Retrieval-Augmented World Models Enhanced with Reinforcement Learning

Anonymous ACL submission

Abstract

World models achieve remarkable success in predicting future states and planning in complex environments and Large Language Models (LLMs) serve as promising foundation to build general world models. However, their performances are usually constrained by the limited knowledge to the specific environments. Existing research attempts to enhance LLM-based world models through prompting or fine-tuning approaches, which are either requiring human knowledge or computational extensive. Therefore, we introduce **R**etrieval-**A**ugmented **W**orld **M**odels (RAWM), a novel framework that leverages retrieval-augmented generation to improve the LLM-based world models. Our main contributions are threefold: (i) We introduce a memory system and design an embedding model to retrieve and incorporate relevant experiences, significantly improving the world model's predictive accuracy. (ii) We develop a reinforcement learning (RL) training pipeline that fine-tunes a small MLP head on the pre-trained embedding model using Proximal Policy Optimization (PPO), further enhancing prediction performance. (iii) We conduct extensive experiments across three diverse RL environments, i.e., Game24, BlocksWorld, and BabyAI, demonstrating that RAWM consistently outperforms baseline models and exhibits strong generalizability. By leveraging external memory and retrieval techniques and training embedding with RL pipeline, RAWM dynamically utilizes relevant historical experiences and equips LLMs with environment-specific knowledge without retraining, enabling more accurate and generalizable predictions.

1 Introduction

The world model (Ha and Schmidhuber, 2018) has demonstrated to be an important module in decision making due to the celebrating success of MuZero (Schrittwieser et al., 2020) and Dreamer (Hafner et al., 2019, 2021, 2023). As

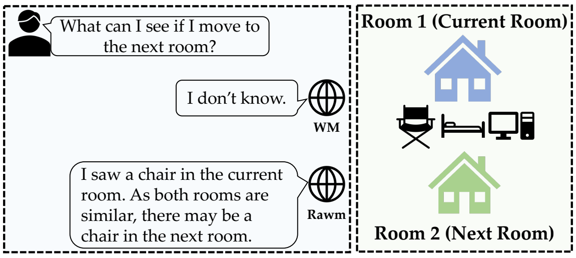


Figure 1: Why retrieval is needed?

learned accurate simulators, world models encode rich representations of the complex dynamics of the environment to predict the future states and the rewards. World models are critical for several key capabilities, such as generalization to novel tasks (Byravan et al., 2020; Robey et al., 2021; Young et al., 2023), efficient planning (Sekar et al., 2020; Hamrick et al., 2021; Schrittwieser et al., 2020), and offline learning (Schrittwieser et al., 2021; Yu et al., 2020, 2021). Beyond decision making, recent works such as Genie (Bruce et al., 2024) and Vista (Gao et al., 2024) demonstrate that world models can serve as general-purpose world simulators and users can directly interact with these models for playing and planning.

The past five years witness the remarkable success of large language models (LLMs) in enormous text generation and understanding tasks (Brown et al., 2020; OpenAI, 2023). LLMs serve as the world model explicitly in Reasoning via Planning (RAP) (Hao et al., 2023) and Reason for Future, Act for Now (RAFA) (Liu et al., 2023), where the LLMs predict the next states based on the actions executed at current states, e.g., the states of blocks in the BlocksWorld (Valmeekam et al., 2023), which is used to assist the planning methods. LLMs serve as the world model implicitly in the widely-used Tree of Thoughts (ToT) (Yao et al., 2023), as well as Graph of Thoughts (GoT) (Besta et al., 2024), where the LLMs need to predict the states and evaluate the thoughts to help the selection of the thoughts to advance the reasoning. The generalizability of LLMs presents them as promising foundations for the world models.

However, the pre-trained LLMs may lack the knowledge of the specific environments, which prohibits them to be accurate world models. For the example displayed in Figure 1, the LLM cannot provide the accurate predictions whether there is a chair in the next room if the next room has never been visited. To address this issue, we can carefully design the prompts to add the specific knowledge to help the LLMs in making predictions, e.g., the rules for objects and actions (Wang et al., 2024; Gu et al., 2024). However, these knowledge is even usually not available for users. Alternatively, we can fine-tune the LLMs on the specific environments (Xiang et al., 2023; Chae et al., 2025). However, the training of LLMs brings additional complexities for building the world models with LLMs and may also hurt the generalizability of LLMs across different tasks.

To tackle these challenges, we propose retrieval-augmented world models (RAWM). Specifically, our contributions are threefold. First, inspired by the retrieval-augmented generation (RAG) (Lewis et al., 2020), we introduce the memory, which stores the pre-collected experiences from the environments, and the embedding model, which is used for querying relevant experience to assist the world model to make predictions. Second, we introduce the reinforcement learning (RL) training pipeline, which adds a small MLP head to the pre-trained embedding model and trains the MLP layer with proximal policy optimization (PPO) (Schulman et al., 2017). Third, we collect the data from Game24, BlocksWorld and BabyAI, and extensive experiments demonstrate RAWM can significantly outperform the world model without retrieved experiences and the pre-trained embedding models and demonstrate the generalizability. RAWM is an efficient way for LLMs to obtain the environment-specific knowledge to build the better world models without training LLMs, and our RL training pipeline can further improve the accuracy of the predictions of LLM-based world models efficiently.

2 Related Work

World Models and LLMs. MuZero (Schrittwieser et al., 2020) and Dreamer (Hafner et al., 2019) are the two prominent examples of the world model for complex decision making tasks. Trajectory transformer (Janner et al., 2021) leverages transformer to model the decision making as a sequence modeling problem. The world models trained in theses methods are environment specific and cannot generalize to other environments. Recently, researchers leverage the generalizability of LLMs to build general world models for reasoning and decision making (Hao et al., 2023; Wang et al., 2024; Yang et al., 2024b; Lin et al., 2024). Specifically, RAP (Hao et al., 2023) and RAFA (Liu et al., 2023) use LLMs to predict the next states explicitly and planning methods for decision making. While ToT (Yao et al., 2023) and GoT (Besta et al., 2024) use LLMs as the world model implicitly to advance and evaluate the different thoughts.

Retrieval-Augmented Generation. RAG is an efficient way for LLMs to incorporate the external knowledge for generation and understanding (Lewis et al., 2020; Gao et al., 2023). Specifically, RAG leverages the retrieval model to query the relevant experiences from the memory, which are further provided to the LLMs as the in-context examples. Different from simple prompting, where the external knowledge is provided by human-written prompts (Wang et al., 2024), and simple in-context learning, where the in-context examples are randomly picked (Hao et al., 2023), RAG can provide better examples for accurate predictions. Compared with fine-tuning (Xiang et al., 2023), RAG can provide a more efficient way of transforming LLMs into world models.

RL for LLM Optimization. RL is a powerful method to train the model with trial and error (Sutton and Barto, 2018). In addition to the applications of RL in games and robotics (Silver et al., 2017) to optimize the LLMs, such as optimizing the prompts (Deng et al., 2022) and the decoding process (Wan et al., 2024), recent works also leverage RL to improve the reasoning capabilities of LLMs (Lambert et al., 2024; Guo et al., 2025). In this work, we leverage the RL method to train the retriever to find the better examples to boost the prediction of the world model.

3 Preliminaries

In this section, we present the preliminaries of RAWM, including the formulation of the decision making, the LLMs, and the world models.

Markov Decision Process (MDP). A decision making problem is usually represented as a Markov decision process (MDP) (Sutton and Barto, 2018), which is defined by the tuple $\mathcal{M} = (S, A, T, R, \gamma)$, where S is the state space, A is the action space, $T : S \times A \rightarrow S$ is the transition dynamics, which specifies the next state s' given the current state s and action a, $R : S \times A \rightarrow \mathbb{R}$ is the reward function, which specifies the agent's reward given the current state s and action a, and γ is the discount factor. The agent's policy is defined by $\pi_\theta : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, parameterized by θ, which takes the state s as the input and outputs the action a to be executed. The objective of the agent is to learn an optimal policy to maximize the expected return, i.e., $\pi^* := \arg\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0\right]$ with initial state s_0.

Large Language Models (LLMs). Large Language models (LLMs) learn from text data using unsupervised learning. LLMs optimize the joint probabilities of variable-length symbol sequences as the product of conditional probabilities by $P(x) = \prod_{i=1}^{n} P(s_i|s_1, ..., s_{i-1})$, where $(s_1, s_2, ..., s_n)$ is the variable-length sequence of symbols. With the billions of parameters and extensive training data, the vast amounts of common knowledge encoded in LLMs lead to the remarkable generalization across various NLP tasks with simple prompting and in-context learning, and without task-specific fine-tuning (Touvron et al., 2023; OpenAI, 2023). Among them, RAG (Lewis et al., 2020) is viewed as a powerful method to incorporate external knowledge to LLMs for generation. Given the generalizability, LLMs present promising foundations for general world models.

World Models. The world model Ω is introduced to predict the dynamics of the environment, thus supporting the decision making process. Specifically, the world model is trained or prompted to predict the next state s', the reward r, and the terminal function d, given the current state s and action a. The world model can be one or multiple neural networks specially trained on the environments for the three prediction tasks (Hafner et al., 2019; Schrittwieser et al., 2020), which cannot generalize across different environments. Recent works leverage LLMs to build the general world models, where the prompting (Xie et al., 2024), in-context learning (Wang et al., 2024), and even fine-tuning methods (Xiang et al., 2023; Lin et al., 2024) are used to build the LLM-based world models. In this work, we primarily focus on the prediction of the next state, which is the most important feature, as both the reward and terminal are usually derived from the next state visited.

4 Retrieval-Augmented World Models

In this section, we introduce **R**etrieval-**A**ugmented **W**orld **M**odels (RAWM). We will first introduce the architecture of RAWM and then introduce the RL training pipeline for the retrieval process.

4.1 Architecture

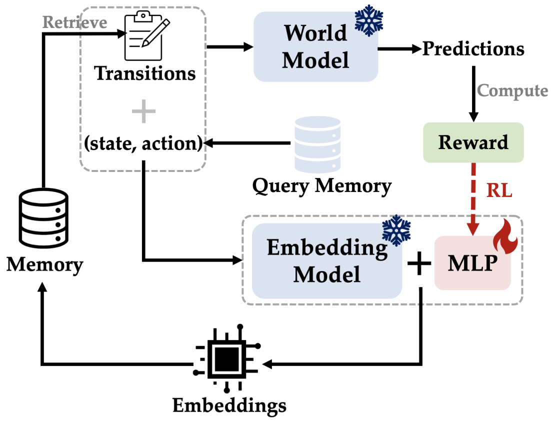


Figure 2: The overview of RAWM.

The architecture of RAWM is displayed in Figure 2. We introduce a memory Ξ, which stores the pre-collected experiences, an embedding model, which is used to rank and retrieve the relevant experiences. Specifically, given the query $q = (s, a) \in \mathcal{Q}$, where $\mathcal{Q}$ is the query dataset, we will use the embedding model to query top K relevant experiences $\boldsymbol{c} = \langle c_k \rangle$, where $c_k = (s_k, a_k, s'_k), k = 1, \ldots, K$. The retrieved experiences $\boldsymbol{c}$ will be concatenated with the query q to form the input to the world model Ω. We note that for the environments where the states are not texts, e.g., BabyAI (Chevalier-Boisvert et al., 2019a), we need to first transform them into the text representation.

Prompt Design. For the prompt design, any information related to the environments will not be provided to the world model, including the tasks, the object and action rules. We expect that all the environment knowledge is provided by the in-context examples retrieved from the memory. The prompt template is displayed as follows:

Prompt Template

System prompt:
"After being given a current state and an action, directly give the next state after performing the action."
Content prompt:
Current state: <text of the current state>
Action: <text of the selected action>
Next state: <text of the next state> or <for prediction>

The system prompt provides a general description of the prediction tasks, and the content prompt includes the query and the context examples. For the context examples, the next state is provided, while for the query, the next state is predicted by the world model Ω. Similarly, this content template is also used to get the embeddings of both query dataset and the memory for the retrieval process.

Trainable Embedding of Transitions. We use the pre-trained embedding model ϕ to encode the transitions into the M-dimensional vector representation. Specifically, for the query dataset, we only encode the state and the action, and for the memory, we encode the state, the action and the next state. However, the embedding model is trained over general corpus, which would be not suitable to the specific environment, so adapting the embedding model is needed. There are several methods to adapt the embedding model to the specific environment: i) fine-tuning all parameters in ϕ, which is not training efficient, ii) low-rank adaption (LoRA) (Hu et al., 2022), which introduces trainable low-rank decomposition matrices for each layer to reduce the parameters to be trained. Though the number of trained parameters is reduced, LoRA still requires to leverage the full embedding model to inference. Besides, both full-parameter fine-tuning and LoRA requires that the access of the parameters of the pre-trained embedding model and cannot be applied to close-source models, e.g., `text-embedding-3`. Therefore, inspired by the linear probe (Radford et al., 2021), we introduce a trainable MLP module above the pre-trained embedding model, which is denoted as ψ. Therefore, the embedding process for both query data and the memory can be represented as:

$$e_q = \psi(\phi(s,a)), \forall (s,a) \in \mathcal{Q}, \tag{1}$$

$$e_c = \psi(\phi(s,a,s'), \forall (s,a,s') \in \Xi. \tag{2}$$

We will introduce the RL training pipeline of ψ in the next section and the parameters in ϕ are frozen. Compared with the full parameter fine-tuning and the LoRA, this method only requires the pre-trained embedding to encode the data in the query dataset and the memory once, and the number of trainable parameters is even significantly less than LoRA.

Retrieval-Augmented Predictions. To query the relevant experiences, a similarity measure, e.g., cosine similarity, is used to rank the examples in the memory, which is denoted as $\mathtt{sim}(\cdot)$. Therefore,

$$\boldsymbol{c} = \{c_k | k \in \mathtt{topK}(\mathtt{sim}(e_q, e_c)), \forall c \in \Xi\}, \tag{3}$$

where $\mathtt{topK}(\cdot)$ is selecting the indices with the top-K maximum values. The K retrieved examples $\boldsymbol{c}$ will be formed the in-context examples and append before the query for the prediction. We concatenate the in-context examples with the query in **a reverse order**, i.e., the examples with larger similarities will be the later examples, and the query is the last one. We found that this reverse order is important for the generalization of the embedding model in different K values, as the reverse order can ensure the last several examples be the same, (e.g., for $K \in \{1,2\}$, the top-1 example is the same, which is the last example before the query in the prompt), thus leading to a more stable generalization performance of the world model.

Evaluation Measure. The evaluation measure is important for the RL training. We follow RAP (Hao et al., 2023) to design the reward: given the output o from the world model, which may include a set of the conditions, e.g., the predicted state of blocks, and s' is the target, we will calculate the accuracy of the prediction, denoted as $v(o,s')$. Alternatively, we can calculate the log likelihood of the target s', which is used in the original RAG (Lewis et al., 2020). However, this may require the access of the logits of the LLMs and cannot be applied to the closed-source models, e.g., GPT-4o.

4.2 Training

In this section, we introduce the efficient RL pipeline to train the embedding models, i.e., training of the MLP head ψ specifically. Typically, the retriever in RAG is trained with supervised learning (Lewis et al., 2020). However, in RAWM, the world models are not trained and we cannot compute the gradient of the embedding directly. Besides, as the retriever needs to explore to choose the examples for the better prediction with the world model, RL is one of the straightforward methods to optimize the embedding model.

One-step Decision Making. To apply RL methods to optimize the embedding model, we need to build the MDP $\mathcal{M}^\psi$ for the embedding ψ[1]:

- State space $S^\psi : \{\phi(s,a), \forall (s,a) \in \mathcal{Q}\} \cup \{\phi(s,a,s'), \forall (s,a,s') \in \Xi\}$, i.e., the embeddings of all data from query dataset and the memory generated by the pre-trained model ϕ.
- Action space $\mathcal{A}^\psi \in \mathbb{R}^M$, where M' is the output dimension of ψ, i.e., ψ will transform the

[1]Please distinguish $\mathcal{M}^\psi$ with the one used for the environment $\mathcal{M}$, where $\mathcal{M}^\psi$ is introduced only for the training.

embeddings by ϕ to M'-dimensional vectors.

- Reward $r = v(\Omega(q, \boldsymbol{c}), s')$, where $\Omega(q, \boldsymbol{c})$ is the output of the world model Ω with the input $(q, \boldsymbol{c})$.

We note that $\mathcal{M}^{\psi}$ is a one-step decision making problem, i.e., $\mathcal{M}^{\psi}$ always ends after the first time step, so the transition function and the discount factor are not necessary for the RL training.

Design of ψ. Before diving into the RL training, we first discuss about the design of ψ. A simple setting for ψ is a randomly initialized MLP, which means this initialization will start with the random embedding for the training and ignore the embeddings generated by the pre-trained model ϕ. On the other hand, we can initialize the MLP with an identify matrix, i.e., $\psi = \boldsymbol{I}$.[2] Both methods have their own advantages and disadvantages: for the random initialization, we can arbitrarily choose the output dimension and the activation function of ψ, but the training will start with a relatively worse performance, while for the identify initialization, the output dimension of ψ must be the same with ϕ, i.e., $M' = M$, and the training will start with the performance of the pre-trained embedding model.

RL Training. RL methods rely on the trail-and-error process to explore the solution space for better policies. The primary RL method is Q-learning (Watkins and Dayan, 1992; Mnih et al., 2015), which can only be used on the problems with discrete actions, and the policy gradient methods are proposed for the problems with both discrete and continuous actions (Sutton et al., 1999; Mnih et al., 2016; Haarnoja et al., 2018). PPO (Schulman et al., 2017) is an on-policy policy gradient method, which is a simplified, but more data efficient and reliable, variant of Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), which leverages the "trust region" to bound the update of the policy to avoid training collapse. Compared with TRPO, PPO is more data efficient and with more reliable performances than TRPO, while only using the first-order optimization for computational efficiency. Specifically, PPO is maximizing the objective

$$J(\psi) = \mathbb{E}\left[\min\left(\rho_{\psi} \cdot r, \; \mathtt{clip}(\rho_{\psi}, 1-\epsilon, 1+\epsilon) \cdot r\right)\right], \quad (4)$$

where ρ_{ψ} is the importance sampling ratio conditional on ψ, r is the reward, and ϵ is the hyperparameter which controls the boundary of the trust region. We note that the advantages in the general PPO implementation is replaced with the reward. We only provide a short introduction of PPO in this section, as we take PPO as a blackbox for optimizing ψ. The full training procedure is displayed in Algorithm 1. Other RL methods, e.g., soft actor critic (SAC) (Haarnoja et al., 2018), can also be used and for more details of RL, we refer readers to the book (Sutton and Barto, 2018).

Algorithm 1 Training of RAWM

```
Input: World model Ω, pre-trained embedding model φ, memory M, Query dataset Q, number of retrieval candidates K
Initialize the MLP ψ.
Computing the embeddings with φ, i.e., Q_φ = {φ(s,a), ∀(s,a) ∈ Q} and M_φ = {φ(s,a,s'), ∀(s,a,s') ∈ M}.
for iter ∈ {1, 2, ...} do
    Update the memory embedding M_ψ = {ψ(φ(s,a,s')), ∀(s,a,s') ∈ M}.
    for (s,a) in Q do
        Compute query embedding ψ(φ(s,a)).
        Select top-K relevant transitions c from M with the embedding in M_ψ.
        Generate the prediction o and compute the reward v(o,s').
    end for
    Train ψ with PPO, i.e., Eq. (4).
end for
```

5 Experiments

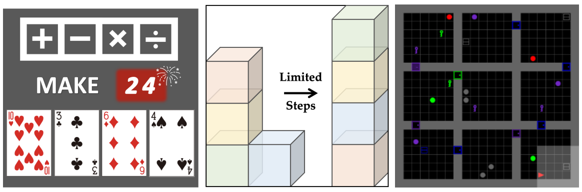
In this section, we present the extensive experiments to evaluate the effectiveness of RAWM. We will first introduce the experiment setup and then the experiment results and analysis.

5.1 Setup

Environments. The environments considered in this work include (as shown in Figure 3)

- **Game24**: a mathematical puzzle game where four numbers are given (e.g., 10, 3, 6, and 4) and the player can only use the basic arithmetic operations, i.e., $(+, -, \times, \div)$, to obtain 24 (e.g., $10 \times (6 \div 3) + 4$). This puzzle is widely used to benchmark the LLMs' reasoning capabilities (Yao et al., 2023) and the LLMs need to generate a sequence of operations to obtain 24. In this game, the world model needs to correctly generate the remaining number when an opera-

[2] With a slight abuse of notations, we use ψ to represent both the MLP and the parameters.



(a) Game24 (b) BlocksWorld (c) BabyAI

Figure 3: Environments

tion is executed, i.e., $10, 2, 4$ are the remaining numbers when $6 \div 3$ is executed.

- **BlocksWorld**: a simple world of blocks where a set of blocks is placed on the plat and the player needs to perform the basic actions, i.e., `pick up`, `put down`, `stack`, and `unstack`, to transform the blocks to a target configuration (Valmeekam et al., 2023; Hao et al., 2023). In this game, the world model needs to predict the states for all blocks (e.g., the blue block is on top of the re block) after an action is executed (e.g., stack blue block on the red block).
- **BabyAI**: a grid world with objects where the agent needs to complete the tasks defined with language instructions (Chevalier-Boisvert et al., 2019a) with the actions, i.e., `turn left`, `turn right`, `move forward` and `pick up`. We use the text description of the states in (Carta et al., 2023) for the environments. In this environment, the world model needs to predict the locations of the objects after performing the action.

Datasets. Given the environments, we need to collect the datasets for the memory, query and test datasets, respectively. We use the query dataset to train the embedding with RL and use the test dataset to validate the performance of the trained models. For Game24 and BlocksWorld, the number of all possible transitions are less than 10K, therefore, we use the Depth-First Search (DFS) to enumerate all transitions to form the full datasets. While for BabyAI, we cannot enumerate all transitions due to the complexity of the environments. Therefore, we utilize the bot provided in (Chevalier-Boisvert et al., 2019b) to collect the data, where we enumerate all valid actions to gather the transitions along the action sequences generated by the bot. After the collection, we choose the separate subsets to form the three datasets without any overlapping to avoid any data leakage. We provide the detailed introduction of the environments and the protocol for data collection in Appendix C.

Model Selection. We use the embedding model `Alibaba-NLP/gte-Qwen2-1.5B-instruct` as the pre-trained ϕ, which is the leading open-source text embedding model on MTEB (Li et al., 2023). For the world model, we choose the Qwen-2.5 instruct model series with the model sizes as $\{1.5\text{B}, 3\text{B}, 7\text{B}\}$ (Yang et al., 2024a)[3]. The AWQ quantized models are chosen for efficient inference. For the configuration of ψ, we consider a three layer MLP with `Tanh()` activation function for the random initialization and a single layer without any activation function for the identity initialization.[4] We provide the details about the model selection and the initialization in Appendix E.

RL Training. For the efficiency, we consider several implementation tricks. i) Compared with the training of the MLP ψ, the inference of the world model is much more time-consuming. Therefore, we enlarge the number of batch sizes and for each batch, we sample multiple times, which can stabilize the training. ii) We also consider fixing the embeddings in the memory, i.e., only the embeddings of the query datasets are trained, and do not observe the advantages. Therefore, we update the embedding of both datasets. iii) The output dimension of the random initialization is much smaller than the output dimension of the identity initialization, which enjoys the training stabilities with larger learning rates and smaller memory usages when retrieval. The hyperparameters for the RL training of ψ is provided in Appendix F.

Methods Evaluated. The methods evaluated in the experiments are: i) zero-shot: the world models give the prediction without any in-context examples (Wang et al., 2024), ii) random: the world models give the prediction with randomly selected in-context examples from $\mathcal{M}$ (Hao et al., 2023), iii) RAWM$_{\psi,\text{rand}}$: RAWM with the randomly initialization of ψ, which differs from the previous method, iv) RAWM$_{\psi,\text{eye}}$: RAWM with the identity initialization of ψ, equivalent to the pre-trained embedding model ϕ, v) RAWM$^{\text{RL}}_{\psi,\text{rand}}$: RAWM with randomly initialized ψ and RL training, and vi) RAWM$^{\text{RL}}_{\psi,\text{eye}}$: RAWM with identity initialized ψ and RL training.

5.2 Evaluation

We present the extensive evaluations of RAWM in this section. There are three main research questions (RQs) investigated:

[3] https://huggingface.co/spaces/Qwen/Qwen2.5

[4] We would note that RAWM can work for both close-source and open-source embedding and world models. We choose open-source models for efficient training and inference.

Model	Method	Game24				BlocksWorld				BabyAI			
		$K=1$		$K=2$		$K=1$		$K=2$		$K=1$		$K=2$	
		train	test	train	test	train	test	train	test	train	test	train	test
1.5B	zero-shot	0.5224	0.5455	0.5224	0.5455	0.3804	0.3849	0.3804	0.3849	0.3786	0.3772	0.3786	0.3772
	random	0.5586	0.5664	0.5714	0.5959	0.4848	0.4822	0.4975	0.4991	0.3851	0.3856	0.3973	0.4030
	RAWM$_{\psi,\text{rand}}$	0.5156	0.5219	0.5322	0.5534	0.5386	0.5402	0.5597	0.5589	0.3415	0.3479	0.3527	0.3484
	RAWM$_{\psi,\text{eye}}$	0.5352	0.5474	0.5510	0.5600	0.5659	0.5697	0.5878	0.5888	0.4427	0.4446	0.4710	0.4671
3B	zero-shot	0.4888	0.4971	0.4888	0.4971	0.3644	0.3661	0.3644	0.3661	0.3303	0.3330	0.3303	0.3330
	random	0.6703	0.6719	0.6984	0.7010	0.4717	0.4706	0.5089	0.5083	0.3912	0.3908	0.4073	0.4052
	RAWM$_{\psi,\text{rand}}$	0.7041	0.7043	0.7269	0.7292	0.5729	0.5739	0.6005	0.6019	0.3855	0.3892	0.3985	0.3991
	RAWM$_{\psi,\text{eye}}$	0.7022	0.7179	0.7313	0.7463	0.6127	0.6102	0.6440	0.6397	0.4355	0.4297	0.4646	0.4633
7B	zero-shot	0.5957	0.6121	0.5957	0.6121	0.5215	0.5207	0.5215	0.5207	0.4201	0.4254	0.4201	0.4254
	random	0.8241	0.8267	0.8712	0.8667	0.5897	0.5838	0.6021	0.6072	0.4084	0.4181	0.4178	0.4221
	RAWM$_{\psi,\text{rand}}$	0.8362	0.8375	0.8724	0.8703	0.6274	0.6240	0.6332	0.6314	0.4301	0.4322	0.4403	0.4355
	RAWM$_{\psi,\text{eye}}$	0.8511	0.8527	0.8781	0.8734	0.6472	0.6452	0.6556	0.6541	0.4484	0.4501	0.4633	0.4693

Table 1: Performance of RAWM with the retrieval mechanism over three environments.

- **RQ1**: Can the retrieved methods in RAWM improve the performance of world model?
- **RQ2**: Can the RL training pipeline in RAWM improve the performance of the world model, compared with pre-trained models?
- **RQ3**: Can the learned model generalize across different settings, e.g., different values of K?

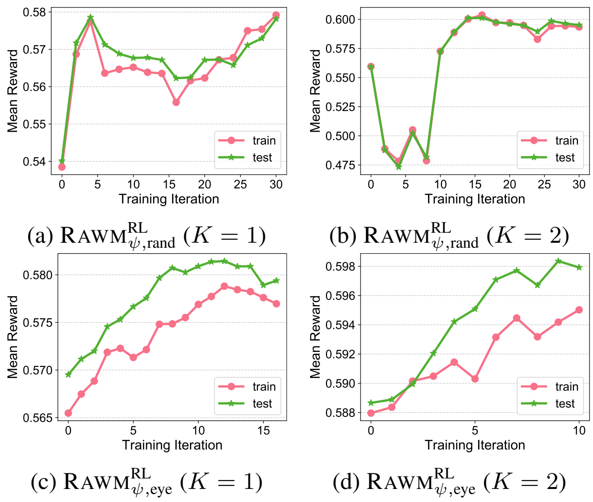
5.2.1 Analysis of RQ1

To investigate the RQ1, we conduct the experiments of RAWM on the different sizes of the world model, i.e., 1.5B, 3B and 7B, over the three environments. We consider the values of K as $\{1, 2\}$. The experiment results are displayed in Table 1. From the results, we observe that the performances over the train and test yield the same trend, which avoids the over-fitting to the specific dataset.

With more in-context examples selected, the performance of the world model is significantly improved, which is consistent with other research (Agarwal et al., 2024). Another interesting observation is that increasing the model sizes of LLMs does not necessarily improve the performance of the world models. For example, the 3B world model performs worse than the 1.5B world model in BabyAI. This is because the LLMs do not have the specific knowledge of the environments and increasing the model size cannot solve this.

We also observe that given the same number of the in-context examples, the pre-trained model (i.e., RAWM$_{\psi,\text{eye}}$) can retrieve more relevant examples for the world models across different sizes in BlocksWorld and BabyAI. While for the 1.5B world model of Game24, the pre-trained models perform worse than the random examples. Therefore, optimizing for a better embedding model can potentially further improve the performance.

5.2.2 Analysis of RQ2

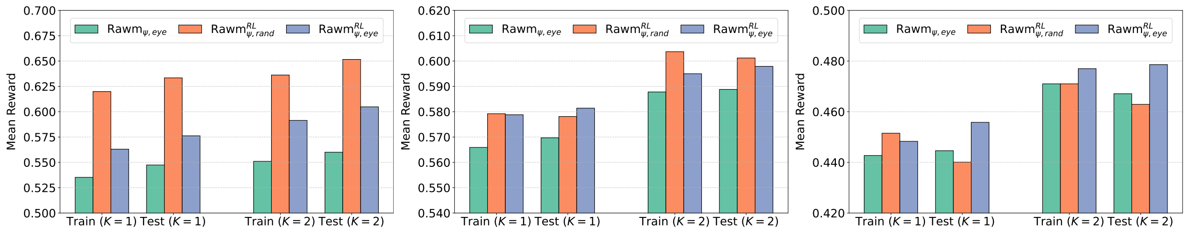


(a) RAWM$^{\text{RL}}_{\psi,\text{rand}}$ ($K=1$)
(b) RAWM$^{\text{RL}}_{\psi,\text{rand}}$ ($K=2$)
(c) RAWM$^{\text{RL}}_{\psi,\text{eye}}$ ($K=1$)
(d) RAWM$^{\text{RL}}_{\psi,\text{eye}}$ ($K=2$)

Figure 4: Training curves on BlocksWorld.

We then present the results of the RL training pipeline of RAWM. Due to the limitation of the resource, we only conduct the training on the world models with 1.5B LLMs. The results of different configurations of ψ across different environments are displayed in Figure 5.

From the results, we observe that the RL training can improve upon the initialization, which indicates the capability of RL to optimize the embedding model through exploration. We observe that both initialization can outperform the pre-trained embedding model, i.e., RAWM$_{\psi,\text{eye}}$, in Game24 and BlocksWorld, while the random initialization fails to find a better embedding than the pre-trained one in BabyAI. The training curves are displayed in



(a) Game24 (b) BlocksWorld (c) BabyAI

Figure 5: Performance of the RL training pipeline in RAWM over three environments.

Method	Game24				BlocksWorld				BabyAI			
	$K=3$		$K=5$		$K=3$		$K=5$		$K=3$		$K=5$	
	train	test	train	test	train	test	train	test	train	test	train	test
random	0.5745	0.5862	0.5669	0.5866	0.5096	0.5125	0.5261	0.5228	0.4044	0.4071	0.4220	0.4165
RAWM$_{\psi,\text{rand}}$	0.5431	0.5443	0.5538	0.5635	0.5702	0.5711	0.5730	0.5738	0.3522	0.3551	0.3696	0.3636
RAWM$_{\psi,\text{eye}}$	0.5533	0.5528	0.5660	0.5765	0.6016	0.5994	0.6178	0.6149	0.4838	0.4753	0.4816	0.4860
RAWM$^{\text{RL}}_{\psi,\text{rand}}$ ($K=1$)	0.5766	0.5974	0.6002	0.6106	0.6001	0.6022	0.6199	0.6200	0.4716	0.4624	0.4745	0.4702
RAWM$^{\text{RL}}_{\psi,\text{eye}}$ ($K=1$)	0.5893	0.5950	0.5976	0.6053	0.6038	0.6042	0.6222	0.6220	0.4878	0.4877	0.4982	0.4872
RAWM$^{\text{RL}}_{\psi,\text{rand}}$ ($K=2$)	0.6097	0.6344	0.5901	0.5999	0.6100	0.6129	0.6198	0.6202	0.4732	0.4711	0.4738	0.4741
RAWM$^{\text{RL}}_{\psi,\text{eye}}$ ($K=2$)	0.5912	0.6020	0.5981	0.6067	0.6049	0.6052	0.6215	0.6205	0.4864	0.4852	0.4976	0.4888

Table 2: Shot generalization of the 1.5B world model trained with RL.

Figure 4. Typically, the random initialization will let the model train from a relative low performance and we observe a drop of the performance due to the exploration for better embedding model (i.e., Figure 4b). And for the identity initialization, the training is more stable with smaller learning rates (i.e., Figures 4c and 4d). These results indicate the effectiveness of our RL training pipeline.[5]

Our RL training pipeline can also be used to diagnose the failure of the retrieval-augmented generation systems. If the RL pipeline cannot find a better embedding to improve the world model's performance, then the user would replace the LLMs for the world models and the datasets.[6]

5.2.3 Analysis of RQ3

The results of shot generation are displayed in Table 2, where the embedding models trained with random and identity initializations of $K \in \{1, 2\}$ are evaluated over the $K \in \{3, 5\}$, i.e., the generalization over shots. From the results, we observe that with larger values of K, the performance of the world model will be further improved. The embedding models trained with RL pipeline demonstrate to be more capable for the generalization over shots, compared with the pre-trained embedding model.

Method	Game24		BlocksWorld		BabyAI	
	Train	Test	Train	Test	Train	Test
RAWM$_{\psi,\text{rand}}$	0.8724	0.8703	0.6332	0.6314	0.4403	0.4355
RAWM$_{\psi,\text{eye}}$	0.8781	0.8734	0.6556	0.6541	0.4633	0.4693
RAWM$^{\text{RL}}_{\psi,\text{rand}}$	0.8799	0.8829	0.6631	0.6630	0.4518	0.4484
RAWM$^{\text{RL}}_{\psi,\text{eye}}$	0.8852	0.8812	0.6597	0.6560	0.4700	0.4721

Table 3: Model generalization of 1.5B →7B ($K = 2$).

We also consider the generalization over different LLMs, which is more difficult than the shot generalization. Table 3 displays the results of generalizing the RL trained embedding model from 1.5B to 7B. We observe that the RL trained embedding admits better generalization performance than the pre-trained embedding model.

6 Conclusions

LLMs present promising foundations to build general world models. However, LLMs usually lack the specific knowledge of environments. Therefore, we introduce **R**etrieval-**A**ugmented **W**orld **M**odels (RAWM), which leverages the retrieval-augmented generation to improve LLM-based world models. We then introduce an efficient RL training pipeline to further improve the performance. Extensive experiments demonstrate the effectiveness and the generalizability of RAWM. RAWM is an efficient method to build the highly capable LLM-based world models without fine-tuning the LLMs.

[5]We note that the improvement that RL training can bring will largely be influenced by the LLMs' capabilities.

[6]Different from the factual QA (Gao et al., 2023) where we can manually check whether the retrieved examples are correct or not, RAWM relies on the LLM's inherit understanding capabilities for the prediction and human cannot manually check the correctness of the retrieval. Therefore, a systematic method, e.g., RL, is needed for diagnosing the system.

Limitations

There are several limitations of current work.

- Current RAWM focuses on prediction and the prediction can be used for decision making. We will extend current RAWM to support the better decision making in future work.
- Current RAWM is based on the pre-collected data, which may require a large number of data to achieve good performance. We will consider to let the model to proactively collect the data and improve the performance automatically.
- Current RAWM is based on LLM and the environments are represented by texts. RAWM can also handle the multi-modal environments, e.g., text and image, where both embedding models and world models will be multi-modal models. We will explore this direction in future work.

We expect that RAWM can be a general framework to build highly capable multi-modal world model with automatically data collection and training, to finally support decision making in complex tasks.

Ethics Statement

We confirm that we have fully complied with the ACL Ethics Policy in this study. All the environments are publicly available and have been extensively used in the research.

References

Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. 2024. Many-shot in-context learning. *arXiv preprint arXiv:2404.11018*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*, pages 17682–17690.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *NeurIPS*, pages 1877–1901.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. 2024. Genie: Generative interactive environments. In *ICML*.

Arunkumar Byravan, Jost Tobias Springenberg, Abbas Abdolmaleki, Roland Hafner, Michael Neunert, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. 2020. Imagined value gradients: Model-based policy optimization with tranferable latent dynamics models. In *CoRL*, pages 566–589.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. 2023. Grounding large language models in interactive environments with online reinforcement learning. *arXiv preprint arXiv:2302.02662*.

Hyungjoo Chae, Namyoung Kim, Kai Tzu iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2025. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *ICLR*.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019a. BabyAI: First steps towards grounded language learning with a human in the loop. In *ICLR*.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019b. BabyAI: First steps towards grounded language learning with a human in the loop. In *ICLR*.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *EMNLP*.

Shenyuan Gao, Jiazhi Yang, Li Chen, Kashyap Chitta, Yihang Qiu, Andreas Geiger, Jun Zhang, and Hongyang Li. 2024. Vista: A generalizable driving world model with high fidelity and versatile controllability. *arXiv preprint arXiv:2405.17398*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. 2024. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122*.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, pages 1861–1870.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2019. Dream to control: Learning behaviors by latent imagination. In *ICLR*.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2021. Mastering Atari with discrete world models. In *ICLR*.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2023. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

Jessica B Hamrick, Abram L. Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Holger Buesing, Petar Veličković, and Theophane Weber. 2021. On the role of planning in model-based deep reinforcement learning. In *ICLR*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *ICLR*.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*.

Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. 2024. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS*, pages 9459–9474.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Jessy Lin, Yuqing Du, Olivia Watkins, Danijar Hafner, Pieter Abbeel, Dan Klein, and Anca Dragan. 2024. Learning to model the world with language. In *ICML*.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. 2023. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *arXiv preprint arXiv:2309.17382*.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763.

Alexander Robey, George J Pappas, and Hamed Hassani. 2021. Model-based domain generalization. In *NeurIPS*, pages 20210–20229.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. 2021. Online and offline reinforcement learning by planning with a learned model. In *NeurIPS*, pages 27580–27591.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *ICML*, pages 1889–1897.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. 2020. Planning to explore via self-supervised world models. In *ICML*, pages 8583–8592.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction*. MIT press.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, pages 1057–1063.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *NeurIPS*, pages 38975–38987.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. Alphazero-like tree-search can guide large language model decoding and training. In *ICML*.

Ruoyao Wang, Graham Todd, Ziang Xiao, Xingdi Yuan, Marc-Alexandre Côté, Peter Clark, and Peter Jansen. 2024. Can language models serve as text-based world simulators? *arXiv preprint arXiv:2406.06485*.

Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning*, 8(3):279–292.

Jiannan Xiang, Tianhua Tao, Yi Gu, Tianmin Shu, Zirui Wang, Zichao Yang, and Zhiting Hu. 2023. Language models meet world models: Embodied experiences enhance language models. *arXiv preprint arXiv:2305.10626*.

Kaige Xie, Ian Yang, John Gunerli, and Mark Riedl. 2024. Making large language models into world models with precondition and effect knowledge. *arXiv preprint arXiv:2409.12278*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024a. Qwen 2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Chang Yang, Xinrun Wang, Junzhe Jiang, Qinggang Zhang, and Xiao Huang. 2024b. Evaluating world models with llm for decision making. *arXiv preprint arXiv:2411.08794*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: deliberate problem solving with large language models. In *NeurIPS*, pages 11809–11822.

Kenny Young, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber. 2023. The benefits of model-based generalization in reinforcement learning. In *ICML*, pages 40254–40276.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. 2021. COMBO: conservative offline model-based policy optimization. In *NeurIPS*, pages 28954–28967.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. 2020. MOPO: Model-based offline policy optimization. In *NeurIPS*, pages 14129–14142.

A Frequently Asked Questions (FAQs)

A.1 Advantages of RAWM

There are several advantages of RAWM, compared with other methods for LLM-based world models:

- RAWM does not require the fine-tuning of LLMs, where the fine-tuning of LLMs is usually time and computation extensive. Besides, the fine-tuning may also hurt the capabilities of LLMs on other tasks. RAWM can be viewed as a plug-and-play framework to transform the LLMs into world models.
- RAWM does not require the manually design of the prompts, i.e., instructions and in-context examples, for LLMs, which is usually labor intensive to optimize the prompts. RAWM automatically retrieve the in-context examples from memory to assist the world models for predictions.
- RAWM introduces the efficient RL training to further improve the world models with retrieval-augmented generation. We note that with the RL training pipeline, RAWM can find the capability limit of the memory and the world model, thus can be used to diagnose the systems.

A.2 Why Focusing on Next State Prediction?

Next state prediction is the most important feature for the world model (Wang et al., 2024). The reward and the terminal can usually derived from the next state. For example, for Game24 and BlocksWorld, we can derive the reward to check whether the remaining number is 24 and whether the next state is the same as the goal state, respectively. Therefore, we focus on next state prediction.

A.3 Why Not Larger LLMs?

We note that `Qwen/Qwen2.5-1.5B-Instruct` is a highly capable LLM, which achieves 60.9% accuracy on the MMLU benchmark. Therefore, we choose this small LLM as the base model for the RL training for the efficiency.

We also consider the models with sizes 3B and 7B for inference, which achieve 65.6% and 72.4% accuracy on MMLU benchmark, respectively.

A.4 What If RL Training Cannot Improve?

RL training is a powerful framework. However, due to the trail-and-error process, RL training may be more complicated than the supervised learning. Here we provide some guidance for the training:

- Smaller learning rate with the identity initialization would be safer for the better performance than pre-trained models. While random initialization can potentially find better embedding models with longer training.

- We would also note that the improvement of RL training may also depend on the data in the memory and the LLMs for the world model. Therefore, if no good hyperparameters for the improvement, please consider larger LLMs and larger memory.

A.5 Code and Dataset Availability

We will release all the code and datasets upon the paper acceptance. The anonymous code can be access at: `https://anonymous.4open.science/r/rawm-acl`.

B Related Work

World Models in Decision Making. World models are actively explored by researchers to further improve the agent's performance and the sample efficiency (Ha and Schmidhuber, 2018; Janner et al., 2019; Hafner et al., 2019; Schrittwieser et al., 2020). Dreamer (Hafner et al., 2019) is a practical model-based reinforcement learning algorithm that introduces the belief over states as a part of the input to the model-free DRL algorithm used. Trajectory Transformer (Janner et al., 2021) trains the transformer to predict the next state and action as a sequence modeling problem for continuous robot control. MuZero (Schrittwieser et al., 2020) is a remarkable success of model-based RL, which learns the world model and conducts the planning in the latent space. The world model with LLM in (Xiang et al., 2023) is trained to gain the environment knowledge, while maintaining other capabilities of the LLMs. Dynalang (Lin et al., 2024) proposes the multi-modal world model, which unifies videos and texts for the future prediction in decision making.

LLMs as World Simulators. World simulators are developed to model the dynamics of the world (Bruce et al., 2024). LLMs serve as the world simulators due to their generalizability across tasks. Specifically, The LLMs (i.e., GPT-3.5 and GPT-4) are evaluated to predict the state transitions, the game progress and scores with the given object, action, and score rules, where these rules are demonstrated to be crucial to the world model predictions (Wang et al., 2024). The world models with LLMs in (Xie et al., 2024) need to additionally identify the valid actions.

World Models in LLMs. The concept of world model also be explored in the deliberation reasoning of LLMs. Specifically, Reasoning via Planning (RAP) (Hao et al., 2023) leverages the planning methods (e.g., Monte Carlo Tree Search (MCTS)) with the world model with LLMs for plan generation and math reasoning, where LLMs need to predict the next state and the reward to guide the search. Tree of Thought (ToT) (Yao et al., 2023) implicitly leverages the LLMs as the world model to predict the next state and the reward for the search over different thoughts. Reason for future, act for now (RAFA) (Liu et al., 2023) combine the planning and reflection with the world model for complex reasoning tasks.

C Environments and Data Collection

C.1 Game24

Game24 is an interesting puzzle game, where four integer numbers in $\{1, 2, 3, \ldots, 13\}$ are given, the player needs to use the basic arithmetic operators, i.e., $+$, $-$, $\times$ and $\div$, and use each number exactly at once to form 24. This puzzle game is used in (Yao et al., 2023) and (Liu et al., 2023) to benchmark the LLM's reasoning capabilities.

Figure 6: Game24

The instances of Game24 used in this work can be accessed at `https://github.com/princeton-nlp/tree-of-thought-llm/blob/master/src/tot/data/24/24.csv`. The state of Game24 is the remaining numbers and the action is applying the operator between two remaining numbers. Here is an example of the transition:

```
{
    "state": (1.0, 1.0, 5.0, 8.0),
    "action": "1.0 + 1.0",
    "next_state": (2.0, 5.0, 8.0),
    "reward": False,
}
```

We provide the python-style code to transform the transitions to natural language examples in Algorithm 2.

Algorithm 2 Transitions to in-context examples for Game24

```python
# transition is the dict with "state", "
    action", "next_state" and "reward"
def transition2example_game24(
    transition, is_query=False,
    is_next_state_prediction=True
):
    example = ""

    example += "current state: {}\n".
    format(transition["state"])
    example += "action: {}\n".format(
    transition["action"])

    if not is_query:
        if is_next_state_prediction:
            example += "next state: {}\n
    ".format(transition["next_state"])
        else:
            example += "reward: {}\n".
    format(transition["reward"])

    return example
```

C.2 BlocksWorld

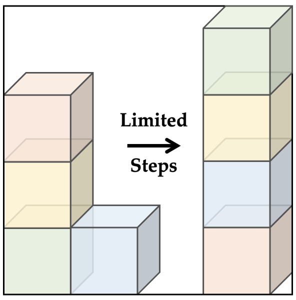


Figure 7: BlocksWorld

BlocksWorld is a widely used benchmark to evaluate the planning capabilities of LLMs (Valmeekam et al., 2023; Hao et al., 2023). All the instances of the BlocksWorld can be accessed at `https://github.com/karthikv792/LLMs-Planning/tree/main/plan-bench/instances/blocksworld`. We build the environment by transforming the instances to MDPs, which can provide the transitions. Here is an example of the transition:

```
{
    "state": "the red block is clear,
    the hand is empty, the orange block
    is on top of the yellow block, the
    red block is on top of the orange
    block, the yellow block is on top of
     the blue block, and the blue block
    is on the table.",
    "action": "unstack the red block
    from on top of the orange block",
    "next_state": "the orange block is
    clear, the red block is in the hand,
     the hand is holding the red block,
    the orange block is on top of the
    yellow block, the yellow block is on
     top of the blue block, and the blue
     block is on the table.",
    "reward": False,
    "info": {
        "goal": "the red block is on top
     of the blue block, the blue block
    is on top of the yellow block and
    the yellow block is on top of the
    orange block"
    },
}
```

We provide the python-style code to transform the transitions to natural language examples in Algorithm 3.

Algorithm 3 Transitions to in-context examples for BlocksWorld

```python
# transition is the dict with "state", "
    action", "next_state" and "reward"
def transition2example_bw(transition,
    is_query=False,
    is_next_state_prediction=True):
    example = ""

    example += "goal state: {}\n".format
    (transition["info"]["goal"])
    example += "current state: {}\n".
    format(transition["state"])
    example += "action: {}\n".format(
    transition["action"])

    if not is_query:
        if is_next_state_prediction:
            example += "next state: {}\n
    ".format(transition["next_state"])
        else:
            example += "reward: {}\n".
    format(transition["reward"])

    return example
```

C.3 BabyAI

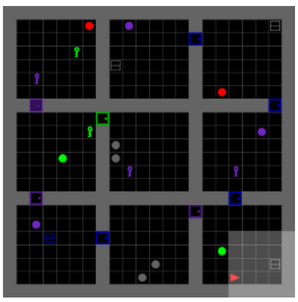


Figure 8: BabyAI

```
{
```

```
    "mission": "go to a red box after
you pick up the purple key",
    "state": [
        "You carry a purple key",
        "You see a wall 2 steps left",
        "You see a blue ball 2 steps
forward",
        "You see a yellow ball 1 step
right and 1 step forward",
        "You see a purple ball 2 steps
right and 2 steps forward",
        "You see a red box 2 steps right
 and 1 step forward",
    ],
    "action": "turn right",
    "reward": 0,
    "done": False,
    "next_state": [
        "You carry a purple key",
        "You see a purple ball 2 steps
left and 2 steps forward",
        "You see a blue ball 2 steps
left",
        "You see a red box 1 step left
and 2 steps forward",
        "You see a yellow ball 1 step
left and 1 step forward",
        "You see a green key 4 steps
forward",
        "You see a green key 1 step
right",
        "You see a red box 2 steps right
 and 1 step forward",
        "You see a yellow key 3 steps
right and 3 steps forward",
        "You see a red ball 3 steps
right",
    ],
}
```

C.4 Statistics of Datasets

Table 4 provides the statistics of the datasets used for the RL training and testing.

	Memory	Query	Test
Game24	2882	2882	5764
BlocksWorld	2416	2416	4833
BabyAI	3124	1562	3124

Table 4: Statistics of the datasets

D Prompts

Design of Prompts. To make the world model as general as possible, we do not specifically design the prompts. The system prompt of the world model is "`After being given a current state and an action, directly give the next state after performing the action.`" We do not provide the description of the task, such as "`I am playing with a set of blocks where I need to arrange the blocks into stacks.`", which is game specific and it needs human to write the specific prompts.

Algorithm 4 Transitions to in-context examples for BabyAI

```
# transition is the dict with "state", "
    action", "next_state" and "reward"
def transition2example_babyai(
    transition, is_query=False,
    is_next_state_prediction=True
):
    def state_to_string(state):
        state_string = ""
        for idx, sta in enumerate(state)
    :
            state_string += sta
            if idx == len(state) - 1:
                continue
            else:
                state_string += ", "
        return state_string

    example = ""

    example += "mission: {}\n".format(
    transition["mission"])

    example += "current state: {}\n".
    format(state_to_string(transition["
    state"]))
    example += "action: {}\n".format(
    transition["action"])

    if not is_query:
        if is_next_state_prediction:
            example += "next state: {}\n
    ".format(
                state_to_string(
    transition["next_state"])
            )
        else:
            example += "reward: {}\n".
    format(transition["reward"])

    return example
```

Content Prompt for LLMs. We present the template for building the full prompt, i.e., the in-context examples and the query, for the LLMs in Algorithm 6.

E Model Selection

E.1 World Models

We expect to transform the LLMs into world models without any manually prompt engineering or fine-tuning of LLMs. Therefore, the world models are the general LLMs. The most capable open-source LLM models are the Qwen-2.5-instruct series models (Yang et al., 2024a). Due to the limited

Algorithm 5 Prompt template

```
system_prompt = (
"After being given a current state and
    an action, "
"directly give the next state after
    performing the action."
)
message = [
    {
        "role": "system",
        "content": system_prompt,
    },
    {"role": "user", "content": prompt},
]
```

resources, we only consider the models with sizes in {1.5B, 3B, 7B} for inference and the 1.5B model for RL training. We note that RAWM can work for both open-source and close-source models.

For the embedding model, we choose the General Text Embedding (gte) family (Li et al., 2023). We choose `Alibaba-NLP/gte-Qwen2-1.5B-instruct` as the embedding model, which is the leading open-source model on MTEB.

Emb. Model ϕ	`Alibaba-NLP/gte-Qwen2-1.5B-instruct`

World Model Ω	`Qwen/Qwen2.5-1.5B-Instruct-AWQ`
	`Qwen/Qwen2.5-3B-Instruct-AWQ`
	`Qwen/Qwen2.5-7B-Instruct-AWQ`

Table 5: LLMs for Embedding and World Models

E.2 Architectures of MLP Head

Algorithm 7 presents the python implementation of the two types of initialization of the MLP. Table 6 displays the comparison of the two initializations.

	Random	Identity
Output dimension	Arbitrary	Same to ϕ
Initial performance	Low	High
Training instabilities	Low	High

Table 6: Comparison between two initialization

F Hyperparameters of RL Training

The hyperparameters of RL training are displayed in Table 7 and Table 8.

G Additional Experiment Results

The training curves for Game24 and BabyAI are shown in Figure 9 and Figure 10 respectively.

Algorithm 6 Generating prompts for LLMs

```
def get_query_examples_prompts(
    query_transitions,
    memory_transitions=None,
    exp_name=None,
):
    query_prompts = []
    for idx in range(len(
    query_transitions)):
        query_prompt =
    transition2example(
            query_transitions[idx],
    is_query=True, exp_name=exp_name
        )
        memory_prompt = ""
        if memory_transitions is not
    None:
            for memory_transition in
    reversed(memory_transitions[idx]):
                memory_prompt +=
    transition2example(
                    memory_transition,
    exp_name=exp_name
                )
        query_memory_prompt =
    memory_prompt + query_prompt + "next
     state:"

        query_prompts.append(
    query_memory_prompt)

    return query_prompts
```

Hyperparameter	Value
norm_adv	True
clip_coef	0.2
entropy_coef	0.2
max_grad_norm	0.2
eps	1e-5

Table 7: Fixed Hyperparameters

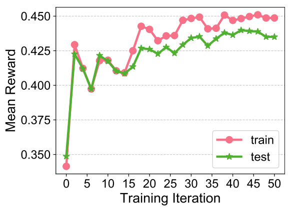
Env	Method	Hyperparameter	Value
Game24	RAWM$^{\text{RL}}_{\psi,\text{rand}}$	learning_rate update_epochs	1e-4 10
	RAWM$^{\text{RL}}_{\psi,\text{eye}}$	learning_rate update_epochs	1e-5 5
BlocksWorld	RAWM$^{\text{RL}}_{\psi,\text{rand}}$	learning_rate update_epochs	1e-4 20
	RAWM$^{\text{RL}}_{\psi,\text{eye}}$	learning_rate update_epochs	1e-5 10
BabyAI	RAWM$^{\text{RL}}_{\psi,\text{rand}}$	learning_rate update_epochs	3e-6 10
	RAWM$^{\text{RL}}_{\psi,\text{eye}}$	learning_rate update_epochs	5e-5 10

Table 8: Modified Hyperparameters

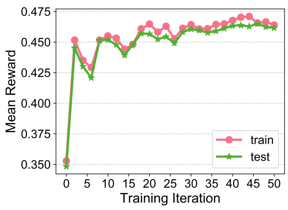
Algorithm 7 MLP initializations

```
# base_emb_dim: dimension of the pre-
    trained embedding model, i.e., 1536
# final_emb_dim: dimension of the MLP,
    36 for rand and 1536 for eye
def layer_init(layer, std=np.sqrt(2),
    bias_const=0.0, with_diag=False):
    if with_diag:
        torch.nn.init.eye_(layer.weight)
        torch.nn.init.constant_(layer.
    bias, 0.0)
    else:
        torch.nn.init.orthogonal_(layer.
    weight, std)
        torch.nn.init.constant_(layer.
    bias, bias_const)
    return layer

mlp_eye = nn.Sequential(
                layer_init(
                    nn.Linear(
    base_emb_dim, final_emb_dim),
    with_diag=True
                ),
            )
mlp_rand = nn.Sequential(
                layer_init(nn.Linear(
    base_emb_dim, 64)),
                nn.Tanh(),
                layer_init(nn.Linear(64,
     64)),
                nn.Tanh(),
                layer_init(nn.Linear(64,
     final_emb_dim), std=0.01),
            )
```

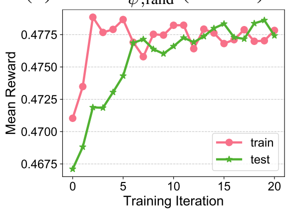


(a) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{rand}}$ $(K=1)$



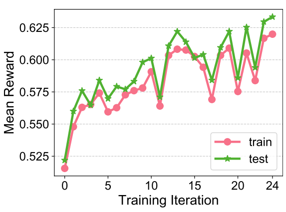
(b) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{rand}}$ $(K=2)$

(c) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{eye}}$ $(K=1)$

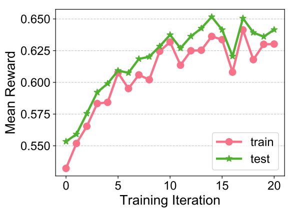


(d) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{eye}}$ $(K=2)$

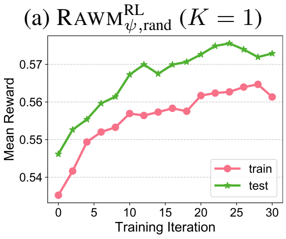
Figure 10: Training curves on BabyAI.



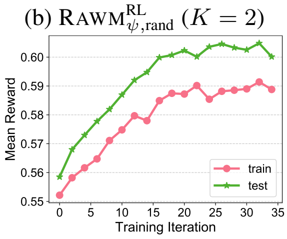
(a) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{rand}}$ $(K=1)$



(b) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{rand}}$ $(K=2)$



(c) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{eye}}$ $(K=1)$



(d) $\textsc{Rawm}^{\text{RL}}_{\psi,\text{eye}}$ $(K=2)$

Figure 9: Training curves on Game24.