Agent Episodic Memory

Nicholas Stranges*
nicholas.stranges@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Khashayar Azad*
khash.azad@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Abstract

Large Language Models (LLMs) face significant challenges in embodied environments due to a lack of short-term memory and their inability to perform long-horizon sequential decision-making. In this work, we introduce a comprehensive episodic memory system for LLM agents operating in the Minecraft environment. Our memory system framework is constructed using Retrieval Augmented Generation (RAG) on the MineRL dataset of human demonstrations of Minecraft gameplay. We introduce and evaluate two retrieval architectures: a unimodal system based on the MineCLIP video embeddings, and a multimodal system that fuses embeddings generated from gameplay video segments and textual descriptions generated by a vision-language model. We evaluate our system on the Treechop task, where the multimodal configuration achieves a performance of 6 wood blocks collected, significantly outperforming the unimodal system (3 blocks) and the baseline agent without memory (0 blocks). Furthermore, we analyze the robustness of our memory database on various indexing techniques, showing that DiskANN offers greater stability against embedding-level and frame-level noise compared to HNSW and IVF_PQ. Our results show that the addition of an episodic memory system positively influences the performance of the agent as it grounds decision-making in past experiences, though this comes at the cost of additional computational latency that affects real-time throughput.

Keywords

Episodic Memory, Retrieval Augmented Generation, Vector Databases, Multimodality, Minecraft, MineRL, Agents

1 Introduction

In recent years, Large Language Models have demonstrated impressive performance in a variety of natural language processing tasks and have shown reasoning abilities that have changed the pace of artificial intelligence research. Models trained on immense amounts of textual data have general world knowledge and can perform complex reasoning chains. However, despite their capabilities, LLMs face significant limitations when being used as embodied agents in visual environments. Since LLMs are trained only on text, they typically struggle to connect their semantic understanding to the spatial and temporal reasoning required by real-world tasks. Although their multimodal variants have indeed narrowed the gap between vision and language, they still tend to lose track of context in tasks that require making a sequence of decisions while interacting with an environment. The most significant challenge LLMs face is their short-term memory. Models have limited working memory and retain only recent information that fits within their current context window. This prevents them from keeping track of information over long periods of time.

Episodic memory refers to the ability to recall specific events in relation to time and space. It is an area that has remained underexplored within LLM research. Huet et al. [7] argue that episodic memory plays a crucial role in how well LLMs can plan, reason, and make sequential decisions. Their work highlights two areas where the lack of episodic memory limits LLMs' potential. The first being that LLMs tend to hallucinate, generating information that is not grounded in reality. Second, LLMs lack mechanisms for long-term storage and retrieval of detailed experiences across their extended interactions with the environment within which they operate [7]. The addition of episodic memory to LLMs enhances their reasoning consistency by grounding outputs in retrievable past experiences rather than relying solely on knowledge encoded in their parameters.

In this work, we present a comprehensive episodic memory system for LLM agents operating in the Minecraft environment. Our contribution lies at the intersection of vector databases and embodied AI agents, specifically, how adding a retrievable episodic memory can help LLMs solve tasks they were not trained to perform. We construct a memory database from past human demonstrations and develop multimodal embeddings from both video and textual observations of gameplay. We evaluate our system on the Treechop task, comparing the performance of an LLM agent equipped with our episodic memory system against a baseline agent without memory augmentation, and additionally evaluating retrieval metrics, including top-k recall, and the resilience of our system to varying levels of embedding noise. The code and data for this project are publicly available.[1]

2 Background

Our work builds upon several previous contributions focused on embodied AI research, particularly those that use a Minecraft environment as a test bed for their sequential decision-making tasks. These prior research works provide essential infrastructure that our research leverages and extends, such as large-scale human demonstration datasets of task gameplay in Minecraft, vision-language alignment models, and LLM-based agent architectures. A common limitation across all these approaches is the lack of a mechanism for storing and retrieving knowledge obtained from past experiences within the same environment. Our work addresses this gap by introducing retrieval augmented generation with multimodal embeddings derived from human demonstrations.

*Both authors contributed equally to this research.

CSC2508 Course Project, University of Toronto

[1] https://github.com/khashazad/agent-episodic-memory

2.1 MineRL

Guss et al. [6] introduced MineRL, a dataset of over 60 million annotated state-action pairs representing human demonstrations across a variety of tasks in Minecraft. The goal of building this dataset was to address sample inefficiency, which is a fundamental challenge in deep reinforcement learning. Each episode in MineRL consists of contiguous state-action pairs sampled at 20 frames per second, where the states are composed of video frames from the player's point of view and game state features such as inventory and player attributes. The authors demonstrated that standard reinforcement learning methods fail to achieve human-level performance on MineRL tasks. Our project uses MineRL as both the source of human demonstrations for building our episodic memory database and as the evaluation environment for measuring the agent's performance. Our project will be focusing on the TreeChop task. This task spawns an agent in a forest biome and the agent's goal is to collect as many wood blocks as possible.

2.2 MineDojo

Fan et al. [4] developed MineDojo, a framework for building open-ended embodied agents that uses internet knowledge about Minecraft. The framework comprises three parts: a simulation platform with a range of diverse tasks, a multimodal knowledge base collected from various internet sources such as YouTube videos and Reddit posts, and MineCLIP, a contrastive video-language model pretrained on over hundreds of thousands of narrated Minecraft videos. MineCLIP calculates correlation scores between goals described in natural language format and 16-frame video snippets, enabling reinforcement learning agents to be trained with dense rewards. Our episodic memory system uses MineCLIP for encoding video observations and textual descriptions into a shared latent space for more efficient retrieval.

2.2.1 *MineCLIP Architecture Details.*

MineCLIP uses a dual-encoder architecture consisting of a video encoder and a text encoder that project their inputs into a shared 512-dimensional latent space. The video encoder takes a 16-frame sequence as input and processes it through a temporal transformer that attends across frames, capturing movement patterns and state transitions. The text encoder uses a standard transformer architecture trained to encode natural language descriptions of Minecraft activities.

The contrastive training objective aligns video-text pairs from narrated gameplay videos. The follow equation represents the loss function used during training:

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(v_i, t_i)/\tau)}{\sum_{j=1}^{N} \exp(\text{sim}(v_i, t_j)/\tau)} \tag{1}$$

where v_i and t_i represent the video and text embeddings for the i-th sample, $\text{sim}(\cdot, \cdot)$ is the cosine similarity, and τ is a learned temperature parameter. This training process enables the construction of an embedding space in which related video and text features cluster, supporting cross-modal retrieval on which our memory system depends.

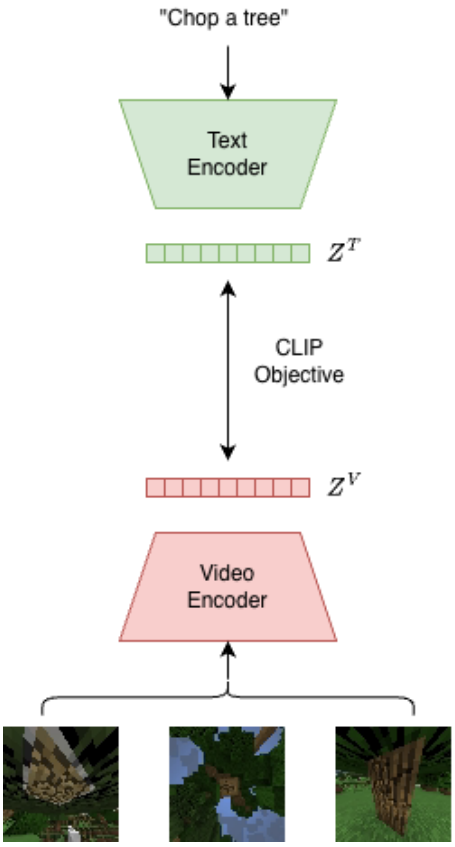

Figure 1: MineCLIP Architecture.

2.3 STEVE-1

Lifshitz et al. [10] introduced STEVE-1, an instruction-tuned agent that can follow short-horizon text and visual instructions using mouse and keyboard controls. Their approach breaks the task of following instructions into two parts. First, the text instruction describing the task is translated into a visual goal that shows what the completed task should look like. Second, a video-pretrained model analyzes the visual goal and decides which control actions should be taken to make the screen match that visual goal. Their method also uses hindsight relabelling, a self-supervised technique in which the model looks at its past trajectories and the resulting states to learn which actions lead to which visual outcomes. The agent shows strong performance on most tasks while requiring a very low compute budget. However, STEVE-1 operates without any retrieval mechanisms or long-term memory integration, so the agent cannot recall prior successful behaviors when faced with similar situations. Our work addresses this limitation by augmenting the agent's decision-making with retrievable episodic memories that provide it contextually relevant action suggestions based on past experiences.

2.4 Voyager

Wang et al. [14] presented Voyager, the first LLM-based embodied agent in Minecraft that continuously explores and learns diverse

skills without any human intervention. The agent operates by progressively building a skill library of executable code based on the feedback it receives from interacting with the environment. The agent interacts with GPT-4 and shows strong in-context learning capabilities. Among the related works, our project most closely resembles Voyager's approach. While Voyager's skill library provides a procedural memory that stores executable programs, the system lacks mechanisms for recalling specific past interactions. Our method enables the agent to retrieve contextually relevant visual observations and action outcomes, guiding it to make better decisions in novel situations.

3 Goals

The main goal of this project is to test if the addition of an episodic memory system will help improve a Minecraft agent's performance. Ideally, there will be a clear separation between the performance of the agent with and without the episodic memory system. To test different agent configurations, a consistent environment that generates measurable rewards is required.

3.1 Indexing Methods

Any memory system that is implemented using vector embeddings produced from an encoder should use a vector database to store memories, as vector databases allow the quick retrieval of vectorized data. Different indexing methods exist within the common vector database packages. A goal of this project is to determine what indexing methods are helpful for this task and what indexing methods should be avoided. Changing the indexing method is an easy way to improve the performance of the entire system.

3.2 Latency

Reducing the overall system latency is another important goal. Minecraft does not require real time processing but, an embodied agent that interacts with the world would. In the case of the current system, there are many forms of latency. The decision making comes from an LLM which is usually many billions of parameters and requires specialized hardware accelerators to run. The environment will have to calculate the next observation based on the requested action. Observations will need to be encoded using an embedding model that could be many millions of parameters. Finally, the vector database will need to perform a k-nearest-neighbours search for the closest data points that match the query. The combination of all of these steps creates a system that could have serious latency and struggle to operate in real time.

3.3 Reranking

The final goal of the project is to extend the standard vector retrieval process by introducing a secondary reranking method similar to Mao et al. [12]. The agent's performance can be compared with and without the reranking system to determine if the extra overhead is worthwhile. As there are once again many ways to implement a working reranking system, the optimal method for this task must be determined.

4 Methods

An episodic memory system requires the creation and validation of different components. Firstly, a stable test environment that can be replication across different tests is needed to ensure the validity of results for different memory techniques. Secondly, an agent framework that is capable enough for the base setup without memory achieve some results is needed. Having a baseline performance for an agent without memory will benchmark how much memory contributes to the problem solving process. Finally, different memory retrieval systems will be created and tested to determine if any method improves the agent's performance on the TreeChop task.

4.1 Environment

Creating a testing environment that is consistent across tests is fundamental to ensure the validity of our experiments. To accomplish this, a docker container with an isolated server was created using the MineRL library [6]. MineRL's TreeChop task is also outdated and unmaintained, adding to the need for a separated environment.

A Flask server acts as the communication layer between the agent and the environment [5]. As MineRL is implemented as an OpenAI Gym environment, the Flask server's main role is to service POST requests made to its /action endpoint [2]. The endpoint receives a dictionary of chosen actions and steps the environment forward. For the purposes of this project, the action space was restricted to only include the options in Table 1.

Table 1: MineRL Action Space Definition

Action	Type	Description
attack	Binary	Executes attack action
back	Binary	Move backward
forward	Binary	Move forward
left	Binary	Strafe left
right	Binary	Strafe right
jump	Binary	Jump action
camera	Continuous	Camera rotation (pitch, yaw) $\in \mathbb{R}^2$

The environment returns an observation, reward, and a done flag which is the result of the chosen action. For the TreeChop task, a reward of +1 is generated when a chopped wood piece is collected and the done flag is only set to true if 64 wood blocks are collected or 18,000 environment steps have been completed. Throughout testing, it was decided that when the attack action is chosen, the attack action should be performed many times in a row. This decreases the burden of having to select the attack action up to ten times in a row to break a wooden block. The observation space is a 64x64 frame that is down sampled from the original Minecraft image. An example of what this frame looks like can be seen in Figure 2.

4.2 Agent

Fundamentally, LLMs produce a distribution over its token vocabulary representing the likelihood of each token given context. The goal of this project is to have the LLM produce the correct sequence of tokens representing the best action to take given a task and

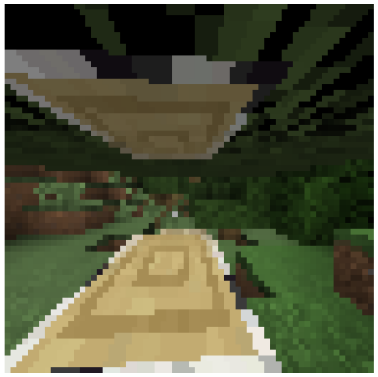

Figure 2: A MineRL observation of a birch tree that has a block destroyed in the middle.

observation. To ensure the robustness of the implementation, the concept of tool calling agents was utilized to create a standard interface for the LLM to interact with the environment. Tool calling LLMs can produce specific outputs based on user defined tools but they must be trained to do this task [13]. The fact that the models have to be trained restricted what models can be used for this setup. To define custom tools, the LangChain framework was used as it handles many burdensome tasks such as parsing LLM outputs [3]. As the observations from the environment are images, the LLMs used in this task must be multimodal so they can process both the images and the text as context [17]. An overview of the agentic system can be seen in Figure 3.

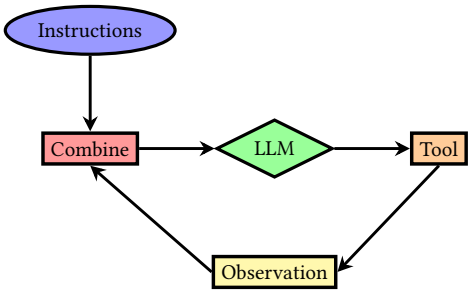

Figure 3: A high level diagram of the agentic system and its components.

4.2.1 Language Model Selection.

Selecting the right large language model to be the backbone of the agent requires balancing between reasoning capability and inference speed. Based on some initial testing with several model architectures, we chose GPT-5-nano as the main model to be used in the agent system. GPT-5-nano is the smallest model in the GPT-5 family and was selected specifically for its balance between reasoning and inference speed. As a reasoning mode, it relies on chain-of-thought processing to break down complex action selection into intermediate reasoning steps, which is more efficient for interpreting visual observations and making appropriate tool calls [16].

The model interacts with the Minecraft environment through the LangChain framework, which handles parsing the outputs for

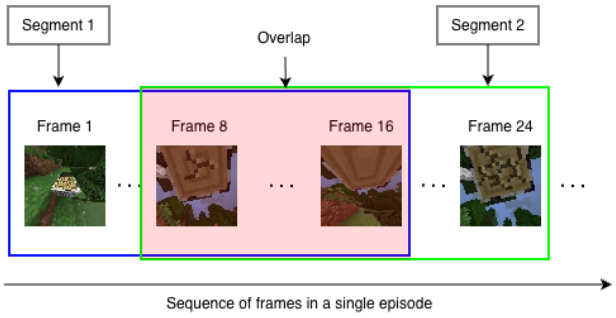

Figure 4: Illustration for the overlapping sliding window segmentation applied to MineRL episodes

action selection. Each call to the model includes the current 64x64 observation frame, the task to collect wood, and, when the memory system is enabled, the retrieved suggested action associated with the matched memory segment. The model reasons over this context to produce a structured action output consistent with the action space described in Table 1.

Despite being the fastest model within the GPT-5 series, there is considerable latency involved due to its reasoning process. The average time taken per Application Programming Interface (API) call for inference was around 8 seconds, which restricted the episode lengths within time and cost budgets. For this reason, we evaluated performance over shorter episodes of 500-700 frames rather than the full 18,000-frame episodes supported by the MineRL environment.

4.3 Memory System

The construction of an episodic memory system for an embodied agent requires consideration of how contiguous gameplay should be partitioned into smaller, retrievable units. The original samples from the MineRL dataset consist of gameplay sequences that can be as long as thousands of frames. This creates a challenge for memory retrieval systems, as long segments dilute semantic representations, while shorter ones fail to capture meaningful context. We address this challenge by using a sliding window mechanism to break long gameplay episodes into smaller, overlapping chunks that are more efficient for our retrieval system.

Each MineRL episode comprises a gameplay video recording captured at 20 frames per second, along with corresponding action data for each frame. This includes movement commands (forwards, backward, left, and right), special actions (jump, sprint, and attack), camera orientation, and reward signals. We segment these episodes into windows of 16 consecutive frames. This window size was selected to capture sufficient behavioural context, such as moving towards a tree or performing an attack action, while keeping the windows short enough to allow retrieval of specific events without mixing multiple unrelated behaviours.

The partitioning process involved using a stride of 8 frames between consecutive windows, resulting in 50% overlap between neighbouring segments. This overlapping structure allows for keeping contextual continuity across the memory collection. For example, during an action sequence where the agent transitions from navigation to collecting wood, partitioning without overlap could

occur at the boundary between these two behaviours, making it impossible to capture the entire context. However, having a 50% overlap guarantees that the transition phases are captured in one of the segments. Additionally, this overlapping redundancy boosts retrieval performance, as episodes are stored from different viewpoints.

4.3.1 *Unimodal Video Embeddings Architecture.*

The initial implementation of our episodic memory system used a unimodal retrieval architecture, in which the video segments generated by the previously described partitioning process serve as both the indexing and query modality. This design leverages MineCLIP, a contrastive vision-language model trained on large-scale Minecraft gameplay, to encode visual experiences into a shared latent space that enables efficient similarity-based retrieval [4].

As shown in the Figure 5, each 16-frame segment generated by the sliding window segmentation is passed through the MineCLIP video encoder, which generates a 512-dimensional embedding vector that encodes the semantic content of the visual sequence. These embeddings serve as the indexing keys in our vector database, encoding information about the environment state, objects visible in the frames, and the activities performed by the agent during the sequence. The values corresponding to these embeddings are the action sequences recorded during those 16 frames. This structure provides a direct mapping between visual contexts and behavioural responses, allowing our memory system to provide experience-based knowledge to the agent during its interactions in the environment.

During inference, the retrieval augmented generation (RAG) system operates by buffering the agent's visual observations until 16 consecutive frames have been collected. This 16-frame sequence is then encoded using the same MineCLIP video encoder, generating a query embedding in the same 512-dimensional latent space as the stored memories. This embedding is then used as a query to perform a nearest-neighbour search against the database, retrieving the top-1 matching segment based on cosine similarity. From the action sequence of the retrieved record, the final action is extracted and provided as additional context to the LLM agent. This retrieved action provides a behavioural suggestion derived from human demonstration in visually similar situations, guiding the agent toward a logical response given its current goal. The assumption behind this architecture is that visual similarity in the latent embedding space correlates with situational similarity and, therefore, actions in a visually similar context will transfer to the agent's current circumstances.

4.3.2 *Multimodal Embedding Architecture.*

While the unimodal approach results in a functional retrieval system, it relies only on visual similarity for memory access, leading it to potentially overlook relationships that are better captured through textual descriptions. Visual embeddings encode low-level features and spatial configurations, but can fail to distinguish between scenarios that are semantically distinct, yet share the same visual characteristics. An example of this would be an agent looking at a tree with the intention of chopping it, versus simply walking past it. To address this limitation, we extended the unimodal architecture to support multimodality, leveraging the aligned latent space of the MineCLIP to combine visual and textual modalities

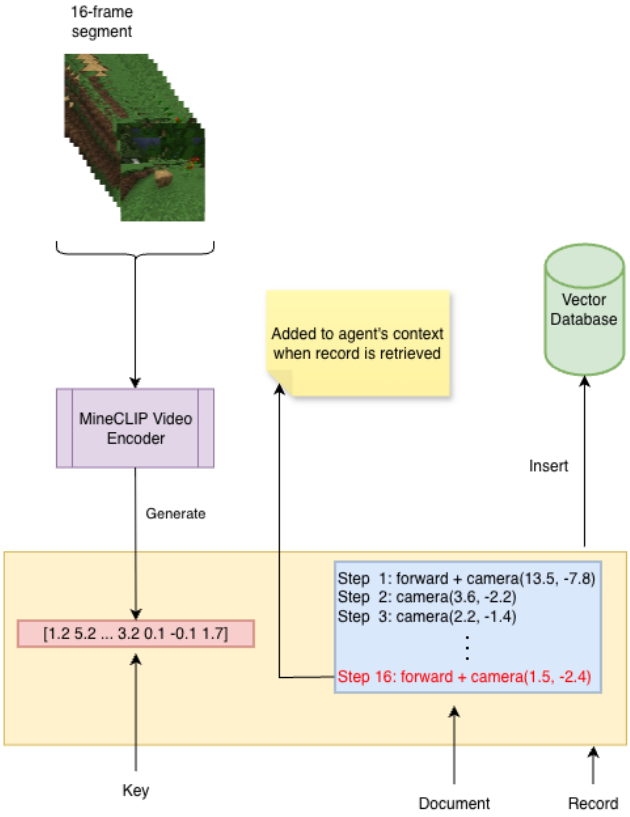

Figure 5: Unimodal Embedding Architecture

into a unified representation. This allowed us to create semantically richer indices for our retrieval systems.

The multimodal pipeline begins with the same video encoding process used in the unimodal architecture, where each 16-frame segment is passed through the MineCLIP video encoder to generate the 512-dimensional embedding. In parallel with the visual encoding, the 16-frame sequence is processed by a local Qwen2.5-VL-7B-Instruct vision-language model to generate a natural language description of the segment's content. To accommodate computational cost constraints, 8 frames are uniformly sampled from the 16-frame window and fed as input to the model. The system prompt contains the model's output in a single sentence that emphasizes environmental context and observable actions. Example outputs include descriptions such as "The agent is looking at oak tree trunks from ground level while chopping wood" or "A forest scene with tall birch trees visible against a blue sky", which capture both spatial context and behavioural state in a short, compact form.

The generated description is then encoded using the MineCLIP text encoder, resulting in a 512-dimensional textual embedding that shares the same latent space as the visual embeddings. This alignment is a property of MineCLIP's architecture, where the model was trained using a contrastive object function that maximizes similarity between matched video and text pairs while minimizing similarity between unmatched pairs, resulting in a shared latent space where semantically related visual and textual features cluster

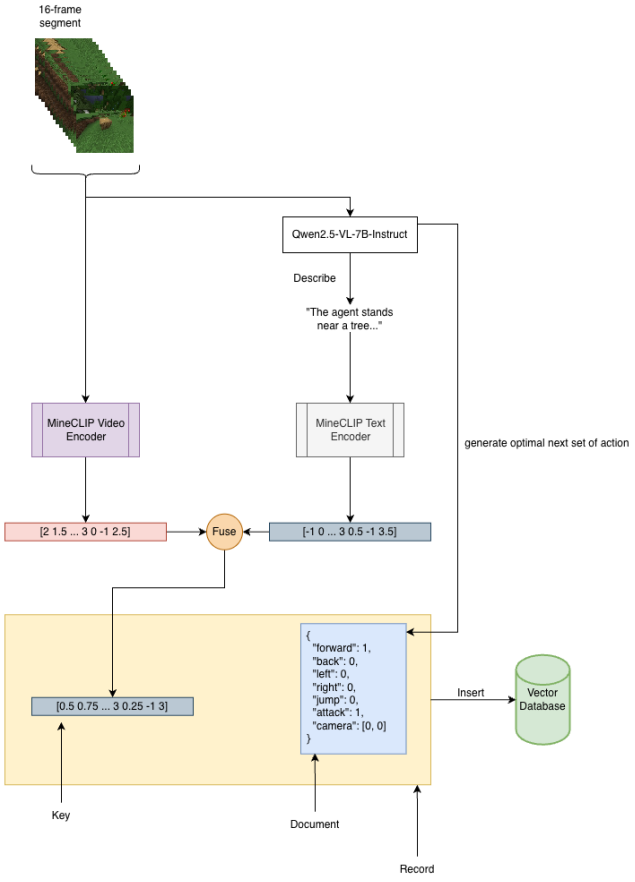

Figure 6: Multimodal Embedding Architecture.

together. This property allows us to perform a naive fusion across embedding modalities without requiring any project layers or alignment mechanisms. We used a simple averaging strategy to fuse the visual and textual embeddings.

Given a visual embedding e_v and textual embedding e_t, the fused multimodal embedding e_m is computed as:

$$e_m = \frac{e_v + e_t}{2}$$

While more sophisticated fusion strategies exist, such as learned weighted combinations and attention-based fusion, the average fusion strategy was chosen for its simplicity. The assumption that justifies using this method is that both modalities provide complementary information of approximately equal importance. Their combination in MineRLs' shared latent space results in a representation that benefits from both perceptual and semantic features. The visual component of the embedding encodes observable environmental features, while the textual component encodes semantics that may help distinguish between visually similar but contextually different scenarios.

The value component of the vector database is also modified within this architecture. Instead of storing the raw action sequence, the video segment is processed by a local Qwen2.5-VL-7B-Instruct vision-language model, which is prompted to generate the optimal next action given the observed context and desired goal. The model analyzes the visual sequence and produces an explicit action recommendation. This predicted action is stored as a retrievable document in our memory database system.

During the inference stage, the agent's observations are buffered into segments of 16 frames and passed through the same multimodal pipeline used to construct the database. The current window of observations is encoded using the MineCLIP video encoder to create the video embedding. The same window is passed to the Qwen2.5-VL-7B-Instruct model to generate the textual description, and then the result is encoded using the MineCLIP text encoder to create the textual embedding. Finally, both video and textual embeddings are fused together using the naive averaging method. The resulting query embedding is matched against the database using nearest-neighbour search based on cosine similarity, retrieving the memory segment whose multimodal representation closely aligns with the agent's current perceptual and semantic context. The action recommendation associated with the retrieved record is then added to the LLM agent's context.

5 Results

We evaluate the performance of our episodic memory framework across three settings on the Treechop task. First, we evaluate the performance of the baseline model without any memory system, second, on a model that uses the unimodal video embedding architecture described in the previous section, and third, on a model that uses a multimodal fused embedding architecture. All of the settings were tested under identical environmental conditions to compare the results. Due to computational constraints, the tests have short episode lengths ranging between 500 and 700 frames. This limitation is due to two main sources of latency: the encoding process required to generate the embeddings from observation frames at each timestep, and the inference latency associated with prompting the LLM agent for action selection. The results presented below represent the best performance achieved across multiple runs per configuration, showing the upper bound of each system's capability rather than average performance.

5.1 Unimodal Memory System

The unimodal memory system, which retrieves actions based only on the video embeddings generated using the MineCLIP video encoder, resulted in an average of 3 wood blocks compared to the baseline agent's 0 wood blocks. This is an improvement over the agent lacking access to the memory system and suggests that visual similarity in the MineCLIP embedding space is useful for action retrieval and allows the agent to make use of appropriate actions associated with human demonstrations in a similar visual context. However, this small margin of improvement also suggests that visual similarity alone may not be an effective way to distinguish scenarios that are semantically different but share similar visual information.

5.2 Multimodal Memory System

The multimodal architecture that averaged the visual and textual embeddings achieved a performance of 6 wood blocks, an increase

of 6 blocks compared to the baseline and 3 blocks above the unimodal approach. This indicates that the generated textual descriptions by the vision-language model provide complementary semantic information that improves retrieval precision.

Table 2: Performance across memory configurations

Configuration	Wood Collected	Δ Baseline	Δ Unimodal
Baseline (No Memory)	0	—	—
Unimodal Memory	3	+3	—
Multimodal Memory	6	+6	+3

While the improvement of 6 wood blocks in performance over the baseline might not seem substantial, it is important to note this improvement in the context of the task difficulty and computational limitations. The Treechop task is challenging for LLM-based agents, as low-resolution 64x64 observation spaces do not convey detailed information about the environment, and wood collection success requires coordinated, consecutive movement and attack commands. In addition, the retrieval pipeline incurs computation overhead, which negatively impacts the overall action throughput of the agent. A single step involves buffering 16 frames, encoding the observation segment, querying the memory database, and adding the retrieved action to the LLM context, each of which adds latency to the agent's response after the environment provides observations. The results show that the benefits of the episodic memory enhancement are limited, but nevertheless helpful. The retrieval robustness analysis presented in Section 6 demonstrates that the retrieval system performs reliably under a range of noise settings, suggesting that performance limitations are a result of the semantic gap between the similarity of embeddings and action relevance.

Aside from the quantitative performance metrics, we performed qualitative analysis of the retrieval system's behavior to determine whether the matched memory segments correspond to any meaningful visual context. Figure 7 shows some of the example instances from the multimodal memory system, including the correspondence between the agent's current observation, the matched memory segment, and the proposed action from the matched record.

The retrieval examples show that the memory system correctly identifies visually and contextually similar scenarios from the human demonstration database. In the first example, the agent observes a forest with oak tree trunks visible at ground level. The retrieved memory segment has a semantically similar environmental context with a similar tree structure and perspective. The proposed action associated with the retrieved memory segment is a forward movement action, which is the correct behavior for the agent's current state, as it has not yet reached the proximity that allows it to attack and harvest the tree.

The second example shows a case where the agent's observation captures an upward view of a tree trunk surface at close range. The matched memory segment shows a visually similar frame capturing tree blocks from a similar viewing angle. In this context, the proposed action is an attack, which indicates the memory system associates close-proximity tree observations with actions leading to harvesting wood. This represents the main functional objective

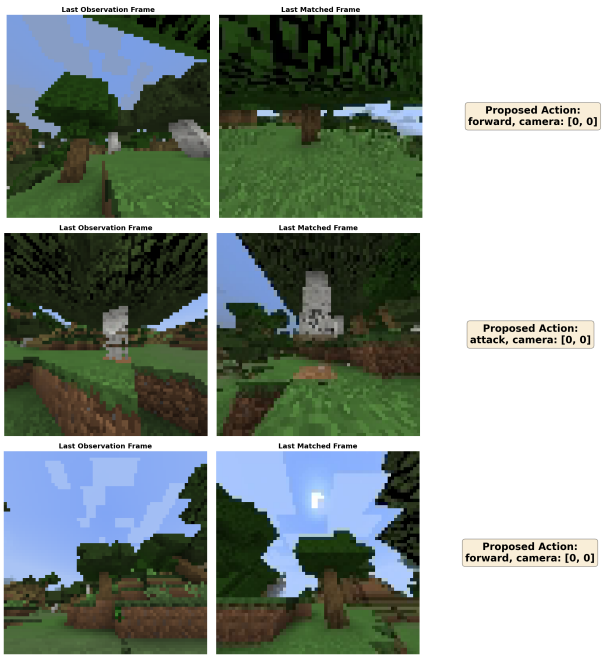

Figure 7: Retrieval examples from the multimodal episodic memory system. Each row presents the agent's current observation (left), the corresponding retrieved memory segment from the memory database (centre), and the proposed action derived from the matched record (right). The examples show semantic correspondence between query and retrieved frames across varying environmental contexts, including navigation scenarios (rows 1 and 3) and wood harvesting situations (row 2).

of the episodic memory system: to provide contextually relevant action suggestions for completing tasks based on demonstrated human behavior in similar situations.

In the third example, the agent observes the tree trunks from a low angle. The retrieved segment matches this with a similar environment containing trees. Because the agent is not in the best position to reach the tree, the proposed action is to move forward while keeping the camera centred since the tree is aligned with the viewing angle.

These qualitative results provide evidence that the MineCLIP learning space indeed represents informative visual relationships relevant to the Treechop task. The matching between the observation frames and the recalled memories suggests that the contrastive objective used in the training process of the model successfully groups similar game situations together, allowing the agent to recall relevant memories. However, the 512-dimensional representation results in some limitations. Deciding whether to move or attack depends on fine details such as precise distance to a tree, which might not get captured by the MineCLIP text and video encoders.

6 Retrieval Robustness

Usually to benchmark the performance of the vector database systems, the actual nearest neighbours are calculated through brute force and those results are compared to the approximate nearest neighbours found by the vector database. For the purposes of this project, the concern was about the performance of the vector database using the MineCLIP video encoder on the low resolution observations produced by MineRL. Also, as the task demands the use of Recall@1 due to the fact that the agent should only receive the most useful action instead of a top-k actions. The choice of Recall@1 makes the benchmark stricter and empathizes how embedding noise affects the stability of the single highest-ranked match.

Two similar experiments were performed to test the effect of noise on the retrieval process. Firstly, noise was added to the existing video embeddings to test index robustness and to gain insight into the decision margin within the vector database itself. Secondly, noise was added to each individual frame in the 16 frame chunk that would then be fed through the MineCLIP encoder to test the encoder's robustness to noise. In both of these tests, noise for each dimension and pixel respectively was generated by sampling noise from a normal distribution as seen in Equation 2. A sweep of different standard deviations were used to visualize Recall@1 vs noise. Once the noise was added, the true answer was compared to the retrieved answer from the database.

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \tag{2}$$

Another natural experiment choice was to compare the performance of different vector database indexing techniques using the noise robustness test. The theory is that specific indexing methods will be more robust to both embedding and frame level noise. The open source vector database Milvus was used as it supports many indexing techniques [15]. Three different indexing techniques were tested: Hierarchical Navigable Small World (HNSW), Inverted File with Product Quantization (IVF_PQ), and DISKANN [8, 9, 11]. For future exploration of the project, the database indexing method could be changed based on these results. It is important to note, all tests in this section were ran using the default Milvus configurations to avoid any influence of hyperparameter tuning.

6.1 Embedding Noise

After adding varying levels of noise to the original embeddings, a clear difference between the three indexing methods is visible. Figure 8 demonstrates the robustness between the three levels. Both DiskANN and HNSW perform similarly but IVF_PQ is the worst performing indexing method for embedding noise robustness. The trend is further demonstrated while analyzing Table 3 which measures the area under the curves (AOC) for each method. Based on the class content from CSC2508, it is understandable that the IVF_PQ method performs the worst. As the noise increases, there is a higher likelihood that the query embedding will be pushed into a different cluster region because the noise error is compounded with the existing quantization error.

The result that IVF_PQ has the worst embedding noise robustness also directly leads to the conclusion that it must also have the smallest decision margin, as defined in Equation 3. In future episodic memory experiments, IVF_PQ should not be used as it could cause the incorrect memories to be retrieved due to its small and unpredictable decision boundary.

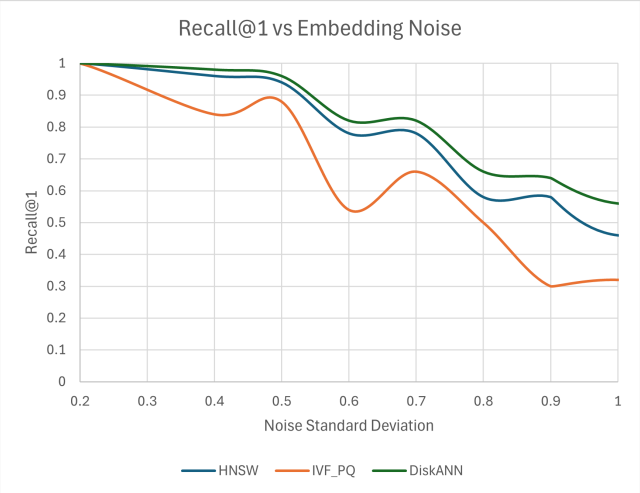

Figure 8: Plot of Recall@1 vs embedding noise for different index methods.

Table 3: Area under the embedding noise curve

Method	AOC
HNSW	1.612
IVF_PQ	1.529
DiskANN	**1.844**

$$\Delta = d\big(q, x_{(2)}\big) - d\big(q, x_{(1)}\big) \tag{3}$$

where q denotes the query embedding, $x_{(1)}$ is the nearest neighbour to q, $x_{(2)}$ is the second-nearest neighbour, and $d(\cdot, \cdot)$ represents the distance metric used for retrieval.

6.2 Frame Noise

The frame level noise tests were mostly to examine how the MineCLIP encoder performed. Again, due to the low resolution images produced by the MineRL environment, slightly different views could have different distortions. These distortions will introduce noise and will require the MineCLIP encoder to be robust to noise.

Figure 9 shows that there are smaller differences between the indexing methods compared to the embedding noise tests. Again, calculating AOC for these different methods shows that DiskANN also performs the best, as seen in Table 4. Mainly the results from Figure 9 demonstrate a mostly linear decreasing trend. A linear decrease as frame noise increases should mean that the MineCLIP encoder preserves neighbourhood structure as the noise increases.

7 Future Work

Throughout the completion of the project, the goals were adjusted based on progress. The main goal of the project was accomplished

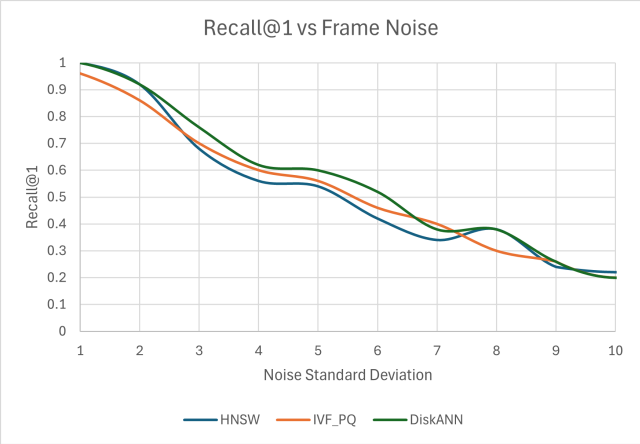

Figure 9: Plot of Recall@1 vs frame noise for different index methods.

Table 4: Area under the frame noise curve

Method	AOC
HNSW	5.689
IVF_PQ	5.701
DiskANN	**6.039**

as there were clear performance differences while using different episodic memory systems compared with a standard agent. The testing of different indexing methods was also successfully able to dictate that IVF_PQ was not a suitable indexing method. The secondary goals of improving latency and the implementation of a reranking system were not able to be completed and will be left for future work.

7.1 Latency

All of the systems tested were not able to run in real time including the agent without episodic memory. The main reason for this was due to the size of the LLM that was used. The GPT-5-nano model was mainly used as the LLM. This model is the smallest and fastest of the GPT-5 family and was chosen because of its balance intelligence and speed. Even being the fastest model in the GPT-5 family, it could not still achieve the necessary speed performance as it is a reasoning model [16]. API calls to the OpenAI API service took an average of 8 seconds to return. As the MineRL environment is 20 frames per second, the current method is roughly 160 times slower than real time processing.

Models that did not use reasoning were tested but none of those models could achieve any satisfactory performance. If reasoning models must be used, specialized accelerators such as the Groq inference chip could be used for open source models [1]. As OpenAI controls all access to GPT-5-nano, there is not much we can do to improve the speed at this time.

7.2 Reranking

A reranking system was not tested but the design of one was discussed. The proposed reranking system would use any base retrieval method that had been developed during this project but instead of only querying the top nearest neighbour, the top-k nearest neighbours would be retrieved. Image embeddings from the single frames can be created so the reranker can find the best frame within the returned top-k episodes. The main reason that the reranking system was not implemented was because of the added latency and the likelihood that a neural network for reranking would need to be post-trained on the Minecraft frames used in this project. Reranking is an interesting future step that we would like to explore as it is commonly used in production RAG systems.

8 Conclusion

In this work, we presented an episodic memory framework designed to counteract some of the limitations of Large Language Models when operating as embodied agents, specifically with regard to long-term memory and sequential decision-making processes. By leveraging the MineRL dataset of human gameplay demonstrations of Minecraft and the MineCLIP contrastive model, we were able to build a retrieval augmented generation framework in which agent actions are based on past human demonstrations instead of relying solely on the model's internal parameters.

Our experiments showed that the addition of the memory system clearly outperforms the baseline agent, which lacked memory capabilities and could not collect any wood blocks. Furthermore, the comparison between unimodal and multimodal embedding architectures highlights the benefits of semantic augmentation. The combination of textual description embeddings and visual embeddings led to a noticeable improvement in the results achieved by the unimodal approach. This validates our hypothesis that multimodal representations capture visual features for spatial context and textual descriptions for semantic context, resulting in higher precision during memory retrieval.

In addition to task performance, our analysis of retrieval robustness showed that the system has stability when varying noise conditions are introduced. Specifically, our comparison of indexing methods shows that, overall, the recall accuracy of DiskANN is better under embedding-level and frame-level noise, while IVF_PQ is the least robust.

Despite these achievements, the system faces the challenge of computational overhead in terms of the latency involved in encoding observations and inference of the agent. Future work should focus on optimizing the inference pipeline to improve real-time interactivity. This research highlights the potential of multimodal episodic memory in filling the gap that exists between static LLM knowledge and dynamic embodied agency.

References

[1] Dennis Abts, Jonathan Ross, Jonathan Sparling, Mark Wong-VanHaren, Max Baker, Tom Hawkins, Andrew Bell, John Thompson, Temesghen Kahsai, Garrin Kimmell, Jennifer Hwang, Rebekah Leslie-Hurd, Michael Bye, E.R. Creswick, Matthew Boyd, Mahitha Venigalla, Evan Laforge, Jon Purdy, Purushotham Kamath, Dinesh Maheshwari, Michael Beidler, Geert Rosseel, Omar Ahmad, Gleb Gagarin, Richard Czekalski, Ashay Rane, Sahil Parmar, Jeff Werner, Jim Sproch, Adrian Macias, and Brian Kurtz. 2020. Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 145–158. doi:10.1109/ISCA45697.2020.00023

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[3] Harrison Chase. 2022. *LangChain.* https://github.com/langchain-ai/langchain Software framework for building applications with large language models.

[4] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. arXiv:2206.08853 [cs.LG] https://arxiv.org/abs/2206.08853

[5] Miguel Grinberg. 2018. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.".

[6] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. 2019. MineRL: A Large-Scale Dataset of Minecraft Demonstrations. arXiv:1907.13440 [cs.LG] https://arxiv.org/abs/1907.13440

[7] Alexis Huet, Zied Ben Houidi, and Dario Rossi. 2025. Episodic Memories Generation and Evaluation Benchmark for Large Language Models. arXiv:2501.13121 [cs.CL] https://arxiv.org/abs/2501.13121

[8] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf

[9] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (Jan. 2011), 117–128. doi:10.1109/TPAMI.2010.57

[10] Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. 2024. STEVE-1: A Generative Model for Text-to-Behavior in Minecraft. arXiv:2306.00937 [cs.AI] https://arxiv.org/abs/2306.00937

[11] Yu. A. Malkov and D. A. Yashunin. 2018. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. arXiv:1603.09320 [cs.DS] https://arxiv.org/abs/1603.09320

[12] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Rider: Reader-Guided Passage Reranking for Open-Domain Question Answering. arXiv:2101.00294 [cs.CL] https://arxiv.org/abs/2101.00294

[13] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Thirty-seventh Conference on Neural Information Processing Systems.* https://openreview.net/forum?id=Yacmpz84TH

[14] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291 [cs.AI] https://arxiv.org/abs/2305.16291

[15] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2614–2627. doi:10.1145/3448016.3457550

[16] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.

[17] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2024. A survey on multimodal large language models. *National Science Review* 11, 12 (Nov. 2024). doi:10.1093/nsr/nwae403