applied sciences

MDPI

Article

Training a Team of Language Models as Options to Build an SQL-Based Memory

Seokhan Lee [1] **and Hanseok Ko** [2,3,*]

[1] College of Informatics, Korea University, Seoul 02841, Republic of Korea
[2] School of Electrical Engineering, Korea University, Seoul 02841, Republic of Korea
[3] Department of Electrical and Computer Engineering, Catholic University of America, Washington, DC 20064, USA
* Correspondence: hsko@korea.ac.kr or koha@cua.edu

Abstract

Despite the rapid progress in the capabilities of large language models, they still lack a reliable and efficient method of storing and retrieving new information conveyed over the course of their interaction with users upon deployment. In this paper, we use reinforcement learning methods to train a team of smaller language models, which we frame as options, on reward-respecting subtasks, to learn to use SQL commands to store and retrieve relevant information to and from an external SQL database. In particular, we train a storage language model on a subtask for distinguishing between user and assistant in the dialogue history, to learn to store any relevant facts that may be required to answer future user queries. We then train a retrieval language model on a subtask for querying a sufficient number of fields, to learn to retrieve information from the SQL database that could be useful in answering the current user query. We find that training our models on their respective subtasks results in much higher performance than training them to directly optimize the reward signal and that the resulting team of language models is able to achieve performance on memory tasks comparable to existing methods that rely on language models orders of magnitude larger in size. In particular, we were able to able to achieve a 36% gain in accuracy over a prompt engineering baseline and a 13% gain over a strong baseline that uses the much larger GPT-3.5 Turbo on the MSC-Self-Instruct dataset.

Keywords: language models; options; memory; SQL

check for updates

Academic Editor: Rui Araújo

Received: 19 September 2025
Revised: 18 October 2025
Accepted: 22 October 2025
Published: 24 October 2025

Citation: Lee, S.; Ko, H. Training a Team of Language Models as Options to Build an SQL-Based Memory. *Appl. Sci.* **2025**, *15*, 11399. https://doi.org/10.3390/app152111399

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Conversational AI has made much progress in recent years largely due to advances in the capabilities of the large language models [1–6] that power its interactions. However, there are many areas that are still in need of improvement, such as the tendency for language models to hallucinate [7–10] and the lack of reliable safety mechanisms [11–15] to prevent their misuse. One particularly notable deficiency that greatly limits their utility is the lack of a reliable mechanism for implementing long-term memory. Language models in their current form possess two different intrinsic means of storing and retrieving information to and from memory. The first, which is analogous to short-term or working memory, is the context of the prompt that is input to the language model. Knowledge that is imparted to the language model by means of the context can be reflected insofar as the language model is sufficiently capable of extracting and making use of the relevant information contained in the entire prompt it is fed. This capability varies widely by model and by the length of the

context. Studies have shown that the longer the context input to a language model, the more difficult it becomes for the language model to reflect the information it contains [16–19]. Furthermore, there is a hard cutoff in the form of the maximum context length supported by the language model, which restricts the amount of text that can physically be input into the language model. Once this maximum has been exceeded, no further information can be conveyed to the language model through the context.

The second means of using memory involves storing knowledge directly in the weights of the model by means of training or finetuning. This allows language models to store and retrieve information contained in the datasets on which they were trained without the need for explicitly entering the information into context and effectively acts as a form of long-term memory. Unfortunately, current deep learning-based architectures are highly brittle in that, after they have undergone an initial pre-training phase, any further finetuning tends to induce catastrophic forgetting, whereby the learning of new information comes at the cost of forgetting much of the information from the data on which it was originally trained [20–26]. Another disadvantage of this approach is that it is impossible to verify exactly what pieces of knowledge have been stored into memory as the knowledge does not take an explicit human-readable form. This entails the further issue of not being able to cleanly isolate and edit or delete specific records that have already been stored.

The limitations of these two means of storing and retrieving information suggest the need for a more lightweight, modular, and interpretable form of memory, whereby information is stored in a database external to the language model. Facts extracted from the dialogue between the language model and user can then be written to the database and relevant facts can be retrieved and entered into context (working memory) when necessary. In the present work, we propose a method for doing this by assigning the task of storing facts to a designated storage language model and the task of retrieving facts to a retrieval language model, which together form an auxiliary team of language models that assist an interchangeable generation language model in managing and making use of an external SQL database as a form of long-term memory. Our choice to use SQL in particular, as opposed to other options such as vector databases and knowledge graphs, was motivated by the rapid progress being made in the coding abilities of language models and in their capabilities as agents. By using SQL as the mode of storage, our framework can easily leverage all the advancements being made on these fronts and will naturally become more powerful and capable with every upgrade in the language models themselves.

We frame our auxiliary language models as options and train them on reward respecting subtasks to optimize them for the tasks they have been assigned and demonstrate empirically that this method results in much higher performance on memory-related tasks than training the language models to maximize the reward signal directly.

1.1. Background of Options and Reward-Respecting Subtasks

The notion of options is introduced in [27] as a means of generalizing actions, which are typically defined for a single time step, to include action sequences that occur over an arbitrary number of time steps, thus enabling temporal abstraction in the decision making of a given agent.

Formally, an option consists of a policy, $\pi(a|s)$, that determines the probability of selecting action $a \in A_s$ from a given state $s \in S$ and a stopping condition $\beta(s)$ that determines the probability of terminating the option at each state $s \in S$, where S is the set of all possible states and A_s is the set of all possible actions from state s. In the case of our language models, we frame the storage model and retrieval model as two distinct options, where the set of states S consists of all possible token sequences and the actions are the next tokens given the current token sequence. Hence, $\pi(a|s)$ gives the probability of each

possible token a being the next token given the current token sequence s and the stopping condition $\beta(s)$ is simply the probability of the next token being the end of sequence token (typically represented in text form as "<EOS>") given the current sequence s which ends the generation of tokens and terminates the option.

Given that our options essentially consist of next-token probability distributions which include the end-of-sequence token as part of the action set for each state, in order to produce distinct options, it is sufficient to add a prefix (and optionally a suffix) to the input sequence to alter the probability distributions for each option. For example, by adding "You are an AI translator. Translate the following text." as a prefix to the input sequence, we obtain next-token probabilities that are different from an option that adds "You are an AI grammar checker. Find any mistakes in the following text." as a prefix to the input sequence. However, we can go further and modify these probability distributions to better serve the intended purpose of each option by training the options (language models in our case) on their respective subtasks. In [28], it was shown that when training options on subtasks, it is often more effective to include the main reward in the learning objective in order to prevent the options from learning behavior that accomplishes their subtasks in ways that diverge from the overall goal of maximizing the main reward. In designing the learning objectives for our options, we follow this approach and demonstrate that training our language models on reward-respecting subtasks results in higher performance than either training on the subtasks alone or training to maximize the reward directly.

1.2. Related Work

Attempts to build a functioning long-term memory for language models fall roughly into the two approaches mentioned above: parametric methods, which aim to store and retrieve information in the weights of the language models; and non-parametric methods, which store information in an external database to be retrieved later and added to context when necessary.

1.2.1. Parametric Methods

In view of the catastrophic forgetting issue mentioned above, there have been recent attempts to develop methods that adjust the model parameters to reflect new information in such a way that minimizes the impact on unrelated, pre-existing knowledge. These methods, commonly known as model editing, include MEND [29], which trains a hypernetwork to take a standard gradient update as input and calculate a one-shot single-rank matrix update to the model weights to reflect the new information; ROME [30] and MEMIT [31], which use causal traces to identify which layers are most responsible for storing facts and then calculate the change in weights in those layers required to maximize the probability of the desired output for a given input; and GRACE [32], which shares more in common with the non-parametric methods than other model editing methods in that it does not adjust the weights of the original model (it adjusts the hidden state vectors output by adaptors that wrap designated layers to modify the final output of specified inputs) and it uses distance between hidden state vectors to determine the scope of the inputs to which its edits are applicable (similar to how retrieval using sentence embeddings uses distance between embedding vectors to determine which sentences to retrieve). However, it should be noted that none of these methods have satisfactorily solved the issue of catastrophic forgetting [33] and that this problem may be unsolvable with the current architectures used in language models.

1.2.2. Non-Parametric Methods

Perhaps the most familiar non-parametric method for storing and retrieving information is the retrieval-augmented generation (RAG) method introduced in [34]. This method

splits all the text that needs to be stored into chunks and uses a sentence embedding model to generate an embedding vector for each chunk. The text chunks and their embeddings are stored in a vector database and in order to search for text chunks that are relevant to the current user query, an embedding is generated for the user query and a search is conducted to find the text chunks in the database with the highest cosine similarity to the user query embedding. These chunks are then added to the prompt input to the language model to help in answering the user query.

A method that stores and retrieves the hidden state key and value vectors from past dialogue (that have been cutoff due to context size limits) to be included along with the current input tokens in attention calculations and thus has overlap with both the RAG method and GRACE is proposed as LongMem [35]. Another method called MemoryBank [24] relies directly on RAG procedures for storing and retrieving past dialogue while adding summaries at various levels of granularity and implementing a mechanism for deleting stale memories based on the Ebbinghaus forgetting curve.

The method that might admit of the closest comparison with our method is MemGPT [36]. MemGPT uses function calls generated by a large language model (LLM) to store and retrieve information into an external archive and adds any retrieved information to the context, which also includes system instructions and a rolling queue of the most recent dialogue history. It should be noted that the performance of this system relies critically on the general capabilities (and hence on the size) of the language model as the instructions for generating function calls are quite lengthy and require the language model to accurately interpret and follow them zero-shot (without any training). In addition to the lengthy system instructions, along with the retrieved information and the rolling dialogue history, a summary of past dialogue is also included in context, from which it becomes clear why the paper only includes results from experiments using the largest models (over 175B parameters in size).

The method we propose is a non-parametric method but differs fundamentally from all the aforementioned methods in that it does not use sentence embeddings to search for relevant information. With our approach, there is no need for any of the accompanying overhead for managing vector databases such as splitting text into chunks, using a sentence embedding model to generate sentence embeddings, implementing a method for efficiently carrying out cosine similarity search, and so forth. In contrast to MemGPT, our method makes use of two very small auxiliary language models to carry out the storage and retrieval functions, which allows easy adaptability to any language model being used for generation (as opposed to being limited to only the largest models), with minimal impact on the performance of memory functionality.

2. Materials and Methods

2.1. Problem Formulation

2.1.1. Dataset

In order to specify the task more precisely, we must begin by describing the content of the dataset we are using to benchmark the performance of our method. The dataset, called MSC-Self-Instruct is taken directly from [36]. It starts with the dialogue data from the original Multi-Session Chat dataset released with [37] wherein each "chat" contains multiple sessions of dialogue between two speakers with distinct personas (fixed throughout the duration of each chat), with arbitrary intervals of time intervening between each session. At the end of each chat, a language model is used to generate a question posed by one of the users that requires knowledge of the prior dialogue history to answer correctly, along with an answer that is taken to be the ground truth.

In Table 1, we show the three fields that we use for one instance of the multi-session chats. The "Personas" field contains short descriptions of each of the speakers taking part in the dialogue sessions. The "Dialog" field contains a total of five dialogue sessions per multi-session chat, a partial selection of which is shown for one session in the table. The "Self-Instruct" field contains a question posed by the second speaker to the first speaker that requires knowledge of the dialogue history to answer correctly, followed by the answer. The dataset consists of 500 total multi-session chats.

Table 1. Sample data from MSC-Self-Instruct.

Personas
["I'm a graduate student. I study psychology and love reading new research.", "I like to kayak. I have diverse music tastes.", "I enjoy a lot of different music including classical and heavy metal.", "I am working on my thesis as a part of my studies. My thesis is on non-verbal communications. I read a lot of science journals. I like autobiographies and science journals.", "I read a scientific article on non-verbal communication.", "I worry about younger generation addicted to latest technology.", "I completed my thesis. I'm working to get my PhD."]
["I teach elementary school, like my parents did. I teach 3rd grade.", "I love board games.", "My favourite band is Up, I've been to a concert of theirs.", "I love teaching reading the most but love teaching all subjects. I have a little reading area with nonfiction and fiction books in my classroom. My students gravitate towards fictional fairy tales and sci-fi more than factual writings.", "My students are genuinely like to learning, playing Dominoes, chutes and ladders. I avoid computer in the classroom."]
Dialog
"Speaker 1": "Now that I've finished my thesis I think I'll finally have time to go out kayaking again." "Speaker 2": "I bet that is very exciting. I hope you have an amazing time. Where will you go to do that at?" "Speaker 1": "I will probably travel to Utah and California, I heard there is great kayaking there." "Speaker 2": "I am glad you know where you are headed! When will you get your grade on the Thesis?" ...
Self-Instruct
"Speaker 2": "Hey, remember that time we talked about music? What did I say was my favorite band?" "Speaker 1": "You said your favorite band is Up."

2.1.2. Main Task

The task that we pose is as follows. Given access to each of the dialogue sessions in a given multi-session chat, store any necessary information in an external database, and upon completion, answer the question from the "Self-Instruct" field by retrieving the relevant information from the database and passing it along with the question into a language model designated for text generation (the generation LLM). The final generated response is passed along with the question and ground truth answer to a language model designated for evaluation (the LLM judge), which generates a label of "CORRECT" or "WRONG". Performance is measured as the percentage of generated answers that are given the label "CORRECT".

2.2. Methodology

Our approach to solving the task involves assigning the function of storing information from dialogue into the database to a designated language model, which we term the storage language model; and likewise, assigning the function of retrieving relevant information

from the database to another language model, which we term the retrieval language model. The storage language model will be denoted $p_{sto}(a_t|s_t, \theta_{sto})$, which indicates that it is a function parameterized by θ_{sto} for generating the probability of selecting a_t as the next token following the input token sequence s_t at time t. Once the use of the storage language model is selected, tokens are generated according to $p_{sto}(a_t|s_t, \theta_{sto})$, adding each new token a_t to s_t at each timestep such that $s_{t+1} = [s_t, a_t]$ (where $[.,.]$ indicates concatenation) until the <EOS> token is generated (with probability $p_{sto}(< EOS > |s_t, \theta_{sto})$), which terminates the use of the model. And likewise, use of the retrieval language model results in tokens being generated according to $p_{ret}(a_t|s_t, \theta_{ret})$ until the <EOS> token is generated with probability $p_{ret}(< EOS > |s_t, \theta_{ret})$. Hence, we can see that these two language models represent two different options that, upon initiation, generate sequences of actions over an indefinite number of timesteps.

In the case of p_{sto}, the inputs s_t that we use consist of each pair of utterances in the dialogue history, entered sequentially with a stride of 1, along with the current contents of the SQL table being used to store information. In Table 2, we show an example of how the lines in the dialogue history are split into pairs to be fed as input into the storage language model. The outputs that we expect from p_{sto} are SQL commands for storing relevant information from the given dialogue into a SQL table. These commands are executed by a Python script to update the SQL table.

Table 2. Sample inputs for the storage function.

Dialog
"Speaker 1": "Now that I've finished my thesis I think I'll finally have time to go out kayaking again." "Speaker 2": "I bet that is very exciting. I hope you have an amazing time. Where will you go to do that at?" "Speaker 1": "I will probably travel to Utah and California, I heard there is great kayaking there." "Speaker 2": "I am glad you know where you are headed! When will you get your grade on the Thesis?"
Sequential Inputs for the Storage Function
"Speaker 1": "Now that I've finished my thesis I think I'll finally have time to go out kayaking again." "Speaker 2": "I bet that is very exciting. I hope you have an amazing time. Where will you go to do that at?"
"Speaker 2": "I bet that is very exciting. I hope you have an amazing time. Where will you go to do that at?" "Speaker 1": "I will probably travel to Utah and California, I heard there is great kayaking there."
"Speaker 1": "I will probably travel to Utah and California, I heard there is great kayaking there." "Speaker 2": "I am glad you know where you are headed! When will you get your grade on the Thesis?"

As for p_{ret}, the inputs s_t consist of the final question for each multi-session chat along with a list of the columns from the current SQL table. The expected outputs are SQL commands for retrieving information that could be helpful in answering the final question. A Python script is used to execute the SQL commands and the retrieved information is added to the prompt along with the question, to be entered into the generation LLM to produce the final answer. The overall process is depicted in Figure 1 and the data schema is presented in Figure 2.

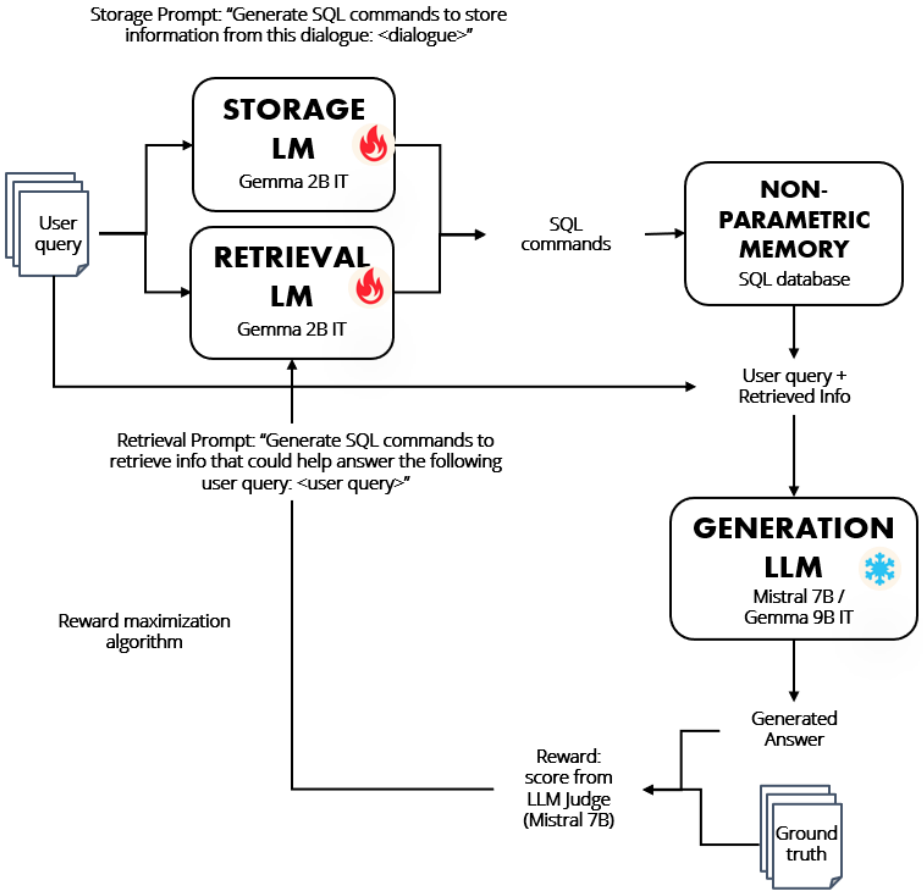

Figure 1. System architecture diagram showing the interaction between the generation model, the auxiliary storage and retrieval models, and the SQL database. Upon receiving a user query, the storage model is prompted to generate an SQL command to store any important new information into the SQL database. At the same time, the retrieval model is prompted to generate an SQL command to retrieve any information relevant to answering the user query. The retrieved information is passed along with the user query to the generation language model, which generates the final response. An LLM judge is used to score the accuracy of the response and the score is used as a reward signal to train the storage and retrieval models.

Table: `persistent_memory`
Primary key: `speaker_id`

`speaker_id`	`TEXT / INT` (PK)
`work_status`	`TEXT NULL`
`current_work_status`	`TEXT NULL`
`movie_watched`	`TEXT NULL`
`favorite_genre`	`TEXT NULL`
...	

Design. Schema-evolving, wide-table layout with one row per interlocutor. When a new fact type is extracted, a new nullable column named after the fact type is added (e.g., `ALTER TABLE persistent_memory ADD COLUMN <fact_type> TEXT`). Only the row(s) for speakers with known values are populated.

Figure 2. Relational data model for the proposed persistent memory. The system maintains a single table keyed by `speaker_id` and adds columns on demand for newly discovered fact types.

2.2.1. Team of Language Models

By assigning different responsibilities to the two language models, we have effectively formed a team with each member responsible for carrying out its assigned function. The

expectation is that forming such a team with each member specialized to carry out a specific function will be more effective than using a larger language model that carries out all functions on its own. To test this hypothesis, we must begin by optimizing each language model to better carry out the role it has been assigned. A first step in this direction is to simply handcraft the prompts that are used to modify the text input to each model. For instance, if $\pi_{sto}(a_t|s_t, \theta_{sto})$ is the base language model used for storage, the function p_{sto} could be defined as:

$$p_{sto}(a_t \mid s_t, \theta_{sto}) = \pi_{sto}(a_t \mid [prefix_{sto}, s_t], \theta_{sto}) \quad (1)$$

where $prefix_{sto}$ are the instructions (including the contents of the current SQL table) that we concatenate with the input dialogue s_t to convey to the language model that it should generate SQL commands for storing information about the speakers contained in the dialogue into a SQL table.

Likewise, we can define p_{ret} as:

$$p_{ret}(a_t \mid s_t, \theta_{ret}) = \pi_{ret}(a_t \mid [prefix_{ret}, s_t], \theta_{ret}) \quad (2)$$

where $prefix_{ret}$ are instructions (that include the columns of the current SQL table) for generating SQL commands to retrieve information relevant for answering the user query s_t. Thus far nothing has been done to modify the underlying models and if the same base model is used for p_{sto} and p_{ret}, then $\theta_{sto} = \theta_{ret}$. To further optimize each member of the team to carry out their respective tasks, we train the underlying models as outlined in the following section.

2.2.2. Storage Subtask

Although we have divided the responsibilities of the two members of our team, their tasks are still part of the main task and serve a common objective. We now introduce subtasks whose objectives diverge from those of the main task, the pursuit of which we hope will help to achieve the objectives of the main task more effectively than directly pursuing the main task directly.

In the case of p_{sto}, one observation we made when using the baseline as described above, is that it would often generate SQL commands that confuse Speaker 1 with Speaker 2. At times, it would store facts about Speaker 1 under the records for Speaker 2 and vice versa. An example is shown in Table 3 where we can see that the storage function has either updated the records for the wrong speaker or mistakenly updated the records for both speakers for every field (this example is not meant to show the typical case, but merely to show that the issue exists). Thus, we postulate that training the storage function to distinguish between Speaker 1 and Speaker 2 could be more effective than optimizing against the main reward itself.

To train the storage function on this subtask, we make use of the "Personas" in the MSC-Self-Instruct dataset. As shown in Table 4, we take the personas in each multi-session chat and form a pseudo-dialog using each line in the persona for each speaker as their line in the pseudo-dialog and alternating between Speaker 1 and Speaker 2. Letting P_x represent the persona for Speaker 1 and P_y represent the persona for Speaker 2, we alternate between each line $x_i \in P_x$ and each line $y_i \in P_y$ to create the pseudo-dialog $[x_1, y_1, x_2, y_2, \ldots]$. We then split the pseudo-dialog into pairs $[x_i, y_i], [y_i, x_{i+1}], [x_{i+1}, y_{i+1}], \ldots$ as explained above, to be used as inputs during training. The advantage of using the personas instead of the actual dialogue sessions from the dataset is that each line in the personas is self-contained in that it does not require knowledge of the preceding line to determine its precise meaning. For instance, in actual dialogue, a line may simply consist of the word "yes", which requires the preceding line to interpret correctly, but with personas, no such context is necessary.

Table 3. Sample SQL table updated by baseline storage function.

Dialog				
"Speaker 1": "Hello what are doing today?" "Speaker 2": "I am good, I just got off work and tired, I have two jobs." "Speaker 1": "I just got done watching a horror movie" "Speaker 2": "I rather read, I've read about 20 books this year."				
Speaker	work_status	current_work_ status	movie_watched	favorite_genre
Speaker 1	Two jobs	Two jobs	Horror movie	None
Speaker 2	Two jobs	None	Horror movie	Horror

Table 4. Sample inputs for training the storage function.

Personas
["I'm a graduate student. I study psychology and love reading new research.", "I like to kayak. I have diverse music tastes." …]
["I teach elementary school, like my parents did. I teach 3rd grade.", "I love board games." …]
Pseudo-Dialog
"Speaker 1": "I'm a graduate student. I study psychology and love reading new research." "Speaker 2": "I teach elementary school, like my parents did. I teach 3rd grade." "Speaker 1": "I like to kayak. I have diverse music tastes." "Speaker 2": "I love board games."
Sequential Inputs for the Storage Function
"Speaker 1": "I'm a graduate student. I study psychology and love reading new research." "Speaker 2": "I teach elementary school, like my parents did. I teach 3rd grade."
"Speaker 2": "I teach elementary school, like my parents did. I teach 3rd grade." "Speaker 1": "I like to kayak. I have diverse music tastes."
"Speaker 1": "I like to kayak. I have diverse music tastes." "Speaker 2": "I love board games."

Using this fact, we are able to generate the ground truth label for the storage function by inputting only the second of the two lines in each pair and instructing the language model (we use a frozen copy p'_{sto} of the storage language model p_{sto}) to generate SQL commands for updating the SQL table with information about the corresponding speaker. Relabeling the indices of the utterance pairs as $[x_i, y_i] = [x, y]^1, [y_i, x_{i+1}] = [y, x]^1, [x_{i+1}, y_{i+1}] = [x, y]^2, [y_{i+1}, x_{i+2}] = [y, x]^2, \ldots$, we delete the first utterance in each pair to get $[y]^1, [x]^1, [y]^2, [x]^2, \ldots$ and generate the SQL commands $z_y^j = [z_0, \ldots, z_n, < EOS >]$ sampled according to $z_t \sim p'_{sto}(a_t \mid [[y]^j, z_0, \ldots, z_{t-1}])$, which become the labels for the utterances ending with Speaker 2 and $z_x^j = [z_0, \ldots, z_n, < EOS >]$ sampled according to $z_t \sim p'_{sto}(a_t \mid [[x]^j, z_0, \ldots, z_{t-1}])$, which become the labels for utterances ending with Speaker 1. The generated SQL commands should only contain information about the last speaker in the pair of utterances and it should only update the records for that speaker since only their line was passed to the language model. The full utterance pairs $[x, y]^j$ and $[y, x]^j$ are then passed as input to the storage language model being trained and it is trained to maximize the likelihood of generating the corresponding ground truth labels z_y^j and z_x^j, respectively. In practice, we aim to minimize the negative log likelihood $L = L_y + L_x$, where

$$L_y = -\sum_{j=1}^{N} \log p_{sto}(z_y^j \mid [x, y]^j, \theta_{sto}) \tag{3}$$

and

$$L_x = -\sum_{j=1}^{N} \log p_{sto}(z_x^j \mid [y, x]^j, \theta_{sto}) \tag{4}$$

which we minimize using standard gradient descent methods, thus encouraging the language model to only output SQL commands corresponding to the latter speaker when presented with a pair of utterances from the dialogue history (and hence achieving the subtask goal of distinguishing between Speaker 1 and Speaker 2). An illustration of this process is shown in Figure 3. This suffices to train p_{sto} to achieve its subtask but to do so in a way that does not diverge from achieving the main task requires us to convert the subtask into a reward-respecting subtask. We do this by making the gradient update contingent on receiving a reward.

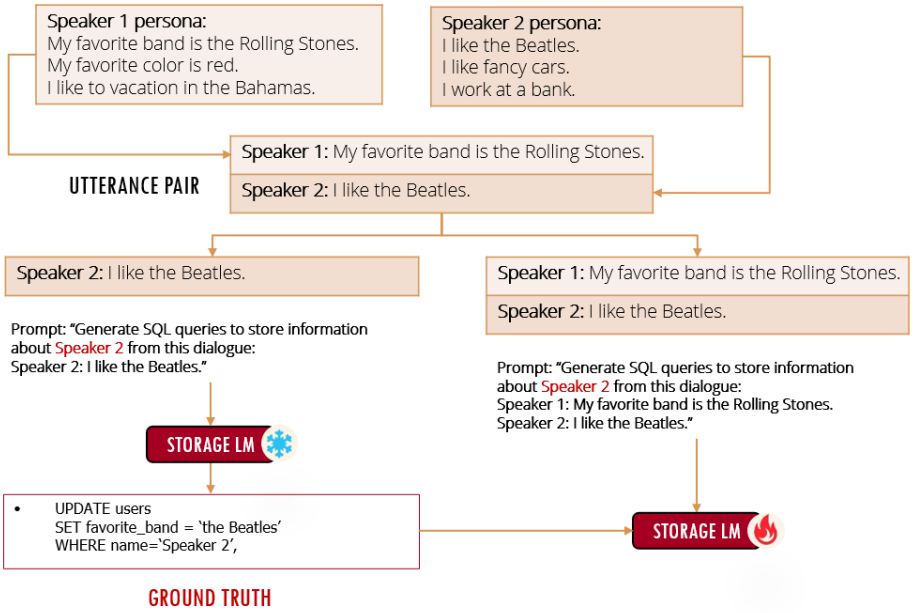

Figure 3. Lines are taken from each speaker's persona to form an utterance pair. The line for Speaker 2 is input to a frozen copy of the storage language model to generate the ground truth label. The full utterance pair is input to the storage language model being trained and the model is trained to maximize the probability of generating the ground truth label.

As we generate the labels z_y^j and z_x^j, which consist of SQL commands, we execute them using a Python script to store facts about the speakers into a SQL table. Once all of the labels for a given multi-session chat have been generated and executed, we pass the final question q into the retrieval model p_{ret} to generate a series of SQL commands $r = [r_0, \ldots, r_n, < EOS >]$, which is sampled according to $r_t \sim p_{ret}(a_t \mid [q, r_0, \ldots, r_{t-1}])$. We then execute the SQL commands in r to retrieve information i from the SQL table and enter the information, along with the question q, into the generation LLM p_{gen} to generate the final predicted answer $\hat{g} = [g_0, \ldots, g_n, < EOS >]$ according to $g_t \sim p_{gen}(a_t \mid [i, q, g_0, \ldots, g_{t-1}])$, where p_{gen} adds its own instructions $prefix_{gen}$ to the prompt for generating answers. Finally, the predicted answer $\hat{g}$ is passed along with the question q and the ground truth answer g to the LLM judge, p_{judge}, which adds instructions $prefix_{judge}$ for generating either the token "CORRECT" or "WRONG" and we obtain the probability of "CORRECT" as $c = p_{judge}("CORRECT" \mid [q, g, \hat{g}])$. If c is greater than some threshold λ, then we give a reward R of 1 and the reward is 0 otherwise.

Now, we multiply the loss function by the reward such that the weight update based on gradient descent is as follows:

$$\begin{aligned} \theta_{sto,t} &= \theta_{sto,t-1} - \alpha R \nabla L \\ &= \theta_{sto,t-1} + \alpha R \nabla \Big(\sum_{j=1}^{N} \log p_{sto}(z_y^j \mid [x,y]^j, \theta_{sto,t-1}) + \sum_{j=1}^{N} \log p_{sto}(z_x^j \mid [y,x]^j, \theta_{sto,t-1}) \Big) \end{aligned} \quad (5)$$

where α is the step-size parameter, ∇L is the gradient of the loss function as specified in Equations (3) and (4), $\theta_{sto,t-1}$ represents the weights of the storage model at time $t-1$, and $\theta_{sto,t}$ are the weights at time t. This turns our update into an instance of the REINFORCE algorithm [38], which ensures that the likelihood is only increased for labels that have ultimately resulted in a correct answer being generated. To address reward sparsity, when the reward is 0, we repeat this process for the same multi-session chat until the reward is 1 up to a specified maximum number of attempts.

2.2.3. Retrieval Subtask

In order to train p_{ret}, we follow the same procedure as above using the SQL commands r generated by p'_{ret}, which is a frozen copy of p_{ret}, as the ground truth label and maximizing the likelihood of r when it results in a correct answer being generated. The loss function is thus defined as:

$$L_r = -\log p_{ret}(r \mid q, \theta_{ret}) \tag{6}$$

And the update is:

$$\begin{aligned} \theta_{ret,t} &= \theta_{ret,t-1} - \alpha R \nabla L_r \\ &= \theta_{ret,t-1} + \alpha R \nabla \log p_{ret}(r \mid q, \theta_{ret}) \end{aligned} \tag{7}$$

Again, repeating the generation of r when the reward is 0 up to a maximum number of attempts. This suffices to make the task reward-respecting, but since it is directly optimizing against the reward, it is not yet a subtask. One weakness that we observed with the baseline retrieval model was that in many cases where the predicted answer was incorrect, the required information was present in the SQL table (but not retrieved) and could have been retrieved if more fields had been queried. Even when instructions to retrieve a specified number of fields are included in $prefix_{ret}$, these instructions are not reliably followed. Thus, we define our retrieval subtask as retrieving a number of fields within a desired range $[n_{low}, n_{high}]$. If the number of fields retrieved n falls within the range, a small bonus is added to the reward R to encourage achieving the subtask of retrieving a sufficient number of fields. Table 5 shows an example of SQL commands generated by p_{ret} before training that omit the necessary information due to not querying a sufficient number of fields.

Table 5. Sample SQL commands generated by baseline retrieval function.

Self-Instruct	
Question: "Hey, remember that time we talked about your favorite food? What do you love to have on your burgers?" Ground Truth: "Bacon!"	
SQL Command	Result
SELECT favorite_food FROM users WHERE name = 'Speaker 2'	None
SELECT favorite_food FROM users WHERE name = 'Speaker 1'	'Hot dogs'

3. Results

3.1. Experimental Setup

To test the effectiveness of our methods, we evaluated the performance of our fine-tuned team of options on 50 held-out, multi-session chats from the MSC-Self-Instruct dataset. For the base storage and retrieval language models, we used separate instances of Gemma 2 2B Instruct (quantized). For the generation language model and LLM judge, we used Mistral 7B Instruct v 0.3 (quantized). Our metric (accuracy) is the average of the

scores (1 for "CORRECT", 0 for "WRONG") output by the LLM judge for each chat given the final question, the ground truth answer, and the generated answer.

Experiments were carried out using a server with a single RTX 4090 GPU, Intel Xeon E5-2620 v4 CPU, 64GB RAM running Ubuntu 24.04.3 LTS with Python 3.12. The environment used PyTorch 2.5.1, Transformers 4.46.2, bitsandbytes 0.44.1, peft 0.13.2, accelerate 1.1.1, sentencepiece 0.2.0, and protobuf 5.28.3. We used SQLite 3 to process the generated SQL commands.

The trainer class was implemented in Python 3.12 using the Hugging Face Transformers and PEFT frameworks. Models were loaded in 4-bit NF4 quantization via the BitsAndBytesConfig to reduce memory overhead. Finetuning was performed with LoRA adapters (rank = 8, α = 32, dropout = 0.05) targeting the query and value projection layers (q_proj, v_proj). The optimizer used AdamW with a learning rate of 10^{-4} and no warm-up steps; a linear scheduler controlled decay across 5000 training steps. We set batch size = 1 due to resource constraints.

3.2. Quantitative Results

In Table 6, the "Baseline Team of Options" uses only text instructions added to the prompt to define the storage and retrieval options. The storage option is then finetuned to maximize the reward only to obtain the "Finetuned Team of Options (reward only)". The "Finetuned Team of Options (subtask only)" is obtained by training the storage option on the subtask without taking the reward signal into account (i.e., reward is fixed to equal 1). The "Finetuned Team of Options (reward-respecting subtask)" is our proposed method which trains the storage option on the reward-respecting subtask described above. The remaining figures are those reported for MemGPT in [36] (as we do not have access to the OpenAI API, we were unable to perform the evaluation of our methods with the exact same specifications as the MemGPT methods, which used GPT 4 as the LLM judge, so the table is not an exact comparison).

Table 6. Performance on MSC-Self-Instruct test set.

Method	Accuracy
Baseline Team of Options	0.56
Finetuned Team of Options (reward only)	0.53
Finetuned Team of Options (subtask only)	0.58
Finetuned Team of Options (reward-respecting subtask)	0.76
GPT-3.5 Turbo + MemGPT	0.67
GPT-4 + MemGPT	0.92
GPT-4 Turbo + MemGPT	0.93

From the table, we can see that finetuning to maximize the reward only actually leads to a drop in performance from 0.56 to 0.53, whereas by finetuning on the subtask, we were able to increase it to 0.58. This confirms our hypothesis that training on the subtask in this instance is more effective than training directly on the main task. By including the reward in the subtask objective, we were able to increase the performance even further to 0.76. This level of accuracy is comparable to the much more complicated MemGPT method that makes use of language models vastly larger in size (i.e., GPT-3 was reported to have approximately 175 billion parameters [4]).

3.3. Qualitative Results

After finetuning, the example shown earlier in Table 3, where the storage function confused the two speakers in storing information from the dialogue into the SQL table, is now fixed as the finetuned model correctly stores information for each speaker in the

correct record, as shown in Table 7. Also, the example shown in Table 5, wherein the retrieved information did not contain the necessary information to answer the given question correctly, now retrieves enough fields to answer the question, as shown in Table 8.

Table 7. Sample SQL table updated by finetuned storage function.

Dialog				
"Speaker 1": "Hello what are doing today?" "Speaker 2": "I am good, I just got off work and tired, I have two jobs." "Speaker 1": "I just got done watching a horror movie" "Speaker 2": "I rather read, I've read about 20 books this year."				
Speaker	work_status	current_work_ status	movie_watched	favorite_genre
Speaker 1	None	'Working on tasks'	'I just got done watching a horror movie'	None
Speaker 2	Two jobs	None	None	'Speaker 2 likes to read and has read 20 books this year.'

Table 8. Sample SQL commands generated by finetuned retrieval function.

Self-Instruct	
Question: "Hey, remember that time we talked about your favorite food? What do you love to have on your burgers?" Ground Truth: "Bacon!"	
SQL Command	Result
SELECT favorite_food FROM users WHERE name = 'Speaker 2'	None
SELECT burger_preference FROM users WHERE name = 'Speaker 2'	None
SELECT cooking_skills FROM users WHERE name = 'Speaker 2'	'Speaker 2: Can cook bacon and make a salad with bacon bits.'
SELECT soap_kitchen_bacon_perference FROM users WHERE name = 'Speaker 2'	None
SELECT campfire_songs_preference FROM users WHERE name = 'Speaker 2'	None
SELECT favorite_food FROM users WHERE name = 'Speaker 1'	'Hot dogs'
SELECT burger_preference FROM users WHERE name = 'Speaker 1'	'I love burgers and bacon so we could do both.'
SELECT cooking_skills FROM users WHERE name = 'Speaker 1'	None
SELECT soap_kitchen_bacon_perference FROM users WHERE name = 'Speaker 1'	'Awesome. Everyone at my soap kitchen would love to hvae bacon, not often on the menu'
SELECT campfire_songs_preference FROM users WHERE name = 'Speaker 1'	'Yes'

4. Discussion

A definitive solution to the problem of long-term memory for language models is still out of reach. Methods that aim to encode the memory into the language model weights, like finetuning or model editing, result in catastrophic forgetting after only a

few rounds of weight updates. Meanwhile, retrieval-based methods cannot make use of all the disparate pieces of information in an external database that may be relevant to generating a satisfactory response. However, our results show that by training separate language models in a team, each specialized to carry out their own distinctive role, we can achieve a system that acts to retrieve the required memories, much as a human would, by using external tools and databases. As progress is made on LLM-based AI agents and, in particular, multi-agent systems [39–44], these capabilities can be made more robust and the problem of long-term memory can be usefully transformed into a problem of taking actions and interacting with an external environment to achieve the intended goal. Hence, being able to capably store and retrieve the right information may be a viable path to achieving what is effectively a form of long-term memory without having to deal with solving the problem of catastrophic forgetting. This is akin to a person with bad memory making use of notes and computers to aid in recording and recalling past events. Even if a reliable parametric method for long-term memory is found, it is unlikely that the memory capacity is unlimited. Hence, even in this case, it will help to add the capability explored here for making use of tools to expand the language models' capacity for knowledge recall beyond what it can store internally in its weights. Hence, the final solution is likely to include both parametric and non-parametric methods—progress from both directions will only serve to further complement each other in systems deployed in the real world.

A natural future development would be to add a higher level controller language model that calls upon the low-level language models (including our storage model and retrieval model along with any other models designed for other functions) when needed and thus serves as a dynamic task partitioning mechanism. This could further enhance our system's generalization ability in different interaction scenarios as each scenario would call for the use of a different set of low-level models.

Limitations

Some limitations of our method worth mentioning include SQL schema scalability for large volumes, query cost, and the need to manually adjust prompts for each submodel. As the number of records increase, it could become increasingly difficult for the auxiliary models to efficiently store and retrieve the right information needed to address each query. This could, in turn, be addressed by training the auxiliary models to perform more complex higher-level tasks to better organize the data as well as make use of more complex SQL commands. As our method makes use of an extra storage and retrieval step during inference, it adds additional overhead, which other methods, i.e., parametric methods, could avoid. Although we use smaller models for retrieval and storage to minimize this cost, future research could aim to minimize the overhead even further. And finally, the prompts that we designed for our auxiliary models were handcrafted for the specific base model we used (Gemma 2) and switching to a different family of models would require rewriting the prompts, which could lead to inconsistent results.

5. Conclusions

We explored a non-parametric method for constructing a mechanism for storing and retrieving memories from an external SQL database. The method trains two language models to specialize in two different tasks such that they can work more effectively as a team in terms of carrying out the actions needed to make use of an external SQL database as a memory store. We defined subtasks for each language model to incorporate prior knowledge about inherent weaknesses of the two models, which when combined with the main objective ultimately led to better overall performance. We showed in our experiments that our method can outperform methods that require language models orders of magni-

tude larger in size and demonstrated the potential for language models to build their own memory when given the ability to use external tools and databases to their advantage.

To summarize, our key contributions were as follows:

- We proposed a method for implementing a mechanism for long-term memory that avoids catastrophic forgetting by training language models to store and retrieve information extracted from dialogue into and from an external SQL database.
- We demonstrated that by specializing each member of a team of language models for a particular role, we can achieve greater efficacy than by training a single language model to carry out all functions.
- We further demonstrated that by training each auxiliary language model, framed as an option, on reward-respecting subtasks, we can achieve more effective performance than by training on reward alone.
- We showed empirically that our method outperforms existing baselines that make use of models that are orders of magnitude larger in size (i.e., GPT-3.5).

Future lines of work that could be pursued to further develop the ideas in this paper include extension to hybrid forms of long-term memory that make use of both parametric and non-parametric methods and integration with multi-role agent architectures. As mentioned above, our method can easily take advantage of the rapidly advancing capabilities of language model-based agents and could be included as part of a more encompassing hierarchy of agent roles, for example, as the memory module in a structure that includes various modules to carry out various functions, all being coordinated by a high-level controller language model.

It seems plausible that the overriding principle of subdividing tasks and training a team of specialized language models could lead to improved performance across a wide range of tasks and we expect that this will play an important part in enhancing the capabilities of language models going forward. That this is, in fact, a generalizable technique would require experimentation on additional tasks to establish and we leave this to future work.

Author Contributions: Conceptualization, S.L. and H.K.; Data curation, S.L.; Formal analysis, S.L.; Funding acquisition, H.K.; Investigation, S.L.; Methodology, S.L.; Project administration, H.K.; Resources, H.K.; Software, S.L.; Supervision, H.K.; Validation, S.L.; Visualization, S.L.; Writing—original draft, S.L.; Writing—review and editing, H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The dataset used in this study can be found at https://huggingface.co/datasets/MemGPT/MSC-Self-Instruct (accessed on 10 December 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LLM	Large language model
SQL	Structured query language

References

1. Radford, A.; Narasimhan, K. Improving Language Understanding by Generative Pre-Training. 2018. Available online: https://www.mikecaptain.com/resources/pdf/GPT-1.pdf (accessed on 18 August 2025).
2. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics, Minneapolis, MN, USA, 2–7 June 2019.
3. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language Models Are Unsupervised Multitask Learners. 2019. Available online: https://storage.prod.researchhub.com/uploads/papers/2020/06/01/language-models.pdf (accessed on 20 August 2025).
4. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *arXiv* **2020**, arXiv:2005.14165. [CrossRef]
5. Kaplan, J.; McCandlish, S.; Henighan, T.J.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling Laws for Neural Language Models. *arXiv* **2020**, arXiv:2001.08361. [CrossRef]
6. Achiam, O.J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. GPT-4 Technical Report. *arXiv* **2023**, arXiv:2303.08774. [CrossRef]
7. Lin, S.C.; Hilton, J.; Evans, O. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, Online, 1–6 August 2021.
8. Li, J.; Cheng, X.; Zhao, W.X.; Nie, J.; rong Wen, J. HaluEval: A Large-Scale Hallucination Evaluation Benchmark for Large Language Models. *arXiv* **2023**, arXiv:2305.11747.
9. Min, S.; Krishna, K.; Lyu, X.; Lewis, M.; tau Yih, W.; Koh, P.W.; Iyyer, M.; Zettlemoyer, L.; Hajishirzi, H. FActScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. *arXiv* **2023**, arXiv:2305.14251.
10. Wei, J.; Yang, C.; Song, X.; Lu, Y.; Hu, N.; Huang, J.; Tran, D.; Peng, D.; Liu, R.; Huang, D.; et al. Long-form factuality in large language models. *arXiv* **2024**, arXiv:2403.18802. [CrossRef]
11. Lee, S.; Kim, M.; Cherif, L.; Dobre, D.; Lee, J.; Hwang, S.J.; Kawaguchi, K.; Gidel, G.; Bengio, Y.; Malkin, N.; et al. Learning diverse attacks on large language models for robust red-teaming and safety tuning. *arXiv* **2024**, arXiv:2405.18540. [CrossRef]
12. Hong, Z.W.; Shenfeld, I.; Wang, T.H.; Chuang, Y.S.; Pareja, A.; Glass, J.; Srivastava, A.; Agrawal, P. Curiosity-driven Red-teaming for Large Language Models. *arXiv* **2024**, arXiv:2402.19464.
13. Inan, H.; Upasani, K.; Chi, J.; Rungta, R.; Iyer, K.; Mao, Y.; Tontchev, M.; Hu, Q.; Fuller, B.; Testuggine, D.; et al. Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations. *arXiv* **2023**, arXiv:2312.06674.
14. Liu, G.; Sun, Q.; Su, H.; Wang, M. Adaptive Cooperative Fault-Tolerant Control for Output-Constrained Nonlinear Multi-Agent Systems Under Stochastic FDI Attacks. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2025**, *72*, 6025–6036. [CrossRef]
15. Liu, G.; Sun, Q.; Su, H.; Hu, Z. Adaptive Tracking Control for Uncertain Nonlinear Multi-Agent Systems With Partially Sensor Attack. *IEEE Trans. Autom. Sci. Eng.* **2025**, *22*, 6270–6279. [CrossRef]
16. Liu, N.F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; Liang, P. Lost in the Middle: How Language Models Use Long Contexts. *Trans. Assoc. Comput. Linguist.* **2023**, *12*, 157–173. [CrossRef]
17. Du, Y.; Tian, M.; Ronanki, S.; Rongali, S.; Bodapati, S.; Galstyan, A.; Wells, A.; Schwartz, R.; Huerta, E.A.; Peng, H. Context Length Alone Hurts LLM Performance Despite Perfect Retrieval. *arXiv* **2025**, arXiv:2510.05381. [CrossRef]
18. Li, T.; Zhang, G.; Do, Q.D.; Yue, X.; Chen, W. Long-context LLMs Struggle with Long In-context Learning. *Trans. Mach. Learn. Res.* **2024**,*2025*.
19. Tang, Z.; Zhou, K.; Li, J.; Ji, B.; Hou, J.; Zhang, M. L-CiteEval: Do Long-Context Models Truly Leverage Context for Responding? *arXiv* **2024**, arXiv:2410.02115.
20. McCloskey, M.; Cohen, N.J. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychol. Learn. Motiv.* **1989**, *24*, 109–165.
21. French, R.M. Catastrophic forgetting in connectionist networks. *Trends Cogn. Sci.* **1999**, *3*, 128–135. [CrossRef] [PubMed]
22. Robins, A.V. Catastrophic forgetting in neural networks: The role of rehearsal mechanisms. In Proceedings of the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, Dunedin, New Zealand, 24–26 November 1993; pp. 65–68.
23. Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.C.; Veness, J.; Desjardins, G.; Rusu, A.A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. USA* **2016**, *114*, 3521–3526. [CrossRef] [PubMed]
24. Zhong, W.; Guo, L.; Gao, Q.F.; Ye, H.; Wang, Y. MemoryBank: Enhancing Large Language Models with Long-Term Memory. *arXiv* **2023**, arXiv:2305.10250. [CrossRef]
25. Li, Z.; Hoiem, D. Learning without Forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *40*, 2935–2947. [CrossRef]
26. Ramasesh, V.V.; Lewkowycz, A.; Dyer, E. Effect of scale on catastrophic forgetting in neural networks. In Proceedings of the International Conference on Learning Representations, Virtual, 25–29 April 2022.

27. Sutton, R.S.; Precup, D.; Singh, S. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.* **1999**, *112*, 181–211. [CrossRef]
28. Sutton, R.S.; Machado, M.C.; Holland, G.Z.; Timbers, D.S.F.; Tanner, B.; White, A. Reward-Respecting Subtasks for Model-Based Reinforcement Learning. *Artif. Intell.* **2023**, *324*, 104001. [CrossRef]
29. Mitchell, E.; Lin, C.; Bosselut, A.; Finn, C.; Manning, C.D. Fast Model Editing at Scale. *arXiv* **2021**, arXiv:2110.11309.
30. Meng, K.; Bau, D.; Andonian, A.; Belinkov, Y. Locating and Editing Factual Associations in GPT. In Proceedings of the Conference on Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022.
31. Meng, K.; Sharma, A.S.; Andonian, A.; Belinkov, Y.; Bau, D. Mass-Editing Memory in a Transformer. *arXiv* **2022**, arXiv:2210.07229.
32. Hartvigsen, T.; Sankaranarayanan, S.; Palangi, H.; Kim, Y.; Ghassemi, M. Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors. *arXiv* **2022**, arXiv:2211.11031.
33. Gupta, A.; Rao, A.; Anumanchipalli, G.K. Model Editing at Scale leads to Gradual and Catastrophic Forgetting. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, Bangkok, Thailand, 11–16 August 2024.
34. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Kuttler, H.; Lewis, M.; tau Yih, W.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv* **2020**, arXiv:2005.11401.
35. Wang, W.; Dong, L.; Cheng, H.; Liu, X.; Yan, X.; Gao, J.; Wei, F. Augmenting Language Models with Long-Term Memory. *arXiv* **2023**, arXiv:2306.07174. [CrossRef]
36. Packer, C.; Fang, V.; Patil, S.G.; Lin, K.; Wooders, S.; Gonzalez, J. MemGPT: Towards LLMs as Operating Systems. *arXiv* **2023**, arXiv:2310.08560.
37. Xu, J.; Szlam, A.; Weston, J. Beyond Goldfish Memory: Long-Term Open-Domain Conversation. *arXiv* **2021**, arXiv:2107.07567.
38. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
39. Guo, T.; Chen, X.; Wang, Y.; Chang, R.; Pei, S.; Chawla, N.; Wiest, O.; Zhang, X. Large Language Model based Multi-Agents: A Survey of Progress and Challenges. In Proceedings of the International Joint Conference on Artificial Intelligence, Jeju Island, Republic of Korea, 3–9 August 2024.
40. Wang, Z.; Moriyama, S.; Wang, W.Y.; Gangopadhyay, B.; Takamatsu, S. Talk Structurally, Act Hierarchically: A Collaborative Framework for LLM Multi-Agent Systems. *arXiv* **2025**, arXiv:2502.11098. [CrossRef]
41. Yuan, S.; Song, K.; Chen, J.; Tan, X.; Li, D.; Yang, D. EvoAgent: Towards Automatic Multi-Agent Generation via Evolutionary Algorithms. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics, Mexico City, Mexico, 16–21 June 2024.
42. Yang, Y.; Chai, H.; Shao, S.; Song, Y.; Qi, S.; Rui, R.; Zhang, W. AgentNet: Decentralized Evolutionary Coordination for LLM-based Multi-Agent Systems. *arXiv* **2025**, arXiv:2504.00587.
43. Händler, T. Balancing Autonomy and Alignment: A Multi-Dimensional Taxonomy for Autonomous LLM-powered Multi-Agent Architectures. *arXiv* **2023**, arXiv:2310.03659.
44. Liu, G.; Liang, H.; Wang, R.; Sui, Z.; Sun, Q. Adaptive Event-Triggered Output Feedback Control for Nonlinear Multiagent Systems Using Output Information Only. *IEEE Trans. Syst. Man Cybern. Syst.* **2025**, *55*, 7639–7650. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.