Knowledge Retrieval in LLM Gaming: A Shift from Entity-Centric to Goal-Oriented Graphs

Anonymous ACL submission

Abstract

Large Language Models (LLMs) demonstrate impressive general capabilities but often struggle with step-by-step reasoning, especially in complex applications such as games. While retrieval-augmented methods like GraphRAG attempt to bridge this gap through cross-document extraction and indexing, their fragmented entity-relation graphs and overly dense local connectivity hinder the construction of coherent reasoning. In this paper, we propose a novel framework based on Goal-Oriented Graphs (GoGs), where each node represents a goal and its associated attributes, and edges encode logical dependencies between goals. This structure enables explicit retrieval of reasoning paths by first identifying high-level goals and recursively retrieving their subgoals, forming coherent reasoning chains to guide LLM prompting. Our method significantly enhances the reasoning ability of LLMs in game-playing tasks, as demonstrated by extensive experiments on the Minecraft testbed, outperforming GraphRAG and other baselines.

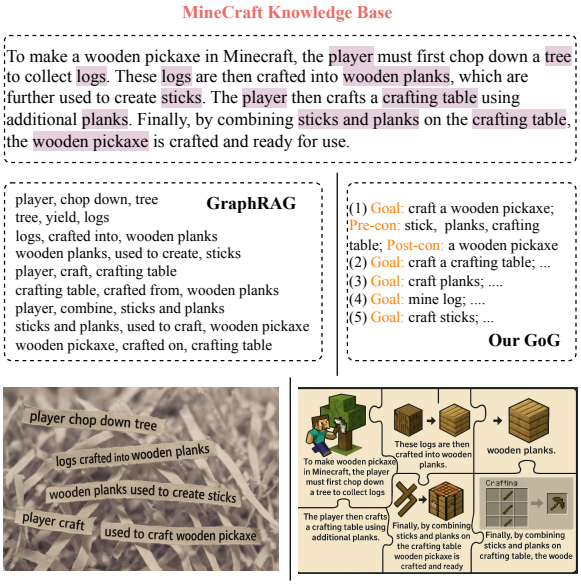

Figure 1: GraphRAG extracts an excessive number of low-granularity entity–relation triples, which hinders effective reasoning over fragmented information. In contrast, our GoG captures procedural knowledge through goal hierarchies, thereby supporting coherent and structured reasoning.

1 Introduction

Leveraging the extensive knowledge embedded in Large Language Models (LLMs) within game environments has the potential to benefit and transform a wide range of sectors, including virtual assistants, non-player characters (NPCs), and other interactive applications. Recent studies have explored the application of LLMs in strategic games such as chess (Feng et al., 2023) and poker (Huang et al., 2024). Building on this progress, agents designed to tackle more challenging open-ended environments like Minecraft—such as Voyager (Wang et al., 2024) and GITM (Zhu et al., 2023)—have played a pivotal role in advancing the capabilities of LLM-based agents.

Since LLMs are generally trained for broad, general-purpose tasks and often exhibit limited domain-specific knowledge in games, various approaches have been proposed to incorporate external game knowledge—either through retrieval from specialized databases (Gao et al.; Edge et al., 2025) or by directly fine-tuning the LLMs for game-specific purposes (Zhai et al., 2024). Notably, GraphRAG-based methods (Guo et al., 2024; Wu et al., 2024) have achieved superior results due to their efficient extraction, organization, and indexing of entity–entity–relation triples, which enable effective retrieval across multiple heterogeneous game documents.

However, GraphRAG suffers from several notable limitations, particularly in tasks that require extensive reasoning, such as Minecraft, where agents must plan and act over multiple steps. A key issue is that GraphRAG fragments documents

into very small pieces, as it prioritizes local entity-level connections over global narrative coherence. For example, in our Minecraft setting, GraphRAG extracts 12,388 nodes and 18,347 edges from the source materials, which hinders the construction of coherent reasoning chains during retrieval. Moreover, the abundance of entities results in overly dense 1-hop neighborhoods, especially around frequently occurring entities. Consequently, many retrieved nodes may be irrelevant to the input query and only loosely related to the central entity. This excessive retrieval introduces noise and can significantly degrade performance—a pitfall we observed when comparing GraphRAG to vanilla retrieval without graph augmentation.

As the saying goes, "tearing paper is easy, putting it back together is hard"—a metaphor that aptly reflects the difficulty of reconstructing reasoning chains in fragmented knowledge graphs. This motivates us to design an efficient knowledge extraction and organization framework that can effectively support multi-step reasoning. In this paper, we propose to construct a directed, goal-oriented graph (GoG), where each node represents a goal along with its associated attributes, and each edge encodes the logical relationship between two goals. Leveraging this novel graph structure, we design an effective retrieval strategy that first retrieves the most relevant goals and then iteratively retrieves their corresponding subgoals, thereby forming a complete reasoning chain. This reasoning chain significantly enhances the capabilities of LLMs in game-playing tasks compared to GraphRAG and its variants, as demonstrated by our extensive experiments on the Minecraft testbed.

The comparison between GraphRAG and our method GoG is illustrated in the Figure 1. In this toy example, GraphRAG extracts 9 triples, making it difficult to summarize the logical reasoning required to craft a wooden pickaxe. In contrast, our method extracts only 5 goals and subgoals, which are logically structured to clearly support the reasoning process for online inference.

Our contributions are summarized as follows:

- We introduce an alternative framework to traditional entity–relationship knowledge graphs-Goal-Oriented Graphs (GoGs), designed to facilitate multi-step reasoning.
- We propose a goal-driven retrieval that identifies the most relevant goals and recursively retrieves their subgoals, explicitly forming a reasoning chain to guide LLM prompting.
- Extensive experiments on Minecraft environments demonstrate that our method achieves superior performance compared to baseline approaches.

2 Related Work

LLM-Based Agents have been proposed for Minecraft. For example, Voyager is an agent that learns skills via lifelong learning (Wang et al., 2024). MP5 (Qin et al., 2024) focuses on leveraging multimodal LLMs to perform planning based on what the agent sees. Unlike these methods, our method aims to improve an agent's performance by using goal-oriented knowledge extracted from text sources. Two methods that are more closely related to ours are GITM (Zhu et al., 2023) and Optimus-1 (Li et al., 2024). GITM uses text knowledge from the Minecraft Wiki and in-game recipes, as does our method. However, we focus on the construction of Goal-Oriented Graphs from text sources and the retrieval process for planning. On the other hand, GITM is presented as a unified agent system. Optimus-1 is a Minecraft agent that contains a hierarchical knowledge graph that is used to decompose goals into subgoal. However, the process in which their graph is created is different from ours, and their agent is focused on the utilization of multimodal memory.

Reasoning using LLMs is a popular research area that aims to enable LLMs to handle more complex tasks. There are a wide variety of methods, including those that alter the prompt to encourage the LLM to output intermediate steps (Wei et al., 2022), provide feedback from the environment to the LLM so that it can adjust its behaviour accordingly (Yao et al., 2023b; Shinn et al., 2023), maintaining and expanding multiple generated reasoning sequences (Yao et al., 2023a), and using majority choice voting (Wang et al., 2023).

RAG-Based Systems aim to address LLMs' knowledge gaps. RAG maintains a vector database of source documents that can be retrieved a given to the LLM as context information according the embedding similarity of the source documents a given query (Lewis et al., 2020). However, if the answer to a given query spans across several documents, then the retrieval process may fail to contain the answer. Therefore, GraphRAG and its variants were proposed to address this issue (Edge et al., 2025; Wu et al., 2024; Guo et al., 2024).

Hierarchical Decomposition is a traditional

method in AI to break complex problems into smaller sub-problems that has been used in areas including planning (Ghallab et al., 2004; Erol et al., 1994) and reinforcement learning (Dietterich et al., 1998; Sutton et al., 1999). These methods typically rely on domain knowledge or hand-crafted rules to establish the hierarchy of tasks. In this work, we aim to construct a hierarchy of goals in the form of a graph by constructing it from text data sources.

3 Methodology

Our method, illustrated in Figure 2, consists of two main phases: goal-oriented knowledge base construction and reasoning-aware inference. In the first phase, we construct a directed graph whose nodes represent goals such as "craft a wooden pickaxe" and edges represent subgoal relationships between goals. During inference, we use the constructed knowledge base to recursively search for step-by-step reasoning paths, which are followed to guide LLM prompting for the given task.

In the following subsections, we describe the two phases of our proposed method, using Minecraft as the testbed for illustration.

3.1 Goal Knowledge Base Construction

In the first phase of our method, we construct a directed graph $G = (V, E)$, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges from source to target nodes. Each node in the graph represents a goal, and each one is associated with a set of attributes consisting of its name, aliases, description, preconditions, and postconditions. The names of the nodes in our graph are goal-oriented phrases that succinctly describe the action involved and expected outcome of the goal, which is detailed further in the node's description. The preconditions are a list of prerequisites that are needed before the goal can be pursued, and postconditions explicitly describe the expected outcome of achieving the goal. Edges represent subgoal relationships between goals, and each edge is associated with a description to briefly explain how two goals are related.

In the case of Minecraft, to craft a wooden pickaxe, our constructed graph includes a set of goals such as "craft a wooden pickaxe" and "craft planks." Since both planks and sticks are required, the graph contains directed edges from the "craft a wooden pickaxe" node to the "craft sticks" and "craft planks" nodes, as shown on the right side of Figure 2. Furthermore, the goal "craft planks" depends on acquiring logs, introducing an additional subgoal relationship. In this manner, high-level goals are recursively decomposed into subgoals until they reach atomic operations. Preconditions, such as required tools and materials, are also encoded in the graph to represent dependencies for actions like crafting tools or mining ores.

To extract the goal-related information from large text-based data sources, we utilize LLMs. However, LLMs have a limit to the amount of tokens that they can process in one query, and therefore the source text needs to be split into smaller chunks that can fit an LLM's context size. For each chunk, we use an LLM to extract goals, their attributes, and subgoal relationships between goals. The prompt used for goal extraction is provided in Appendix B.

3.2 Goal Merge and Subgoal Derivation

After extracting goals and subgoals from all text chunks, two critical challenges must be addressed: (1) duplicate goals may be extracted from different chunks—how can they be effectively merged? and (2) a goal identified in one chunk may have a subgoal relationship with a goal from another chunk—how can the complete subgoal relations be derived?

The equivalence between new and existing goals is determined by the similarity between their name embeddings, and further verified using the preconditions and postconditions of each goal pair. To perform the aggregation process, we use an text embedding function $f : \mathcal{T} \to \mathbb{R}^d$ to map a given text $t \in \mathcal{T}$ to a d-dimensional vector. In particular, for each newly extracted goal, we retrieve the most similar goals from existing goal base by comparing the cosine similarity between their name embeddings. Then, we compare the pre- and postconditions of the pairs of new goals and their most similar existing goals. For a given new goal and its most similar goal, we sort each of their lists of preconditions and postconditions in alphabetical order, and then we calculate the cosine similarity between the embeddings of the pair after sorting. We define a threshold to binarize the similarity of text embedding.

If all pairs of corresponding conditions (preconditions and postconditions) have similarity scores above a threshold θ, the new goal is considered equivalent to an existing one. Otherwise, the new goal is added to the goal base as a distinct entry.

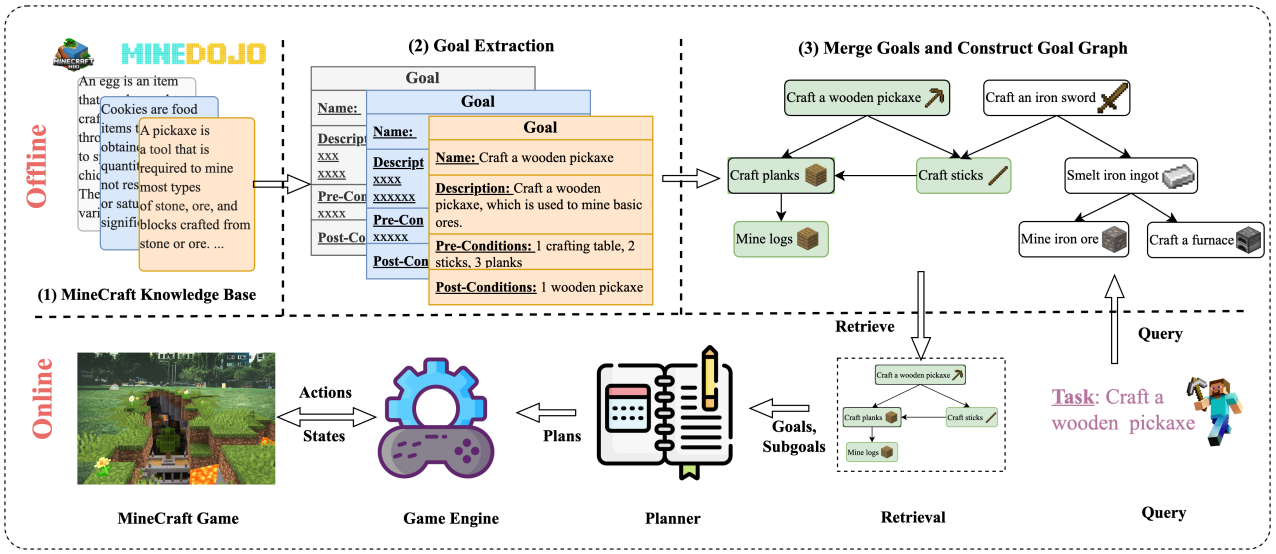

Figure 2: An overview of our proposed method GoG. First, we construct a knowledge base of goals from source text documents. Then, given a task instruction, we retrieve goal-oriented knowledge from the knowledge base to use for plan generation.

In cases where the condition similarity is high but the name embedding similarity is lower than θ, we treat the new goal as an alias and append its name to the existing goal's alias list. If both condition and name similarities are high, the goals are treated as identical.

Once we have added a newly extracted goal to the knowledge base, the next step is to determine whether it has any subgoal relationships with any existing goals in our knowledge base. To do this, we use a process similar to the goal equivalence comparison described previously, but instead we will match preconditions of the goal to the postconditions of existing goals, and vice versa. If there is a match, then we add a new edge to the knowledge base.

After deduplicating goals and completing subgoal relations across all chunks, we obtain a directed goal-oriented graph from the knowledge source, as shown in Figure 2.

3.3 Goal Selection and Planning

After constructing the knowledge base, we leverage it to extract goal-oriented knowledge for downstream tasks such as planning. The overall process is illustrated in Figure 3.

Given a task query, such as "craft a wooden sword", we retrieve the top-k goals from the our knowledge base according to the cosine similarity between the query and the goals' names in the knowledge base. If $k > 1$, we need to determine which goal is the best match for the query. To do this, we utilize LLMs to select the goal from the top-k retrieved options by giving it the goal names, descriptions, and postconditions. The prompt given to an LLM to select the best matching goal from the k candidates is provided in Appendix B.

After selecting the goal that matches the query, we retrieve all subgoals using depth-first search (DFS) starting from the selected goal. We avoid infinite loops caused by cyclical subgoal relations by not going deeper when a node that has already been visited is seen again. After retrieving subgoals, it may be the case that there are multiple ways to achieve the overall goal. In this case, a procedure to select the best set of subgoals is needed, which may depend on the use case scenario.

With the set of goals, their attributes, and edges indicating subgoal relationships, we end up with a goal tree. We traverse the tree to obtain a complete list of preconditions which, in Minecraft, is a list of materials and tools needed to achieve the overall goal. At last, we perform planning by providing this information to an LLM and prompting it to generate a plan. The planning prompt we use is provided in Appendix B.

3.4 GraphRAG vs GoG

Here we summarize the major differences between GraphRAG and our proposed GoG. GraphRAG organizes knowledge into fragmented, low-granularity entity–relation triples, which limits its ability to support coherent reasoning—much like trying to reconstruct evidence from shredded pieces

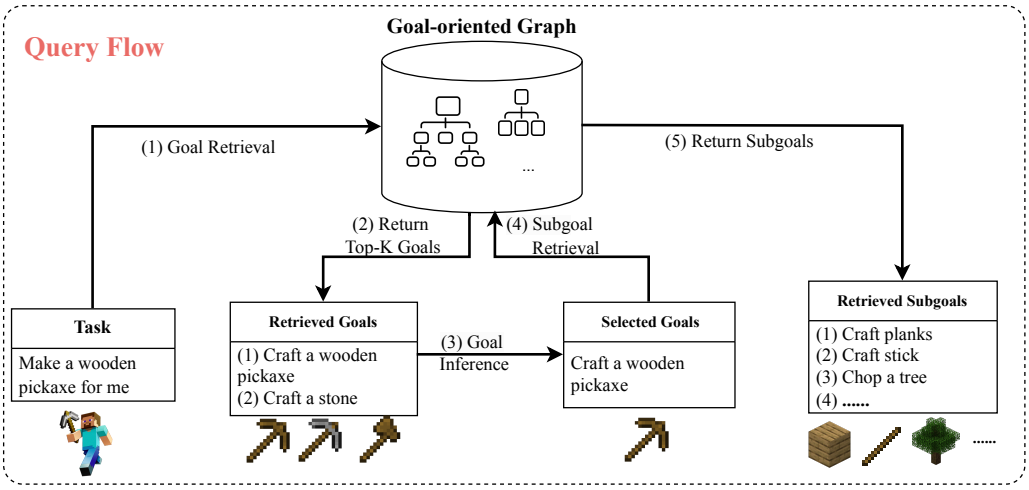

Figure 3: The query pipeline. For a given task, the top-k goals based on embedding similarity between the query and the goals' names are retrieved.

of paper. In contrast, we explicitly model abstract goals and subgoals as graph nodes, preserving the task hierarchy and enabling multi-step reasoning. For example, our GoG consists of 703 nodes and 1,653 edges, whereas the knowledge graph constructed using GraphRAG contains 12,388 nodes and 18,347 edges from the same source. This more compact and structured goal graph is better tailored for goal achievement and planning, as evidenced by the following results on the Minecraft testbed.

4 Experiments

4.1 Experimental Setup

In this work, we use the Minecraft environment and tasks provided by (Li et al., 2024) to develop and evaluate our method. There are 66 tasks that are categorized into 7 groups: wood, stone, iron, gold, diamond, redstone, and armor. The difficulty of the task groups from easiest to hardest is wood, stone, iron, gold, then diamond. The redstone and armor groups contain tasks of mixed difficulty ranging from iron to diamond. More difficult tasks require longer plans to be generated and require the agent to find rarer materials. The agent has a limited number of in-game steps, defined based on the task's group, to complete the task. The complete list of tasks and details can be found in Appendix C.

Baselines. We compare our proposed method **GoG** to three baselines:

Vanilla. This method uses few-shot examples of tasks and directly generate plans for given tasks from a given task instruction, without any further context information.

GraphRAG. We prompt GPT-4o mini to construct a knowledge graph from the Minecraft Wiki pages provided by MineDojo (Fan et al., 2022) and in-game recipes using the method proposed by Edge et al. (2025). During goal inference and planning, context information is retrieved using local search and inserted into the prompt given to the LLM. This method uses the same source as our GoG.

Hierarchical Knowledge Graph (HKG). This baseline constructs a knowledge graph from in-game recipe files as implemented by Li et al. (2024), where each node is an item and edges are dependencies between them. For a given task, the HKG matches the task to a node and extracts a list of materials and tools, which is given to an LLM to produce a plan. We exclude the multimodal memory module proposed by Li et al. (2024) from this baseline as we aim to compare the knowledge graphs used by each method.

Crafting recipes in the Minecraft Wiki are displayed as images. Therefore, we supplement the Wiki pages with recipes from the Minecraft game files to provide a text-based representation of the recipes. We remove pages that are unrelated to our experimental setting, such as those about real people related to the game, or game patch notes. Additionally, we filter pages by including those that have titles including the names of items contained in the in-game recipe files, or if the title is included in the name of the item. We only include text from the Wiki pages, leaving out images associated with the pages. The final set of documents used to construct our knowledge base consists of 514 Minecraft Wiki pages and 859 recipe files.

Models. We utilize three LLMs in our experiments: Llama 3.2-Vision 90b, Gemma 3 27b, and Qwen 2.5-VL 32b. We use multimodal LLMs because we provide game frames to the LLM when prompting them to perform goal inference and planning. We use a temperature of 0, context length of 32K, $\theta = 0.92$ with nomic-text-embed-v1.5 (Nussbaum et al., 2024) as the embedding model, and use $k = 3$ for both GraphRAG and our method.

Evaluation metrics. For our main experiments, we report the *success rates* of completing tasks and *the number of in-game steps* used by the agent over 30 runs of each task for each combination of baseline method and LLM. Higher success rates and lower required steps indicate better performance.

We conduct an ablation study in Section 4.4 and use metrics based on classical planning literature to assess the quality of the plans generated (Ghallab et al., 2004). We use goal satisfaction, soundness, completeness, and efficiency. The definitions of the metrics can be found in Appendix E. The metrics are calculated using information from the retrieved goal tree for a given task. Using these metrics instead of success rate and in-game steps allows us to focus on the differences in generated plans without needing to consider other factors that may cause the agent to fail, such as the stochasticity of environment.

After we generate a plan using the baselines and our method, we adopt STEVE-1 to convert text instructions into keyboard and mouse controls (Lifshitz et al., 2023). However, STEVE-1 is incapable of directly performing complex tasks that require multiple steps, hence the need for a planner that can decompose complex tasks into simpler ones. More details about the experimental setup can be found in Appendix D.

4.2 Main Results

The main results in Table 1 show that GoG performs much better than the baseline models in more complex task groups. For simpler task groups (e.g., wood and stone), all methods achieve comparable performance in terms of success rate and the average number of steps required to complete the tasks. However, for more challenging task groups—such as iron, gold, and armor—our GoG demonstrates a significant advantage over the baselines. For example, the success rate on the gold task is three times higher than that of HKG, and on the armor task, GoG outperforms HKG by 57.84% with Llama 3.2 Vision model. For the most difficult tasks, such as gold and diamond, all baseline methods consistently fail to achieve the goal within the maximum allowed in-game steps. In contrast, our method generally maintains a success rate above 50%, demonstrating its robustness in long-horizon planning scenarios. These successful results empirically demonstrate that our GoG enhances the LLM's reasoning capabilities by providing goal structures and material lists that fill knowledge gaps and support accurate planning.

A key observation from our experiments is a notable pitfall of GraphRAG: it sometimes performs even worse than the Vanilla baseline, despite having access to additional retrieved content. The retrieval process of GraphRAG returns many irrelevant but connected nodes from the 1-hop neighbourhood. For example, a general entity such as stone can be used to craft many different tools. When tasked with making a stone axe, the LLM often lacks the reasoning to select the appropriate objects from its neighbourhood and instead returns all connected entities. This results in highly noisy context, which can significantly degrade downstream task execution. As illustrated by our example of crafting a stone axe, the community reports of GraphRAG fail to translate into practical benefits—even with additional computational costs for clustering and summarization. This outcome echoes the old saying: reconstructing evidence from shredded pieces remains inherently difficult.

4.3 Hyperparameter Analysis of k

Here, we analyze the effect of k when retrieving candidates to match a given query to a goal in our GoG. In test tasks, all of the text instructions of the tasks are provided in a similar structure in the form of "<*verb*> <*item*>" with a limited number of verbs, such as "craft a wooden pickaxe". Therefore, in order to increase the diversity of the task instructions for this analysis, we use GPT 4o-mini to generate 10 rewordings for each of the 66 task instructions used in our experiments. Then, for each of the 660 generated instructions, we retrieve the top-k goals from our GoG, and use a vLLM to select the best match from the retrieved goals. In the case of $k = 0$, no retrieval is performed and the LLM is directly asked to determine the goal. For $k = 1$, the LLM is not required to perform any selection because there is only one option, hence the results for $k = 1$ are the same across all LLMs.

The results in Table 2 show the accuracies of various combinations of k and LLMs in determining

Table 1: The results of our main experiments. Bolded numbers are the best result of each task group.

Group	Metric	Llama 3.2 Vision				Gemma			
		Vanilla	GraphRAG	HKG	GoG	Vanilla	GraphRAG	HKG	GoG
Wood	SR↑	83.67	80.00	93.67	**95.67**	88.67	91.00	**95.67**	93.33
	AS↓	1268.69	1497.273	**1004.17**	1027.21	1264.81	1080.68	**989.55**	1044.69
Stone	SR ↑	57.41	37.77	46.67	**80.00**	47.41	47.78	**71.11**	69.63
	AS↓	3898.67	4610.74	4371.23	**3079.80**	4311.94	4296.61	3456.20	**3405.20**
Iron	SR ↑	19.79	19.17	54.38	**74.17**	15.94	11.78	57.01	**66.11**
	AS↓	21340.94	21137.67	14857.19	**10770.56**	21459.40	22523.79	14049.04	**12215.19**
Gold	SR↑	5.28	0.00	0.00	**70.00**	0.00	0.00	5.56	**72.22**
	AS↓	35321.25	∞	∞	**15886.11**	∞	∞	34448.38	**15336.51**
Diamond	SR↑	0.00	0.00	0.00	**66.11**	11.49	3.33	0.00	**31.39**
	AS↓	∞	∞	∞	**19717.08**	34601.33	35940.58	∞	**27361.79**
Redstone	SR↑	0.00	0.00	14.85	**49.44**	0.00	0.00	12.65	**47.46**
	AS↓	∞	∞	32188.80	**22309.15**	∞	∞	32733.51	**25633.46**
Armor	SR↑	25.83	26.15	34.44	**54.36**	23.59	6.15	35.38	**55.30**
	AS↓	28507.39	28335.48	25785.84	**20668.21**	29774.12	34177.48	25652.54	**20244.88**

- success rate (SR), average step (AS), ∞ (failed after reaching the max steps).

k	Gemma 3	Qwen 2.5 VL	Llama 3.2
0	.8742	.9258	.9515
1	.9712	.9712	.9712
2	.9846	.9879	**.9879**
3	**.9879**	.9894	.9864
4	.9818	.9862	.9863
5	**.9879**	**.9954**	.9848

Table 2: Accuracies for different values of k using various LLMs on goal inference for our retrieval method. The bolded numbers are the best results in each column.

the correct goal by checking that the postcondition of the selected goal matches the task. First, we observe that when $k > 1$, our goal inference clearly improves the retrieval of matching goals compared to the case of $k = 0$, where the LLM must rely solely on its own knowledge to infer the postconditions for the query task. Notably, Gemma 3 exhibited the lowest performance at $k = 0$, but its accuracy increased significantly—approaching that of other LLMs—when given access to our knowledge base. Second, we find that increasing k leads to only a marginal improvement in query accuracy, highlighting the robustness of our goal matching strategy, which remains effective even with a small number of retrieved goals.

When $k = 0$, corresponding to the baseline methods that rely solely on internal knowledge to infer the next steps, a significant question arises: why does performance vary considerably in our main experiments between GoG and baselines, even though retrieval performance remains relatively consistent across models? We attribute this performance gap to the recursive retrieval of subgoals in our algorithm, which explicitly constructs a reasoning path—something the baselines lack. However, there is still room for improvement in our approach, which motivates us to further explore how to enable deeper, more structured reasoning over tasks, rather than relying on superficial question analysis.

4.4 Plan Quality Ablation

Remember that two main components are extracted in our inference stage: the goal tree and the materials and tools list in Section 3.3. In this study, we analyze the effects of the two main components of the prompt for plan generation. We generate a plan for each of the 66 tasks for each LLM in the ablation study, and alter the information given in the prompt based on the components removed according to the corresponding variant. We experiment with 4 variants: full context information, goal tree information only, material and tool list only, and neither. The detailed evaluation is introduced in Appendix E.

The results are presented in Table 3. The material list refers to the set of materials required to achieve the target goal, and its inclusion significantly improves all four evaluation scores—particularly in terms of using the correct materials, in the correct order, and with greater efficiency—across all three LLMs. Overall, providing both goal information and the material list

LLM	Goal Info	Material List	Goal Satisfaction	Soundness	Completeness	Efficiency
Llama 3.2 Vision	✗	✗	.3231	.7901	.8152	.3589
	✓	✗	.1385	.3819	.3263	.1385
	✗	✓	.9538	.9850	.9891	.9538
	✓	✓	**1.000**	**1.000**	**1.000**	**1.000**
Qwen 2.5 VL	✗	✗	.2615	.7705	.7893	.2939
	✓	✗	.2154	.7622	.7511	.2301
	✗	✓	.9692	.9947	.9970	.9534
	✓	✓	**.9846**	**.9983**	**.9982**	**.9843**
Gemma 3	✗	✗	.3231	.8197	.8253	.3417
	✓	✗	.3692	.7804	.8083	.3438
	✗	✓	**.9538**	**.9905**	**.9940**	**.8769**
	✓	✓	.8308	.8704	.9088	.8292

Table 3: Plan quality ablation results, where bolded numbers show the highest scores on each metric for each LLM.

enables the LLM to generate higher-quality plans, as the incorporation of the goal tree—a topological structure that captures goal dependencies—further enhances multi-step reasoning. An exception is observed with Gemma 3, where adding the goal tree slightly reduces performance—likely due to difficulty handling hierarchical input or increased context length from overlapping information with the material list. We leave further investigation of this issue to future work.

4.5 Use-case Demonstration

To illustrate why our method performs better than the baselines, we compare the plans generated by our method and the baselines on the "craft a diamond axe" task using Llama 3.2-Vision in Appendix F.1. Two types of errors are observed from the plans generated by the baselines.

First, we observe hallucinations occurring in intermediate planning steps. For example, the plan generated by HKG includes the step "smelt diamond," in Table 5, which is invalid under Minecraft's rules—diamonds cannot be smelted. We hypothesize that this error stems from the LLM's lack of understanding of game mechanics. Although HKG supplies a list of materials and tools, this information alone is insufficient for constructing a valid plan. In contrast, our top-down approach provides structured goal decomposition, which better aligns with the procedural nature of in-game tasks.

Second, both GraphRAG and the vanilla baseline exhibit similar shortcomings in long-horizon planning. In particular, both generate plans that produce too few sticks, forcing the agent to replan and ultimately fail to complete the task, as shown in Tables 7, 8. This limitation arises from the LLM's inability to reason about quantities—successfully completing the task requires both knowledge of crafting recipes and the capacity to compute the required number of items based on those recipes.

Our GoG is constructed from practical Minecraft resources, such as the Minecraft Wiki and in-game crafting recipes, which provide rich information about both crafting procedures and the quantities of required materials. Unlike the baselines, GoG not only models goal-to-goal dependencies but also explicitly encodes preconditions and postconditions for each goal—for example, "1 crafting table, 2 sticks, and 3 planks" for crafting a wooden pickaxe. This structured representation significantly mitigates the two types of errors identified in our analysis, as shown in Table 6.

5 Conclusion

We proposed a novel framework for constructing Goal-Oriented Graphs (GoGs) from source documents to support planning tasks. We show that existing GraphRAG-based approaches are ill-suited for such domains due to the extremely fine granularity of entity–relation triple extraction, which makes it difficult to retrieve concise and complete reasoning paths. To address this, our GoGs represent goals at different abstraction levels as nodes, with edges indicating logical relationships between connected goals. This design streamlines retrieval by allowing all relevant subgoals of a target goal to be efficiently identified. Experiments on the Minecraft testbed demonstrate that our method significantly outperforms GraphRAG and its variants, especially on complex tasks that require long-horizon reasoning.

Limitations

In this work, we focused on constructing a goal-oriented graph that explicitly facilitates reasoning for complex tasks in Minecraft. While our approach achieves superior performance compared to baseline methods, several limitations remain.

First, our approach is currently tailored to the Minecraft environment, where goals and crafting logic are well-defined and richly documented. Applying GoG to less structured or poorly documented domains may require additional adaptation or domain-specific tuning.

Second, the construction of GoG relies on LLMs to extract goals, subgoals, and pre/postconditions. Errors in this extraction process—such as misinterpreting instructions or failing to capture implicit dependencies—can propagate through the graph and affect downstream performance. Future work could pay more attention to reduce the construction errors.

Lastly, as the number of knowledge sources increases, the number of goals and their interdependencies also grows. This can lead to increasingly complex graphs, potentially introducing retrieval inefficiencies or injecting noisy context during inference. We plan to explore the scalability of our approach in future work to ensure its effectiveness in larger and more diverse domains.

References

Thomas G Dietterich and 1 others. 1998. The maxq method for hierarchical reinforcement learning. In *ICML*, volume 98, pages 118–126.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From local to global: A graph rag approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

Kutluhan Erol, James Hendler, and Dana S Nau. 1994. Htn planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128.

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. In *Advances in Neural Information Processing Systems*, volume 35, pages 18343–18362. Curran Associates, Inc.

Xidong Feng, Yicheng Luo, Ziyan Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. 2023. Chessgpt: Bridging policy learning and language modeling. In *Advances in Neural Information Processing Systems*, volume 36, pages 7216–7262. Curran Associates, Inc.

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

Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: theory and practice*. Elsevier.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation.

Chenghao Huang, Yanbo Cao, Yinlong Wen, Tao Zhou, and Yanru Zhang. 2024. Pokergpt: An end-to-end lightweight solver for multi-player texas hold'em via large language model. *arXiv preprint arXiv:2401.06781*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. 2024. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. 2023. Steve-1: A generative model for text-to-behavior in minecraft. In *Advances in Neural Information Processing Systems*, volume 36, pages 69900–69929. Curran Associates, Inc.

Zach Nussbaum, John X Morris, Brandon Duderstadt, and Andriy Mulyar. 2024. Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*.

Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing Shao. 2024. Mp5: A multi-modal open-ended embodied system in minecraft via active perception. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16307–16316.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.

Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Junde Wu, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, and Vicente Grau. 2024. Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation. *arXiv preprint arXiv:2408.04187*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and Sergey Levine. 2024. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *Preprint*, arXiv:2305.17144.

A LLM in Goal Selection and Planning

As shown in Figure 4, we prompt the LLM to first select the most appropriate goal (e.g., "craft a wooden pickaxe") and retrieve relevant subgoals from the goal-oriented graph. We then prompt the LLM again to convert the selected goal and its subgoals into a coherent multi-step plan.

B Prompts

In this section, we provide various prompts used by our method.

C Experimental Tasks

Table 4 presents the list of tasks for each task group, along with the corresponding maximum number of in-game steps allowed. In total, there are 66 tasks, categorized into seven groups: wood, stone, iron, gold, diamond, redstone, and armor. As expected, more complex tasks require a greater number of steps to complete.

D Additional Experiment Details

In this section, we provide more details about ours experimental settings.

At the start of each experimental run, the agent starts with an empty inventory and is given an instruction as a string, such as "craft a wooden sword". Then, as described in the previous section, we retrieve the top-k goals from our knowledge base and use an LLM to perform goal inference by selecting a candidate from the top-k. Based on the selected goal, we retrieve all subgoals for the goal. Items in Minecraft often have multiple ways to be crafted, resulting in many possible ways to craft an item. However, almost all such variations are cyclical. For example, it is possible to obtain the "iron ingot" item from a "block of iron", however, a "block of iron" is obtained by combining "iron ingots", resulting in a cyclical relationship. Given that the agent in our experiments starts with an empty inventory, we exclude such paths from the result of DFS. This results in only 1 set of subgoals that achieve the given task.

After retrieving the subgoals, we parse the resulting goal tree, which consists of the goal, its subgoals, and all their attributes, to produce a list of items and materials needed to accomplish the task. Then, we provide the goal tree and list of items to an LLM to produce a plan to accomplish the task. The prompt used for planning can be found in Appendix B. The plan consists of a sequence of subtasks and expected items to obtain from each subtask. This plan is given to the agent, which then attempts to execute the plan in Minecraft.

During execution of the plan, if a step fails due to missing tool or materials, the Minecraft environment provides feedback to the agent about the missing item and quantity. This triggers the agent to replan, which follows a similar procedure as the original planning step at the beginning of the trajectory. For GoG, the list of materials and tools for the missing item is calculated from the goal tree, which is then given to the LLM to produce a plan to obtain missing materials. All baseline methods are able to perform replanning, with HKG also generating a list of materials using its knowledge graph. The generated replanning steps are then inserted into the original plan.

E Plan Quality Metrics

Here we provide the definitions of the metrics used in the plan quality ablation study.

Goal Satisfaction assesses overall plan quality by measuring whether a plan, when executed, can possibly achieve the overall task. This means that the plan must include steps to obtain all required tools and materials, and the ordering of the steps should be such that pre-conditions of each step are not violated. We assign a satisfaction score $g \in \{0, 1\}$ to a given plan, with 0 representing a plan that cannot achieve the given task.

Soundness checks that each step of a plan is formulated correctly and is executed with all preconditions being satisfied. For example, "mine a wooden sword" would be an invalid plan step. A plan is assigned a soundness score $s \in \{0, 1\}$, where 0 indicates that at least one step is invalid. Soundness is an upper bound on goal satisfaction; a plan cannot satisfy a goal if it is not sound. However, a sound plan does not necessarily mean that it will satisfy the goal.

Completeness measures the proportion of materials and tools needed to achieve a task are obtained by a plan. More formally, the score is assigned as:

$$c = \min\left(\frac{n_{\text{obtained}}}{n_{\text{needed}}}, 1\right). \quad (1)$$

This means that if a plan obtains more materials than necessary, it can still obtain a completeness score of 1.

Efficiency determines whether the plan contains more steps than required. A plan may have high

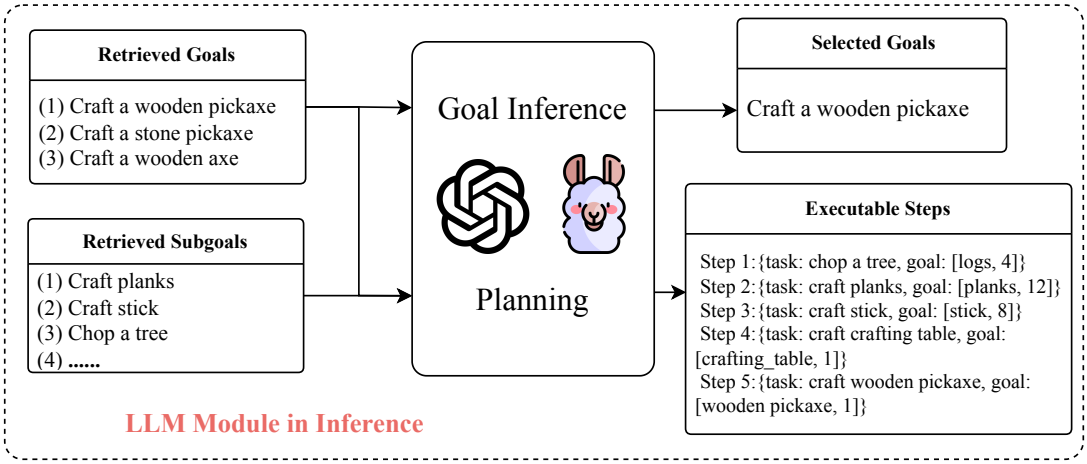

Figure 4: The LLM used during goal selection and planning stage.

Prompt for Extracting Goals and Subgoals, Part 1

-Goal-
Given a portion of a document and relevant in-game recipes about the game Minecraft, extract actionable in-game goals that a player can achieve. Use only the content from the given document and in-game recipe JSONs to construct goals and subgoals. Do not infer or add goals beyond what is explicitly described. Focus solely on the core Minecraft experience. Exclude any content related to Minecraft spinoff games (e.g., Minecraft Dungeons, Minecraft Legends).
-Steps-
1. Identify relevant goals that a player can achieve in the game. For each goal, extract the following attributes:
- name: Name of the goal. Use short, specific names in the form of "<action> <minecraft_item>", such as "craft planks", "mine cobblestone", or "smelt charcoal". For tools with different grades such as "wooden" or "stone", use "<action> <grade> <minecraft_tool>", such as "craft a wooden pickaxe" or "craft a stone sword".
- description: A concise explanation of what the goal entails.
- req_tools: Needed tools to complete the goal, as a JSON object where keys are Minecraft tools and values are 1. For tools with multiple grades (e.g. wooden or stone), specify the tool grade and only include the lowest grade needed. Crafting tables and furnaces are considered as tools, and their usage can be determined by document text, recipes, and summaries. Smelting using a furnace always requires "fuel" as a tool. Use "None" (just as a standalone string, not as a JSON object or set or list) if no tools are needed.
- req_materials: Needed materials to complete the goal, as a JSON object where keys are Minecraft items and values are needed quantities of that item. If no materials are needed, set this to "None" (just as a standalone string, not as a JSON object or set or list).
- postconditions: The resulting state or item after completing the goal, as a JSON object where the keys are Minecraft items and values are the quantity. If there are no post-conditions, set this to "None" (just as a standalone string, not as a JSON object or set or list).
Before writing each goal, generate reasoning as to where the information about the goal comes from. If it comes from a shaped crafting recipe, you must use the format as described above, otherwise write a brief sentence.
Format each goal as a tuple:
("goal"{tuple_delimiter}"<name>"{tuple_delimiter}"<description>"{tuple_delimiter}
"<req_tools>"{tuple_delimiter}"<req_materials>"{tuple_delimiter}"<postconditions>")
... (continued in Figure 6) ...

Figure 5: The prompt used to extract goals and subgoals from source texts to build our knowledge base.

Prompt for Extracting Goals and Subgoals, Part 2

2. From the goals identified in step 1, identify subgoals that are needed for the achievement of the goal. For every goal, establish subgoal relationships between the goal and associated subgoals for each required material and tool that must be obtained or crafted, as identified by <req_tools> or <req_materials>. For each goal-subgoal relationship, extract the following information:
- goal_name: Name of the higher-level goal, which must exist in the goals identified in step 1.
- subgoal_name: Name of the subgoal that is used by the goal.
- relationship_description: Explanation as to how and why the higher-level goal and the subgoal are related to each other.
Format each relationship as a tuple:
("subgoal"{tuple_delimiter}"<goal_name>"{tuple_delimiter}"<subgoal_name>"{tuple_delimiter}
"<relationship_description>")
3. Return a single list of tuples of all goals and subgoals as extracted from steps 1 and 2. Use **{record_delimiter}** as the list delimiter. If either tools or materials are ambiguous or missing, omit the goal. Do not repeat the same goal in the list.
4. When finished, output {completion_delimiter}.
Only output the list as instructed without any explanation, summary, or other text. If there is no relevant information in the document, just output {completion_delimiter}.
Here are some examples:
{examples}
######################
-Real Data-
######################
Document Text:
{input_text}
— End of Document —
Goals and Subgoals:

Figure 6: The second part of the prompt used to extract goals and subgoals from source texts to build our knowledge base.

Task Group	#Tasks	Task description	Max Steps
Wood	10	craft a wooden shovel, craft a wooden pickaxe, craft a wooden axe, craft a wooden hoe, craft a stick, craft a crafting table, craft a wooden sword, craft a chest, craft a bowl, craft a ladder	2400
Stone	9	craft a stone shovel, craft a stone pickaxe, craft a stone axe, craft a stone hoe, smelt a charcoal, craft a smoker, craft a stone sword, craft a furnace, craft a torch	6000
Iron	16	craft a iron shovel, craft a iron pickaxe, craft a iron axe, craft a iron hoe, craft a bucket, craft a hopper, craft a rail, craft a iron sword, craft a shears, craft a smithing table, craft a tripwire hook, craft a chain, craft an iron bars, craft an iron nugget, craft a blast furnace, craft a stonecutter	24000
Gold	6	craft a golden shovel, craft a golden pickaxe, craft a golden axe, craft a golden hoe, craft a golden sword, smelt and craft a gold ingot	36000
Diamond	6	craft a diamond shovel, craft a diamond pickaxe, craft a diamond axe, craft a diamond hoe, craft a diamond sword, craft a jukebox	36000
Redstone	6	craft a piston, craft a redstone torch, craft an activator rail, craft a compass, craft a dropper, craft a note block	36000
Armor	13	craft shield, craft iron chestplate, craft iron boots, craft iron leggings, craft iron helmet, craft diamond helmet, craft diamond chestplate, craft diamond leggings, craft diamond boots, craft golden helmet, craft golden leggings, craft golden boots, craft golden chestplate	36000

Table 4: Tasks used in our main experiments.

Prompt for Goal Inference

You are a MineCraft game expert and you can guide agents to complete complex tasks. For a given game screen, task, and context information, you need to complete "goal inference" and "visual inference".
The context information is a set of possible goals to choose from for "goal inference".
"goal inference": According to the task, you need to select the goal from given options that best matches the given query.
"visual inference": According to the game screen, you need to infer the following aspects: health bar, food bar, hotbar, environment.
{Examples}
Here is a game screen and task, you MUST respond in JSON format as shown in the example outputs WITHOUT further explanation, introduction, or extra text. Complete "goal inference" by setting it to the value of the "name" of the option that best matches the given task as shown in the example. Other fields should be completed based on the given game screen.
<task>: {task}
<context>:
{context}
Output:

Figure 7: The prompt used for goal inference for our proposed method. "Context" consists of the top-k retrieved goals from the knowledge base.

goal satisfaction, soundness, and completeness if it obtains much more materials than required, but this would not mean that it is a good plan. Therefore, efficiency checks that a plan consists of only necessary steps, measured as:

$$e = \begin{cases} \frac{s_{\text{minimal}}}{s_{\text{plan}}} & \text{if} s_{\text{plan}} \geq s_{\text{minimal}}, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

The efficiency of a plan is 0 if the number of steps in the plan s_{plan} is less than the minimum steps required s_{minimal} because that would mean that the plan would fail.

To assign scores using these metrics, we use the goal tree retrieved from our knowledge base for a given task. From the goal tree, we calculate the required list of items to complete the task, and the order that the items should be obtained used the preconditions of the goals. We use the goal names in the tree to determine correct wordings, as the goal names are in the form of "<action> <item>".

F Craft a Diamond Axe Case Study

F.1 Plans

In this section, we provide plans generated using Llama3.2-Vision 90b for the "craft a diamond axe" task.

F.2 Retrieved Content

In this section, we show the retrieved contents for various methods.

Prompt for Plan Generation

You are a MineCraft game expert and you can guide agents to complete complex tasks. For a given overall goal, game screen, hierarchy of goals, and list of needed materials, construct a ordered plan that completes the given task. The goal hierarchy is structured as a JSON object whose keys are names of goals, and values are information about the goal and its and subgoals. You will be given a list of tools and materials and amounts needed for you to obtain and craft to complete the overall goal. Based on the information from the goal hierarchy and the list of tools and materials, create a plan in JSON format as shown in the following example:
####################
-Example-
####################
{example}
####################
####################
-Real Task-
####################
<goal>
{goal}
<visual info>
{visual_info}
<goal hierarchy>
{goal_hierarchy}
<materials and tools>
{materials_and_tools}
<planning>
Complete <planning> for the given overall <goal> with valid JSON as instructed and in the format shown in the example. Use the information in the goal hierarchy, game screen, and list of tools and materials and their amounts to generate "task" and "goal" in each step of the plan. Use the same wording styles and patterns for the "task" in each step as shown in the example plan. Only output the plan as a valid JSON object with no additional text, introduction, or explanation. Do not use Markdown.

Figure 8: The prompt used for planning for our proposed method.

Step	Instruction	Target Item	Amount
0	chop a tree	logs	9
1	craft planks	planks	27
2	craft stick	stick	8
3	craft crafting table	crafting table	1
4	craft wooden pickaxe	wooden pickaxe	1
5	equip wooden pickaxe	wooden pickaxe	1
6	dig down and break down cobblestone	cobblestone	19
7	craft stone pickaxe	stone pickaxe	1
8	equip stone pickaxe	stone pickaxe	1
9	craft furnace	furnace	1
10	dig down and break down iron ore	iron ore	3
11	smelt iron ore	iron ingot	3
12	craft iron pickaxe	iron pickaxe	1
13	equip iron pickaxe	iron pickaxe	1
14	dig down and mine diamond	diamond ore	3
15	smelt diamond	diamond	3
16	craft diamond axe	diamond axe	1

Table 5: A plan generated using HKG for "craft a diamond axe". At step 15, the plan tries to "smelt diamond", which is unnecessary and invalid.

Step	Instruction	Target Item	Amount
0	chop a tree	logs	4
1	craft planks	planks	12
2	craft stick	stick	8
3	craft crafting table	crafting table	1
4	craft wooden pickaxe	wooden pickaxe	1
5	equip wooden pickaxe	wooden pickaxe	1
6	dig down and break down cobblestone	cobblestone	11
7	craft stone pickaxe	stone pickaxe	1
8	equip stone pickaxe	stone pickaxe	1
9	dig down and break down iron ore	iron ore	3
10	craft a furnace	furnace	1
11	smelt iron ingot	iron ingot	3
12	craft an iron pickaxe	iron pickaxe	1
13	equip iron pickaxe	iron pickaxe	1
14	dig down and break down diamond ore	diamond	3
15	craft diamond axe	diamond axe	1

Table 6: A plan generated using our proposed method for "craft a diamond axe", which leads to a successful completion of the task.

Step	Instruction	Target Item	Amount
0	chop a tree	logs	5
1	craft planks	planks	15
2	craft stick	stick	3
3	craft crafting table	crafting table	1
4	craft wooden pickaxe	wooden pickaxe	1
5	equip wooden pickaxe	wooden pickaxe	1
6	dig down and break down cobblestone	cobblestone	11
7	craft stone pickaxe	stone pickaxe	1
8	equip stone pickaxe	stone pickaxe	1
9	craft furnace	furnace	1
10	dig down and break down iron ore	iron ore	2
11	smelt iron ore	iron ingot	2
12	chop tree	logs	2
13	craft planks	planks	2
14	craft sticks	stick	2
15	craft wooden pickaxe	wooden pickaxe	1
16	mine iron ore	iron ore	1
17	smelt iron ore	iron ingot	1
18	craft iron pickaxe	iron pickaxe	1
19	equip iron pickaxe	iron pickaxe	1
20	dig down and break down diamond ore	diamond ore	3
21	craft diamond axe	diamond axe	1

Table 7: A plan generated using GraphRAG for “craft a diamond axe”. The plan contains replanning steps because the plan failed to obtain enough sticks in step 3.

Step	Instruction	Target Item	Amount
0	chop a tree	logs	5
1	craft planks	planks	16
2	craft stick	stick	4
3	craft crafting table	crafting table	1
4	craft wooden pickaxe	wooden pickaxe	1
5	equip wooden pickaxe	wooden pickaxe	1
6	dig down and break down cobblestone	cobblestone	11
7	craft stone pickaxe	stone pickaxe	1
8	equip stone pickaxe	stone pickaxe	1
9	craft furnace	furnace	1
10	dig down and break down iron ore	iron ore	2
11	smelt iron ore	iron ingot	2
12	chop tree	logs	2
13	craft planks	planks	2
14	craft sticks	sticks	2
15	craft stone pickaxe	stone pickaxe	1
16	mine iron ore	iron ore	1
17	smelt iron ore	iron ingot	1
18	craft iron pickaxe	iron pickaxe	1
19	equip iron pickaxe	iron pickaxe	1
20	dig down and break down diamond ore	diamond ore	3
21	craft diamond axe	diamond axe	1

Table 8: A plan generated using vanilla prompting for "craft a diamond axe". The plan contains replanning steps because the plan failed to obtain enough sticks in step 3.

1. logs: 4
2. planks: 12
3. stick: 8
4. crafting_table: 1
5. wooden_pickaxe: 1
6. cobblestone: 11
7. stone_pickaxe: 1
8. iron_ore: 3
9. furnace: 1
10. iron_ingot: 3
11. iron_pickaxe: 1
12. diamond: 3
13. diamond_axe: 1

Table 9: The generated material list for "craft a diamond axe" using our proposed method. The list is generated by parsing the goal tree and calculating all required materials.

Listing 1 A portion of the retrieved goal tree for "craft a diamond axe" using our proposed method.


```
{
    "craft a diamond axe": {
        "description": "Crafts a diamond axe, which is used for chopping wood.",
        "aliases": [],
        "tools": {
            "crafting_table": 1
        },
        "materials": {
            "stick": 2,
            "diamond": 3
        },
        "postconditions": {
            "diamond_axe": 1
        },
        "subgoals": [
            {
                "subgoal": "craft a crafting table",
                "relationship_description": "craft a crafting table is used by craft a diamond axe"
            },
            {
                "subgoal": "craft sticks",
                "relationship_description": "craft sticks is used by craft a diamond axe"
            },
            {
                "subgoal": "mine diamond ore",
                "relationship_description": "Diamonds are required to craft a diamond axe"
            }
        ]
    },
    "craft a crafting table": {
        "description": "Craft a crafting table, which is used to craft more complex items.",
        "aliases": [],
        "tools": "None",
        "materials": {
            "planks": 4
        },
        "postconditions": {
            "crafting_table": 1
        },
        "subgoals": [
            {
                "subgoal": "craft planks",
                "relationship_description": "craft planks is used by craft a crafting table"
            }
        ]
    },
    "craft planks": {
        "description": "Craft planks, a basic crafting material.",
        "aliases": [],
        "tools": "None",
        "materials": {
            "logs": 1
        },
        "postconditions": {
            "planks": 4
        },
        "subgoals": [
            {
                "subgoal": "mine log",
                "relationship_description": "mine log is used by craft planks"
            }
        ]
    },
    ... (continued) ...
```

...
Role of Toolsmith Villagers
Toolsmith villagers play a significant role in the community by trading various axes for emeralds. This relationship highlights the economic aspect of resource management, where players can acquire high-quality tools through trade. The presence of toolsmith villagers emphasizes the importance of community interactions and resource exchange in enhancing gameplay.
Utility of Different Axe Materials
Each type of axe, from wooden to netherite, offers varying levels of efficiency and durability, impacting how players approach resource gathering. For instance, diamond axes are known for their high efficiency, while wooden axes are less durable but easier to craft. Understanding the utility of different axe materials allows players to make informed decisions based on their resource availability and gameplay needs.
Sound and Interaction Events
The community includes various sound events associated with logs and axes, such as breaking and placing sounds. These auditory cues enhance the immersive experience of gameplay, providing feedback to players as they interact with the environment. The relationship between sound events and actions taken with axes and logs contributes to the overall engagement of players in the game.
Crafting Efficiency and Strategy
The crafting relationships among axes, logs, and other materials necessitate strategic planning for resource management. Players must consider the most efficient ways to gather materials and craft tools, balancing their immediate needs with long-term resource sustainability. This strategic element adds depth to gameplay, encouraging players to think critically about their actions.
Environmental Interactions
Axes and logs interact with various environmental elements, such as beehives and vines, showcasing their versatility in gameplay. For example, axes are the most efficient tool for breaking beehives, allowing players to harvest honey quickly. This interaction with the environment emphasizes the importance of axes in not only resource gathering but also in engaging with the game's ecosystem." 2, "# Axe Community and Its Variants
The community revolves around various types of axes, including the standard AXE and the high-quality DIAMOND AXE, along with their associated enchantments and crafting materials. These entities are interconnected through crafting relationships and usage in gameplay, highlighting their importance in resource gathering and combat.
Central Role of the AXE
The AXE serves as a fundamental tool in the community, primarily used for chopping wood and dealing damage to entities. Its versatility allows players to gather resources efficiently, making it a crucial item for crafting and building. The AXE can be crafted using various materials, including sticks and different types of ingots, which further emphasizes its importance in the game.
DIAMOND AXE as a High-Quality Variant
The DIAMOND AXE is a high-tier variant of the standard AXE, crafted from diamonds, which makes it the most durable and efficient option for chopping wood. Its crafting requires both diamonds and sticks, and it is highly sought after for its superior performance in resource gathering and combat scenarios. The DIAMOND AXE's significance is underscored by its ability to be enchanted, enhancing its capabilities even further.
...

Table 10: A portion of the community reports generated for "craft a diamond axe" using GraphRAG.

The formula to find the total lifetime damage is (durability ÷ 2) × damage per hit = minimum lifetime damage. The durability is halved because axes take double durability when used as a weapon. The formula also ignores enchantments and critical hits, and assumes each attack is performed at maximum charge.
Enchantments
An axe can receive the following enchantments:
— Table Start —
Headers: Name, Max Level, Method
Cells:
Fortune[note 1], III,
Silk Touch[note 1], I,
Efficiency, V,
Unbreaking, III,
Sharpness[note 2], V,
Smite[note 2], V,
Bane of Arthropods[note 2], V,
Fire Aspect[upcoming: JE Combat Tests], II,
Looting[upcoming: JE Combat Tests], III,
Knockback[upcoming: JE Combat Tests], II,
Cleaving[upcoming: JE Combat Tests][note 2], III,
Sweeping Edge[upcoming: JE Combat Tests][note 3], III,
Mending, I,
Curse of Vanishing, I,
— Table End —
Silk Touch and Fortune are mutually exclusive.
Sharpness, Smite, Bane of Arthropods, and Cleaving[upcoming: JE Combat Tests] are mutually exclusive.
Sweeping edge currently exists, but it can be used only for swords.
Fuel
Wooden axes can be used as a fuel in furnaces, smelting 1 item per axe.
Smelting ingredient
— Table Start —
Headers: Name, Ingredients, Smelting recipe, Iron Nugget or
Gold Nugget
Cells:
Iron Axe or
Golden Axe +
Any fuel, 0.1
— Table End —

Table 11: A portion of the retrieved documents for “craft a diamond axe” using GraphRAG.