long text.... What is the value of x?' where long text is 6166 characters long, to which the memory-unsupported model responds that 'The value of x is not mentioned in the given text'. Thus, the memory-unsupported model fails to remember the value of x due to its internal memory being exceeded.

Now, introducing our external memory-supported model, prompt 3 is [R]x=10[/R], to which our model responds giving the key-value assignment format of 'x': 10.0. Passing in the same 6166 character long text as prompt 4, the final prompt (prompt 5), queries for the value of x using the 'Query' tags: [Q]x[/Q], to which the memory-supported model successfully replies that 'The answer is 10.0'.

As shown, our memory-supported model succeeds in remembering simple mathematical information when the gpt-3.5-turbo model's internal memory capacity is exceeded and therefore passes on our REM-X example.

6 Future Works

Future work on this external memory system protocol would include the expansion of the compatible prompt domain. Currently, only simple mathematical inquiries are supported. By implemented an additional natural language processing model to support the reformatting of prompts to be viably stored in terms of key-values, importance and summary functions would have to be implemented to denote important strings to remember and efficiently minimized representations of these important strings to be stored. This advancement would allow for our system's success against the BROKE-LEG example, which currently is unsupported by our model. Another advancement could be the use of different data structures.

As discussed, to increase prompt memory for enhanced contextual understanding, an LLM can be paired with an external data structure. A Trie, also known as a prefix tree, could be implemented as a search tree to store prompts in the word space. Each node represents a character, and the strings are paths from the root to leaf nodes. Descendants of a node share a common prefix. By virtue of using a Trie, the following benefits arise:

1. Efficient storage and retrieval: prompts can be checked for earlier encounters and retrieve associated information
2. Pattern matching: prompts sharing a given pattern can be found, helping with conversational contexts
3. Memory management: by storing prefixes exactly once, Tries save memory.

The Tree could also implement the following key components:

1. Timestamp: Each node will include a timestamp to mark when the node has been inserted, retrieved, or updated.
2. Summarization
3. Forgetting mechanism: each node will be assigned a timestamp and include a "decay" value. Following the decay of human memory using the Ebbinghaus forgetting curve, memory retention will decrease exponentially over time unless the information is reviewed or recalled. Thus, each

Prompt Number	Accuracy	Prompt
One	0.2828	"Rewrite the following prompt in the format [R]variable=value[\R]"
Two	0.4646	"Rewrite the following prompt in the format [R]variable=value[\R] using operators and variables"
Three	0.5353	"Rewrite the following prompt in the format [R]variable=value[\R] or the form [R]variable+...+variable=value[\R]"
Four	0.6939	"Rewrite the following prompt in the format [R]variable=value[\R] or [R]equation[\R] where equation is expressed in terms of operators, variables, and or numbers"
Five	0.7959	"Turn the following list of sentences, expressed in human language, into the form [R]equation[\R], where equation is the equation expressed in natural language by each prompt:\n"
Six	0.8265	"Rewrite the following prompt in the format [R]equation[\R] where equation is an equation expressed in terms of variables and operators"
Seven	0.8469	"Rewrite the following prompt in the format [R]variable=value[\R] or the form [R]variable+...+variable=value[\R] or [R]variable=variable[\R] or [R]variable+...+variable=variable+...+variable[\R] or [R]variable+=variable[\R] or [R]equation[\R] where equation is expressed in terms of operators, variables, and or numbers:"

Figure 5: Accuracy varies across prompts used to transform a "Remember" prompt in natural language to our defined meta language.

Prompt Number	Accuracy	Prompt
One	0.2222	"Rewrite the following prompt in the format [Q]value[\Q] where value is expressed in terms of variables and operators"
Two	0.2323	"Rewrite the following prompt in the format [Q]equation[\Q] where equation is expressed in terms of operators, variables, and or numbers"
Three	0.6333	"Rewrite the following prompt in the format [Q]value[\Q] where value is equivalent to the value of the expression, using operators and variables as needed"
Four	0.7545	"Rewrite the following prompt in the format [Q]value[\Q] where value is equivalent to the value of the expression, remove all spaces in the answer"
Five	0.78	"Rewrite the following prompt in the format [Q]value[\Q] where value is equivalent to the value of the expression, using operators and variables as needed, remove all spaces in the answer"
Six	0.78	"Rewrite the following prompt in the format [Q]value[\Q] where value is equivalent to the value of the expression, using operators, variables, and numbers as needed, remove all spaces in the answer"

Figure 6: Accuracy varies across prompts used to transform a "Question" prompt in natural language to our defined meta language

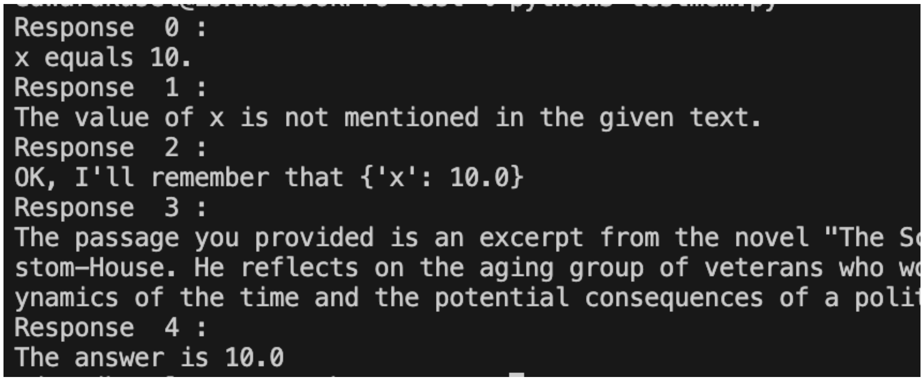

Figure 7: A memory-unsupported gpt-3.5-turbo model and a memory-supported gpt-3.5-turbo model are compared

node will evanesce as dictated by its decay value and time passed from its timestamp to present.

4. Priority markers: critical information in a given prompt can be marked by a critical tag, labeling the information as critical to memory. These prompts will contain a lower decay value or no decay value, making them less likely to be removed or non removable.

Hard memory will be remembered forever unless explicitly removed and is therefore similar to long-term memory. Soft memory will disappear after a small number of iterations unless the same information appears before the forgetting period, causing a delay in forgetting. Soft memory is therefore similar to short-term memory. As stated above, the forgetting mechanism can implement soft memory and a critical information marker can implement hard memory. The mechanism of hard and soft memory in the external tree data structure assists in maintaining optimal data volume by forgetting information unnecessary to context and remembering information important to context. This could be used to succeed where adequately representing important information becomes difficult.

7 Conclusions

In conclusion, this paper has highlighted the limitations of current LMaaS due to their lack of memory, which can lead to failures in certain applications. We have demonstrated these failures through the REM-X and BROKE-LEG examples, where the models fail to remember important information after their internal memory is artificially exceeded.

Our model enhances the statefulness of LMaaS through summarization and an external memory system. Our meta-language allows for the efficient storage and retrieval of information in a data structure such as a hash table. Our implementation of an external memory system shows that our memory-supported model can successfully remember simple mathematical information even when the internal memory capacity of gpt-3.5-turbo is exceeded.

Our implementation requires explicit user limitations and as such, future work could focus on expanding the compatible user prompts and making the process more automatic and intuitive. In summary, our model has proposed a new way to enhance the current capabilities of LMaaS and provides a foundation for future research in this area.

References

[1] A Baddeley, MW Eysenck, and MC Anderson. Memory. hove, england & new york, 2009.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[4] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[5] OpenAI. Introducing ChatGPT, 2022. Accessed on April 11, 2023.

[6] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[7] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[8] Dale Schuurmans. Memory augmented large language models are computationally universal. *arXiv preprint arXiv:2301.04589*, 2023.

[9] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[11] Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *arXiv preprint arXiv:2306.07174*, 2023.

[12] Wanjun Zhong, Lianghong Guo, Qiqi Gao, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023.