wiki-BALROG: Enhancing Language Model Agents Through Retrieval-Augmented Game Knowledge

Team 3

Abstract

NetHack is a challenging procedurally generated game environment for AI agents, requiring sophisticated reasoning and long-term planning skills. While recent language model agents show promise in text-based environments, their performance in NetHack remains limited by the game's vast action space and sparse reward signals, as well as complex game mechanics. We present a retrieval-augmented generation (RAG) approach that enhances LLMs by dynamically integrating game knowledge from the NetHack Wiki (NetHack Wiki contributors, 2024).Our RAG-enabled agents achieve a progression score of 1.57 ± 0.35, outperforming the BALROG baselines (Paglieri et al., 2024). By using FAISS-indexed document retrieval, we demonstrate that LLM agents with RAG can surpass the performance of those without retrieval augmentation. This work establishes RAG as a critical component for scaling agentic capabilities in complex, knowledge-intensive environments while maintaining computational efficiency.

1 Introduction

Research in Large Language Models (LLMs) continues to improve the decision making and reasoning capabilities of AI agents. Challenging benchmarks are essential for inspiring innovation and exposing weaknesses in state-of-the-art (SOTA) techniques. For example, the game Go inspired self-play and deep neural networks with tree search (Silver et al., 2017), many Atari games were made trivial with the introduction of Deep Q-Networks (Mnih et al., 2013) and Starcraft II made large advances in multi-agent reinforcement-learning and long-horizon coordination (Vinyals et al., 2017, 2019). More recently, the MuJoCo Physics engine generated challenging physical control simulations that drove improvements in transfer and representation learning (Todorov et al., 2012).

Figure 2: An example of the NetHack Learning Environment (NLE), showing the language observations and dungeon map, which are provided as context to the LLM agent.

NetHack is one such challenging benchmark, with a vast action space encompassing thousands of possible interactions between the player and their environment. NetHack originated as a single-player roguelike terminal game, requiring good exploration, long-horizon planning and reasoning skills (NetHack Development Team, 1987). The game is also open source, facilitating the creation of the NetHack Learning Environment (NLE), which was intially intended for RL agents (Küttler et al., 2020). Like most roguelike games, NetHack is procedurally generated and death is permenant, so no two games are identical. This tests the generalisation capabilities of the player.

Figure 2 displays an example of the NLE environment in action. Players are shown a dungeon map and some natural language observations, and simple keyboard commands dictate how the player (`@` symbol) moves and interacts with the environment. The aim of the game is to explore many levels of the dungeon, retrieve `the amulet of Yendor`, offer it to the player's God and escape the dungeon alive. As the user interface is fully ASCII based, evaluating

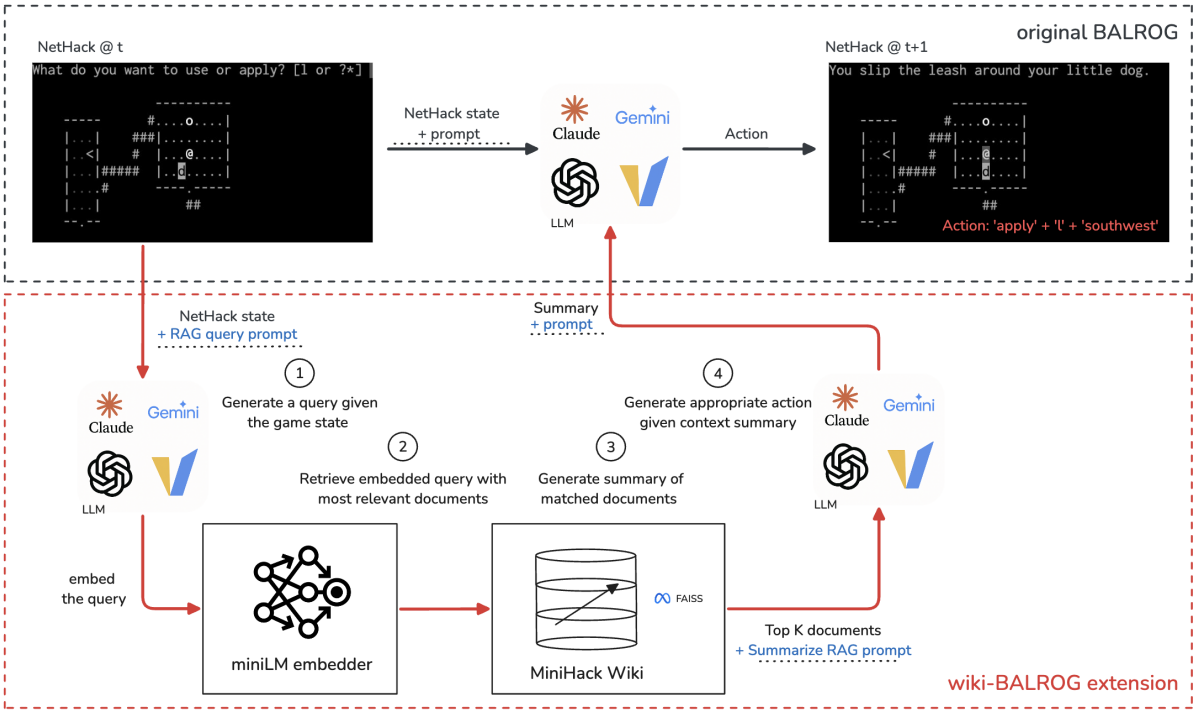

Figure 1: RAG-augmented wiki-BALROG architecture. This extension of the original BALROG benchmark integrates Retrieval-Augmented Generation (RAG) to provide NetHack agents with external game knowledge. The system generates a query from the current NetHack state using an LLM, embeds it with MiniLM, and retrieves relevant documents from the MiniHack Wiki via FAISS. These documents are summarized and combined with chain-of-thought (CoT) instructions to guide the LLM in generating an informed action. This RAG pipeline enriches decision-making by injecting context-aware knowledge into the agent's prompting process.

LLMs on the game is feasible and fast to compute, which was demonstrated in Paglieri et al. (2024). This simplicity of the ASCII-based UI allows researchers to focus primarily on testing the reasoning and decision-making capabilities of LLMs, making NetHack an ideal environment for studying pure cognitive aspects of language models.

Our work builds upon BALROG, a framework for benchmarking large language and vision models in interactive environments. BALROG provides an angentic structure where LLM or VLM agents can interact with both text and image-based environments, including NetHack. Crucially, BALROG demonstrated that SOTA LLM agents struggle to achieve meaningful progress in NetHack, with most models unable to progress beyond the early levels. The sentiment was reflected in the 2021 NeurIPS NetHack Challenge, in which symbolic AI bots systematically beat deep neural agents (Hambro et al., 2022). None were able to beat the game.

In this work, we present a novel agent configuration within the BALROG framework that equips LLM agents with Retrieval-Augmented Generation (RAG) capabilities, leveraging the NetHack Wiki as our primary knowledge source (NetHack Wiki contributors, 2024). One of NetHack's most compelling aspects is its extensive and continually evolving community knowledge base, where dedicated players have contributed detailed strategies, nuanced gameplay tips, and comprehensive explanations of game mechanics over decades. By integrating the Wiki into our RAG pipeline, our agents can dynamically retrieve and apply this curated knowledge during gameplay. To the best of our knowledge, no prior work has attempted to use RAG for NetHack. However PlanCraft in 2024 attempted to enable RAG for LLM Minecraft-playing agents, gaining improvements when tasked to find diamonds in the game (Dagan et al., 2024).

Our choice of RAG for NetHack is motivated by how human players approach the game - they frequently consult the NetHack Wiki and other external sources while playing; a process known in the community as 'spoiling'. RAG mirrors this hu-

man gameplay pipeline, where agents can dynamically retrieve and apply game knowledge during evaluation. This is particularly important given the poor performance of existing language models on NetHack, with most BALROG agents failing to progress beyond early dungeon levels or achieve progression scores above $\sim 2\%$ (Paglieri et al., 2024). The need for additional knowledge grounding is evident in the performance data, where even SOTA models achieve only modest progression in their best performing configurations.

The primary contributions of this work are as follows:

1. The development and integration of a Retrieval-Augmented Generation (RAG) pipeline within the BALROG benchmark, thereby facilitating the rapid prototyping of novel RAG agents.
2. The formulation of a prompt specifically designed to mitigate common failure cases in NetHack, including instances of repetitive or ill-advised actions.
3. The demonstration of enhanced BALROG leaderboard performance for NetHack through the implementation of the newly developed RAG agents.

2 Related Works

Retrieval-Augmented Generation RAG has emerged as a powerful technique for enhancing language models by incorporating external knowledge during inference (Lewis et al., 2020). The core mechanism combines two primary components: a *retrieval module* which efficiently searches through knowledge bases to identify relevant information, and a *generation module* that synthesizes coherent responses using the retrieved content (Lewis et al., 2020). This approach addresses fundamental limitations of traditional language models; by giving them access to data beyond their training data we mitigate their tendency to hallucinate, forget context or generate factually incorrect information (Gao et al., 2024).

Multiple RAG architectures have been proposed to tackle different aspects of knowledge integration. The original RAG framework introduced two key variants: *RAG-Sequence*, which conditions the entire generation on a single retrieved document, and *RAG-Token*, which allows dynamic document retrieval at each generation step (Lewis et al., 2020). More recent approaches like FiD (Fusion-in-Decoder) (Izacard et al., 2022) improved upon these by enabling the model to reason over multiple retrieved passages simultaneously, achieving SOTA performance on knowledge-intensive tasks while maintaining computational efficiency. The success of these approaches has led to the development of specialized RAG variants for specific domains, from question-answering to code generation (Setty et al., 2024; Siriwardhana et al., 2023).

RAG has been applied in game-playing contexts, as demonstrated in a recent Minecraft study (Dagan et al., 2024) where one of the benchmark agents used RAG with the Minecraft Wiki site as a source, although it yielded only modest improvements in agent performance. Unlike Minecraft, NetHack does not involve any complex visual components, so we might expect a RAG agent to be more successful in this environment.

AI Agents and Reasoning Agentic AI systems refer to autonomous agents that independently make decisions and devise strategies without direct human oversight. These systems are capable of integrating external tools to enhance their planning and execution in complex environments (Acharya et al., 2025).

Recent developments in AI agents have focused on combining reasoning with action through frameworks such as ReAct (Yao et al., 2023), which enables language models to interleave thought and action in a more human-like manner. The ReAct framework leverages chain-of-thought reasoning combined with external tool use, allowing agents to dynamically adapt their plans based on new information and feedback from their environment. These iterative processes enable agents to refine their strategies over time, mirroring human cognitive approaches.

In addition to ReAct, other paradigms such as "Plan-Then-Execute" (He et al., 2025) emphasize the importance of structured planning followed by sequential execution. This approach has been shown to improve user trust and collaborative performance, particularly in scenarios requiring step-by-step decision-making.

Despite these advancements, challenges persist in environments where action spaces are vast or consequences are delayed. For example, strategy games such as Diplomacy often require agents to infer outcomes based on incomplete information or stochastic dynamics (Wongkamjan et al., 2024;

META FUNDAMENTAL AI RESEARCH DIPLOMACY TEAM (FAIR) et al., 2022). Addressing these challenges involves developing sophisticated inference mechanisms and probabilistic reasoning models.

2021 NetHack Challenge The NeurIPS 2021 NetHack Challenge revealed important insights about the current state of AI approaches to NetHack (Hambro et al., 2022). Traditional symbolic methods significantly outperformed deep learning approaches, with the best symbolic bot achieving nearly three times the median score of neural agents (Küttler et al., 2020). While deep reinforcement learning showed progress with almost 5x improvement over the IMPALA baseline (Espeholt et al., 2018), the challenge demonstrated that NetHack remains an unsolved problem - the best agents' median scores were orders of magnitude below typical human capabilities.

Specifically, symbolic bots excelled at incorporating domain knowledge and implementing strategy-like subroutines (Sypetkowski, 2024), where neural approaches struggled with long-term credit assignment, such as reasoning which items to collect early in the game to assist later on. Hybrid approaches, such as combination of rule-based systems and neural networks, leveraged the strengths of both paradigms but did not reach human performance.

The competition also highlighted the importance of role-specific strategies and the challenge of managing the game's vast action space, with some teams implementing separated action spaces and role-specific training to improve performance (Petrenko et al., 2020).

Nethack Agents Language models have recently shown promise in game-playing tasks, particularly in text-based environments. NetPlay (Jeurissen et al., 2024) demonstrated application of LLMs as zero-shot agents for NetHack, highlighting both the potential and limitations of current approaches. While these agents can understand game mechanics and formulate basic strategies, they often struggle with the complex, long-term planning required for successful NetHack gameplay. Our RAG-based approach addresses these limitations by providing agents with access to curated game knowledge from the NetHack Wiki (NetHack Wiki contributors, 2024), enabling more informed decision-making while maintaining the benefits of zero-shot learning.

Other complex environments Recent research has demonstrated significant progress in applying Large Language Models (LLMs) to complex game environments, showcasing their potential for problem-solving and decision-making in interactive settings.

For example, in Minecraft, a complex 3D game environment, VOYAGER (Wang et al., 2023) has proven to be a powerful agent that can explore and acquire diverse skills without human intervention. The finetuned model has shown improved reasoning and command usage in Minecraft gameplay.

For Overcooked, a cooperative cooking game, we have seen progress in using LLMs as collaborative agents (Sun et al., 2025; Agashe et al., 2024). Other examples include the Toolformer (Schick et al., 2023), where agents teach themselves to use simple tools like the calculator and calendar, and Hanabi, a card game which forces cooperation as part of gameplay (Bard et al., 2020; Agashe et al., 2024).

These studies collectively demonstrate the progress in leveraging LLMs for diverse gaming applications, from strategic reasoning in mathematical games to complex interactions in open-world settings.

MiniHack (Samvelyan et al., 2021) is a framework for reinforcement learning built on the NetHack Learning Environment. It allows fast environment creation with ASCII specifications and focuses on isolated skill challenges like spatial exploration, tactical planning, or memory tasks. The action space is limited compared to NetHack, depending on the environment, with core commands like movement and basic interactions. We mention this as the MiniHack Python package includes a cleaned version of the NetHack wiki that we will use as a knowledge source.

3 Methods

3.1 Data

We evaluated our RAG-enabled agents using two data sources:

- **NetHack Wiki**: The full contents of the official NetHack Wiki. [1]
- **MiniHack Dataset**: A refined subset of the NetHack Wiki with user comments and forum discussions removed.

[1] nethackwiki.com/wiki

The full NetHack Wiki is expansive and includes substantial noise from user-generated content such as comments and discussion threads. Cleaning this dataset requires in-depth parsing of the XML dump. Therefore, we opted to use the preprocessed MiniHack Wiki dataset provided within the MiniHack package Samvelyan et al. (2021). This dataset offers a structured and cleaned version of the wiki content (NetHack Wiki contributors, 2024), which facilitates more efficient and relevant information retrieval.

To construct our data chunks, we leveraged MiniHack's preprocessing utilities, which provide the cleaned data in JSON format. We applied additional filtering steps to remove irrelevant information and duplicate entries, further improving dataset quality. The final dataset consists of 3,111 document chunks, each containing the fields: `Title`, `Categories`, and `Text`. A sample of the processed data is shown in Figure 6 in the Appendix.

3.2 RAG Implementation

We introduce a RAG client that queries an indexed version of the data source to retrieve relevant documents. Given that the processed data is inherently structured into chunks, we leverage this structure to construct a FAISS index.

The FAISS index is built by extracting the title and textual content from each document chunk, which are then embedded using `all-MiniLM-L6-v2`, a compact transformer-based embedding model[2]. When a query is made, its embedding is computed and compared against the indexed document embeddings using the L2 distance metric. The top k most similar documents are subsequently retrieved.

FAISS-based indexing and similarity retrieval significantly outperform naive brute-force search methods for large-scale knowledge sources. While a brute-force approach requires comparing the query embedding against every document, resulting in linear complexity and prohibitive computational costs, FAISS employs approximate nearest-neighbor search techniques and quantizes embeddings to optimize memory usage. This results in a substantial improvement in query efficiency, particularly when processing multiple queries concurrently. Although FAISS may not always return the exact nearest neighbor, its use of GPU parallelization and optimized search algorithms renders it a highly efficient method for our purposes (Douze et al., 2024; Johnson et al., 2019).

[2]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

3.3 Improved Prompts

User Prompt We introduce the new ROBUST* COT agent as an enhancement to the existing BALROG agent ROBUST COT. The original BALROG prompts frequently generated invalid actions, particularly when used with models like GPT-3.5 and various Gemini versions. These issues stemmed from insufficient examples of structured prompting and inadequate separation between reasoning and action output. Our new variant implements revised user prompts (Figure 9 that:

- Enforce valid action formats with demonstrated examples
- Eliminate problematic reasoning loops
- Provide clearer guidance for context-appropriate decision making

System Prompt We introduce modifications to the BALROG NetHack Environment instructions as illustrated in Figure 7 in the Appendix. These system prompt changes are specifically designed to emulate human gameplay by explicitly providing contextual information that human players would naturally possess while playing the game. These enhancements should help the agent:

- Make more informed and realistic decisions
- Process environmental cues more effectively
- Align its behaviour more closely with human-like reasoning patterns

3.4 RAG context for the LLM

Generating RAG Queries Our initial approach involved passing the agent's observations, either short-term context (e.g., inventory, status, ASCII map) or a combination of short-term and long-term context—directly to the RAG client as the query. This approach provided a static context for each call, as the observations did not change significantly between consecutive game steps. The length of the resulting prompt is influenced by the following:

- `max_cot_history`: Number of previous chain-of-thought steps retained in the prompt.
- `max_history`: Number of previous interaction messages retained.

With values such as `max_cot_history` ≥ 4 and `max_history` ≥ 8, the prompt becomes sufficiently long to cause the model to lose focus when

selecting an action. This often led to less effective or ambiguous behavior.

We attempted to reduce prompt size by limiting context to short-term observations (e.g., current inventory), but this caused repeated retrieval of nearly identical documents, since inventory changes slowly. As a further brute-force method, we truncated retrieved documents to their first 500 characters. However, this approach proved ineffective, as it stripped away important contextual information and failed to meaningfully inform the agent's decisions. It became clear that the query itself needed to be more focused and semantically meaningful. Given that the underlying LLMs already possess knowledge of NetHack gameplay mechanics, we saw that the model could be tasked with generating its own retrieval query based on its understanding of the current game state.

AutoRAG: The AutoRAG mechanism enables the agent to generate a targeted keyword-phrase query based on its observations and gameplay knowledge. These pointed queries substantially narrow the retrieval space and improve document relevance. For instance, a query at the start of the game might be "Explore starting level, find stairs down", whereas later-stage queries dynamically adapt to nearby threats or opportunities, such as inventory items or adjacent enemies (Figure 5).

Passing Retrieved Context to the Agent Initially, we passed the full retrieved documents directly to the game agent. However, the large volume of content often overwhelmed the prompt, making it difficult for the model to identify and focus on relevant information. This negatively impacted decision quality.

To address this, we introduced an additional LLM call, separate from the active gameplay loop. In this step, we provide the model with the current observations and the retrieved documents, and ask it to generate a concise summary containing only the most relevant information. This summarized context is then passed to the game agent, enabling more informed decision-making without significantly increasing the prompt length.

The complete system pipeline is illustrated in Figure 1. The red-dotted box highlights the original BALROG inference loop, while the grey-dotted box shows our extended RAG pipeline. This retrieval-augmented component is modular and can be independently integrated into any agent configuration. It encapsulates query generation, document retrieval, summarization, and integration into the decision-making prompt, enabling more context-aware action selection.

4 Experiments

We evaluated our approach using the NetHack Learning Environment with a fixed seed across all runs to ensure comparable initial conditions. All experiments used the Gemini Flash 2.0 model as the underlying LLM. Performance was measured using the progression score, which reflects how far the agent advanced in the game. For each configuration, we recorded the best average score across five parallel episodes. Due to the procedurally generated nature of NetHack, even with a fixed seed, the progression and outcome of each run varies significantly.

To understand the contribution of each component in our approach, we conducted a series of ablation studies:

Prompt Engineering We evaluated the impact of our prompt engineering improvements on agent performance. Figure 3 shows the performance gains achieved by various combinations of RAG and improved system and user ($\star$) prompts, both individually and in combination.

RAG Context Summarization We also performed an ablation study to evaluate the impact of RAG context summarization. Table 1 shows the results of passing full documents versus summarized documents to the agent.

5 Results and Discussion

Figure 3 presents our main results comparing various agent configurations against the BALROG baseline. Our best-performing agent, which combines improved system prompts, enhanced user prompts, and RAG, achieves an average progression score of 1.57 ± 0.35, significantly outperforming the BALROG benchmark established by Claude-3.5-Haiku-2024-10-22 (1.2 ± 0.4). In contrast, the baseline BALROG agents achieve a progression score of 0.00 in all configurations except for the CoT variant, which attains an average score of 0.37.

The plots (b) and (c) in Figure 3 demonstrate that our best agents reach significantly higher Dungeon Levels (up to Level 7) and Experience Levels compared to the baseline, surpassing the best dungeon level reported in Paglieri et al. (2024). These

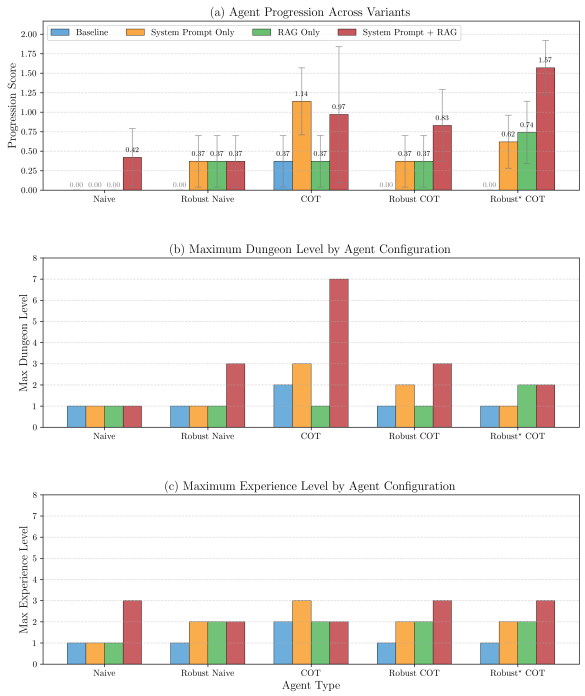

Figure 3: Comparison of the results obtained on different agents by fixing a seed value (same initial starting point), and taking the highest progression of 5 episodes over 3 runs. Plot (a): Average progression and standard deviation, Plot (b): Corresponding maximum Dungeon Levels reached, Plot (c): corresponding maximum experience level reached. We can see that combination of prompt changes and RAG consistently improves the progression score as well as the max dungeon level reached.

improvements stem from more informed decisions such as efficiently defeating monsters or strategically using potions to gain experience.

The results demonstrate that even without retrieval augmentation, well-structured and informative prompts reasonably reduce the chance of errors in the output of LLM agents. Our newly introduced ROBUST* COT agent, which builds upon the original Robust CoT variant with enhanced prompt design, improves upon the performance of ROBUST COT.

5.1 RAG Context Summarization

Following the setup described in Section 3.4, we compared the effects of passing full documents versus summarized documents to the agent. Table 1 shows that passing entire retrieved documents increased the prompt length and possibly interfered with decision-making. By summarizing retrieved content before integration, the agent's progression score increased, demonstrating the effectiveness of condensed context for guiding reasoning and action.

Sample RAG Query, Summary, Reasoning

- **Rag Query:** How to remove curses?
- **RAG Summary:**
  - Remove curse can be achieved through the spellbook of remove curse, the scroll of remove curse, or a magic trap.
  - A limited remove curse effect (magic trap, uncursed scroll, or basic spell) uncurses the wielded item, worn armor/rings/blindfold/amulet, and carried loadstones, and reverses punishment.
  - A full remove curse effect (blessed scroll or skilled+ spell) uncurses all inventory items and reverses punishment.
  - Praying can uncurse items. Cursed blindfolds, towels, levitation items, two-handed weapons, and wearing both a cursed weapon and cursed shield, or a cursed amulet of strangulation are treated as major problems ...
- **Agent Reasoning:** The RAG results confirm that praying can uncurse items, and since I'm being punished, it's worth trying again. The RAG results also indicate that the cursed items I have are considered a minor problem, so there's a chance that praying will uncurse them. I'm going to pray again and hope for the best.

Figure 4: Example of improved chain of thought reasoning with RAG integration

RAG Context	Progression
Full	0.4 ± 0.4
Summarized	0.7 ± 0.4

Table 1: Comparison of progression scores using Gemini-2.0-Flash as the LLM and Robust CoT RAG as the agent, evaluated under different RAG context lengths. *Full RAG context* refers to passing the entire retrieved document chunks to the LLM, while *Summarized context* refers to passing LLM-generated summaries of those documents.

5.2 Combined Effects

The ROBUST + RAG, ROBUST* WITH SYSTEM PROMPT IMPROVEMENTS + RAG and ROBUST* WITH SYSTEM PROMPT IMPROVEMENTS bars in Figure 3 illustrate the synergistic effects of combining our improvements. The results indicate that while each component provides individual benefits, their combination yields the most substantial performance gains. This suggests that prompt design and retrieval work synergistically, together producing better results than either component alone.

Sample RAG Queries Generated

- How to fight kobold throwing darts?
- When to use magic mapping scrolls?
- Dip dagger in fountain, good idea?
- What to do with goblin corpse?
- What to do with scroll labeled ASHPD SODALG?
- Attack tame dog or move away?
- When to read scroll of light?

Figure 5: Examples of targeted queries generated by the AutoRAG system

5.3 Discussion

A key contributor to our performance gains is the integration of RAG. By enabling agents to generate context-specific queries based on the current game state, RAG allows for the retrieval of highly relevant information from the MiniHack Wiki. This targeted retrieval supports the development of more coherent chain-of-thought reasoning, ultimately leading to more effective decision-making. As illustrated in Figure 5, the agents are capable of posing contextually appropriate questions and subsequently acting on the retrieved knowledge. This behavior contrasts sharply with baseline BALROG agents, which often failed to exhibit strategic planning or incorporate external knowledge. The impact of this improvement can be further observed in Figure 4, where the LLM successfully utilizes summarized RAG context to select a well-informed action.

Another noteworthy observation is that our agents tend to die in ways that the original BALROG agents did not. For instance, some of the failure cases include deaths such as "Killed by the wrath of Kos" or "Killed by a potion of holy water". These examples suggest that, following the integration of RAG, the agents exhibit more exploratory behavior, attempting actions like engaging with in-game deities or using unfamiliar items, which were not observed in the baseline BALROG agents. This indicates that the enhanced prompts and knowledge integration encourage the agents to experiment and interact more dynamically with the environment, even at the risk of failure.

However, we also observed cases where the integration of RAG led to a reduction in the agent's average survival duration. This is primarily because the agent occasionally attempts actions inspired by information retrieved from external documents, such as executing advanced strategies intended for later stages of the game — for example, attempting to fight shopkeepers — rather than focusing on basic objectives like fully exploring the current level. As a result, the agent may engage in unnecessary or premature behaviors that increase the risk of failure.

6 Conclusion and Future Work

In this work, we evaluated the effect of integrating Retrieval-Augmented Generation (RAG) and prompt optimization on language model agents in the NetHack environment. Our results show that these additions significantly improve agent performance, both in terms of progression score and overall gameplay depth. The agents not only achieved higher dungeon and experience levels but also demonstrated more strategic and exploratory behaviors.

While the use of external knowledge led to improved decision-making, it also introduced new failure modes, particularly when agents acted on information that was contextually inappropriate for their current state. This highlights the challenge of aligning retrieved knowledge with in-game objectives.

Overall, the results suggest that retrieval and prompting techniques hold promise for improving the reasoning capabilities of LLM-based agents in complex environments.

Future work could explore several promising directions. First, developing more adaptive knowledge integration mechanisms that dynamically adjust to the agent's progression in the environment would help provide more contextually relevant information. Second, refining system prompts to better handle edge cases and prevent misinterpretation of retrieved knowledge could address failure modes observed in our experiments (see Figure 8). Third, expanding the knowledge base to incorporate the entire NetHack wiki, including valuable forum discussions and user feedback, would provide agents with a more comprehensive understanding of game mechanics and strategies. Finally, implementing a reward model that better aligns with long-term strategic goals rather than immediate rewards could help guide agent decision-making toward more successful gameplay trajectories.

References

Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B. Divya. 2025. Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey. *IEEE Access*, 13:18912–18936.

Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. 2024. LLM-Coordination: Evaluating and Analyzing Multi-agent Coordination Abilities in Large Language Models. *arXiv preprint*. ArXiv:2310.03903 [cs] version: 2.

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. 2020. The Hanabi Challenge: A New Frontier for AI Research. *Artificial Intelligence*, 280:103216. ArXiv:1902.00506 [cs].

Gautier Dagan, Frank Keller, and Alex Lascarides. 2024. Plancraft: an evaluation dataset for planning with llm agents. *arXiv preprint arXiv:2412.21033*.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, and 1 others. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint*. ArXiv:2312.10997 [cs].

Eric Hambro, Sharada Mohanty, Dmitrii Babaev, Minwoo Byeon, Dipam Chakraborty, Edward Grefenstette, Minqi Jiang, Jo Daejin, Anssi Kanervisto, Jongmin Kim, and 1 others. 2022. Insights from the neurips 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 41–52. PMLR.

Gaole He, Gianluca Demartini, and Ujwal Gadiraju. 2025. Plan-then-execute: An empirical study of user trust and team performance when using llm agents as a daily assistant. *arXiv preprint arXiv:2502.01390*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Nicola Cancedda, Sebastian Rokhlenko, and Sebastian Riedel. 2022. Atlas: Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*.

Dominik Jeurissen, Diego Perez-Liebana, Jeremy Gow, Duygu Cakmak, and James Kwan. 2024. Playing nethack with llms: Potential & limitations as zero-shot agents. In *2024 IEEE Conference on Games (CoG)*, pages 1–8.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. 2020. The NetHack Learning Environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.

Heinrich Küttler, Nantas Nardelli, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. 2020. The NetHack learning environment. *arXiv preprint arXiv:2006.13760*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

META FUNDAMENTAL AI RESEARCH DIPLOMACY TEAM (FAIR), Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, Athul Paul Jacob, Mojtaba Komeili, Karthik Konath, Minae Kwon, Adam Lerer, Mike Lewis, Alexander H. Miller, Sasha Mitts, Adithya Renduchintala, and 8 others. 2022. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074. Publisher: American Association for the Advancement of Science.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv preprint*. ArXiv:1312.5602 [cs].

NetHack Development Team. 1987. NetHack: The game. *NetHack*.

NetHack Wiki contributors. 2024. Nethack wiki. Accessed: 2024.

Davide Paglieri, Bartłomiej Cupiał, Sam Coward, Ulyana Piterbarg, Maciej Wołczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. 2024. Benchmarking agentic LLM and VLM reasoning on games. *arXiv preprint arXiv:2411.13543*.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. 2020. Sample factory: Egocentric 3d control from pixels at 100000

fps with asynchronous reinforcement learning. In *ICML*.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. 2021. MiniHack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*.

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. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Spurthi Setty, Harsh Thakkar, Alyssa Lee, Eden Chung, and Natan Vidra. 2024. Improving retrieval for rag based question answering models on financial documents. *arXiv preprint arXiv:2404.07221*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2023. Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering. *Transactions of the Association for Computational Linguistics*, 11:1–17. Place: Cambridge, MA Publisher: MIT Press.

Haochen Sun, Shuwen Zhang, Lei Ren, Hao Xu, Hao Fu, Caixia Yuan, and Xiaojie Wang. 2025. Collab-overcooked: Benchmarking and evaluating large language models as collaborative agents. *arXiv preprint arXiv:2502.20073*.

Sypetkowski. 2024. Autoascend: A symbolic approach to nethack. *arXiv preprint arXiv:2403.00690*.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. ISSN: 2153-0866.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, and 23 others. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354. Publisher: Nature Publishing Group.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, and 6 others. 2017. StarCraft II: A New Challenge for Reinforcement Learning. *arXiv preprint*. ArXiv:1708.04782 [cs].

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 preprint arXiv:2305.16291*.

Wichayaporn Wongkamjan, Feng Gu, Yanze Wang, Ulf Hermjakob, Jonathan May, Brandon M. Stewart, Jonathan K. Kummerfeld, Denis Peskoff, and Jordan Lee Boyd-Graber. 2024. More Victories, Less Cooperation: Assessing Cicero's Diplomacy Play. *arXiv preprint*. ArXiv:2406.04643 [cs].

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

A Appendix

A.1 Wiki Data Processing

Figure 6 shows a sample of how we processed and structured the NetHack Wiki data into a JSON format suitable for retrieval. Each entry contains the title, categories, and raw text extracted from the original wiki pages. This structured format enables efficient indexing and retrieval of relevant game knowledge during gameplay.

Cleaned Wiki Data

```
{
  "high-elf": {
    "title": "high-elf",
    "categories": ["Defunct monsters"],
    "raw_text": "High-elf\nThe High-elf is a defunct feature in NetHack ..."
  },
  "samurai": {
    "title": "samurai",
    "categories": ["Roles"],
    "raw_text": "Samurai\nSamurai are Japanese warriors who adhere to ..."
  },
  ...
}
```

Figure 6: Sample of the processed wiki data in JSON format; entries are keyed by name and contain values including what category of item they are and text derived from the original NetHack Wiki page.

A.2 Additional Gameplay Prompts

Figure 7 details the supplementary prompts we added to guide the LLM's behavior in the NetHack environment. These prompts address common failure modes and encourage more strategic gameplay by providing important game mechanics context and general gameplay principles.

Additional prompts in the NetHack Environment

- **Taking the stairs up without Amulet of Yendor will quit the game and you will lose**.
  - *The basic game structure of NetHack is that the player needs keep descending through the dungeon levels, retrieve the Amulet of Yendor, and then ascend back. As the LLM is not made aware of this, it takes the stairs up and loses the game.*
- **Learn the special abilities of your character class and use them to your advantage.**
  - *As the character class contain special abilities, the LLM should be made aware that it can use it to properly mimic a human game play strategy.*
- **Do not get stuck trying to achieve a specific short term goal. The goal is to explore as much as possible.**
  - *The LLM often gets stuck trying to do a particular task (e.g., picking something up) leading to the termination of the game.*

Figure 7: Additional Prompts added to the NetHack Environment (with the reasoning) that are being passed to the LLM as a context at every step to emulate human-like gameplay.

A.3 Common Failure Modes

Figure 8 illustrates a typical failure case where the agent gets stuck in a loop trying to pick up items. This example highlights the importance of proper action formatting and the need for better handling of multi-step interactions.

A.4 Prompt Engineering

Figure 9 shows the evolution of our prompting strategy, comparing the original Robust CoT prompt with our improved Robust* CoT prompt. The enhanced prompt includes explicit action verification and format

Failure Examples

```
CoT: "I should pick up the key and rocks."
Output Action: pickup
System Message: "Pick up what?
Tools
a - a key
Gems/Stones
b - 26 uncursed rocks
(end)"

CoT: "I should pick up the rocks to use as projectiles."
Output Action: "b"
System Message: "Pick up what?
Tools
a + a key
Gems/Stones
b + 26 uncursed rocks
(end)"

....
```

Figure 8: Example of one of the cases where the Agent keeps failing. The agent wants to pickup something, and keeps on outputting the action ”a” or ”b”.

examples to reduce errors in action selection and output formatting.

User Prompt

Robust CoT prompt: First, think about the best course of action. Then, you must choose exactly one of the listed actions and output it strictly in the following format:
$< |ACTION| > YOUR\_CHOSEN\_ACTION < |END| >$
Replace $YOUR\_CHOSEN\_ACTION$ with the chosen action.

Robust* CoT User Prompt: First, think about the best course of action. Then, you must choose exactly one of the listed actions and output it strictly in the following format:
$< |ACTION| > YOUR\_CHOSEN\_ACTION < |END| >$
Replace $YOUR\_CHOSEN\_ACTION$ with the chosen action. Verify that the action you provided is a valid action from the list of actions given.
In case you want to choose the action ”go forward”, you must output: $< |ACTION| > goforward < |END| >$
Explain your action choice in not more than 20 words.

Figure 9: The change in User Prompt.