

EXPERIENCE-BASED KNOWLEDGE CORRECTION FOR ROBUST PLANNING IN MINECRAFT

000
001
002
003
004
005 **Anonymous authors**
006 Paper under double-blind review
007
008
009
010

ABSTRACT

011 Large Language Model (LLM)-based planning has advanced embodied agents in
012 long-horizon environments such as Minecraft, where acquiring latent knowledge of
013 goal (or item) dependencies and feasible actions is critical. However, LLMs often
014 begin with flawed priors and fail to correct them through prompting, even with
015 feedback. We present XENON (eXpErience-based kNOwnledge correctioN), an
016 agent that algorithmically revises knowledge from experience, enabling robustness
017 to flawed priors and sparse binary feedback. XENON integrates two mechanisms:
018 Adaptive Dependency Graph, which corrects item dependencies using past suc-
019 cesses, and Failure-aware Action Memory, which corrects action knowledge using
020 past failures. Together, these components allow XENON to acquire complex
021 dependencies despite limited guidance. Experiments across multiple Minecraft
022 benchmarks show that XENON outperforms prior agents in both knowledge learn-
023 ing and long-horizon planning. Remarkably, with only a 7B open-weight LLM,
024 XENON surpasses agents that rely on much larger proprietary models.
025
026

1 INTRODUCTION

027 Large Language Model (LLM)-based planning has advanced in developing embodied AI agents that
028 tackle long-horizon goals in complex, real-world-like environments (Szot et al., 2021; Fan et al.,
029 2022). Among such environments, Minecraft has emerged as a representative testbed for evaluating
030 planning capability that captures the complexity of such environments (Wang et al., 2023b;c; Zhu
031 et al., 2023; Yuan et al., 2023; Feng et al., 2024; Li et al., 2024b). Success in these environments often
032 depends on agents acquiring planning knowledge, including the dependencies among goal items and
033 the valid actions needed to obtain them. For instance, to obtain an iron nugget, an agent should
034 first possess an iron ingot, which can only be obtained by the action *smelt*.
035

036 However, LLMs often begin with flawed priors about these dependencies and actions. This issue is
037 indeed critical, since a lack of knowledge for a single goal can invalidate all subsequent plans that
038 depend on it (Guss et al., 2019; Lin et al., 2021; Mao et al., 2022). We find several failure cases
039 stemming from these flawed priors, a problem that is particularly pronounced for the lightweight
040 LLMs suitable for practical embodied agents. First, an LLM often fails to predict planning knowledge
041 accurately enough to generate a successful plan (Figure 1b), resulting in a complete halt in progress
042 toward more challenging goals. Second, an LLM cannot robustly correct its flawed knowledge, even
043 when prompted to self-correct with failure feedback (Shinn et al., 2023; Chen et al., 2024), often
044 repeating the same errors (Figures 1c and 1d). To improve self-correction, one can employ more
045 advanced techniques that leverage detailed reasons for failure (Zhang et al., 2024; Wang et al., 2023a).
046 Nevertheless, LLMs often stubbornly adhere to their erroneous parametric knowledge (i.e. **knowledge**
047 **implicitly stored in model parameters**), as evidenced by Stechly et al. (2024) and Du et al. (2024).
048

049 In response, we propose XENON (eXpErience-based kNOwnledge correctioN), an agent that robustly
050 learns planning knowledge from only binary success/failure feedback. To this end, instead of relying
051 on an LLM for correction, XENON algorithmically and directly revises its external knowledge
052 memory using its own experience, which in turn guides its planning. XENON learns this planning
053 knowledge through two synergistic components. The first component, Adaptive Dependency Graph
(ADG), revises flawed dependency knowledge by leveraging successful experiences to propose
plausible new required items. The second component, Failure-aware Action Memory (FAM), builds
and corrects its action knowledge by exploring actions upon failures. In the challenging yet practical
setting of using only binary feedbacks, FAM enables XENON to disambiguate the cause of a failure,
054

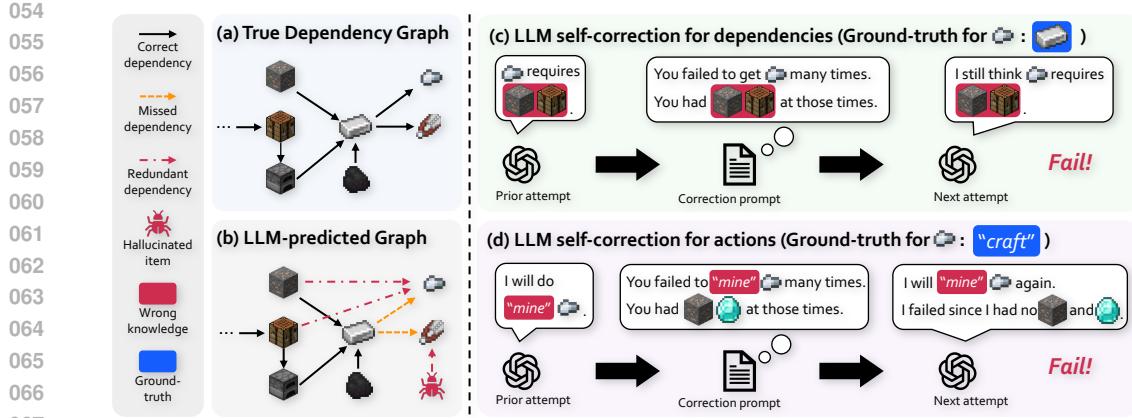


Figure 1: **An LLM exhibits flawed planning knowledge and fails at self-correction.** (b) The dependency graph predicted by Qwen2.5-VL-7B (Bai et al., 2025) contains multiple errors (e.g., missed dependencies, hallucinated items) compared to (a) the ground truth. (c, d) The LLM fails to correct its flawed knowledge about dependencies and actions from failure feedbacks, often repeating the same errors. See Appendix B for the full prompts and LLM’s self-correction examples.

distinguishing between flawed dependency knowledge and invalid actions, which in turn triggers a revision in ADG for the former.

Extensive experiments in three Minecraft testbeds show that XENON excels at both knowledge acquisition and planning. XENON outperforms prior agents in learning knowledge, showing unique robustness to LLM hallucinations and modified ground-truth environmental rules. Furthermore, with only a 7B LLM, XENON significantly outperforms prior agents that rely on much larger proprietary models like GPT-4 in solving diverse long-horizon goals. These results suggest that robust algorithmic knowledge management can be a promising direction for developing practical embodied agents with lightweight LLMs (Belcak et al., 2025).

Our contributions are as follows. First, we propose XENON, an LLM-based agent that robustly learns planning knowledge from experience via algorithmic knowledge correction, instead of relying on the LLM to self-correct its own knowledge. We realize this idea through two synergistic mechanisms that explicitly store planning knowledge and correct it: Adaptive Dependency Graph (ADG) for correcting dependency knowledge based on successes, and Failure-aware Action Memory (FAM) for correcting action knowledge and disambiguating failure causes. Second, extensive experiments demonstrate that XENON significantly outperforms prior state-of-the-art agents in both knowledge learning and long-horizon goal planning in Minecraft.

2 RELATED WORK

2.1 LLM-BASED PLANNING IN MINECRAFT

Prior work has often addressed LLMs’ flawed planning knowledge in Minecraft using impractical methods. For example, such methods typically involve directly injecting knowledge through LLM fine-tuning (Zhao et al., 2023; Feng et al., 2024; Liu et al., 2025; Qin et al., 2024) or relying on curated expert data (Wang et al., 2023c; Zhu et al., 2023; Wang et al., 2023a).

Another line of work attempts to learn planning knowledge via interaction, by storing the experience of obtaining goal items in an external knowledge memory. However, these approaches are often limited by unrealistic assumptions or lack robust mechanisms to correct the LLM’s flawed prior knowledge. For example, ADAM and Optimus-1 artificially simplify the challenge of predicting and learning dependencies via shortcuts like pre-supplied items, while also relying on expert data such as learning curriculum (Yu & Lu, 2024) or Minecraft wiki (Li et al., 2024b). They also lack a robust way to correct wrong action choices in a plan: ADAM has none, and Optimus-1 relies on unreliable LLM self-correction. Our most similar work, DECKARD (Nottingham et al., 2023), uses an LLM to predict item dependencies but does not revise its predictions for items that repeatedly fail, and when a plan fails, it cannot disambiguate whether the failure is due to incorrect dependencies or incorrect

108 actions. In contrast, our work tackles the more practical challenge of learning planning knowledge
 109 and correcting flawed priors from only binary success/failure feedback.
 110

111 **2.2 LLM-BASED SELF-CORRECTION**
 112

113 LLM self-correction, i.e., having an LLM correct its own outputs, is a promising approach to
 114 overcome the limitations of flawed parametric knowledge. However, for complex tasks like planning,
 115 LLMs struggle to identify and correct their own errors without external feedback (Huang et al.,
 116 2024; Tyen et al., 2024). To improve self-correction, prior works fine-tune LLMs (Yang et al.,
 117 2025) or prompt LLMs to correct themselves using environmental feedback (Shinn et al., 2023) and
 118 tool-execution results (Gou et al., 2024). While we also use binary success/failure feedbacks, we
 119 directly correct the agent’s knowledge in external memory by leveraging experience, rather than
 120 fine-tuning the LLM or prompting it to self-correct.
 121

122 **3 PRELIMINARIES**
 123

124 We aim to develop an agent capable of solving long-horizon goals by learning planning knowledge
 125 from experience. As a representative environment which necessitates accurate planning knowledge,
 126 we consider Minecraft as our testbed. Minecraft is characterized by strict dependencies among game
 127 items (Guss et al., 2019; Fan et al., 2022), which can be formally represented as a directed acyclic
 128 graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$, where \mathcal{V}^* is the set of all items and each edge $(u, q, v) \in \mathcal{E}^*$ indicates that q
 129 quantities of an item u are required to obtain an item v .¹ A goal is to obtain an item $g \in \mathcal{V}^*$. To
 130 obtain g , an agent must possess all of its prerequisites as defined by \mathcal{G}^* in its inventory, and perform
 131 the valid high-level action in $\mathcal{A} = \{\text{“mine”}, \text{“craft”}, \text{“smelt”}\}$.
 132

133 **Framework: Hierarchical agent with graph-augmented planning.** We employ a hierarchical
 134 agent with an LLM planner and a low-level controller, adopting a graph-augmented planning strat-
 135 egy (Li et al., 2024b; Nottingham et al., 2023). In this strategy, agent maintains its knowledge graph
 136 \mathcal{G} and plans with \mathcal{G} to decompose a goal g into subgoals in two stages. First, the agent identifies pre-
 137 requisite items it does not possess by traversing $\hat{\mathcal{G}}$ backward from g to nodes with no incoming edges
 138 (i.e., basic items with no known requirements), and aggregates them into a list of (quantity, item)
 139 tuples, $((q_1, u_1), \dots, (q_{L_g}, u_{L_g})) = (1, g)$. Second, the planner LLM converts this list into executable
 140 language subgoals $\{(a_l, q_l, u_l)\}_{l=1}^{L_g}$, where it takes each u_l as input and outputs a high-level action
 141 a_l to obtain u_l . Then the controller executes each subgoal, i.e., it takes each language subgoal as
 142 input and outputs a sequence of low-level actions in the environment to achieve it. After each subgoal
 143 execution, the agent receives only binary success/failure feedback.

144 **Problem formulation: Dependency and action learning.** To plan correctly, the agent must acquire
 145 knowledge of the true dependency graph \mathcal{G}^* . However, $\hat{\mathcal{G}}$ is latent, making it necessary for the agent to
 146 learn this structure from experience. We model this as revising a learned graph, $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$, where $\hat{\mathcal{V}}$
 147 contains known items and $\hat{\mathcal{E}}$ represents the agent’s current belief about item dependencies. Following
 148 Nottingham et al. (2023), whenever the agent obtains a new item v , it identifies the *experienced*
 149 *requirement set* $\mathcal{R}_{\text{exp}}(v)$, the set of (item, quantity) pairs consumed during this item acquisition. The
 150 agent then updates $\hat{\mathcal{G}}$ by replacing all existing incoming edges to v with the newly observed $\mathcal{R}_{\text{exp}}(v)$.
 151 The detailed update procedure is in Appendix C.

152 We aim to maximize the accuracy of learned graph $\hat{\mathcal{G}}$ against true graph \mathcal{G}^* . We define this accuracy
 153 $N_{\text{true}}(\hat{\mathcal{G}})$ as the number of items whose incoming edges are identical in $\hat{\mathcal{G}}$ and \mathcal{G}^* , i.e.,
 154

$$N_{\text{true}}(\hat{\mathcal{G}}) := \sum_{v \in \mathcal{V}^*} \mathbb{I}(\mathcal{R}(v, \hat{\mathcal{G}}) = \mathcal{R}(v, \mathcal{G}^*)) , \quad (1)$$

155 where the dependency set, $\mathcal{R}(v, \mathcal{G})$, denotes the set of all incoming edges to the item v in the graph \mathcal{G} .
 156

157
 158 ¹In our actual implementation, each edge also stores the resulting item quantity, but we omit it from the
 159 notation for presentation simplicity, since most edges have resulting item quantity 1 and this multiplicity is not
 160 essential for learning item dependencies.
 161

162 **4 METHODS**
 163

164 XENON is an LLM-based agent with two core components: Adaptive Dependency Graph (ADG)
 165 and Failure-aware Action Memory (FAM), as shown in Figure 3. ADG manages dependency
 166 knowledge, while FAM manages action knowledge. The agent learns this knowledge in a loop
 167 that starts by selecting an unobtained item as an exploratory goal (detailed in Appendix G). Once
 168 an item goal g is selected, ADG, our learned dependency graph $\hat{\mathcal{G}}$, traverses itself to construct
 169 $((q_1, u_1), \dots, (q_{L_g}, u_{L_g}) = (1, g))$. For each u_l in this list, FAM either reuses a previously successful
 170 action for u_l or, if none exists, the planner LLM selects a high-level action $a_l \in \mathcal{A}$ given u_l and action
 171 histories from FAM. The resulting actions form language subgoals $\{(a_l, q_l, u_l)\}_{l=1}^{L_g}$. The controller
 172 then takes each subgoal as input, executes a sequence of low-level actions to achieve it, and returns
 173 binary success/failure feedback, which is used to update both ADG and FAM. The full procedure is
 174 outlined in Algorithm 1 in Appendix D. We next detail each component, beginning with ADG.
 175

176 **4.1 ADAPTIVE DEPENDENCY GRAPH (ADG)**
 177

178 **Dependency graph initialization.** To make the most of
 179 the LLM’s prior knowledge, albeit incomplete, we ini-
 180 tialize the learned dependency graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ us-
 181 ing an LLM. We follow the initialization process of
 182 **DECKARD** (Nottingham et al., 2023), which consists
 183 of two steps. First, $\hat{\mathcal{V}}$ is assigned \mathcal{V}_0 , which is the set of
 184 goal items whose dependencies must be learned, and $\hat{\mathcal{E}}$
 185 is assigned \emptyset . Second, for each item v in $\hat{\mathcal{V}}$, the LLM
 186 is prompted to predict its requirement set (i.e. incoming
 187 edges of v), aggregating them to construct the initial graph.
 188

189 However, those LLM-predicted requirement sets often
 190 include items not present in the initial set \mathcal{V}_0 , which is a
 191 phenomenon overlooked by DECKARD. Since \mathcal{V}_0 may
 192 be an incomplete subset of all possible game items \mathcal{V}^* ,
 193 we cannot determine whether such items are genuine required items or hallucinated items which
 194 do not exist in the environment. To address this, we provisionally accept all LLM requirement set
 195 predictions. We iteratively expand the graph by adding any newly mentioned item to $\hat{\mathcal{V}}$ and, in
 196 turn, querying the LLM for its own requirement set. This expansion continues until a requirement
 197 set has been predicted for every item in $\hat{\mathcal{V}}$. **Since we assume that the true graph \mathcal{G}^* is a DAG, we**
 198 **algorithmically prevent cycles in $\hat{\mathcal{G}}$; see Appendix E.2 for the cycle-check procedure.** The quality of
 199 this initial LLM-predicted graph is analyzed in detail in Appendix K.1.

200 **Dependency graph revision.** Correcting the agent’s flawed dependency knowledge involves two
 201 challenges: (1) detecting and handling hallucinated items from the graph initialization, and (2)
 202 proposing a new requirement set. Simply prompting an LLM for corrections is ineffective, as it
 203 often predicts a new, flawed requirement set, as shown in Figures 1c and 1d. Therefore, we revise $\hat{\mathcal{G}}$
 204 algorithmically using the agent’s experiences, without relying on the LLM.
 205

206 To implement this, we introduce a dependency revision procedure called `RevisionByAnalogy`
 207 and a revision count $C(v)$ for each item $v \in \hat{\mathcal{V}}$. This procedure outputs a revised graph by taking item
 208 v whose dependency needs to be revised, its revision count $C(v)$, and the current graph $\hat{\mathcal{G}}$ as inputs,
 209 leveraging the required items of previously obtained items. When a revision for an item v is triggered
 210 by FAM (Section 4.2), the procedure first discards v ’s existing requirement set (i.e., $\mathcal{R}(v, \hat{\mathcal{G}}) \leftarrow \emptyset$).
 211 It increments the revision count $C(v)$ for v . Based on whether $C(v)$ exceeds a hyperparameter c_0 ,
 212 `RevisionByAnalogy` proceeds with one of the following two cases:
 213

- 214 • **Case 1: Handling potentially hallucinated items ($C(v) > c_0$).** If an item v remains unobtainable
 215 after excessive revisions, the procedure flags it as *inadmissible* to signify that it may be a halluci-
 216 nated item. This reveals a critical problem: if v is indeed a hallucinated item, any of its descendants
 217 in $\hat{\mathcal{G}}$ become permanently unobtainable. To enable XENON to try these descendant items through
 218 alternative paths, we recursively call `RevisionByAnalogy` for all of v ’s descendants in $\hat{\mathcal{G}}$,
 219

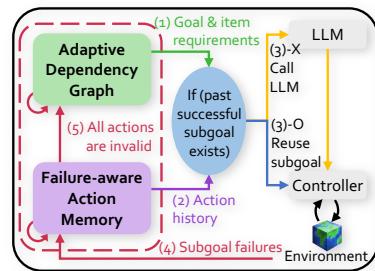


Figure 3: Overview. XENON updates Adaptive Dependency Graph and Failure-aware Action Memory with environmental experiences.

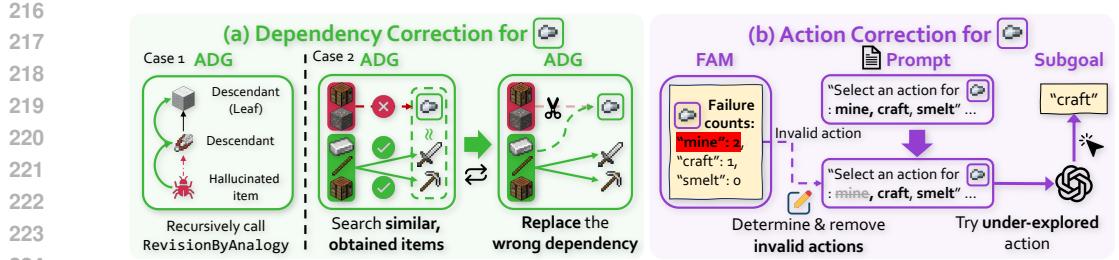


Figure 4: XENON’s algorithmic knowledge correction. (a) Dependency Correction via RevisionByAnalogy. Case 1: For an inadmissible item (e.g., a hallucinated item), its descendants are recursively revised to remove the flawed dependency. Case 2: A flawed requirement set is revised by referencing similar, obtained items. (b) Action Correction via FAM. FAM prunes invalid actions from the LLM’s prompt based on failures, guiding it to select an under-explored action.

removing their dependency on the inadmissible item v (Figure 4a, Case 1). Finally, to account for cases where v may be a genuine item that is simply difficult to obtain, its requirement set $\mathcal{R}(v, \hat{\mathcal{G}})$ is reset to a general set of all resource items (i.e. items previously consumed for crafting other items), each with a quantity of hyperparameter α_i .

• **Case 2: Plausible revision for less-tried items** ($C(v) \leq c_0$). The item v ’s requirement set, $\mathcal{R}(v, \hat{\mathcal{G}})$, is revised to determine both a plausible set of new items and their quantities. First, for plausible required items, we use an idea that similar goals often share similar preconditions (Yoon et al., 2024). Therefore, we set the new required items referencing the required items of the **top- K** similar, successfully obtained items (Figure 4a, Case 2). **We compute this item similarity as the cosine similarity between the Sentence-BERT (Reimers & Gurevych, 2019) embeddings of item names.** Second, to determine their quantities, the agent should address the trade-off between sufficient amounts to avoid failures and an imperfect controller’s difficulty in acquiring them. Therefore, the quantities of those new required items are determined by gradually scaling with the revision count, $\alpha_s C(v)$.

Here, the hyperparameter c_0 serves as the revision count threshold for flagging an item as inadmissible. α_i and α_s control the quantity of each required item for inadmissible items (Case 1), and for less-tried items (Case 2), respectively, to maintain robustness when dealing with an imperfect controller. **K determines the number of similar, successfully obtained items to reference for (Case 2).** Detailed pseudocode of RevisionByAnalogy is in Appendix E.3, **Algorithm 3**.

4.2 FAILURE-AWARE ACTION MEMORY (FAM)

FAM is designed to address two challenges of learning only from binary success/failure feedback: (1) discovering valid high-level actions for each item, and (2) disambiguating the cause of persistent failures between invalid actions and flawed dependency knowledge. This section first describes FAM’s core mechanism, and then details how it addresses each of these challenges in turn.

Core mechanism: empirical action classification. FAM classifies actions as either *empirically valid* or *empirically invalid* for each item, based on their history of past subgoal outcomes. Specifically, for each item $v \in \hat{\mathcal{V}}$ and action $a \in \mathcal{A}$, FAM maintains the number of successful and failed outcomes, denoted as $S(a, v)$ and $F(a, v)$ respectively. Based on these counts, an action a is classified as *empirically invalid* for v if it has failed repeatedly, (i.e., $F(a, v) \geq S(a, v) + x_0$); otherwise, it is classified as *empirically valid* if it has succeeded at least once (i.e., $S(a, v) > 0$ and $S(a, v) > F(a, v) - x_0$). The hyperparameter x_0 controls the tolerance for this classification, accounting for the possibility that an imperfect controller might fail even with an indeed valid action.

Addressing challenge 1: discovering valid actions. FAM helps XENON discover valid actions by avoiding repeatedly failed actions when making a subgoal $sg_l = (a_l, q_l, u_l)$. Only when FAM has no empirically valid action for u_l , XENON queries the LLM to select an under-explored action for constructing sg_l . **To accelerate this search for a valid action, we query the LLM with (i) the current subgoal item u_l , (ii) empirically valid actions for top- K similar items successfully obtained and stored in FAM (using Sentence-BERT similarity as in Section 4.1), and (iii) candidate actions for**

270 u_l that remain after removing all empirically invalid actions from \mathcal{A} (Figure 4b). We prune action
 271 candidates rather than include the full failure history because LLMs struggle to effectively utilize long
 272 prompts (Li et al., 2024a; Liu et al., 2024). If FAM already has an empirically valid one, XENON
 273 reuses it to make sg_l without using LLM. Detailed procedures and prompts are in Appendix F.
 274

275 **Addressing challenge 2: disambiguating failure causes.** By ensuring systematic action exploration,
 276 FAM allows XENON to determine that persistent subgoal failures stem from flawed dependency
 277 knowledge rather than from the actions. Specifically, once FAM classifies all actions in \mathcal{A} for an item
 278 as empirically invalid, XENON concludes that the error lies within ADG and triggers its revision.
 279 Subsequently, XENON resets the item’s history in FAM to allow for a fresh exploration of actions
 280 with the revised ADG.
 281

282 4.3 ADDITIONAL TECHNIQUE: CONTEXT-AWARE REPROMPTING (CRe) FOR CONTROLLER

283 In real-world-like environments, an imperfect low-level controller can stall (e.g., in deep water). To
 284 address this, XENON employs context-aware reprompting (CRe) technique, which uses an LLM
 285 that takes the current image observation and the controller’s language subgoal for item acquisition
 286 as input, and outputs whether to replace that subgoal and, if so, a new temporary subgoal to escape
 287 the stalled state (e.g., “get out of the water”). Our CRe is adapted from Optimus-1 (Li et al., 2024b)
 288 to be suitable for smaller LLMs, with two key differences: (1) a two-stage reasoning process that
 289 separates image observation captioning from the subsequent text-only decision to replace the subgoal
 290 or not, and (2) a conditional trigger that activates only when the subgoal for item acquisition makes
 291 no progress, rather than at fixed intervals. See Appendix H for details.
 292

293 5 EXPERIMENTS

295 5.1 SETUPS

296 **Environments.** We conduct experiments in three Minecraft environments, which we separate into
 297 two categories based on their controller capacity. First, as realistic, visually-rich embodied AI
 298 environments, we use MineRL (Guss et al., 2019) and Mineflayer (PrismarineJS, 2023) with imperfect
 299 low-level controllers: STEVE-1 (Lifshitz et al., 2023) in MineRL and hand-crafted codes (Yu & Lu,
 300 2024) in Mineflayer. Second, we use MC-TextWorld (Zheng et al., 2025) as a controlled testbed
 301 with a perfect controller. Each experiment in this environment is repeated over 15 runs; in our
 302 results, we report the mean and standard deviation, omitting the latter when it is negligible. In all
 303 environments, the agent starts with an empty inventory. Further details on environments are provided
 304 in Appendix J. Additional experiments in a household task planning domain other than Minecraft are
 305 reported in Appendix A, where XENON also exhibits robust performance.
 306

307 **Evaluation metrics.** For both dependency
 308 learning and planning evaluations, we utilize
 309 the 67 goals from 7 groups proposed in the long-
 310 horizon task benchmark (Li et al., 2024b). To
 311 evaluate dependency learning with an intuitive
 312 performance score between 0 and 1, we report
 313 $N_{\text{true}}(\hat{\mathcal{G}})/67$, where $N_{\text{true}}(\hat{\mathcal{G}})$ is defined in
 314 Equation (1). We refer to this normalized score as
 315 Experienced Graph Accuracy (EGA). To eval-
 316 uate planning performance, we follow the bench-
 317 mark setting (Li et al., 2024b): at the beginning
 318 of each episode, a goal item is specified exter-
 319 nally for the agent, and we measure the average success rate (SR) of obtaining this goal item in
 320 MineRL. See Table 10 for the full list of goals.
 321

322 **Implementation details.** For the planner, we use Qwen2.5-VL-7B (Bai et al., 2025). The learned
 323 dependency graph is initialized with human-written plans for three goals (“craft an iron sword ”,
 324 “craft a golden sword ”, “mine a diamond ”), providing minimal knowledge; the agent must learn
 325 dependencies for over 80% of goal items through experience. We employ CRe only for long-horizon

Table 1: Comparison of knowledge correction mechanisms across agents. ○: Our proposed mech-
 anism (XENON), △: LLM self-correction, ✗: No correction, -: Not applicable.

Agent	Dependency Correction	Action Correction
XENON	○	○
SC	△	△
DECKARD	✗	✗
ADAM	-	✗
RAND	✗	✗

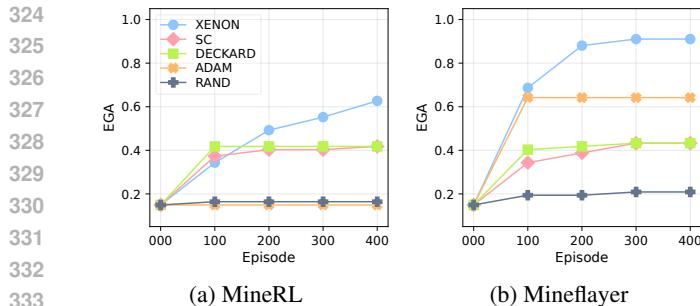


Figure 5: **Robustness against flawed prior knowledge.** EGA over 400 episodes in (a) MineRL and (b) Mineflayer. XENON consistently outperforms the baselines.

Table 2: **Robustness to LLM hallucinations.** The number of correctly learned dependencies of items that are descendants of a hallucinated item in the initial LLM-predicted dependency graph (out of 12).

Agent	Learned descendants of hallucinated items
XENON	0.33
SC	0
ADAM	0
DECKARD	0
RAND	0

goal planning in MineRL. All hyperparameters are kept consistent across experiments. Further details on hyperparameters and human-written plans are in Appendix I.

Baselines. As no prior work learns dependencies in our exact setting, we adapt four baselines, whose knowledge correction mechanisms are summarized in Table 1. For dependency knowledge, (1) **LLM Self-Correction (SC)** starts with an LLM-predicted dependency graph and prompts the LLM to revise it upon failures; (2) **DECKARD** (Nottingham et al., 2023) also relies on an LLM-predicted graph but with no correction mechanism; (3) **ADAM** (Yu & Lu, 2024) assumes that any goal item requires all previously used resource items, each in a sufficient quantity; and (4) **RAND**, the simplest baseline, uses a static graph similar to DECKARD. Regarding action knowledge, all baselines except for RAND store successful actions. However, only the SC baseline attempts to correct its flawed knowledge upon failures. The SC prompts the LLM to revise both its dependency and action knowledge using previous LLM predictions and interaction trajectories, as done in many self-correction methods (Shinn et al., 2023; Stechly et al., 2024). See Appendix B for the prompts of SC and Appendix J.1 for detailed descriptions of these baselines. To evaluate planning on diverse long-horizon goals, we further compare XENON with recent planning agents that are provided with oracle dependencies: DEPS Wang et al. (2023b), Jarvis-1 Wang et al. (2023c), Optimus-1 Li et al. (2024b), and Optimus-2 Li et al. (2025b).

5.2 ROBUST DEPENDENCY LEARNING AGAINST FLAWED PRIOR KNOWLEDGE

XENON demonstrates robust dependency learning from flawed prior knowledge, consistently outperforming baselines with an EGA of approximately 0.6 in MineRL and 0.9 in Mineflayer (Figure 5), despite the challenging setting with imperfect controllers. This superior performance is driven by its algorithmic correction mechanism, `RevisionByAnalogy`, which corrects flawed dependency knowledge while also accommodating imperfect controllers by gradually scaling required items quantities. The robustness of this algorithmic correction is particularly evident in two key analyses of the learned graph for each agent from the MineRL experiments. First, as shown in Table 2, XENON is uniquely robust to LLM hallucinations, learning dependencies for descendant items of non-existent, hallucinated items in the initial LLM-predicted graph. Second, XENON outperforms the baselines in learning dependencies for items that are unobtainable by the initial graph, as shown in Table 13.

Our results demonstrate the unreliability of relying on LLM self-correction or blindly trusting an LLM’s flawed knowledge; in practice, SC achieves the same EGA as DECKARD, with both plateauing around 0.4 in both environments.

We observe that controller capacity strongly impacts dependency learning. This is evident in ADAM, whose EGA differs markedly between MineRL (≈ 0.1), which has a limited controller, and Mineflayer (≈ 0.6), which has a more competent controller. While ADAM unrealistically assumes a controller can gather large quantities of all resource items before attempting a new item, MineRL’s controller STEVE-1 (Lifshitz et al., 2023) cannot execute this demanding strategy, causing ADAM’s EGA to fall below even the simplest baseline, RAND. Controller capacity also accounts for XENON’s lower EGA in MineRL. For instance, XENON learns none of the dependencies of the Redstone group items, as STEVE-1 cannot execute XENON’s strategy for *inadmissible items* (Section 4.1). In contrast, the

378
 379 **Table 3: Performance on long-horizon task benchmark.** Average success rate of each group on the
 380 long-horizon task benchmark Li et al. (2024b) in MineRL. *Oracle* indicates that the true dependency
 381 graph is known in advance, *Learned* indicates that the graph is learned via experience across 400
 382 episodes. For fair comparison across LLMs, we include Optimus-1[†], our reproduction of Optimus-1
 383 using Qwen2.5-VL-7B. Due to resource limits, results for DEPS, Jarvis-1, Optimus-1, and Optimus-2
 384 are cited directly from (Li et al., 2025b). See Appendix K.12 for the success rate on each goal.
 385

Method	Dependency	Planner LLM	Overall	Wood	Stone	Iron	Diamond	Gold	Armor	Redstone
DEPS	-	Codex	0.22	0.77	0.48	0.16	0.01	0.00	0.10	0.00
Jarvis-1	Oracle	GPT-4	0.38	0.93	0.89	0.36	0.08	0.07	0.15	0.16
Optimus-1	Oracle	GPT-4V	0.43	0.98	0.92	0.46	0.11	0.08	0.19	0.25
Optimus-2	Oracle	GPT-4V	0.45	0.99	0.93	0.53	0.13	0.09	0.21	0.28
Optimus-1 [†]	Oracle	Qwen2.5-VL-7B	0.34	0.92	0.80	0.22	0.10	0.09	0.17	0.04
XENON *	Oracle	Qwen2.5-VL-7B	0.79	0.95	0.93	0.83	0.75	<u>0.73</u>	0.61	0.75
XENON	Learned	Qwen2.5-VL-7B	<u>0.54</u>	0.85	0.81	0.46	<u>0.64</u>	0.74	0.28	0.00

392
 393 more capable Mineflayer controller executes this strategy successfully, allowing XENON to learn
 394 the correct dependencies for 5 of 6 Redstone items. This difference highlights the critical role of
 395 controllers for dependency learning, as detailed in our analysis in Appendix K.3

396 397 5.3 EFFECTIVE PLANNING TO SOLVE DIVERSE GOALS

398 As shown in Table 3, XENON significantly outperforms baselines in solving diverse long-horizon
 399 goals despite using the lightweight Qwen2.5-VL-7B LLM (Bai et al., 2025), while the baselines
 400 rely on large proprietary models such as Codex (Chen et al., 2021), GPT-4 (OpenAI, 2024), and
 401 GPT-4V (OpenAI, 2023). Remarkably, even with its *learned* dependency knowledge (Section 5.2),
 402 XENON surpasses the baselines with the oracle knowledge on challenging late-game goals, achieving
 403 high SRs for item groups like Gold (0.74) and Diamond (0.64).

404 XENON’s superiority stems from two key factors. First, its FAM provides systematic, fine-grained
 405 action correction for each goal. Second, it reduces reliance on the LLM for planning in two ways: it
 406 shortens prompts and outputs by requiring it to predict one action per subgoal item, and it bypasses
 407 the LLM entirely by reusing successful actions from FAM. In contrast, the baselines lack a systematic,
 408 fine-grained action correction mechanism and instead make LLMs generate long plans from lengthy
 409 prompts—a strategy known to be ineffective for LLMs (Wu et al., 2024; Li et al., 2024a). This
 410 challenge is exemplified by Optimus-1[†]. Despite using a knowledge graph for planning like XENON,
 411 its long-context generation strategy causes LLM to predict incorrect actions or omit items explicitly
 412 provided in its prompt, as detailed in Appendix K.5.

413 We find that accurate knowledge is critical for long-horizon planning, as its absence can make even a
 414 capable agent ineffective. The Redstone group from Table 3 provides an example: while XENON*
 415 with oracle knowledge succeeds (0.75 SR), XENON with learned knowledge fails entirely (0.00 SR),
 416 because it failed to learn the dependencies for Redstone goals due to the controller’s limited capacity
 417 in MineRL (Section 5.2). This finding is further supported by our comprehensive ablation study,
 418 which confirms that accurate dependency knowledge is most critical for success across all goals (See
 419 Table 17 in Appendix K.7).

420 421 5.4 ROBUST DEPENDENCY LEARNING AGAINST KNOWLEDGE CONFLICTS

422 To isolate dependency learning from controller capacity,
 423 we shift to the MC-TextWorld environment with a perfect
 424 controller. In this setting, we test each agent’s robustness to
 425 conflicts with its prior knowledge (derived from the LLM’s
 426 initial predictions and human-written plans) by introducing
 427 arbitrary perturbations to the ground-truth required items and
 428 actions. These perturbations are applied with an intensity
 429 level; a higher intensity affects a greater number of items,
 430 as shown in Table 4. This intensity is denoted by a tuple
 431 (α, β) for required items and actions, respectively. $(0, 0)$
 432 represents the vanilla setting with no
 433 perturbations. See Figure 21 for the detailed perturbation process.

434 Table 4: Effect of ground-truth pertur-
 435 bations on prior knowledge.

Perturbation Intensity	Goal items obtainable via prior knowledge
0	16 (no perturbation)
1	14 (12 %)
2	11 (31 %)
3	9 (44 %)

435 represents the vanilla setting with no

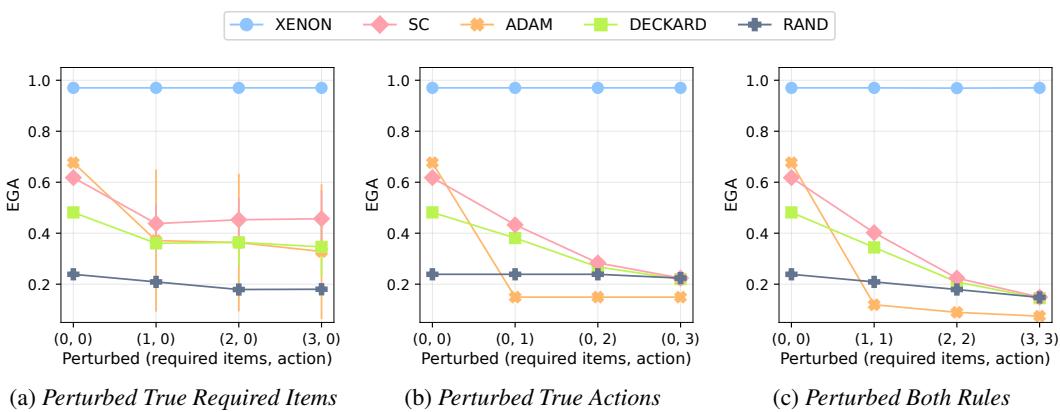


Figure 6: **Robustness against knowledge conflicts.** EGA after 3,000 environment steps in MC-TextWorld under different perturbations of the ground-truth rules. The plots show performance with increasing intensities of perturbation applied to: (a) requirements only, (b) operations only, and (c) both (see Table 4).

Figure 6 shows XENON’s strong robustness to knowledge conflicts, as it consistently maintains a near-perfect EGA (≈ 0.97). In contrast, the performance of all baselines degrades as perturbation intensity increases across all three perturbation scenarios (required items, actions, or both). Notably, we find that prompting an LLM to self-correct is ineffective when the ground truth conflicts with its parametric knowledge: SC shows no significant advantage over DECKARD, which lacks a correction mechanism. ADAM is vulnerable to action perturbations; its strategy of gathering all resource items before attempting a new item fails when the valid actions for those resources are perturbed, effectively halting its learning.

5.5 ABLATION STUDIES ON KNOWLEDGE CORRECTION MECHANISMS

As shown in Table 5, to analyze XENON’s knowledge correction mechanisms for dependencies and actions, we conduct ablation studies in MC-TextWorld. While dependency correction is generally more important for overall performance, action correction becomes vital under action perturbations. In contrast, LLM self-correction is ineffective for complex scenarios: it offers minimal gains for dependency correction even in the vanilla setting and fails entirely for perturbed actions. Its effectiveness is limited to simpler scenarios, such as action correction in the vanilla setting.

These results demonstrate that our algorithmic knowledge correction approach enables robust learning from experience, overcoming the limitations of both LLM self-correction and flawed initial knowledge.

Table 5: Ablation study of knowledge correction mechanisms. \circ : XENON; \triangle : LLM self-correction; \times : No correction. All entries denote the EGA after 3,000 environment steps. Columns denote the perturbation setting (r, a). For LLM self-correction, we use the same prompt as the SC baseline (see Appendix B).

Dependency Correction	Action Correction	(0, 0)	(3, 0)	(0, 3)	(3, 3)
\circ	\circ	0.97	0.97	0.97	0.97
\circ	\triangle	0.93	0.93	0.12	0.12
\circ	\times	0.84	0.84	0.12	0.12
\triangle	\circ	0.57	0.30	0.57	0.29
\times	\circ	0.53	0.13	0.53	0.13
\times	\times	0.46	0.13	0.19	0.11

5.6 ABLATION STUDIES ON HYPERPARAMETERS

To validate XENON’s stability to its hyperparameters, we conduct comprehensive ablation studies in both MC-TextWorld and MineRL. In these studies, we vary one hyperparameter at a time while keeping the others fixed to their default values ($c_0 = 3$, $\alpha_i = 8$, $\alpha_s = 2$, $x_0 = 2$).

Our results (Figure 7, Figure 8) show that although XENON is generally stable across hyperparameters, an effective learning strategy should account for controller capacity when the controller

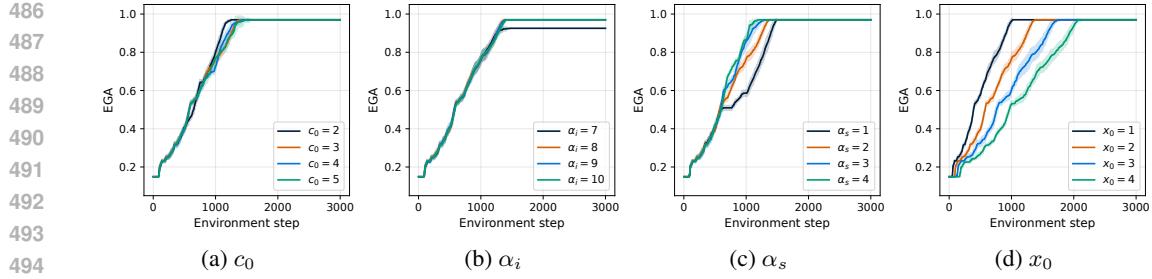


Figure 7: **Hyperparameter ablation study in MC-TextWorld.** EGA over 3,000 environment steps under different hyperparameters. The plots show EGA when varying: (a) c_0 (revision count threshold for inadmissible items), (b) α_i (required items quantities for inadmissible items), (c) α_s (required items quantities for less-tried items), and (d) x_0 (invalid action threshold). Each study varies one hyperparameter while keeping the others fixed to their default values ($c_0 = 3$, $\alpha_i = 8$, $\alpha_s = 2$, $x_0 = 2$).

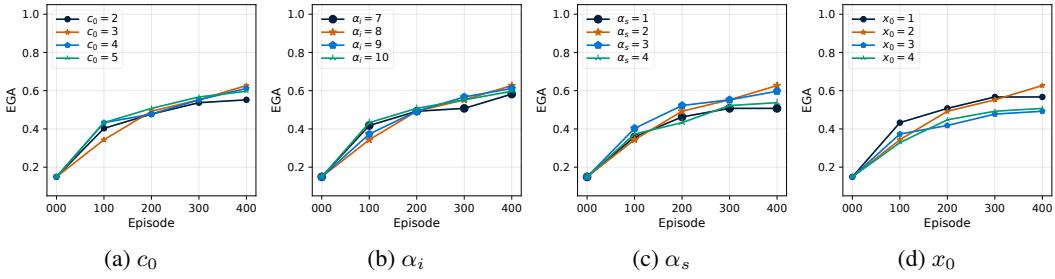


Figure 8: **Hyperparameter ablation study in MineRL.** EGA over 400 episodes under different hyperparameters. The plots show EGA when varying: (a) c_0 (revision count threshold for inadmissible items), (b) α_i (required items quantities for inadmissible items), (c) α_s (required items quantities for less-tried items), and (d) x_0 (invalid action threshold). Each study varies one hyperparameter while keeping the others fixed to their default values ($c_0 = 3$, $\alpha_i = 8$, $\alpha_s = 2$, $x_0 = 2$).

is imperfect. In MC-TextWorld (Figure 7), XENON maintains near-perfect EGA across a wide range of all tested hyperparameters, confirming its stability when a perfect controller is used. In MineRL (Figure 8), with an imperfect controller, the results demonstrate two findings. First, while influenced by hyperparameters, XENON still demonstrates robust performance, showing EGA after 400 episodes for all tested values remains near or above 0.5, outperforming baselines that plateau around or below 0.4 (Figure 5a). Second, controller capacity should be considered when designing dependency and action learning strategies. For example, the ablation on α_s (Figure 8c) shows that while gathering a sufficient quantity of items is necessary ($\alpha_s = 1$), overburdening the controller with excessive items ($\alpha_s = 4$) also degrades performance. Similarly, the ablation on x_0 (Figure 8d) shows the need to balance tolerating controller failures against wasting time on invalid actions.

6 CONCLUSION

We address the challenge of robust planning via experience-based algorithmic knowledge correction. With XENON, we show that directly revising external knowledge through experience enables an LLM-based agent to overcome flawed priors and sparse feedback, surpassing the limits of LLM self-correction. Experiments across diverse Minecraft benchmarks demonstrate that this approach not only strengthens knowledge acquisition and long-horizon planning, but also enables an agent with a lightweight 7B open-weight LLM to outperform prior methods that rely on much larger proprietary models. Our work delivers a key lesson for building robust LLM-based embodied agents: LLM priors should be treated with skepticism and continuously managed and corrected algorithmically.

Limitations. Despite its contributions, XENON faces a limitation. XENON’s performance is influenced by the underlying controller; in MineRL, STEVE-1 (Lifshitz et al., 2023) controller struggles with spatial exploration tasks, making a performance gap compared to more competent controllers like Mineflayer. Future work could involve jointly training the planner and controller, potentially using hierarchical reinforcement learning.

540 REFERENCES
541

542 Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang,
543 Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan,
544 Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng,
545 Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv*
546 preprint [arXiv:2502.13923](https://arxiv.org/abs/2502.13923), 2025.

547 Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon
548 Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching
549 unlabeled online videos, 2022. URL <https://arxiv.org/abs/2206.11795>.

550 Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Ce-
551 line Lin, and Pavlo Molchanov. Small language models are the future of agentic ai, 2025. URL
552 <https://arxiv.org/abs/2506.02153>.

553 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared
554 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,
555 Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan,
556 Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian,
557 Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios
558 Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino,
559 Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders,
560 Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa,
561 Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob
562 McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating
563 large language models trained on code. 2021.

564 Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual:
565 Constructing instruction manuals by llm agents via interactive environmental learning, 2024.

566 Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James
567 Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and
568 Adam Trischler. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532,
569 2018.

570 Kevin Du, Vésteinn Snæbjarnarson, Niklas Stoehr, Jennifer White, Aaron Schein, and Ryan Cot-
571 terell. Context versus prior knowledge in language models. In Lun-Wei Ku, Andre Martins, and
572 Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Compu-
573 tational Linguistics (Volume 1: Long Papers)*, pp. 13211–13235, Bangkok, Thailand, August
574 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.714. URL
575 [https://aclanthology.org/2024.acl-long.714/](https://aclanthology.org/2024.acl-long.714).

576 Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang,
577 De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied
578 agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing
579 Systems Datasets and Benchmarks Track*, 2022. URL [https://openreview.net/forum?
580 id=rc8o_j8I8PX](https://openreview.net/forum?id=rc8o_j8I8PX).

581 Yicheng Feng, Yuxuan Wang, Jiazheng Liu, Sipeng Zheng, and Zongqing Lu. LLaMA-rider:
582 Spurring large language models to explore the open world. In Kevin Duh, Helena Gomez,
583 and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL
584 2024*, pp. 4705–4724, Mexico City, Mexico, June 2024. Association for Computational Linguis-
585 tics. doi: 10.18653/v1/2024.findings-naacl.292. URL [https://aclanthology.org/2024.
587 findings-naacl.292/](https://aclanthology.org/2024.
586 findings-naacl.292/).

588 Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen.
589 Critic: Large language models can self-correct with tool-interactive critiquing, 2024. URL
590 <https://arxiv.org/abs/2305.11738>.

591 William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso,
592 and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations, 2019. URL
593 <https://arxiv.org/abs/1907.13440>.

594 Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song,
 595 and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024. URL <https://arxiv.org/abs/2310.01798>.

597 Jiajian Li, Qi Wang, Yunbo Wang, Xin Jin, Yang Li, Wenjun Zeng, and Xiaokang Yang. Open-world
 598 reinforcement learning over long short-term imagination. In *ICLR*, 2025a.

600 Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhui Chen. Long-context llms struggle with
 601 long in-context learning, 2024a.

603 Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1:
 604 Hybrid multimodal memory empowered agents excel in long-horizon tasks. *Advances in neural*
 605 *information processing systems*, 37:49881–49913, 2024b.

606 Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-2:
 607 Hybrid multimodal minecraft agent with goal-observation-action conditioned policy. In *2025 IEEE/CVF*
 608 *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2025b.

610 Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative
 611 model for text-to-behavior in minecraft. 2023.

612 Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. Juewu-mc:
 613 Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint*
 614 *arXiv:2112.04907*, 2021.

616 Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and
 617 Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the*
 618 *Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL
 619 <https://aclanthology.org/2024.tacl-1.9/>.

620 Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng,
 621 and Mingli Song. Odyssey: Empowering minecraft agents with open-world skills. In *International*
 622 *Joint Conference on Artificial Intelligence*, 2025.

623 Hangyu Mao, Chao Wang, Xiaotian Hao, Yihuan Mao, Yiming Lu, Chengjie Wu, Jianye Hao, Dong
 624 Li, and Pingzhong Tang. Seihai: A sample-efficient hierarchical ai for the minerl competition. In
 625 *Distributed Artificial Intelligence: Third International Conference, DAI 2021, Shanghai, China,*
 626 *December 17–18, 2021, Proceedings 3*, pp. 38–51. Springer, 2022.

628 Microsoft, :, Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, Nguyen
 629 Bach, Jianmin Bao, Alon Benhaim, Martin Cai, Vishrav Chaudhary, Congcong Chen, Dong Chen,
 630 Dongdong Chen, Junkun Chen, Weizhu Chen, Yen-Chun Chen, Yi ling Chen, Qi Dai, Xiyang
 631 Dai, Ruchao Fan, Mei Gao, Min Gao, Amit Garg, Abhishek Goswami, Junheng Hao, Amr Hendy,
 632 Yuxuan Hu, Xin Jin, Mahmoud Khademi, Dongwoo Kim, Young Jin Kim, Gina Lee, Jinyu Li,
 633 Yunsheng Li, Chen Liang, Xihui Lin, Zeqi Lin, Mengchen Liu, Yang Liu, Gilsinia Lopez, Chong
 634 Luo, Piyush Madan, Vadim Mazalov, Arindam Mitra, Ali Mousavi, Anh Nguyen, Jing Pan, Daniel
 635 Perez-Becker, Jacob Platin, Thomas Portet, Kai Qiu, Bo Ren, Liliang Ren, Sambuddha Roy,
 636 Ning Shang, Yelong Shen, Saksham Singhal, Subhajit Som, Xia Song, Tetyana Sych, Praneetha
 637 Vaddamanu, Shuohang Wang, Yiming Wang, Zhenghao Wang, Haibin Wu, Haoran Xu, Weijian
 638 Xu, Yifan Yang, Ziyi Yang, Donghan Yu, Ishmam Zabir, Jianwen Zhang, Li Lyna Zhang, Yunan
 639 Zhang, and Xiren Zhou. Phi-4-mini technical report: Compact yet powerful multimodal language
 640 models via mixture-of-loras, 2025. URL <https://arxiv.org/abs/2503.01743>.

641 Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer
 642 Singh, and Roy Fox. Do embodied agents dream of pixelated sheep? embodied decision making
 643 using language guided world modelling. In *Proceedings of the 40th International Conference on*
 644 *Machine Learning*, ICML’23. JMLR.org, 2023.

645 OpenAI. Gpt-4v(ision) system card, 2023. URL <https://openai.com/index/gpt-4v-system-card/>.

647 OpenAI. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>.

648 PrismarineJS. Prismarinejs/mineflayer, 2023. URL <https://github.com/PrismarineJS/mineflayer>.
 649 <https://github.com/PrismarineJS/mineflayer>.

650

651 Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing
 652 Shao. Mp5: A multi-modal open-ended embodied system in minecraft via active perception. In
 653 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16307–
 654 16316. IEEE, 2024.

655 Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks.
 656 In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.
 657 Association for Computational Linguistics, 11 2019. URL <https://arxiv.org/abs/1908.10084>.

658

659 Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu
 660 Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

661

662 Kaya Stechly, Karthik Valmeeekam, and Subbarao Kambhampati. On the self-verification limitations
 663 of large language models on reasoning and planning tasks, 2024. URL <https://arxiv.org/abs/2402.08115>.

664

665 Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre,
 666 Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus,
 667 Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun,
 668 Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange
 669 their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

670

671 Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. LLMs cannot find
 672 reasoning errors, but can correct them given the error location. In Lun-Wei Ku, Andre Martins,
 673 and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*,
 674 pp. 13894–13908, Bangkok, Thailand, August 2024. Association for Computational Linguistics.
 675 doi: 10.18653/v1/2024.findings-acl.826. URL <https://aclanthology.org/2024.findings-acl.826>.

676

677 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and
 678 Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*, 2023a.

679

680 Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian (Shawn) Ma, and Yitao Liang.
 681 Describe, explain, plan and select: Interactive planning with llms enables open-world multi-task
 682 agents. *Advances in Neural Information Processing Systems*, 36:34153–34189, 2023b.

683

684 Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng
 685 He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. Jarvis-1: Open-world multi-task
 686 agents with memory-augmented multimodal language models. *arXiv preprint arXiv: 2311.05997*,
 687 2023c.

688

689 Yuhao Wu, Ming Shan Hee, Zhiqing Hu, and Roy Ka-Wei Lee. Longgenbench: Benchmarking long-
 690 form generation in long context llms, 2024. URL <https://arxiv.org/abs/2409.02076>.

691

692 Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E Gonzalez, Bin Cui, and Shuicheng
 693 Yan. Supercorrect: Supervising and correcting language models with error-driven insights. In
 694 *International Conference on Learning Representations*, 2025.

695

696 Youngsik Yoon, Gangbok Lee, Sungsoo Ahn, and Jungseul Ok. Breadth-first exploration on adaptive
 697 grid for reinforcement learning. In *Forty-first International Conference on Machine Learning*,
 698 2024.

699

700 Shu Yu and Chaochao Lu. Adam: An embodied causal agent in open-world environments. *arXiv preprint arXiv:2410.22194*, 2024.

701

702 Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing
 703 Lu. Plan4MC: Skill reinforcement learning and planning for open-world Minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023.

702 Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee,
 703 Honglak Lee, and Lu Wang. Small language models need strong verifiers to self-correct rea-
 704 soning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Associa-
 705 tion for Computational Linguistics: ACL 2024*, pp. 15637–15653, Bangkok, Thailand, August
 706 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.924. URL
 707 <https://aclanthology.org/2024.findings-acl.924/>.

708 Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel:
 709 Llm agents are experiential learners. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam
 710 Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-
 711 Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth
 712 Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024,
 713 Vancouver, Canada*, pp. 19632–19642. AAAI Press, 2024. doi: 10.1609/aaai.v38i17.29936. URL
 714 <https://ojs.aaai.org/index.php/AAAI/article/view/29936>.

715 Zhonghan Zhao, Wenhao Chai, Xuan Wang, Li Boyi, Shengyu Hao, Shidong Cao, Tian Ye, Jenq-
 716 Neng Hwang, and Gaoang Wang. See and think: Embodied agent in virtual environment. *arXiv*
 717 *preprint arXiv:2311.15209*, 2023.

718 Xinyue Zheng, Haowei Lin, Kaichen He, Zihao Wang, Zilong Zheng, and Yitao Liang. Mcu: An
 719 evaluation framework for open-ended game agents, 2025. URL <https://arxiv.org/abs/2310.08367>.

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

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756 This appendix is organized as follows:
757

- 758 • [Appendix A: Experiments in a domain other than Minecraft \(Microsoft TextWorld Cooking\)](#).
- 759 • [Appendix B: Prompts and qualitative results of LLM self-correction in our experiments](#).
- 760 • [Appendix C: Detailed procedure for experienced requirement set determination and depen-](#)
- 761 [dency graph updates, as discussed in Section 3.](#)
- 762 • [Appendix E: Detailed pseudocode and the prompt for ADG in Section 4.1.](#)
- 763 • [Appendix F: Detailed pseudocode and the prompt for step-by-step planning using FAM in Section 4.2.](#)
- 764 • [Appendix H: Detailed descriptions and the prompt for CRe in Section 4.3.](#)
- 765 • [Appendix I: Detailed descriptions of implementation, human-written plans, and hyperpa-](#)
- 766 [rameters.](#)
- 767 • [Appendix J: Detailed descriptions of the baselines and experimental environments in Sec-](#)
- 768 [tion 5.](#)
- 769 • [Appendix K: Analysis of experimental results and additional experimental results.](#)
- 770 • [Appendix L: Descriptions about LLM usage.](#)

771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809

810
 811 Table 6: Success rates in the TextWorld Cooking environment, comparing XENON against the SC
 812 (LLM self-correction) and DECKARD baselines from Section 5.1. We report the mean \pm standard
 813 deviation over 3 independent runs, where each run consists of 100 episodes.

	DECKARD	SC	XENON
Success Rate	0.09 ± 0.02	0.75 ± 0.04	1.00 ± 0.00

817 818 819 A ADDITIONAL EXPERIMENTS IN ANOTHER DOMAIN 820 821

822 To assess generalization beyond Minecraft, we evaluate XENON on the Microsoft TextWorld Cooking
 823 environment (Côté et al., 2018), a text-based household task planning benchmark. We demonstrate
 824 XENON can correct an LLM’s flawed knowledge of preconditions (e.g., required tools) and valid
 825 actions for plans using ADG and FAM in this domain as well. We note that XENON is applied with
 826 minimal modification: FAM is applied without modification, while ADG is adapted from its original
 827 design, which supports multiple incoming edges (preconditions) for a node, to one that allows only a
 828 single incoming edge, as this domain requires only a single precondition per node.

830 831 A.1 EXPERIMENT SETUP 832

833 **Environment Rules.** The goal is to prepare and eat a meal by reading a cookbook, which provides
 834 a plan as a list of (action, ingredient) pairs, e.g., (“fry”, “pepper”). We note that an agent cannot
 835 succeed by naively following this plan. This is because the agent must solve two key challenges: (1)
 836 it must discover the valid **tool** required for each cookbook action, and (2) it must discover the valid,
 837 **executable action** for each cookbook action, as some cookbook actions are not directly accepted by
 838 the environment (i.e., not in its action space).

839 Specifically, to succeed a cookbook’s (action, ingredient) pair, an agent must make a subgoal,
 840 formatted as (**executable action**, ingredient, **tool**), where the **executable action** and
 841 **tool** must be valid for the cookbook action. For example, the cookbook’s (“fry”, “pepper”) pair
 842 requires the agent to make a subgoal (cook, “pepper”, stove). The available executable action
 843 space consists of { “chop”, “close”, “cook”, “dice”, “drop”, “eat”, “examine”, “slice”, “prepare” },
 844 and the available tools are { “knife”, “oven”, “stove”, “fridge”, “table”, “counter” }.

845 **Baselines and Evaluation.** All agents use an LLM (Qwen2.5-VL-7B) to make subgoals. The **tool**
 846 for each cookbook action is predicted by the LLM from the available tools before an episode begins.
 847 At each timestep during the episode, given a cookbook action, the LLM predicts an **executable**
 848 **action** from the executable action space, constructing a subgoal from this predicted **executable**
 849 **action**, the input ingredient, and the predicted **tool**.

850 To isolate the challenge of planning knowledge correction, we assume a competent controller gathers
 851 all ingredients and tools; thus, an agent starts each episode with all necessary ingredients and tools.
 852 An episode (max 50 timesteps) is successful if the agent completes the plan.

854 855 A.2 RESULTS 856

857 Table 6 shows that XENON achieves a perfect success rate (1.00 ± 0.00), significantly outperforming
 858 both SC (0.75 ± 0.04) and DECKARD (0.09 ± 0.02). These results demonstrate that XENON’s core
 859 mechanisms (ADG and FAM) are generalizable, effectively correcting flawed planning knowledge in
 860 a domain that requires the agent to discover valid symbolic actions and preconditions. Notably, the
 861 SC baseline fails to achieve high performance, even in the TextWorld Cooking environment which is
 862 simpler than Minecraft. This reinforces our claim that relying on LLM self-correction is less reliable
 863 than XENON’s experience-based algorithmic knowledge correction.

864 **B PROMPTS AND QUALITATIVE RESULTS OF LLM SELF-CORRECTION**
865866 **B.1 DEPENDENCY CORRECTION**
867868 Figure 9 shows the prompt used for dependency correction.
869

```

870 1 You are a professional game analyst. For a given <item_name>, you need to
871     make <required_items> to get the item.
872 2 If you make <required_items> well, I will give you 1 $.
873 3
874 4 I will give you recent transitions.
875 5 % Recent failed trajectories are given
876 6 [Failed example]
877 7 <item_name>: {item_name}
878 8 <hypothesized_required_items>: {original_prediction}
879 9 <inventory>: {inventory}
880 10 <plan>: {failed_subgoal}
881 11 <success>: false
882 12
883 13 I will give you learned items similar to <item_name>, and their validated
884     required items, just for reference.
885 14 % K similar experienced items and their requirements are given
886 15 [Success Example]
887 16 <item_name>: {experienced_item}
888 17 <required_items> {experienced_requirements}
889 18
890 19 % Make a new predicted requirement set
891 20 [Your turn]
892 21 Here is <item_name>, you MUST output <required_items> to obtain the item
893     in JSON format. Remember <required_items> MUST be in JSON format.
894 22
895 23 <item_name>: {item_name}
896 24 <required_items>:

```

894 Figure 9: Prompt used for LLM self-correction about dependencies.
895896
897 We provide some examples of actual prompts and LLM outputs in Figure 10, Figure 11
898899 **B.2 ACTION CORRECTION**
900901 Figure 12 shows the prompt used self-reflection for failed actions.
902903 We provide some examples of actual prompts and LLM outputs in Figure 13, Figure 14
904

```

918
919
920
921
922
923
924
925
926
927
928
929
930
1 You are a professional game analyst. For a given <item_name>, you need to make <required_items>
931     > to get the item.
932 2 If you make <required_items> well, I will give you 1 $.
933 3
934 4 I will give you recent transitions.
935 5
936 6 [Failed example]
937 7 <item_name>: iron_nugget
938 8 <hypothesized_required_items>: {'iron_ore': 1, 'crafting_table': 1}
939 9 <inventory>: {'crafting_table': 1, 'wooden_sword': 1, 'wooden_pickaxe': 1, 'torch': 4, 'furnace': 1, 'stone_pickaxe': 1, 'iron_axe': 1, 'iron_shovel': 1, 'stick': 2, 'iron_pickaxe': 1, 'diamond': 3, 'iron_ingot': 2, 'iron_ore': 2, 'gold_ore': 1, 'coal': 1}
940 10 <plan>: dig down and mine iron_nugget
941 11 <success>: false
942 12
943 13 I will give you learned items similar to <item_name>, and their validated required items, just
944     for reference.
945 14 [Success Example]
946 15 <item_name>:
947 16 iron_ingot
948 17 <required_items>:
949 18 {'recipe': {'furnace': 1, 'iron_ore': 1, 'coals': 1}}
950 19 [Success Example]
951 20 <item_name>:
952 21 iron_pickaxe
953 22 <required_items>:
954 23 {'recipe': {'stick': 2, 'iron_ingot': 3, 'crafting_table': 1}}
955 24 [Success Example]
956 25 <item_name>:
957 26 iron_shovel
958 27 <required_items>:
959 28 {'recipe': {'stick': 2, 'iron_ingot': 1, 'crafting_table': 1}}
960 29
961 30 [Your turn]
962 31 Here is <item_name>, you MUST output <required_items> to obtain the item in JSON format.
963     Remember <required_items> MUST be in JSON format.
964
965
966
967
968
969
970
971

```

Figure 10: Example of dependency self-correction for iron_nugget.

```

972
973
974
975
976
977
978
979
980
981
982
983
984
1  You are a professional game analyst. For a given <item_name>, you need to make <required_items>
2   > to get the item.
3
4  If you make <required_items> well, I will give you 1 $.
5
6  I will give you recent transitions.
7
8  [Failed example]
9  <item_name>: charcoal
10 <hypothesized_required_items>: {'oak_log': 8}
11 <inventory>: {'dirt': 1, 'oak_log': 2, 'crafting_table': 1, 'wooden_hoe': 1, 'wooden_pickaxe': 1, 'torch': 4, 'stone_axe': 1, 'furnace': 1, 'stone_pickaxe': 1, 'stick': 2, 'iron_pickaxe': 1, 'diamond': 1, 'iron_ingot': 3, 'iron_ore': 2, 'coal': 2}
12 <action>: craft charcoal
13 <success>: false
14
15 I will give you learned items similar to <item_name>, and their validated required items, just
16   for reference.
17 [Success Example]
18 <item_name>:
19 coals
20 <required_items>:
21 {'recipe': {'wooden_pickaxe': 1}}
22 [Success Example]
23 <item_name>:
24 furnace
25 <required_items>:
26 {'recipe': {'cobblestone': 8, 'crafting_table': 1}}
27 [Success Example]
28 <item_name>:
29 diamond
30 <required_items>:
31 {'recipe': {'iron_pickaxe': 1}}
32 [Your turn]
33 Here is <item_name>, you MUST output <required_items> to achieve charcoal in JSON format.
34   Remember <required_items> MUST be in JSON format.
35
36 <item_name>:
37 charcoal
38 <required_items>:
39 % LLM output: {'recipe': {'oak_log': 8}}
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
925
926
927
927
928
929
929
930
931
932
933
934
935
936
936
937
938
938
939
939
940
941
942
943
944
945
945
946
947
947
948
949
949
950
951
952
953
954
955
956
956
957
958
958
959
959
960
961
962
963
964
965
965
966
967
967
968
969
969
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
16
```

```

1026
1027 1 % LLM self-reflection to analyze failure reasons
1028 2 You are a professional game analyst.
1029 3 For a given <item_name> and <inventory>, you need to analyze why <plan>
1030 4 failed to get the item.
1031 5 I will give you examples of analysis as follow.
1032 6
1033 7 [Example]
1034 8 <item_name>: wooden_pickaxe
1035 9 <inventory>: {'stick': 4, 'planks': 4, 'crafting_table': 1}
1036 10 <plan>: smelt wooden_pickaxe
1037 11 <failure_analysis>
1038 12 {"analysis": "You failed because you cannot smelt a wooden_pickaxe. You
1039 13 should craft it instead."}
1040 14
1041 15 [Example]
1042 16 <item_name>: stone_pickaxe
1043 17 <inventory>: {'stick': 4, 'planks': 4, 'crafting_table': 1}
1044 18 <plan>: craft stone_pickaxe
1045 19 <failure_analysis>
1046 20 {"analysis": "You failed because you do not have enough cobblestones."}
1047 21
1048 22 [Your turn]
1049 23 Here is <item_name>, <inventory> and <plan>, you MUST output <failure_
1050 24 analysis> concisely in JSON format.
1051 25
1052 26 <item_name>: {item_name}
1053 27 <inventory>: {inventory}
1054 28 <plan>: {plan}
1055 29 <failure_analysis>
1056 30 % Then, using the self-reflection results, LLM self-correct its actions.
1057 31 For an item name, you need to make a plan, by selecting one among
1058 32 provided options.
1059 33 I will give you examples of which plans are needed to achieve an item,
1060 34 just for reference.
1061 35 [Example]
1062 36 <item name>
1063 37 {similar_item}
1064 38 <task planning>
1065 39 {successful_plan}
1066 40
1067 41 Here are some analyses on previous failed plans for this item.
1068 42 [Analysis]
1069 43 {'item_name': {item}, 'inventory': {inventory}, 'plan': '{plan}', '
1070 44 failure_analysis': '{self-reflection}'}
1071 45
1072 46 [Your turn]
1073 47 Here is <item name>, you MUST select one from below <options>, to make <
1074 48 task planning>.
1075 49 you MUST select one from below <options>. DO NOT MAKE A PLAN NOT IN <
1076 50 options>.
1077 51
1078 52 <options>:
1079 53 1: {"task": "dig down and mine {item}", "goal": [{item}, {quantity}]}
2: {"task": "craft {item}", "goal": [{item}, {quantity}]}
3: {"task": "smelt {item}", "item": [{item}, {quantity}]}
<item name>
{item}
<task planning>

```

Figure 12: Prompts used for LLM self-correction about actions.

```

1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092 1 For an item name, you need to make a plan, by selecting one among provided options.
1093 2 I will give you examples of which plans are needed to achieve an item, just for reference.
1094 3
1095 4 [Example]
1096 5 <item name>
1097 6 iron_ingot
1098 7 <task planning>
1099 8 {"task": "smelt iron_ingot", "goal": ["iron_ingot", 1]}
1100 9
1101 10 [Example]
1102 11 <item name>
1103 12 iron_pickaxe
1104 13 <task planning>
1105 14 {"task": "craft iron_pickaxe", "goal": ["iron_pickaxe", 1]}
1106 15
1107 16 [Example]
1108 17 <item name>
1109 18 iron_shovel
1110 19 <task planning>
1111 20 {"task": "craft iron_shovel", "goal": ["iron_shovel", 1]}
1112 21
1113 22 Here are some analyses on previous failed plans for this item.
1114 23 [Analysis]
1115 24 {'item_name': 'iron_nugget',
1116 25 'inventory': {'crafting_table': 1, 'wooden_sword': 1, 'wooden_pickaxe': 1, 'torch': 4, 'furnace': 1, 'stone_pickaxe': 1, 'iron_axe': 1, 'iron_shovel': 1, 'stick': 2, 'iron_pickaxe': 1, 'diamond': 3, 'iron_ingot': 2, 'iron_ore': 2, 'gold_ore': 1, 'coal': 1},
1117 26 'plan': 'dig down and mine iron_nugget',
1118 27 'failure_analysis': 'You failed because you do not have any iron ore or diamond ore to mine
1119 28 for iron nuggets.'}
1120 29
1121 30 [Your turn]
1122 31 Here is <item name>, you MUST select one from below <options>, to make <task planning>.
1123 32 you MUST select one from below <options>. DO NOT MAKE A PLAN NOT IN <options>.
1124 33
1125 34 <options>
1126 35 1. {"task": "dig down and mine iron_nugget", "goal": ["iron_nugget", 1]}
1127 36 2. {"task": "craft iron_nugget", "goal": ["iron_nugget", 1]}
1128 37 3. {"task": "smelt iron_nugget", "goal": ["iron_nugget", 1]}
1129 38
1130 39 <item name>
1131 40 iron_nugget
1132 % LLM output: '{"task": "dig down and mine iron_nugget", "goal": ["iron_nugget", 1]}'

```

Figure 13: Example of action self-correction for iron_nugget.

```

1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

```

```

1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1 For an item name, you need to make a plan, by selecting one among provided options.
1146 2 I will give you examples of which plans are needed to achieve an item, just for reference.
1147 3
1148 4 [Example]
1149 5 <item name>
1150 6 coals
1151 7 <task planning>
1152 8 {"task": "dig down and mine coals", "goal": ["coals", 1]}
1153 9
1154 10 [Example]
1155 11 <item name>
1156 12 furnace
1157 13 <task planning>
1158 14 {"task": "craft furnace", "goal": ["furnace", 1]}
1159 15
1160 16 [Example]
1161 17 <item name>
1162 18 diamond
1163 19 <task planning>
1164 20 {"task": "dig down and mine diamond", "goal": ["diamond", 1]}
1165 21
1166 22 Here are some analyses on previous failed plans for this item.
1167 23 [Analysis]
1168 24 {'item_name': 'charcoal',
1169 25 'inventory': {'dirt': 1, 'oak_log': 2, 'crafting_table': 1, 'wooden_hoe': 1, 'wooden_pickaxe': 1,
1170 26 'torch': 4, 'stone_axe': 1, 'furnace': 1, 'stone_pickaxe': 1, 'stick': 2, 'iron_pickaxe': 1, 'diamond': 1, 'iron_ingot': 3, 'iron_ore': 2, 'coal': 2},
1171 27 'plan': 'mine iron_nugget',
1172 28 'failure_analysis': 'You failed because you already have enough charcoal.'}
1173 29
1174 30 [Your turn]
1175 31 Here is <item name>, you MUST select one from below <options>, to make <task planning>.
1176 32 you MUST select one from below <options>. DO NOT MAKE A PLAN NOT IN <options>.
1177 33
1178 34 <options>
1179 35 1. {"task": "mine iron_nugget", "goal": ["charcoal", 1]}
1180 36 2. {"task": "craft charcoal", "goal": ["charcoal", 1]}
1181 37 3. {"task": "smelt charcoal", "goal": ["charcoal", 1]}
1182 38
1183 39 <item name>
1184 40 charcoal
1185 41 <task planning>
1186 42 % LLM output: {'task": "craft charcoal", "goal": ["charcoal", 1]}
1187

```

Figure 14: Example of action self-correction for charcoal.

1188 C EXPERIENCED REQUIREMENT SET AND DEPENDENCY GRAPH UPDATE
11891190 We note that the assumptions explained in this section are largely similar to those in the implementa-
1191 tion of DECKARD (Nottingham et al., 2023)².
11921193 **Determining experienced requirement set.** When the agent obtains item v while executing a
1194 subgoal (op, q, u) , it determines the experienced requirement set $\mathcal{R}_{exp}(v)$ differently depending on
1195 whether the operation op is “mine” or falls under “craft” or “smelt”. If op is “mine”, the agent
1196 determines $\mathcal{R}_{exp}(v)$ based on the pickaxe in its inventory. If no pickaxe is held, $\mathcal{R}_{exp}(v)$ is \emptyset .
1197 Otherwise, $\mathcal{R}_{exp}(v)$ becomes $\{(the\ highest\ tier\ pickaxe\ the\ agent\ has, 1)\}$, where the highest-tier
1198 pickaxe is determined following the hierarchy: “wooden_pickaxe”, “stone_pickaxe”, “iron_pickaxe”,
1199 “diamond_pickaxe”. If op is “craft” or “smelt”, the agent determines the used items and their quantities
1200 as $\mathcal{R}_{exp}(v)$ by observing inventory changes when crafting or smelting v .
12011202 **Dependency graph update.** When the agent obtains an item v and its $\mathcal{R}_{exp}(v)$ for the first time, it
1203 updates its dependency graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$. Since $\mathcal{R}_{exp}(v)$ only contains items acquired before v , no
1204 cycles can be introduced to ADG during learning. The update proceeds as follows: The agent adds v
1205 to both the set of known items $\hat{\mathcal{V}}$. Then, it updates the edge set $\hat{\mathcal{E}}$ by replacing v ’s incoming edges
1206 with $\mathcal{R}_{exp}(v)$; it removes all of v ’s incoming edges $(u, \cdot, v) \in \hat{\mathcal{E}}$ and adds new edges (u_i, q_i, v) to $\hat{\mathcal{E}}$
1207 for every $(u_i, q_i) \in \mathcal{R}_{exp}(v)$.
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241

 ²<https://github.com/DeckardAgent/deckard>

```

1242
1243 Algorithm 1: Pseudocode of XENON
1244 input : invalid operation threshold  $x_0$ , inadmissible item threshold  $c_0$ , less-explored item scale
1245  $\alpha_s$ , inadmissible item scale  $\alpha_i$ 
1246 1 Initialize dependency  $\hat{\mathcal{G}} \leftarrow (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ , revision counts  $C[v] \leftarrow 1$  for all  $v \in \hat{\mathcal{V}}$ 
1247 2 Initialize memory  $S(a, v) = 0, F(a, v) = 0$  for all  $v \in \hat{\mathcal{V}}, a \in \mathcal{A}$ 
1248 3 while learning do
1249   4 Get an empty inventory  $inv$ 
1250   5  $v_g \leftarrow \text{SelectGoalWithDifficulty}(\hat{\mathcal{G}}, C[\cdot])$  // DEX Appendix G
1251   6 while  $H_{episode}$  do
1252     7 Series of aggregated requirements  $((q_l, u_l))_{l=1}^{L_{v_g}}$  using  $\hat{\mathcal{G}}$  and  $inv$ 
1253       // from Section 3
1254     8 Plan  $P \leftarrow ((a_l, q_l, u_l))_{l=1}^{L_{v_g}}$  by selecting  $a_l$  for each  $u_l$ , using LLM,  $M, x_0$ 
1255     9 foreach subgoal  $(a, q, u) \in P$  do
1256       10 Execute  $(a, q, u)$  then get the execution result  $success$ 
1257       11 Get an updated inventory  $inv$ , dependency graph  $\hat{\mathcal{G}}$  // from Section 3
1258       12 Update memory  $S(a, u) += success, F(a, u) += \neg success$ 
1259       13 if not  $success$  then
1260         14 if All actions are invalid then
1261           15  $\hat{\mathcal{G}}, C \leftarrow \text{RevisionByAnalogy}(\hat{\mathcal{G}}, u, C[\cdot], c_0, \alpha_s, \alpha_i)$ 
1262             // ADG Section 4.1
1263           16 Reset memory  $M[u, \cdot] \leftarrow (0, 0)$ 
1264           17  $v_g \leftarrow \text{SelectGoalWithDifficulty}(\hat{\mathcal{G}}, C[\cdot])$ 
1265         18 break
1266

```

D FULL PROCEDURE OF XENON

The full procedure of XENON is outlined in Algorithm 1.

1296 E DETAILS IN ADAPTIVE DEPENDENCY LEARNING (ADG)
12971298 E.1 RATIONALE FOR INITIAL KNOWLEDGE
1299

1300 In real-world applications, a human user may wish for an autonomous agent to accomplish certain
1301 goals, yet the user themselves may have limited or no knowledge of how to achieve them within a
1302 complex environment. We model this scenario by having a user specify goal items without providing
1303 the detailed requirements, and then the agent should autonomously learn how to obtain these goal
1304 items. The set of 67 goal item names (\mathcal{V}_0) provided to the agent represents such user-specified goal
1305 items, defining the learning objectives.

1306 To bootstrap learning in complex environments, LLM-based planning literature often utilizes minimal
1307 human-written plans for initial knowledge (Zhao et al., 2024; Chen et al., 2024). In our case, we
1308 provide the agent with 3 human-written plans (shown in Appendix I). By executing these plans, our
1309 agent can experience items and their dependencies, thereby bootstrapping the dependency learning
1310 process.

1311
1312 E.2 DETAILS IN DEPENDENCY GRAPH INITIALIZATION
1313

1314
1315 **Keeping ADG acyclic during initialization.** During initialization, XENON prevents cycles in
1316 ADG algorithmically and maintains ADG as a directed acyclic graph, by, whenever adding an LLM-
1317 predicted requirement set for an item, discarding any set that would make a cycle and instead assign
1318 an empty requirement set to that item. Specifically, we identify and prevent cycles in three steps when
1319 adding LLM-predicted incoming edges for an item v . First, we tentatively insert the LLM-predicted
1320 incoming edges of v into the current ADG. Second, we detect cycles by checking whether any of v 's
1321 parents now appears among v 's descendants in the updated graph. Third, if a cycle is detected, we
1322 discard the LLM-predicted incoming edges for v and instead assign an empty set of incoming edges
1323 to v in the ADG.

1324 Pseudocode is shown in Algorithm 2. The prompt is shown in Figure 15.

```
1325 1 You are a professional game analyst. For a given <item_name>, you need to  
1326 2     make <required_items> to get the item.  
1327 3 If you make <required_items> well, I will give you 1 $.  
1328 4 I will give you some examples <item_name> and <required_items>.  
1329 5 [Example] % TopK similar experienced items are given as examples  
1330 6 <item_name>: {experienced_item}  
1331 7 <required_items>: {experienced_requirement_set}  
1332 8  
1333 9 [Your turn]  
1334 10 Here is a item name, you MUST output <required_items> in JSON format.  
1335 11     Remember <required_items> MUST be in JSON format.  
1336 12  
1337 13 <item_name>: {item_name}  
1338 14 <required_items>:  
1339 15
```

1340 Figure 15: Prompt for requirement set prediction for dependency graph initialization
1341

1342
1343 E.3 PSEUDOCODE OF REVISIONBYANALOGY
1344

1345 Pseudocode is shown in Algorithm 3.
1346
1347
1348
1349

1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1360
 1361
 1362
 1363
 1364

Algorithm 2: GraphInitialization

1365 **input** :Goal items \mathcal{V}_0 , (optional) human written plans \mathcal{P}_0
 1366 **output** :Initialized dependency graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$, experienced items \mathcal{V}
 1367 1 Initialize a set of known items $\hat{\mathcal{V}} \leftarrow \mathcal{V}_0$, edge set $\hat{\mathcal{E}} \leftarrow \emptyset$
 1368 2 Initialize a set of experienced items $\mathcal{V} \leftarrow \emptyset$
 1369 3 **foreach** *plan* in \mathcal{P}_0 **do**
 1370 4 Execute the plan and get experienced items and their experienced requirement sets
 1371 $\{(v_n, \mathcal{R}_{exp}(v_n))\}_{n=1}^N$
 1372 5 **foreach** $(v, \mathcal{R}_{exp}(v)) \in \{(v_n, \mathcal{R}_{exp}(v_n))\}_{n=1}^N$ **do**
 1373 6 **if** $v \notin \mathcal{V}$ **then**
 1374 7 /* graph update from Appendix C */
 1375 8 $\mathcal{V} \leftarrow \mathcal{V} \cup \{v\}$, $\hat{\mathcal{V}} \leftarrow \hat{\mathcal{V}} \cup \{v\}$
 1376 /* Add edges to $\hat{\mathcal{E}}$ according to $\mathcal{R}_{exp}(v)$ */
 1377 /* Graph construction using LLM predictions */
 1378 9 **while** $\exists v \in \hat{\mathcal{V}} \setminus \mathcal{V}$ whose requirement set $\mathcal{R}(v)$ has not yet been predicted by the LLM **do**
 1379 10 Select such an item $v \in \hat{\mathcal{V}} \setminus \mathcal{V}$ (i.e., $\mathcal{R}(v)$ has not yet been predicted)
 1380 11 Select $\mathcal{V}_K \subseteq \mathcal{V}$ based on Top-K semantic similarity to v , $|\mathcal{V}_K| = K$
 1381 12 Predict $\mathcal{R}(v) \leftarrow LLM(v, \{(u, \mathcal{R}(u, \hat{\mathcal{G}}))\}_{u \in \mathcal{V}_K})$
 1382 13 **foreach** $(u_j, q_j) \in \mathcal{R}(v)$ **do**
 1383 14 $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(u_j, q_j, v)\}$
 1384 15 **if** $u_j \notin \hat{\mathcal{V}}$ **then**
 1385 16 $\hat{\mathcal{V}} \leftarrow \hat{\mathcal{V}} \cup \{u_j\}$
 1386
 1387
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1399
 1400
 1401
 1402
 1403

1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414

Algorithm 3: RevisionByAnalogy

1415
1416 **input** : Dependency graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$, an item to revise v , exploration counts $C[\cdot]$,
1417 inadmissible item threshold c_0 , less-explored item scale α_s , inadmissible item scale α_i
1418 **output** : Revised dependency graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$, exploration counts $C[\cdot]$
1419 1 Consider cases based on $C[v]$:
1420 2 **if** $C[v] > c_0$ **then**
1421 /* v is inadmissible */
1422 /* resource set: items previously consumed for crafting
1423 other items */
1424 3 $\mathcal{R}(v) \leftarrow \{(u, \alpha_i) \mid u \in \text{“resource” set}\}$
1425 /* Remove all incoming edges to v in $\hat{\mathcal{E}}$ and add new edges */
1426 4 $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \setminus \{(x, q, v) \mid (x, q, v) \in \hat{\mathcal{E}}\}$
1427 5 **foreach** $(u, \alpha_i) \in \mathcal{R}(v)$ **do**
1428 $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(u, \alpha_i, v)\}$
1429 /* Revise requirement sets of descendants of v */
1430 7 Find the set of all descendants of v in $\hat{\mathcal{G}}$ (excluding v): $\mathcal{W} \leftarrow \text{FindAllDescendants}(v, \hat{\mathcal{G}})$
1431 8 **for** each item w in \mathcal{W} **do**
1432 $\text{Invoke RevisionByAnalogy for } w$
1433
1434 10 **else**
1435 /* v is less explored yet. Revise based on analogy */
1436 11 Find similar **successfully obtained** items $\mathcal{V}_K \subseteq \hat{\mathcal{V}}$ based on Top-K semantic similarity to v
1437 12 Candidate items $U_{cand} \leftarrow \{u \mid \exists w \in \mathcal{V}_K, (u, \cdot, w) \in \hat{\mathcal{E}}\}$ /* all items required
1438 to obtain similar **successfully obtained** items \mathcal{V}_K */
1439 13 Start to construct a requirement set, $\mathcal{R}(v) \leftarrow \emptyset$
1440 14 **for** each item u in U_{cand} **do**
1441 **if** u is in “resource” set **then**
1442 $\text{Add } (u, \alpha_s \times C[v]) \text{ to } \mathcal{R}(v)$
1443 **else**
1444 $\text{Add } (u, 1) \text{ to } \mathcal{R}(v)$
1445 19 Update $\hat{\mathcal{G}}$: Remove all incoming edges to v in $\hat{\mathcal{E}}$, and add new edges (u, q, v) to $\hat{\mathcal{E}}$ for each
1446 $(u, q) \in \mathcal{R}(v)$
1447

1448
1449
1450
1451
1452
1453
1454
1455
1456
1457

1458 F STEP-BY-STEP PLANNING USING FAM
1459

1460 Given a sequence of aggregated requirements $((q_l, v_l))_{l=1}^L$, XENON employs a step-by-step planning
1461 approach, iteratively selecting an operation op_l for each requirement item v_l to make a subgoal
1462 (op_l, q_l, v_l) . This process considers the past attempts to obtain v_l using specific operations. Specif-
1463 ically, for a given item v_l , if FAM has an empirically valid operation, XENON reuses it without
1464 prompting the LLM. Otherwise, XENON prompts the LLM to select an operation, leveraging infor-
1465 mation from (i) valid operations for items semantically similar to v_l , (ii) empirically invalid operations
1466 for v_l .

1467 The pseudocode for this operation selection process is detailed in Algorithm 4. The prompt is shown
1468 in Figure 16.
1469

1470 **Algorithm 4:** Step-by-step Planning
1471

1472 **input** : A requirement item v , Fine-grained Failure-aware Operation Memory $M[\cdot, \cdot]$, invalid
1473 operation threshold x_0
1474 **output** : Selected operation $op_{selected}$
1475 1 Retrieve history for item v : $M_v \leftarrow \{(op, n_{v,op}^{succ}, n_{v,op}^{fail})\}_{op \in \mathcal{OP}}$
1476 /* Find empirically valid and invalid operations for v according to M_v */
1477 2 $\mathcal{OP}_v^{valid} \leftarrow \{op \in \mathcal{OP} \mid n_{v,op}^{succ} > 0 \wedge n_{v,op}^{succ} - n_{v,op}^{fail} > -x_0\}$
1478 3 $\mathcal{OP}_v^{invalid} \leftarrow \{op \in \mathcal{OP} \mid n_{v,op}^{succ} - n_{v,op}^{fail} \leq -x_0\}$
1479 4 **if** $\mathcal{OP}_v^{valid} \neq \emptyset$ **then**
1480 /* Empirically valid operation found (at most one), reuse it */
1481 /* */
1482 5 Select $op_{selected}$ as the single element in \mathcal{OP}_v^{valid}
1483 **return** $op_{selected}$
1484 6 /* Otherwise, prompt LLM */
1485 7 **else**
1486 /* 1. Collect (item, valid operation) pairs into ItemValidOpPairs */
1487 /* */
1488 ItemValidOpPairs $\leftarrow \emptyset$
1489 8 **foreach** $u \in \text{keys}(M)$ **do**
1490 Retrieve history $M_u \leftarrow \{(op', n_{u,op'}^{succ}, n_{u,op'}^{fail})\}_{op' \in \mathcal{OP}}$
1491 9 **foreach** $op' \in \mathcal{OP}$ such that $n_{u,op'}^{succ} > 0 \wedge n_{u,op'}^{succ} - n_{u,op'}^{fail} > -x_0$ **do**
1492 10 Add (u, op') to ItemValidOpPairs
1493 11 /* 2. Find K pairs from ItemValidOpPairs whose items are Top-K similar to v */
1494 12 $\mathcal{V}_{with_valid_op} \leftarrow \{u \mid (u, op') \in \text{ItemValidOpPairs}\}$
1495 13 Find similar items $\mathcal{V}_K \subseteq \mathcal{V}_{with_valid_op}$ based on Top-K semantic similarity to v
1496 14 $\{(u_k, op_k)\}_{k=1}^K \leftarrow \{(u, op') \in \text{ItemValidOpPairs} \mid u \in \mathcal{V}_K\}$
1497 15 /* 3. Make candidate operations for LLM, excluding invalid ones for v */
1498 16 $\mathcal{OP}_v^{cand} \leftarrow \mathcal{OP} \setminus \mathcal{OP}_v^{invalid}$
1499 17 **if** $\mathcal{OP}_v^{cand} = \emptyset$ **then**
1500 18 $\mathcal{OP}_v^{cand} \leftarrow \mathcal{OP}$
1501 19 $op_{selected} \leftarrow LLM(v, \{(u_k, op_k)\}_{k=1}^K, \mathcal{OP}_v^{cand})$
1502 20 **return** $op_{selected}$

1503
1504
1505
1506
1507
1508
1509
1510
1511

```

1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527 1 For an item name, you need to make a plan, by selecting one among
1528   provided options.
1529 2 I will give you examples of which plans are needed to achieve an item,
1530   just for reference.
1531 3
1532 4 % Similar items and their successful plans are given
1533 5 [Example]
1534 6 <item name>
1535 7 {similar_item}
1536 8 <task planning>
1537 9 {successful_plan}
1538 10
1539 11 [Your turn]
1540 12 Here is <item name>, you MUST select one from below <options>, to make <
1541   task planning>.
1542 13 you MUST select one from below <options>. DO NOT MAKE A PLAN NOT IN <
1543   options>.
1544 14
1545 15 % Three actions are given, excluding any that were empirically invalid
1546 16 <options>:
1547 17 1: {"task": "dig down and mine {item}", "goal": [{item}, {quantity}]}
1548 18 2: {"task": "craft {item}", "goal": [{item}, {quantity}]}
1549 19 3: {"task": "smelt {item}", "item": [{item}, {quantity}]}
1550 20
1551 21 <item name>
1552 22 {item}
1553 23 <task planning>
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565

```

Figure 16: Prompt for action selection

1566 **G DIFFICULTY-BASED EXPLORATION (DEX)**
1567

1568 For autonomous dependency learning, we introduce DEX. DEX strategically selects items that (1)
 1569 appear easier to obtain, prioritizing those (2) under-explored for diversity and (3) having fewer
 1570 immediate prerequisite items according to the learned graph $\hat{\mathcal{G}}$. (line 5 in Algorithm 1). First, DEX
 1571 defines the previously unobtained items but whose required items are all obtained according to
 1572 learned dependency $\hat{\mathcal{G}}$ as the frontier F . Next, the least explored frontier set $\mathcal{F}_{min} := \{f \in F \mid$
 1573 $C(f) = \min_{f' \in F} C(f')\}$ is identified, based on revision counts $C(\cdot)$. For items $f' \in \mathcal{F}_{min}$, difficulty
 1574 $D(f')$ is estimated as $L_{f'}$, the number of distinct required items needed to obtain f' according to
 1575 $\hat{\mathcal{G}}$. The intrinsic goal g is then selected as the item in \mathcal{F}_{min} with the minimum estimated difficulty:
 1576 $g = \arg \min_{f' \in \mathcal{F}_{min}} D(f')$. Ties are broken uniformly at random.

1577 While our frontier concept is motivated by DECKARD (Nottingham et al., 2023), DEX’s selection
 1578 process differs significantly. DECKARD selects randomly from $\{v \in \mathcal{F} \mid C(v) \leq c_0\}$, but if this set
 1579 is empty, it selects randomly from the union of frontier set and previously obtained item set. This
 1580 risks inefficient attempts on already obtained items. In contrast, DEX exclusively selects goals from
 1581 \mathcal{F}_{min} , inherently avoiding obtained items. This efficiently guides exploration towards achievable,
 1582 novel dependencies.

1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

1620 **H CONTEXT-AWARE REPROMPTING (CRE)**
16211622 Minecraft, a real-world-like environment can lead to situations where the controller stalls (e.g., when
1623 stuck in deep water or a cave). To assist the controller, the agent provides temporary prompts to guide
1624 it (e.g., "get out of the water and find trees"). XENON proposes a context-aware reprompting scheme.
1625 It is inspired by Optimus-1 Li et al. (2024b) but introduces two key differences:
16261627 (a) **Two-stage reasoning.** When invoked, in Optimus-1, LLM simultaneously interprets
1628 image observations, decides whether to reprompt, and generates new prompts. XENON
1629 decomposes this process into two distinct steps:
1630 (i) the LLM generates a caption for the current image observation, and
1631 (ii) using *text-only* input (the generated caption and the current subgoal prompt), the LLM
1632 determines if reprompting is necessary and, if so, produces a temporary prompt.
1633 (b) **Trigger.** Unlike Optimus-1, which invokes the LLM at fixed intervals, XENON calls the
1634 LLM only if the current subgoal item has not been obtained within that interval. This
1635 approach avoids unnecessary or spurious interventions from a smaller LLM.
16361637 The prompt is shown in Figure 17.
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

```

1674 1 % Prompt for the first step: image captioning
1675 2 Given a Minecraft game image, describe nearby Minecraft objects, like
1676 3 tree, grass, cobblestone, etc.
1677 4 [Example]
1678 5 "There is a large tree with dark green leaves surrounding the area."
1679 6 "The image shows a dark, cave-like environment in Minecraft. The player
1680 7 is digging downwards. There are no visible trees or grass in this
1681 8 particular view."
1682 9 "The image shows a dark, narrow tunnel made of stone blocks. The player
1683 10 is digging downwards."
1684 11 [Your turn]
1685 12 Describe the given image, simply and clearly like the examples.
1686 13
1687 14 % Prompt for the second step: reasoning whether reprompting is needed or
1688 15 not
1689 16 Given <task> and <visual_description>, determine if the player needs
1690 17 intervention to achieve the goal. If intervention is needed, suggest
1691 18 a task that the player should perform.
1692 19 I will give you examples.
1693 20 [Example]
1694 21 <task>: chop tree
1695 22 <visual_description>: There is a large tree with dark green leaves
1696 23 surrounding the area.
1697 24 <goal_item>: logs
1698 25 <reasoning>:
1699 26 {{ {
1700 27     "need_intervention": false,
1701 28     "thoughts": "The player can see a tree and can chop it down to get
1702 29         logs.",
1703 30     "task": ""
1704 31 }}}
1705 32 [Example]
1706 33 <task>: chop tree
1707 34 <visual_description>: The image shows a dirt block in Minecraft. There is
1708 35 a tree in the image, but it is too far from here.
1709 36 <goal_item>: logs
1710 37 <reasoning>:
1711 38 {{ {
1712 39     "need_intervention": true,
1713 40     "thoughts": "The player is far from trees. The player needs to move
1714 41         to the trees.",
1715 42     "task": "explore to find trees",
1716 43 }}}
1717 44 [Example]
1718 45 <task>: dig down to mine iron_ore
1719 46 <visual_description>: The image shows a dark, narrow tunnel made of stone
1720 47 blocks. The player is digging downwards.
1721 48 <goal_item>: iron_ore
1722 49 <reasoning>:
1723 50 {{ {
1724 51     "need_intervention": false,
1725 52     "thoughts": "The player is already digging down and is likely to find
1726 53         iron ore.",
1727 54     "task": ""
1728 55 }}}
1729 56 [Your turn]
1730 57 Here is the <task>, <visual_description>, and <goal_item>.
1731 58 You MUST output the <reasoning> in JSON format.
1732 59 <task>: {task} % current prompt for the controller
1733 60 <visual_description>: {visual_description} % caption from the step 1
1734 61 <goal_item>: {goal_item} % current subgoal item
1735 62 <reasoning>:
1736 63
1737 64
1738 65
1739 66
1740 67
1741 68
1742 69
1743 70
1744 71
1745 72
1746 73
1747 74
1748 75
1749 76
1750 77
1751 78
1752 79
1753 80
1754 81
1755 82
1756 83
1757 84
1758 85
1759 86
1760 87
1761 88
1762 89
1763 90
1764 91
1765 92
1766 93
1767 94
1768 95
1769 96
1770 97
1771 98
1772 99
1773 100
1774 101
1775 102
1776 103
1777 104
1778 105
1779 106
1780 107
1781 108
1782 109
1783 110
1784 111
1785 112
1786 113
1787 114
1788 115
1789 116
1790 117
1791 118
1792 119
1793 120
1794 121
1795 122
1796 123
1797 124
1798 125
1799 126
1800 127
1801 128
1802 129
1803 130
1804 131
1805 132
1806 133
1807 134
1808 135
1809 136
1810 137
1811 138
1812 139
1813 140
1814 141
1815 142
1816 143
1817 144
1818 145
1819 146
1820 147
1821 148
1822 149
1823 150
1824 151
1825 152
1826 153
1827 154
1828 155
1829 156
1830 157
1831 158
1832 159
1833 160
1834 161
1835 162
1836 163
1837 164
1838 165
1839 166
1840 167
1841 168
1842 169
1843 170
1844 171
1845 172
1846 173
1847 174
1848 175
1849 176
1850 177
1851 178
1852 179
1853 180
1854 181
1855 182
1856 183
1857 184
1858 185
1859 186
1860 187
1861 188
1862 189
1863 190
1864 191
1865 192
1866 193
1867 194
1868 195
1869 196
1870 197
1871 198
1872 199
1873 200
1874 201
1875 202
1876 203
1877 204
1878 205
1879 206
1880 207
1881 208
1882 209
1883 210
1884 211
1885 212
1886 213
1887 214
1888 215
1889 216
1890 217
1891 218
1892 219
1893 220
1894 221
1895 222
1896 223
1897 224
1898 225
1899 226
1900 227
1901 228
1902 229
1903 230
1904 231
1905 232
1906 233
1907 234
1908 235
1909 236
1910 237
1911 238
1912 239
1913 240
1914 241
1915 242
1916 243
1917 244
1918 245
1919 246
1920 247
1921 248
1922 249
1923 250
1924 251
1925 252
1926 253
1927 254
1928 255
1929 256
1930 257
1931 258
1932 259
1933 260
1934 261
1935 262
1936 263
1937 264
1938 265
1939 266
1940 267
1941 268
1942 269
1943 270
1944 271
1945 272
1946 273
1947 274
1948 275
1949 276
1950 277
1951 278
1952 279
1953 280
1954 281
1955 282
1956 283
1957 284
1958 285
1959 286
1960 287
1961 288
1962 289
1963 290
1964 291
1965 292
1966 293
1967 294
1968 295
1969 296
1970 297
1971 298
1972 299
1973 300
1974 301
1975 302
1976 303
1977 304
1978 305
1979 306
1980 307
1981 308
1982 309
1983 310
1984 311
1985 312
1986 313
1987 314
1988 315
1989 316
1990 317
1991 318
1992 319
1993 320
1994 321
1995 322
1996 323
1997 324
1998 325
1999 326
2000 327
2001 328
2002 329
2003 330
2004 331
2005 332
2006 333
2007 334
2008 335
2009 336
2010 337
2011 338
2012 339
2013 340
2014 341
2015 342
2016 343
2017 344
2018 345
2019 346
2020 347
2021 348
2022 349
2023 350
2024 351
2025 352
2026 353
2027 354
2028 355
2029 356
2030 357
2031 358
2032 359
2033 360
2034 361
2035 362
2036 363
2037 364
2038 365
2039 366
2040 367
2041 368
2042 369
2043 370
2044 371
2045 372
2046 373
2047 374
2048 375
2049 376
2050 377
2051 378
2052 379
2053 380
2054 381
2055 382
2056 383
2057 384
2058 385
2059 386
2060 387
2061 388
2062 389
2063 390
2064 391
2065 392
2066 393
2067 394
2068 395
2069 396
2070 397
2071 398
2072 399
2073 400
2074 401
2075 402
2076 403
2077 404
2078 405
2079 406
2080 407
2081 408
2082 409
2083 410
2084 411
2085 412
2086 413
2087 414
2088 415
2089 416
2090 417
2091 418
2092 419
2093 420
2094 421
2095 422
2096 423
2097 424
2098 425
2099 426
2100 427
2101 428
2102 429
2103 430
2104 431
2105 432
2106 433
2107 434
2108 435
2109 436
2110 437
2111 438
2112 439
2113 440
2114 441
2115 442
2116 443
2117 444
2118 445
2119 446
2120 447
2121 448
2122 449
2123 450
2124 451
2125 452
2126 453
2127 454
2128 455
2129 456
2130 457
2131 458
2132 459
2133 460
2134 461
2135 462
2136 463
2137 464
2138 465
2139 466
2140 467
2141 468
2142 469
2143 470
2144 471
2145 472
2146 473
2147 474
2148 475
2149 476
2150 477
2151 478
2152 479
2153 480
2154 481
2155 482
2156 483
2157 484
2158 485
2159 486
2160 487
2161 488
2162 489
2163 490
2164 491
2165 492
2166 493
2167 494
2168 495
2169 496
2170 497
2171 498
2172 499
2173 500
2174 501
2175 502
2176 503
2177 504
2178 505
2179 506
2180 507
2181 508
2182 509
2183 510
2184 511
2185 512
2186 513
2187 514
2188 515
2189 516
2190 517
2191 518
2192 519
2193 520
2194 521
2195 522
2196 523
2197 524
2198 525
2199 526
2200 527
2201 528
2202 529
2203 530
2204 531
2205 532
2206 533
2207 534
2208 535
2209 536
2210 537
2211 538
2212 539
2213 540
2214 541
2215 542
2216 543
2217 544
2218 545
2219 546
2220 547
2221 548
2222 549
2223 550
2224 551
2225 552
2226 553
2227 554
2228 555
2229 556
2230 557
2231 558
2232 559
2233 560
2234 561
2235 562
2236 563
2237 564
2238 565
2239 566
2240 567
2241 568
2242 569
2243 570
2244 571
2245 572
2246 573
2247 574
2248 575
2249 576
2250 577
2251 578
2252 579
2253 580
2254 581
2255 582
2256 583
2257 584
2258 585
2259 586
2260 587
2261 588
2262 589
2263 590
2264 591
2265 592
2266 593
2267 594
2268 595
2269 596
2270 597
2271 598
2272 599
2273 600
2274 601
2275 602
2276 603
2277 604
2278 605
2279 606
2280 607
2281 608
2282 609
2283 610
2284 611
2285 612
2286 613
2287 614
2288 615
2289 616
2290 617
2291 618
2292 619
2293 620
2294 621
2295 622
2296 623
2297 624
2298 625
2299 626
2300 627
2301 628
2302 629
2303 630
2304 631
2305 632
2306 633
2307 634
2308 635
2309 636
2310 637
2311 638
2312 639
2313 640
2314 641
2315 642
2316 643
2317 644
2318 645
2319 646
2320 647
2321 648
2322 649
2323 650
2324 651
2325 652
2326 653
2327 654
2328 655
2329 656
2330 657
2331 658
2332 659
2333 660
2334 661
2335 662
2336 663
2337 664
2338 665
2339 666
2340 667
2341 668
2342 669
2343 670
2344 671
2345 672
2346 673
2347 674
2348 675
2349 676
2350 677
2351 678
2352 679
2353 680
2354 681
2355 682
2356 683
2357 684
2358 685
2359 686
2360 687
2361 688
2362 689
2363 690
2364 691
2365 692
2366 693
2367 694
2368 695
2369 696
2370 697
2371 698
2372 699
2373 700
2374 701
2375 702
2376 703
2377 704
2378 705
2379 706
2380 707
2381 708
2382 709
2383 710
2384 711
2385 712
2386 713
2387 714
2388 715
2389 716
2390 717
2391 718
2392 719
2393 720
2394 721
2395 722
2396 723
2397 724
2398 725
2399 726
2400 727
2401 728
2402 729
2403 730
2404 731
2405 732
2406 733
2407 734
2408 735
2409 736
2410 737
2411 738
2412 739
2413 740
2414 741
2415 742
2416 743
2417 744
2418 745
2419 746
2420 747
2421 748
2422 749
2423 750
2424 751
2425 752
2426 753
2427 754
2428 755
2429 756
2430 757
2431 758
2432 759
2433 760
2434 761
2435 762
2436 763
2437 764
2438 765
2439 766
2440 767
2441 768
2442 769
2443 770
2444 771
2445 772
2446 773
2447 774
2448 775
2449 776
2450 777
2451 778
2452 779
2453 780
2454 781
2455 782
2456 783
2457 784
2458 785
2459 786
2460 787
2461 788
2462 789
2463 790
2464 791
2465 792
2466 793
2467 794
2468 795
2469 796
2470 797
2471 798
2472 799
2473 800
2474 801
2475 802
2476 803
2477 804
2478 805
2479 806
2480 807
2481 808
2482 809
2483 810
2484 811
2485 812
2486 813
2487 814
2488 815
2489 816
2490 817
2491 818
2492 819
2493 820
2494 821
2495 822
2496 823
2497 824
2498 825
2499 826
2500 827
2501 828
2502 829
2503 830
2504 831
2505 832
2506 833
2507 834
2508 835
2509 836
2510 837
2511 838
2512 839
2513 840
2514 841
2515 842
2516 843
2517 844
2518 845
2519 846
2520 847
2521 848
2522 849
2523 850
2524 851
2525 852
2526 853
2527 854
2528 855
2529 856
2530 857
2531 858
2532 859
2533 860
2534 861
2535 862
2536 863
2537 864
2538 865
2539 866
2540 867
2541 868
2542 869
2543 870
2544 871
2545 872
2546 873
2547 874
2548 875
2549 876
2550 877
2551 878
2552 879
2553 880
2554 881
2555 882
2556 883
2557 884
2558 885
2559 886
2560 887
2561 888
2562 889
2563 890
2564 891
2565 892
2566 893
2567 894
2568 895
2569 896
2570 897
2571 898
2572 899
2573 900
2574 901
2575 902
2576 903
2577 904
2578 905
2579 906
2580 907
2581 908
2582 909
2583 910
2584 911
2585 912
2586 913
2587 914
2588 915
2589 916
2590 917
2591 918
2592 919
2593 920
2594 921
2595 922
2596 923
2597 924
2598 925
2599 926
2600 927
2601 928
2602 929
2603 930
2604 931
2605 932
2606 933
2607 934
2608 935
2609 936
2610 937
2611 938
2612 939
2613 940
2614 941
2615 942
2616 943
2617 944
2618 945
2619 946
2620 947
2621 948
2622 949
2623 950
2624 951
2625 952
2626 953
2627 954
2628 955
2629 956
2630 957
2631 958
2632 959
2633 960
2634 961
2635 962
2636 963
2637 964
2638 965
2639 966
2640 967
2641 968
2642 969
2643 970
2644 971
2645 972
2646 973
2647 974
2648 975
2649 976
2650 977
2651 978
2652 979
2653 980
2654 981
2655 982
2656 983
2657 984
2658 985
2659 986
2660 987
2661 988
2662 989
2663 990
2664 991
2665 992
2666 993
2667 994
2668 995
2669 996
2670 997
2671 998
2672 999
2673 1000
2674 1001
2675 1002
2676 1003
2677 1004
2678 1005
2679 1006
2680 1007
2681 1008
2682 1009
2683 1010
2684 1011
2685 1012
2686 1013
2687 1014
2688 1015
2689 1016
2690 1017
2691 1018
2692 1019
2693 1020
2694 1021
2695 1022
2696 1023
2697 1024
2698 1025
2699 1026
2700 1027
2701 1028
2702 1029
2703 1030
2704 1031
2705 1032
2706 1033
2707 1034
2708 1035
2709 1036
2710 1037
2711 1038
2712 1039
2713 1040
2714 1041
2715 1042
2716 1043
2717 1044
2718 1045
2719 1046
2720 1047
2721 1048
2722 1049
2723 1050
2724 1051
2725 1052
2726 1053
2727 1054
2728 1055
2729 1056
2730 1057
2731 1058
2732 1059
2733 1060
2734 1061
2735 1062
2736 1063
2737 1064
2738 1065
2739 1066
2740 1067
2741 1068
2742 1069
2743 1070
2744 1071
2745 1072
2746 1073
2747 1074
2748 1075
2749 1076
2750 1077
2751 1078
2752 1079
2753 1080
2754 1081
2755 1082
2756 1083
2757 1084
2758 1085
2759 1086
2760 1087
2761 1088
2762 1089
2763 1090
2764 1091
2765 1092
2766 1093
2767 1094
2768 1095
2769 1096
2770 1097
2771 1098
2772 1099
2773 1100
2774 1101
2775 1102
2776 1103
2777 1104
2778 1105
2779 1106
2780 1107
2781 1108
2782 1109
2783 1110
2784 1111
2785 1112
2786 1113
2787 1114
2788 1115
2789 1116
2790 1117
2791 1118
2792 1119
2793 1120
2794 1121
2795 1122
2796 1123
2797 1124
2798 1125
2799 1126
2800 1127
2801 1128
2802 1129
2803 1130
2804 1131
2805 1132
2806 1133
2807 1134
2808 1135
2809 1136
2810 1137
2811 1138
2812 1139
2813 1140
2814 1141
2815 1142
2816 1143
2817 1144
2818 1145
2819 1146
2820 1147
2821 1148
2822 1149
2823 1150
2824 1151
2825 1152
2826 1153
2827 1154
2828 1155
2829 1156
2830 1157
2831 1158
2832 1159
2833 1160
2834 1161
2835 1162
2836 1163
2837 1164
2838 1165
2839 1166
2840 1167
2841 1168
2842 1169
2843 1170
2844 1171
2845 1172
2846 1173
2847 1174
2848 1175
2849 1176
2850 1177
2851 1178
2852 1179
2853 1180
2854 1181
2855 1182
2856 1183
2857 1184
2858 1185
2859 1186
2860 1187
2861 1188
2862 
```

1728

I IMPLEMENTATION DETAILS

1729
 1730 To identify similar items, semantic similarity between two items is computed as the cosine similarity
 1731 of their Sentence-BERT (all-MiniLM-L6-v2 model) embeddings (Reimers & Gurevych, 2019).
 1732 This metric is utilized whenever item similarity comparisons are needed, such as in Algorithm 2,
 1733 Algorithm 3, and Algorithm 4.

1734

I.1 HYPERPARAMETERS

1735 Table 7: Hyperparameters used in our experiments.
 1736

1739 Hyperparameter	1740 Notation	1741 Value
1741 Failure threshold for invalid action	x_0	-2
1742 Revision count threshold for inadmissible items	c_0	3
1743 Required items quantity scale for less explored items	α_s	2
1744 Required items quantity scale for inadmissible items	α_i	8
1745 Number of top-K similar experienced items used	K	3

1746 For all experiments, we use consistent hyperparameters across environments. The hyperparameters,
 1747 whose values are determined with mainly considering robustness against imperfect controllers. All
 1748 hyperparameters are listed in Table 7. The implications of increasing each hyperparameter’s value
 1749 are detailed below:

- 1750 • x_0 (failure threshold for empirically invalid action): Prevents valid operations from being
 1751 misclassified as invalid due to accidental failures from an imperfect controller or environmental
 1752 stochasticity. Values that are too small or large hinder dependency learning and
 1753 planning by hampering the discovery of valid actions.
- 1754 • c_0 (exploration count threshold for inadmissible items): Ensures an item is sufficiently
 1755 attempted before being deemed ‘inadmissible’ and triggering a revision for its descendants.
 1756 Too small/large values could cause inefficiency; small values prematurely abandon potentially
 1757 correct LLM predictions for descendants, while large values prevent attempts on
 1758 descendant items.
- 1759 • α_s (required items quantity scale for less explored items): Controls the gradual increase of
 1760 required quantities for revised required items. Small values make learning inefficient by
 1761 hindering item obtaining due to insufficient required items, yet large values lower robustness
 1762 by overburdening controllers with excessive quantity demands.
- 1763 • α_i (required items quantity scale for inadmissible items): Ensures sufficient acquisition of
 1764 potential required items before retrying inadmissible items to increase the chance of success.
 1765 Improper values reduce robustness; too small leads to failure in obtaining items necessitating
 1766 many items; too large burdens controllers with excessive quantity demands.
- 1767 • K (Number of similar items to retrieve): Determines how many similar, previously suc-
 1768 cessful experiences are retrieved to inform dependency revision (Algorithm 3) and action
 1769 selection (Algorithm 4).

1771

I.2 HUMAN-WRITTEN PLANS

1772 We utilize three human-written plans (for iron sword, golden sword, and diamond, shown in Plan 18,
 1773 19, and 20, respectively), the format of which is borrowed from the human-written plan examples in
 1774 the publicly released Optimus-1 repository³. We leverage the experiences gained from executing
 1775 these plans to initialize XENON’s knowledge.

1776
 1777
 1778
 1779
 1780
 1781 ³<https://github.com/JiuTian-VL/Optimus-1/blob/main/src/optimus1/example.py>

```

1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795 1 iron_sword: str = """
1796 2 <goal>: craft an iron sword.
1797 3 <requirements>:
1798 4 1. log: need 7
1799 5 2. planks: need 21
1800 6 3. stick: need 5
1801 7 4. crafting_table: need 1
1802 8 5. wooden_pickaxe: need 1
1803 9 6. cobblestone: need 11
1804 10 7. furnace: need 1
1805 11 8. stone_pickaxe: need 1
1806 12 9. iron_ore: need 2
1807 13 10. iron_ingot: need 2
1808 14 11. iron_sword: need 1
1809 15 <plan>
1810 16 {
1811 17 "step 1": {"prompt": "mine logs", "item": ["logs", 7]},
1812 18 "step 2": {"prompt": "craft planks", "item": ["planks", 21]},
1813 19 "step 3": {"prompt": "craft stick", "item": ["stick", 5]},
1814 20 "step 4": {"prompt": "craft crafting_table", "item": ["crafting_table",
1815 21 1]},
1816 22 "step 5": {"prompt": "craft wooden_pickaxe", "item": ["wooden_pickaxe",
1817 23 1]},
1818 24 "step 6": {"prompt": "mine cobblestone", "item": ["cobblestone", 11]},
1819 25 "step 7": {"prompt": "craft furnace", "item": ["furnace", 1]},
1820 26 "step 8": {"prompt": "craft stone_pickaxe", "item": ["stone_pickaxe",
1821 27 1]},
1822 28 "step 9": {"prompt": "mine iron_ore", "item": ["iron_ore", 2]},
1823 29 "step 10": {"prompt": "smelt iron_ingot", "item": ["iron_ingot", 2]},
1824 30 "step 11": {"prompt": "craft iron_sword", "item": ["iron_sword", 1]}
1825 31 }
1826 32 """
1827
1828
1829
1830
1831
1832
1833
1834
1835

```

Figure 18: Human-written plan for crafting an iron sword.

```

1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846 1 golden_sword: str = """
1847 2 <goal>: craft a golden sword.
1848 3 <requirements>:
1849 4 1. log: need 9
1850 5 2. planks: need 27
1851 6 3. stick: need 7
1852 7 4. crafting_table: need 1
1853 8 5. wooden_pickaxe: need 1
1854 9 6. cobblestone: need 11
1855 10 7. furnace: need 1
1856 11 8. stone_pickaxe: need 1
1857 12 9. iron_ore: need 3
1858 13 10. iron_ingot: need 3
1859 14 11. iron_pickaxe: need 1
1860 15 12. gold_ore: need 2
1861 16 13. gold_ingot: need 2
1862 17 14. golden_sword: need 1
1863 18 <plan>
1864 19 {
1865 20   "step 1": {"prompt": "mine logs", "item": ["logs", 7]},
1866 21   "step 2": {"prompt": "craft planks", "item": ["planks", 21]},
1867 22   "step 3": {"prompt": "craft stick", "item": ["stick", 5]},
1868 23   "step 4": {"prompt": "craft crafting_table", "item": ["crafting_table",
1869 24     1]},
1870 25   "step 5": {"prompt": "craft wooden_pickaxe", "item": ["wooden_pickaxe",
1871 26     1]},
1872 27   "step 6": {"prompt": "mine cobblestone", "item": ["cobblestone", 11]},
1873 28   "step 7": {"prompt": "craft furnace", "item": ["furnace", 1]},
1874 29   "step 8": {"prompt": "craft stone_pickaxe", "item": ["stone_pickaxe",
1875 30     1]},
1876 31   "step 9": {"prompt": "mine iron_ore", "item": ["iron_ore", 3]},
1877 32   "step 10": {"prompt": "smelt iron_ingot", "item": ["iron_ingot", 3]},
1878 33   "step 11": {"task": "craft iron_pickaxe", "goal": ["iron_pickaxe", 1]},
1879 34   "step 12": {"prompt": "mine gold_ore", "item": ["gold_ore", 2]},
1880 35   "step 13": {"prompt": "smelt gold_ingot", "item": ["gold_ingot", 2]},
1881   "step 14": {"task": "craft golden_sword", "goal": ["golden_sword", 1]}
1882 }
1883 """
1884
1885
1886
1887
1888
1889

```

Figure 19: Human-written plan for crafting a golden sword.

```

1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902 1 diamond: str = """
1903 2 <goal>: mine a diamond.
1904 3 <requirements>:
1905 4 1. log: need 7
1906 5 2. planks: need 21
1907 6 3. stick: need 6
1908 7 4. crafting_table: need 1
1909 8 5. wooden_pickaxe: need 1
1910 9 6. cobblestone: need 11
1911 10 7. furnace: need 1
1912 11 8. stone_pickaxe: need 1
1913 12 9. iron_ore: need 3
1914 13 10. iron_ingot: need 3
1915 14 11. iron_pickaxe: need 1
1916 15 12. diamond: need 1
1917 16 <plan>
1918 17 {
1919 18 "step 1": {"prompt": "mine logs", "item": ["logs", 7]},
1920 19 "step 2": {"prompt": "craft planks", "item": ["planks", 21]},
1921 20 "step 3": {"prompt": "craft stick", "item": ["stick", 5]},
1922 21 "step 4": {"prompt": "craft crafting_table", "item": ["crafting_table",
1923 22 1]},
1924 23 "step 5": {"prompt": "craft wooden_pickaxe", "item": ["wooden_pickaxe",
1925 24 1]},
1926 25 "step 6": {"prompt": "mine cobblestone", "item": ["cobblestone", 11]},
1927 26 "step 7": {"prompt": "craft furnace", "item": ["furnace", 1]},
1928 27 "step 8": {"prompt": "craft stone_pickaxe", "item": ["stone_pickaxe",
1929 28 1]},
1930 29 "step 9": {"prompt": "mine iron_ore", "item": ["iron_ore", 2]},
1931 30 "step 10": {"prompt": "smelt iron_ingot", "item": ["iron_ingot", 2]},
1932 31 "step 11": {"prompt": "craft iron_pickaxe", "item": ["iron_pickaxe", 1]},
1933 32 "step 12": {"prompt": "mine diamond", "item": ["diamond", 1]}
1934 33 }
1935 34 """
1936
1937
1938
1939
1940
1941
1942
1943

```

Figure 20: Human-written plan for mining a diamond.

1944

J DETAILS FOR EXPERIMENTAL SETUP

1945

1946

1947

J.1 COMPARED BASELINES FOR DEPENDENCY LEARNING

1948

We compare our proposed method, XENON, against four baselines: LLM self-correction (SC), DECKARD Nottingham et al. (2023), ADAM (Yu & Lu, 2024), and RAND (the simplest baseline). As no prior baselines were evaluated under our specific experimental setup (i.e., empty initial inventory, pre-trained low-level controller), we adapted their implementation to align with our environment. SC is implemented following common methods that prompt the LLM to correct its own knowledge upon plan failures (Shinn et al., 2023; Stechly et al., 2024). A summary of all methods compared in our experiments is provided in Table 8. All methods share the following common experimental setting: each episode starts with an initial experienced requirements for some items, derived from human-written plans (details in Appendix I). Additionally, all agents begin each episode with an initial empty inventory.

1958

1959

1960

Table 8: Summary of methods compared in our experiments.

1961

1962

1963

1964

1965

1966

Method	Predicted Requirement Set	Action Memory	Intrinsic Goal Selection
XENON	LLM-generated (with revision)	Success & Failure	DEX
SC	LLM-generated (with revision)	Success & Failure	Random
ADAM Yu & Lu (2024)	“8 × resources”	Success-only	Random
DECKARD Nottingham et al. (2023)	LLM-generated (without revision)	Success-only	Frontier + obtained items
RAND	LLM-generated (without revision)	None	Random

LLM self-correction (SC). While no prior work specifically uses LLM self-correction to learn Minecraft item dependencies in our setting, we include this baseline to demonstrate the unreliability of this approach. For predicted requirements, similar to XENON, SC initializes its dependency graph with LLM-predicted requirements for each item. When a plan for an item fails repeatedly, it attempts to revise the requirements using LLM. SC prompts the LLM itself to perform the correction, providing it with recent trajectories and the validated requirements of similar, previously obtained items in the input prompt. SC’s action memory stores both successful and failed actions for each item. Upon a plan failure, the LLM is prompted to self-reflect on the recent trajectory to determine the cause of failure. When the agent later plans to obtain an item on which it previously failed, this reflection is included in the LLM’s prompt to guide its action selection. Intrinsic goals are selected randomly from the set of previously unobtained items. The specific prompts used for the LLM self-correction and self-reflection in this baseline are provided in Appendix B.

DECKARD. The original DECKARD utilizes LLM-predicted requirements for each item but does not revise these initial predictions. It has no explicit action memory for the planner; instead, it trains and maintains specialized policies for each obtained item. It selects an intrinsic goal randomly from less explored frontier items (i.e., $\{v \in \mathcal{F} \mid C(v) \leq c_0\}$). If no such items are available, it selects randomly from the union of experienced items and all frontier items.

In our experiments, the DECKARD baseline is implemented to largely mirror the original version, with the exception of its memory system. Its memory is implemented to store only successful actions without recording failures. This design choice aligns with the original DECKARD’s approach, which, by only learning policies for successfully obtained items, lacks policies for unobtained items.

ADAM. The original ADAM started with an initial inventory containing 32 quantities of experienced resource items (i.e., items used for crafting other items) and 1 quantity of tool items (e.g., pickaxes, crafting table), implicitly treating those items as a predicted requirement set for each item. Its memory recorded which operations were used for each subgoal item without noting success or failure, and its intrinsic goal selection was guided by an expert-defined exploration curriculum.

In our experiments, ADAM starts with an empty initial inventory. The predicted requirements for each goal item in our ADAM implementation assume a fixed quantity of 8 for all resource items. This quantity was chosen to align with α_i , the hyperparameter for the quantity scale of requirement items for inadmissible items, thereby ensuring a fair comparison with XENON. The memory stores successful operations for each item, but did not record failures. This modification aligns the memory mechanism with SC and DECKARD baselines, enabling a more consistent comparison across

1998 baselines in our experimental setup. Intrinsic goal selection is random, as we do not assume such an
 1999 expert-defined exploration curriculum.
 2000

2001 **RAND.** RAND is a simple baseline specifically designed for our experimental setup. It started
 2002 with an empty initial inventory and an LLM-predicted requirement set for each item. RAND did
 2003 not incorporate any action memory. Its intrinsic goal selection involved randomly selecting from
 2004 unexperienced items.

2005 **J.2 MINERL ENVIRONMENT**

2006 **J.2.1 BASIC RULES**

2009 Minecraft has been adopted as a suitable testbed for validating performance of AI agents on long-
 2010 horizon tasks (Mao et al., 2022; Lin et al., 2021; Baker et al., 2022; Li et al., 2025a), largely because
 2011 of the inherent dependency in item acquisition where agents must obtain prerequisite items before
 2012 more advanced ones. Specifically, Minecraft features multiple technology levels—including wood,
 2013 stone, iron, gold, diamond, etc. —which dictate item and tool dependencies. For instance, an agent
 2014 must first craft a lower-level tool like a wooden pickaxe to mine materials such as stone. Subsequently,
 2015 a stone pickaxe is required to mine even higher-level materials like iron. An iron pickaxe is required
 2016 to mine materials like gold and diamond. Respecting the dependency is crucial for achieving complex
 2017 goals, such as crafting an iron sword or mining a diamond.

2018 **J.2.2 OBSERVATION AND ACTION SPACE**

2019 First, we employ MineRL (Guss et al., 2019) with Minecraft version 1.16.5.

2022 **Observation.** When making a plan, our agent receives inventory information (i.e., item with their
 2023 quantities) as text. When executing the plan, our agent receives an RGB image with dimensions of
 2024 640×360 , including the hotbar, health indicators, food saturation, and animations of the player’s
 2025 hands.

2026 **Action space.** Following Optimus-1 (Li et al., 2024b), our low-level action space primarily consists
 2027 of keyboard and mouse controls, except for craft and smelt high-level actions. Crucially, craft and
 2028 smelt actions are included into our action space, following (Li et al., 2024b). This means these
 2029 high-level actions automatically succeed in producing an item if the agent possesses all the required
 2030 items and a valid actions for that item is chosen; otherwise, they fail. This abstraction removes the
 2031 need for complex, precise low-level mouse control for these specific actions. For low-level controls,
 2032 keyboard presses control agent movement (e.g., jumping, moving forward, backward) and mouse
 2033 movements control the agent’s perspective. The mouse’s left and right buttons are used for attacking,
 2034 using, or placing items. The detailed action space is described in Table 9.

2035 **Table 9: Action space in MineRL environment**

2038 Index	2039 Action	2040 Human Action	2041 Description
1	Forward	key W	Move forward.
2	Back	key S	Move back.
3	Left	key A	Move left.
4	Right	key D	Move right.
5	Jump	key Space	Jump. When swimming, keeps the player afloat.
6	Sneak	key left Shift	Slowly move in the current direction of movement.
7	Sprint	key left Ctrl	Move quickly in the direction of current movement.
8	Attack	left Button	Destroy blocks (hold down); Attack entity (click once).
9	Use	right Button	Place blocks, entity, open items or other interact actions defined by game.
10	hotbar [1-9]	keys 1-9	Selects the appropriate hotbar item.
11	Open/Close Inventory	key E	Opens the Inventory. Close any open GUI.
12	Yaw	move Mouse X	Turning; aiming; camera movement. Ranging from -180 to +180.
13	Pitch	move Mouse Y	Turning; aiming; camera movement. Ranging from -180 to +180.
14	Craft	-	Execute crafting to obtain new item
15	Smelt	-	Execute smelting to obtain new item.

2052 J.2.3 GOALS
2053

2054 We consider 67 goals from the long-horizon tasks benchmark suggested in (Li et al., 2024b). These
2055 goals are categorized into 7 groups based on Minecraft’s item categories: Wood , Stone , Iron ,
2056 Gold , Diamond , Redstone , and Armor . All goal items within each group are listed
2057 in Table 10.

2058 Table 10: Setting of 7 groups encompassing 67 Minecraft long-horizon goals.
2059

Group	Goal Num.	All goal items
 Wood	10	bowl, crafting_table, chest, ladder, stick, wooden_axe, wooden_hoe, wooden_pickaxe, wooden_shovel, wooden_sword
 Stone	9	charcoal, furnace, smoker, stone_axe, stone_hoe, stone_pickaxe, stone_shovel, stone_sword, torch
 Iron	16	blast_furnace, bucket, chain, hopper, iron_axe, iron_bars, iron_hoe, iron_nugget, iron_pickaxe, iron_shovel, iron_sword, rail, shears, smithing_table, stonecutter, tripwire_hook
 Gold	6	gold_ingot, golden_axe, golden_hoe, golden_pickaxe, golden_shovel, golden_sword
 Redstone	6	activator_rail, compass, dropper, note_block, piston, redstone_torch
 Diamond	7	diamond, diamond_axe, diamond_hoe, diamond_pickaxe, diamond_shovel, diamond_sword, jukebox
 Armor	13	diamond_boots, diamond_chestplate, diamond_helmet, diamond_leggings, golden_boots, golden_chestplate, golden_helmet, golden_leggings, iron_boots, iron_chestplate, iron_helmet, iron_leggings, shield

2083 **Additional goals for scalability experiments.** To evaluate the scalability of XENON with respect to
2084 the number of goals Appendix K.9, we extend the above 67-goal set (Table 10) by adding additional
2085 goal items to construct two larger settings with 100 and 120 goals; the added goals are listed in
2086 Table 11.

2087 Specifically, in the setting with 100 goals, we add 33 goals in total by introducing new “leather”,
2088 “paper”, and “flint” groups and by adding more items to the existing “wood” and “stone” groups. In
2089 the setting with 120 goals, we further add 20 goals in the “iron”, “gold”, “redstone”, and “diamond”
2090 groups.

2092 J.2.4 EPISODE HORIZON
2093

2094 The episode horizon varies depending on the experiment phase: dependency learning or long-
2095 horizon goal planning. During the dependency learning phase, each episode has a fixed horizon of
2096 36,000 steps. In this phase, if the agent successfully achieves an intrinsic goal within an episode,
2097 it is allowed to select another intrinsic goal and continue exploration without the episode ending.
2098 After dependency learning, when measuring the success rate of goals from the long-horizon task
2099 benchmark, the episode horizon differs based on the goal’s category group. And in this phase, the
2100 episode immediately terminates upon success of a goal. The specific episode horizons for each group
2101 are as follows: Wood: 3,600 steps; Stone: 7,200 steps; Iron: 12,000 steps; and Gold, Diamond,
2102 Redstone, and Armor: 36,000 steps each.

2103 J.2.5 ITEM SPAWN PROBABILITY DETAILS
2104

2105 Following Optimus-1’s public implementation, we have modified environment configuration different
from original MineRL environment (Guss et al., 2019). In Minecraft, obtaining essential resources

2106 Table 11: Additional goals used for the scalability experiments. The setting with 100 goals extends
 2107 the 67-goal set in Table 10 by adding all items in the top block; the setting with 120 goals further
 2108 includes both the top and bottom blocks.

Group	Goal Num.	Added goal items
<i>Additional items in the setting with 100 goals (33 items)</i>		
leather	7	leather, leather_boots, leather_chestplate, leather_helmet, leather_leggings, leather_horse_armor, item_frame
paper	5	map, book, cartography_table, bookshelf, lectern
flint	4	flint, flint_and_steel, fletching_table, arrow
wood	8	bow, boat, wooden_slab, wooden_stairs, wooden_door, wooden_sign, wooden_fence, woodenfence_gate
stone	9	cobblestone_slab, cobblestone_stairs, cobblestone_wall, lever, stone_slab, stone_button, stone_pressure_plate, stone_bricks, grindstone
<i>Additional items only in the setting with 120 goals (20 more items)</i>		
iron	7	iron_trapdoor, heavy_weighted_pressure_plate, iron_door, crossbow, minecart, cauldron, lantern
gold	4	gold_nugget, light_weighted_pressure_plate, golden_apple, golden_carrot
redstone	7	redstone, powered_rail, target, dispenser, clock, repeater, detector_rail
diamond	2	obsidian, enchanting_table

2135 such as iron, gold, and diamond requires mining their respective ores. However, these ores are
 2136 naturally rare, making them challenging to obtain. This inherent difficulty can significantly hinder an
 2137 agent’s goal completion, even with an accurate plan. **This challenge in resource gathering due to an**
 2138 **imperfect controller is a common bottleneck**, leading many prior works to employ environmental
 2139 modifications to focus on planning. For example, DEPS (Wang et al., 2023b) restricts the controller’s
 2140 actions based on the goal items⁴. Optimus-1 (Li et al., 2024b) also made resource items easier to
 2141 obtain by increasing item ore spawn probabilities. To focus on our primary goal of robust planning
 2142 and isolate this challenge, we follow Optimus-1 and adopt its item ore spawn procedure directly from
 2143 the publicly released Optimus-1 repository, without any modifications to its source code⁵.

2144 The ore spawn procedure probabilistically spawns ore blocks in the vicinity of the agent’s current
 2145 coordinates (x, y, z) . Specifically, at each timestep, the procedure has a 10% chance of activating.
 2146 When activated, it spawns a specific type of ore block based on the agent’s y -coordinate. Furthermore,
 2147 for any given episode, the procedure is not activate more than once at the same y -coordinate. The
 2148 types of ore blocks spawned at different y -levels are as follows:

-  **Coal Ore**: between $y=45$ and $y=50$.
-  **Iron Ore**: between $y=26$ and $y=43$.
-  **Gold Ore**: between $y=15$ and $y=26$
-  **Redstone Ore**: between $y=15$ and $y=26$
-  **Diamond Ore**: below $y=14$

2158 ⁴<https://github.com/CraftJarvis/MC-Planner/blob/main/controller.py>

2159 ⁵<https://github.com/JiuTian-VL/Optimus-1/blob/main/src/optimus1/env/wrapper.py>

2160 J.3 MINEFLAYER ENVIRONMENT
21612162 We use the Mineflayer (PrismarineJS, 2023) environment with Minecraft version 1.19. In Mineflayer,
2163 resource item spawn probabilities do not need to be adjusted, unlike in MineRL Appendix J.2.5.
2164 This is because the controller, JavaScript APIs provided by Mineflayer, is competent to gather many
2165 resource items.2166
2167 J.3.1 OBSERVATION AND ACTION SPACE2168 The agent’s observation space is multimodal. For planning, the agent receives its current inventory
2169 (i.e., item names and their quantities) as text. For plan execution, it receives a first-person RGB image
2170 that includes the hotbar, health and food indicators, and player hand animations. For action space,
2171 following ADAM (Yu & Lu, 2024), we use the JavaScript APIs provided by Mineflayer for low-level
2172 control. Specifically, our high-level actions, such as “craft”, “smelt”, and “mine”, are mapped to
2173 corresponding Mineflayer APIs like `craftItem`, `smeltItem`, and `mineBlock`.2174
2175 J.3.2 EPISODE HORIZON2176 For dependency learning, each episode has a fixed horizon of 30 minutes, which is equivalent to
2177 36,000 steps in the MineRL environment. If the agent successfully achieves a goal within this horizon,
2178 it selects another exploratory goal and continues within the same episode.2180 J.4 MC-TEXTWORLD
21812182 MC-Textworld is a text-based environment based on Minecraft game rules (Zheng et al., 2025). We
2183 employ Minecraft version 1.16.5. In this environment, basic rules and goals are the same as those in
2184 the MineRL environment Appendix J.2. Furthermore, resource item spawn probabilities do not need
2185 to be adjusted, unlike in MineRL Appendix J.2.5. This is because an agent succeeds in mining an
2186 item immediately without spatial exploration, if it has a required tool and “mine” is a valid operation
2187 for that item.2188 In the following subsections, we detail the remaining aspects of experiment setups in this environment:
2189 the observation and action space, and the episode horizon.2190
2191 J.4.1 OBSERVATION AND ACTION SPACE2192 The agent receives a text-based observation consisting of inventory information (i.e., currently
2193 possessed items and their quantities). Actions are also text-based, where each action is represented as
2194 an high-level action followed by an item name (e.g., “mine diamond”). Thus, to execute a subgoal
2195 specified as (a, q, v) (high-level action a , quantity q , item v), the agent repeatedly performs the action
2196 (a, v) until q units of v are obtained.2197
2198 J.4.2 EPISODE HORIZON2199 In this environment, we conduct experiments for dependency learning only. Each episode has a fixed
2200 horizon of 3,000 steps. If the agent successfully achieves an intrinsic goal within an episode, it is then
2201 allowed to select another intrinsic goal and continue exploration, without termination of the episode.2202
2204 J.4.3 PERTURBATION ON GROUND TRUTH RULES2205 To evaluate each agent’s robustness to conflicts with its prior knowledge, we perturb the ground-truth
2206 rules (required items and actions) for a subset of goal items, as shown in Figure 21. The perturbation
2207 is applied at different intensity levels (from 1 to 3), where higher levels affect a greater number of
2208 items. These levels are cumulative, meaning a Level 2 perturbation includes all perturbations from
2209 Level 1 plus additional ones.2210
2211 • **Vanilla Setting:** In the setting with no perturbation (Figure 21, a), the ground-truth rules are
2212 unmodified. In the figure, items in the black solid boxes are the goal items, and those with
2213 arrows pointing to them are their true required items. Each goal item has “craft” as a valid
action.

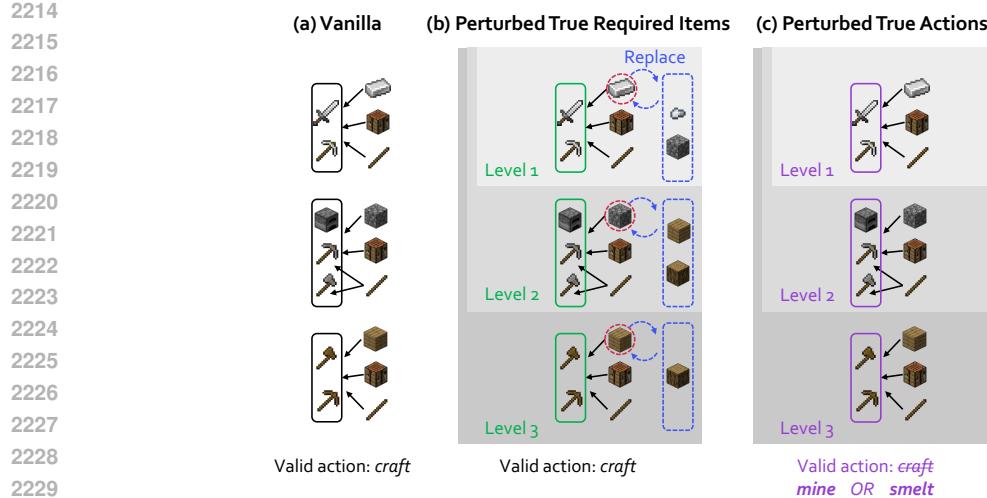


Figure 21: Illustration of the ground-truth rule perturbation settings. (a) in the vanilla setting, goal items (black boxes) have standard required items (incoming edges) and “craft” is the valid action; (b) in the Perturbed Requirements setting, one required item (red dashed circle) is replaced by a new one randomly from a candidate pool (blue dashed box); (c) in the Perturbed Actions setting, the valid action is changed to either “mine” or “smelt”.

- **Perturbed True Required Items:** In this setting (Figure 21, b), one of the true required items (indicated by a red dashed circle) for a goal is replaced. The new required item is chosen uniformly at random from a candidate pool (blue dashed box). The valid action remains craft.
- **Perturbed True Actions:** In this setting (Figure 21, c), the valid action for a goal is randomly changed from “craft” to either “mine” or “smelt”. The required items are not modified.
- **Perturbed Both Rules:** In this setting, both the required items and the valid actions are modified according to the rules described above.

2268 **K ADDITIONAL EXPERIMENTAL RESULTS**
22692270 **K.1 LLM-PREDICTED INITIAL DEPENDENCY GRAPH ANALYSIS**
22712272
2273 Table 12: Performance analysis for the initial LLM-predicted requirement sets over 75 Minecraft
2274 items, used to build the initial dependency graph. Note that while we began the prediction process
2275 with 67 goal items, the total number of predicted items expanded to 75. This expansion occurred
2276 because, as the LLM predicted requirement sets for items in the dependency graph (initially for those
2277 goal items), any newly mentioned items that were not yet part of the graph are also included. This
2278 iterative process is detailed in Section 4.1 (Dependency graph initialization) of our method.
2279

Metric	Value
<i>Requirement Set Prediction Accuracy</i>	
Correct items (ignoring quantities)	23%
Exact items & quantities	8%
<i>Non-existent Item Rates</i>	
Non-existent items	8%
Descendants of non-existent items	23%
<i>Required Items Errors</i>	
Unnecessary items included	57%
Required items omitted	57%
<i>Required Item Quantity Prediction Errors</i>	
Standard deviation of quantity error	2.74
Mean absolute quantity error	2.05
Mean signed quantity error	-0.55

2294 The initial dependency graph, constructed from predictions by Qwen2.5-VL-7B (Bai et al., 2025),
2295 forms the initial planning knowledge for XENON (Section 4.1). This section analyzes its quality,
2296 highlighting limitations that necessitate our adaptive dependency learning.
22972298 As shown in Table 12, the 7B LLM’s initial requirement sets exhibit significant inaccuracies. Ac-
2299 curacy for correct item types was 23%, dropping to 8% for exact items and quantities. Errors in
2300 dependency among items are also prevalent: 57% of items included unnecessary items, and 57%
2301 omitted required items. Furthermore, 8% of predicted items were non-existent (hallucinated), making
2302 23% of descendant items unattainable. Quantity predictions also showed substantial errors, with a
2303 mean absolute error of 2.05.2304 These results clearly demonstrate that the LLM-generated initial dependency graph is imperfect. Its
2305 low accuracy and high error rates underscore the unreliability of raw LLM knowledge for precise
2306 planning, particularly for smaller models like the 7B LLM which are known to have limited prior
2307 knowledge on Minecraft, as noted in previous work (ADAM, Yu & Lu (2024), Appendix A. LLMs’
2308 Prior Knowledge on Minecraft). This analysis therefore highlights the importance of the adaptive
2309 dependency learning within XENON, which is designed to refine this initial, imperfect knowledge
2310 for robust planning.2311 Table 13: Ratio of dependencies learned for items which are unobtainable by the flawed initial
2312 dependency graph (out of 51). Analysis is based on the final learned graphs from the MineRL
2313 experiments.
2314

Agent	Learned ratio (initially unobtainable items)
XENON	0.51
SC	0.25
DECKARD	0.25
ADAM	0.00
RAND	0.02

2322 K.2 ADDITIONAL ANALYSIS OF LEARNED DEPENDENCY GRAPH
2323

2324 As shown in Table 13, XENON demonstrates significantly greater robustness to the LLM’s flawed
2325 prior knowledge compared to all baselines. It successfully learned the correct dependencies for over
2326 half (0.51) of the 51 items that were initially unobtainable by the flawed graph. In contrast, both
2327 DECKARD (with no correction) and the SC baseline (with LLM self-correction) learned only a
2328 quarter of these items (0.25). This result strongly indicates that relying on the LLM to correct its own
2329 errors is as ineffective as having no correction mechanism at all in this setting. The other baselines,
2330 ADAM and RAND, failed almost completely, highlighting the difficulty of this challenge.

2331
2332 K.3 IMPACT OF CONTROLLER CAPACITY ON DEPENDENCY LEARNING
2333

2334 We observe that controller capacity significantly impacts an agent’s ability to learn dependencies
2335 from interaction. Specifically, in our MineRL experiments, we find that ADAM fails to learn any new
2336 dependencies due to the inherent incompatibility between its strategy and the controller’s limitations.
2337 In our realistic setting with empty initial inventories, ADAM’s strategy requires gathering a sufficient
2338 quantity (fixed at 8, same with our hyperparameter α_i ⁶) of all previously used resources before
2339 attempting a new item. This list of required resource items includes gold ingot , because of an
2340 initially provided human-written plan for golden sword; however, the controller STEVE-1 never
2341 managed to collect more than seven units of gold in a single episode across all our experiments.
2342 Consequently, this controller bottleneck prevents ADAM from ever attempting to learn new items,
2343 causing its dependency learning to stall completely.

2343 Although XENON fails to learn dependencies for the Redstone group items in MineRL, our analysis
2344 shows this stems from controller limitations rather than algorithmic ones. Specifically, in MineRL,
2345 STEVE-1 cannot execute XENON’s exploration strategy for *inadmissible items*, which involves
2346 gathering a sufficient quantity of all previously used resources before a retry (Section 4.1). The
2347 Redstone group items become inadmissible because the LLM’s initial predictions for them are entirely
2348 incorrect. This lack of a valid starting point prevents XENON from ever experiencing the core item,
2349 redstone, being used as a requirement for any other item. Consequently, our *RevisionByAnalogy*
2350 mechanism has no analogous experience to propose redstone as a potential required item for other
2351 items during its revision process.

2352 In contrast, with more competent controllers, XENON successfully overcomes even such severely
2353 flawed prior knowledge to learn the challenging Redstone group dependencies, as demonstrated
2354 in Mineflayer and MC-TextWorld. First, in Mineflayer, XENON learns the correct dependencies
2355 for 5 out of 6 Redstone items. This success is possible because its more competent controller can
2356 execute the exploration strategy for *inadmissible items*, which increases the chance of possessing
2357 the core required item (redstone) during resource gathering. Second, with a perfect controller in
2358 MC-TextWorld, XENON successfully learns the dependencies for all 6 Redstone group items in
2359 every single episode.

2360 K.4 IMPACT OF CONTROLLER CAPACITY IN LONG-HORIZON GOAL PLANNING
2361

2362 Because our work focuses on building a robust planner, to isolate the planning from the significant
2363 difficulty of item gathering—a task assigned to the controller—our main experiments for long-horizon
2364 tasks (Section 5.3) uses a modified MineRL environment following the official implementation of
2365 Optimus-1. This modification makes essential resource items like iron, gold, and diamond easier
2366 for the controller to find, allowing for a clearer evaluation of planning algorithms (modifications are
2367 detailed in Appendix J.2.5). However, to provide a more comprehensive analysis, we also evaluated
2368 our agent and baselines in the unmodified, standard MineRL environment. In this setting, items like
2369 iron, gold, and diamond are naturally rare, making item gathering a major bottleneck.

2370 The results are shown in Table 14. Most importantly, XENON* consistently outperforms the baselines
2371 in both the modified and standard MineRL. Notably, in the standard environment, XENON*’s
2372 performance on the Iron group (0.24 SR) is comparable to that of the *OracleActionPlanner* (0.27 SR),
2373 which always generates correct plans for all goals. This comparison highlights the severity of the
2374 controller bottleneck: even the *OracleActionPlanner* achieves a 0.00 success rate for the Diamond

2375 ⁶The scaling factor for required item quantities for inadmissible items.

2376
 2377 Table 14: Long-horizon task success rate (SR) comparison between the **Modified MineRL** (a setting
 2378 where resource items are easier to obtain) and **Standard MineRL** environments. All methods are
 2379 provided with the correct dependency graph. DEPS \dagger and Optimus-1 \dagger are our reproductions of the
 2380 respective methods using Qwen2.5-VL-7B as a planner. *OracleActionPlanner*, which generates the
 2381 correct plan for all goals, represents the performance upper bound. SR for Optimus-1 \dagger and XENON*
 2382 in the **Modified MineRL** column are taken from Table 3 in Section 5.3.

2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429	2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429	Method	Dependency	Modified MineRL			Standard MineRL		
2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429	2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429	Method	Dependency	Iron	Diamond	Gold	Iron	Diamond	Gold
DEPS \dagger	-	DEPS \dagger	-	0.02	0.00	0.01	0.01	0.00	0.00
Optimus-1 \dagger	Oracle	Optimus-1 \dagger	Oracle	0.23	0.10	0.11	0.13	0.00	0.00
XENON*	Oracle	XENON*	Oracle	0.83	0.75	0.73	0.24	0.00	0.00
<i>OracleActionPlanner</i>	Oracle	<i>OracleActionPlanner</i>	Oracle	-	-	-	0.27	0.00	0.00

2390
 2391 and Gold groups in the standard MineRL. This shows that the failures are due to the controller’s
 2392 inability to gather rare resources in the standard environment.

2394 K.5 LONG-HORIZON TASK BENCHMARK EXPERIMENTS ANALYSIS

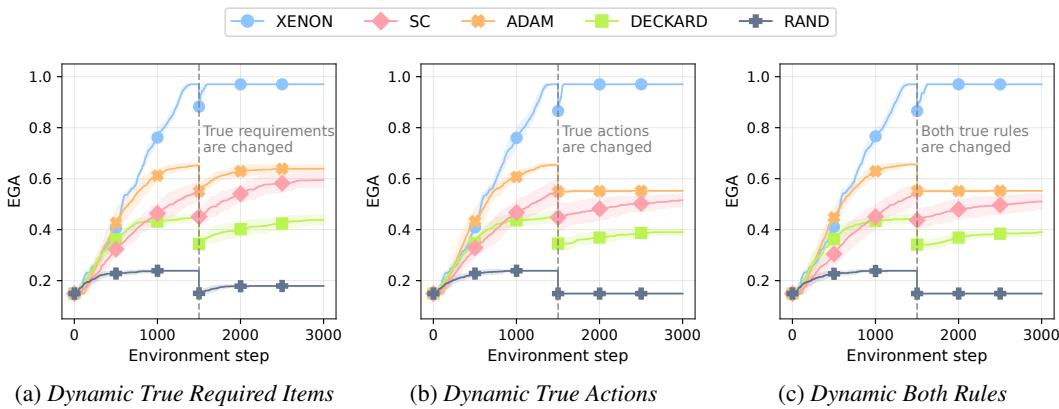
2396 This section provides a detailed analysis of the performance differences observed in Table 3 between
 2397 Optimus-1 \dagger and XENON* on long-horizon tasks, even when both access to a true dependency graph
 2398 and increased item spawn probabilities (Appendix J.2.5). We specifically examine various plan errors
 2399 encountered when reproducing Optimus-1 \dagger using Qwen2.5-VL-7B as the planner, and explain how
 2400 XENON* robustly constructs plans through step-by-step planning with FAM.

2402 Table 15: Analysis of primary plan errors observed in Optimus-1 \dagger and XENON* during long-horizon
 2403 tasks benchmark experiments. This table presents the ratio of specified plan error among the failed
 2404 episodes for Optimus-1 \dagger and XENON* respectively. *Invalid Action* indicates errors where an invalid
 2405 action is used for an item in a subgoal. *Subgoal Omission* refers to errors where a necessary subgoal
 2406 for a required item is omitted from the plan. Note that these plan error values are not exclusive; one
 2407 episode can exhibit multiple types of plan errors.

2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429	2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429	Plan Error Type	Optimus-1 \dagger Error Rate (%)	XENON* Error Rate (%)
Invalid Action			37	2
Subgoal Omission			44	0

2413 Optimus-1 \dagger has no fine-grained action knowledge correction mechanism. Furthermore, Optimus-
 2414 1 \dagger ’s LLM planner generates a long plan at once with a long input prompt including a sequence of
 2415 aggregated requirements $((q_1, u_1), \dots, (q_{L_v}, u_{L_v})) = (1, v)$ for the goal item v . Consequently, as
 2416 shown in Table 15, Optimus-1 generates plans with invalid actions for required items, denoted as
 2417 Invalid Action. Furthermore, Optimus-1 omits necessary subgoals for required items, even they are
 2418 in the input prompts, denoted as Subgoal Omission.

2419 In contrast, XENON discovers valid actions by leveraging FAM, which records the outcomes of each
 2420 action for every item, thereby enabling it to avoid empirically failed ones and reuse successful
 2421 ones. Furthermore, XENON mitigates the problem of subgoal omission through constructing a plan
 2422 by making a subgoal for each required item one-by-one.

2430
2431 K.6 ROBUST DEPENDENCY LEARNING UNDER DYNAMIC TRUE KNOWLEDGE
2432
24332446
2447 Figure 22: **Robustness against dynamic true knowledge.** EGA over 3,000 environment steps in the
2448 where the true item acquisition rules are changed during the learning process.
2449

2450 Additionally, We show XENON is also applicable to sce-
2451 narios where the latent true knowledge changes dynam-
2452 ically. We design three dynamic scenarios where the
2453 environment begins with the vanilla setting, $(0, 0)$, for the
2454 first 1,500 steps, then transitions to a level-3 perturbation
2455 setting for the subsequent 1,500 steps: either required
2456 items-only $(3, 0)$, action-only $(0, 3)$, or both $(3, 3)$.
2457 Upon this change, the agent is informed of which items’
2458 rules are modified but not what the new rules are, for-
2459 cing it to relearn from experience. As shown in Figure 22,
2460 XENON rapidly adapts by re-learning the new depen-
2461 dencies and recovering its near-perfect EGA in all three
2462 scenarios. In contrast, all baselines fail to adapt effectively,
2463 with their performance remaining significantly degraded
2464 after the change. Specifically, for the 7 items whose rules
2465 are altered, Table 16 shows that XENON achieves a perfect re-learning ratio of 1.0 in all scenarios,
2466 while all baselines score 0.0 whenever actions are modified.
2467

2468 K.7 ABLATION STUDIES FOR LONG-HORIZON GOAL PLANNING
24692470 Table 17: Ablation experiment results for long-horizon goal planning in MineRL. Without Learned
2471 Dependency, XENON employs a dependency graph initialized with LLM predictions and human-
2472 written examples. Without Action Correction, XENON saves and reuses successful actions in FAM,
2473 but it does not utilize the information of failed actions.
2474

Learned Dependency	Action Correction	CRe	Wood	Stone	Iron	Diamond	Gold	Armor	Redstone
			0.54	0.39	0.10	0.26	0.45	0.0	0.0
	✓		0.54	0.38	0.09	0.29	0.45	0.0	0.0
✓			0.82	0.69	0.36	0.59	0.69	0.22	0.0
✓	✓		0.82	0.79	0.45	0.59	0.68	0.21	0.0
✓	✓	✓	0.85	0.81	0.46	0.64	0.74	0.28	0.0

2480 To analyze how each of XENON’s components contributes to its long-horizon planning, we conducted
2481 an ablation study in MineRL, with results shown in Table 17. The findings first indicate that without
2482 accurate dependency knowledge, our action correction using FAM provides no significant benefit
2483 on its own (row 1 vs. row 2). The most critical component is the learned dependency graph, which

2444
2445 Table 16: The ratio of correctly learned
2446 dependencies among whose rules are dy-
2447 namically changed (out of 7 total) by
2448 each agent. Columns correspond to the
2449 type of ground-truth rules changed dur-
2450 ing learning: requirements only, actions
2451 only, or both.
2452

Agent	(3,0)	(0,3)	(3,3)
XENON	1.0	1.0	1.0
SC	0.80	0.0	0.0
ADAM	0.83	0.0	0.0
DECKARD	0.49	0.0	0.0
RAND	0.29	0.0	0.0

2484 dramatically improves success rates across all item groups (row 3). Building on this, adding FAM’s
 2485 action correction further boosts performance, particularly for the Stone and Iron groups where
 2486 it helps overcome the LLM’s flawed action priors (row 4). Finally, Context-aware Reprompting
 2487 (CRe, Section 4.3) provides an additional performance gain on more challenging late-game items,
 2488 such as Iron, Gold, and Armor. This is likely because their longer episode horizons offer more
 2489 opportunities for CRe to rescue a stalled controller.

2490

2491 K.8 THE NECESSITY OF KNOWLEDGE CORRECTION EVEN WITH EXTERNAL SOURCES

2492

2493 Even when an external source is available to initialize
 2494 an agent’s knowledge, correcting that knowledge from
 2495 interaction remains essential for dependency and action
 2496 learning, because such sources can be flawed or outdated.
 2497 To support this, we evaluate XENON and the baselines in
 2498 the MC-TextWorld environment where each agent’s depen-
 2499 dency graph is initialized from an oracle graph, while the
 2500 environment’s ground-truth dependency graph is perturbed
 2501 (perturbation level 3 in Table 4). We measure performance
 2502 as the ratio of the 67 goal items obtained within a single
 2503 episode. We use an intrinsic exploratory item selection
 2504 method for all agents (i.e., which item each agent chooses
 2505 on its own to try to obtain next): they choose, among items
 2506 not yet obtained in the current episode, the one with the
 2507 fewest attempts so far.

2508 As shown in Figure 23, this experiment demonstrates that,
 2509 even when an external source is available, (1) interaction
 2510 experience-based knowledge correction remains crucial
 2511 when the external source is mismatched with the environ-
 2512 ment, and (2) XENON is also applicable and robust in this
 2513 scenario. By continually revising its dependency knowledge,
 2514 XENON achieves a much higher ratio of
 2515 goal items obtained in an episode than all baselines. In contrast, the baselines either rely on unreliable
 2516 LLM self-correction (e.g., SC) or do not correct flawed knowledge at all (e.g., DECKARD, ADAM,
 2517 RAND), and therefore fail to obtain many goal items. Their performance is especially poor because
 2518 there are dependencies between goals: for example, when the true required items for stone pickaxe
 2519 and iron pickaxe are perturbed, the baselines cannot obtain these items and thus cannot obtain other
 2520 goal items that depend on them.

2521 K.9 SCALABILITY OF DEPENDENCY AND ACTION LEARNING WITH MORE GOALS AND 2522 ACTIONS

2523 To evaluate the scalability of XENON’s dependency and action learning, we vary the number of goal
 2524 items and available actions in the MC-TextWorld environment. For the goal-scaling experiment, we
 2525 increase the number of goals from 67 to 100 and 120 by adding new goal items (see Table 11 for the
 2526 added goals), while keeping the original three actions “mine”, “craft”, and “smelt” fixed. For the
 2527 action-scaling experiment, we increase the available actions from 3 to 15, 30, and 45 (e.g., “harvest”,
 2528 “hunt”, “place”), while keeping the original 67 goals fixed.

2529 The results in Figure 24 show that XENON maintains high EGA as both the number of goals and
 2530 the number of actions grow, although the number of environment steps required for convergence
 2531 naturally increases. As seen in Figure 24a, increasing the number of goals from 67 to 100 and 120
 2532 only moderately delays convergence (from around 1,400 to about 2,100 and 2,600 steps). In contrast,
 2533 Figure 24b shows a larger slowdown when increasing the number of actions (from about 1,400 steps
 2534 with 3 actions to roughly 4,000, 7,000, and 10,000 steps with 15, 30, and 45 actions), which is
 2535 expected because XENON only revises an item’s dependency after all available actions for that item
 2536 have been classified as empirically invalid by FAM. We believe this convergence speed could be
 2537 improved with minimal changes, such as by lowering x_0 , the failure count threshold for classifying
 2538 an action as invalid, or by triggering dependency revision once the agent has failed to obtain an item
 2539 a fixed number of times, regardless of which actions were tried in subgoals.

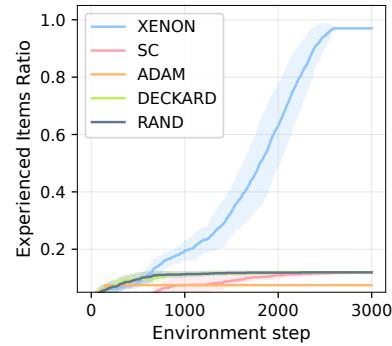


Figure 23: Ratio of goal items obtained in one MC-TextWorld episode when each agent’s dependency graph is initialized from an oracle graph while the environment’s ground-truth dependency graph is perturbed. Solid lines denote the mean over 15 runs; shaded areas denote the standard deviation.

XENON achieves a much higher ratio of goal items obtained in an episode than all baselines. In contrast, the baselines either rely on unreliable LLM self-correction (e.g., SC) or do not correct flawed knowledge at all (e.g., DECKARD, ADAM, RAND), and therefore fail to obtain many goal items. Their performance is especially poor because there are dependencies between goals: for example, when the true required items for stone pickaxe and iron pickaxe are perturbed, the baselines cannot obtain these items and thus cannot obtain other goal items that depend on them.

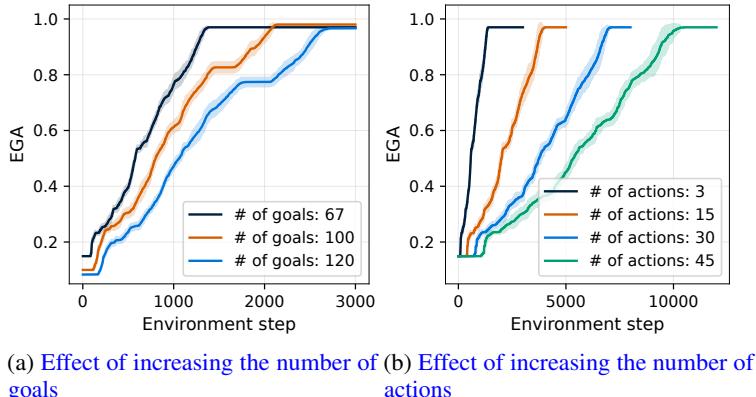


Figure 24: **Scalability of XENON with more goals and actions.** EGA over environment steps in MC-TextWorld when (a) increasing the number of goal items and (b) increasing the number of available actions. In (a), we keep the three actions (“mine”, “craft”, “smelt”) fixed, while in (b) we keep the 67 goal items fixed. Solid lines denote the mean over 15 runs; shaded areas denote the standard deviation.

K.10 ABLATION ON ACTION SELECTION METHODS FOR MAKING SUBGOALS

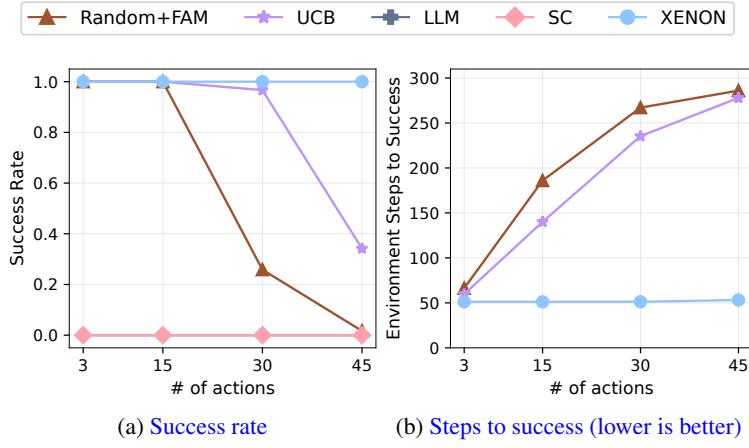


Figure 25: **Ablation on action selection methods for subgoal construction.** We evaluate different action selection methods for solving long-horizon goals given an oracle dependency graph, as the size of the available action set increases. (a) Success rate and (b) number of environment steps per successful episode. Note that in (a), the curves for LLM and SC overlap at 0.0 because they fail on all episodes, and in (b), they are omitted since they never succeed.

We find that, while LLMs can in principle accelerate the search for valid actions, they do so effectively *only when their flawed knowledge is corrected algorithmically*. To support this, we study how different action selection methods for subgoal construction affect performance on long-horizon goals. In this ablation, the agent is given an oracle dependency graph and a long-horizon goal, and only needs to output one valid action from the available actions for each subgoal item to achieve that goal. Each episode specifies a single goal item, and it is counted as successful if the agent obtains this item within 300 environment steps in MC-TextWorld. To study scalability with respect to the size of the available action set, we vary the number of actions as 3, 15, 30, and 45 by gradually adding actions such as “harvest” and “hunt” to the original three actions (“mine”, “craft”, “smelt”).

Methods and metrics. We compare five action selection methods: **Random+FAM** (which randomly samples from available actions that have not yet repeatedly failed and reuses past successful actions),

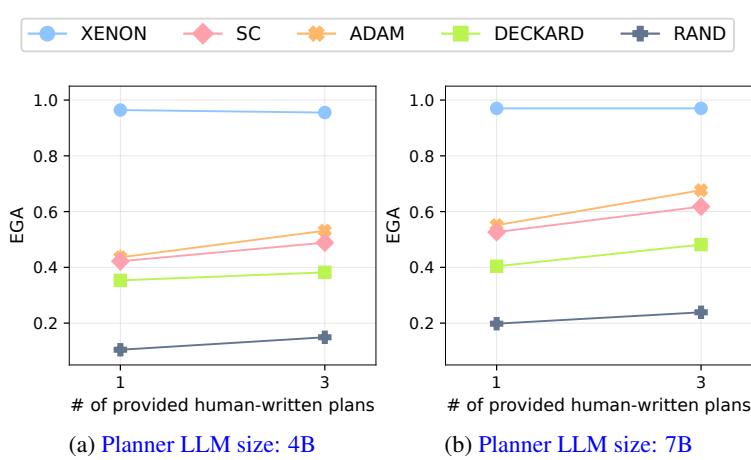


Figure 26: **Effect of planner LLM size and initial dependency graph quality in dependency and action learning.** The plots show EGA after 3,000 environment steps of dependency and action learning in MC-TextWorld, obtained by varying the planner LLM size and the amount of correct knowledge in the initial dependency graph (controlled by the number of provided human-written plans). In (a), the planner is Phi-4-mini (4B) (Microsoft et al., 2025); in (b), the planner is Qwen2.5-VL-7B (7B) (Bai et al., 2025).

UCB, LLM without memory, LLM self-correction (SC), and XENON, which combines an LLM with FAM. We report the average success rate and the average number of environment steps to success over 20 runs per goal item, where goal items are drawn from the Redstone group.

As shown in Figure 25, among the three LLM-based methods (LLM, SC, XENON), only XENON—which corrects the LLM’s action knowledge by removing repeatedly failed actions from the set of candidate actions the LLM is allowed to select—solves long-horizon goals reliably, maintaining a success rate of 1.0 and requiring roughly 50 environment steps across all sizes of the available action set. In contrast, LLM and SC never succeed in any episode, because they keep selecting incorrect actions for subgoal items (e.g., redstone), and therefore perform worse than the non-LLM baselines, Random+FAM and UCB. Random+FAM and UCB perform well when the number of available actions is small, but become increasingly slow and unreliable as the number of actions grows, often failing to reach the goal within the episode horizon.

K.11 ROBUSTNESS TO SMALLER PLANNER LLMs AND LIMITED INITIAL KNOWLEDGE

We further evaluate robustness of XENON and the baselines to limited prior knowledge by measuring dependency and action learning in MC-TextWorld while (i) varying the planner LLM size and (ii) degrading the quality of the initial dependency graph. For the planner LLM, we compare a 7B model (Qwen2.5-VL-7B (Bai et al., 2025)) against a 4B model (Phi-4-mini (Microsoft et al., 2025)); for the initial graph quality, we vary the number of provided human-written plans used to initialize the graph from three (“craft iron_sword”, “mine diamond”, “craft golden_sword”) to one (“craft iron_sword”).

As shown in Figure 26, XENON remains robust across all these settings: its EGA stays near-perfect even with the smaller 4B planner and the weakest initial graph, indicating that leveraging experiences can quickly compensate for weak priors. In contrast, baselines that rely on LLM self-correction (SC) or that strongly depend on the LLM or initial graph (ADAM, DECKARD) suffer substantial drops in EGA as the planner LLM becomes smaller and the initial graph contains less correct prior knowledge. This suggests that, in our setting, algorithmic knowledge correction is more critical than scaling up the planner LLM or richer initial human-provided knowledge.

2646 K.12 FULL RESULTS ON THE LONG-HORIZON TASKS BENCHMARK
26472648
2649 In this section, we report XENON’s performance on each goal within the long-horizon tasks bench-
2650 mark, detailing metrics such as the goal item, number of sub-goals, success rate (SR), and evaluation
2651 episodes.2652 Table 18 and 19 present XENON’s results when utilizing the dependency graph learned through
2653 400 episodes of exploration. Conversely, Table 20 and 21 display XENON*’s performance, which
2654 leverages an oracle dependency graph.2655
2656
2657 Table 18: The results of XENON (with dependency graph learned via exploration across 400 episodes)
2658 on the Wood group, Stone group, and Iron group. SR denotes success rate.
2659

Group	Goal	Sub-Goal Num.	SR	Eval Episodes
Wood	bowl	4	92.68	41
	chest	4	95.24	42
	crafting_table	3	95.83	48
	ladder	5	0.00	31
	stick	3	95.45	44
	wooden_axe	5	90.91	44
	wooden_hoe	5	95.35	43
	wooden_pickaxe	5	93.02	43
	wooden_shovel	5	93.75	48
Stone	wooden_sword	5	95.35	43
	charcoal	8	87.50	40
	furnace	7	88.10	42
	smoker	8	0.00	47
	stone_axe	7	97.78	45
	stone_hoe	7	90.70	43
	stone_pickaxe	7	95.45	44
	stone_shovel	7	89.58	48
	stone_sword	7	89.80	49
Iron	torch	7	93.02	43
	blast_furnace	13	0.00	42
	bucket	11	0.00	47
	chain	12	0.00	42
	hopper	12	0.00	47
	iron_axe	11	75.56	45
	iron_bars	11	80.43	46
	iron_hoe	11	89.13	46
	iron_nugget	11	79.55	44
	iron_pickaxe	11	77.08	48
	iron_shovel	11	75.56	45
	iron_sword	11	84.78	46
	rail	11	0.00	44
	shears	11	0.00	43
	smithing_table	11	93.75	48
	stonecutter	12	0.00	43
	tripwire_hook	11	78.43	51

2700 Table 19: The results of XENON (with dependency graph learned via exploration across 400 episodes)
 2701 on the Gold group, Diamond group, Redstone group, and Armor group. SR denotes success rate.
 2702

Group	Goal Item	Sub Goal Num.	SR	Eval Episodes
Gold	gold_ingot	13	76.92	52
	golden_axe	14	72.00	50
	golden_hoe	14	66.67	48
	golden_pickaxe	14	76.00	50
	golden_shovel	14	71.74	46
	golden_sword	14	78.26	46
Diamond	diamond	12	87.76	49
	diamond_axe	13	72.55	51
	diamond_hoe	13	63.79	58
	diamond_pickaxe	13	60.71	56
	diamond_shovel	13	84.31	51
	diamond_sword	13	76.79	56
Redstone	jukebox	13	0.00	48
	activator_rail	14	0.00	3
	compass	13	0.00	3
	dropper	13	0.00	3
	note_block	13	0.00	4
	piston	13	0.00	12
Armor	redstone_torch	13	0.00	19
	diamond_boots	13	64.29	42
	diamond_chestplate	13	0.00	44
	diamond_helmet	13	67.50	40
	diamond_leggings	13	0.00	37
	golden_boots	14	69.23	39
Armor	golden_chestplate	14	0.00	39
	golden_helmet	14	60.53	38
	golden_leggings	14	0.00	38
	iron_boots	11	94.44	54
	iron_chestplate	11	0.00	42
	iron_helmet	11	4.26	47
Armor	iron_leggings	11	0.00	41
	shield	11	0.00	46

K.13 EXPERIMENTS COMPUTE RESOURCES

2747 All experiments were conducted on an internal computing cluster equipped with RTX3090, A5000,
 2748 and A6000 GPUs. We report the total aggregated compute time from running multiple parallel experiments.
 2749 For the dependency learning, exploration across 400 episodes in the MineRL environment,
 2750 the total compute time was 24 days. The evaluation on the long-horizon tasks benchmark in the
 2751 MineRL environment required a total of 34 days of compute. Experiments within the MC-TextWorld
 2752 environment for dependency learning utilized a total of 3 days of compute. We note that these values
 2753 represent aggregated compute time, and the actual wall-clock time for individual experiments was
 significantly shorter due to parallelization.

2754 Table 20: The results of XENON* (with oracle dependency graph) on the Wood group, Stone group,
 2755 and Iron group. SR denotes success rate.

2756

2757	Group	Goal Item	Sub-Goal Num.	SR	Eval Episodes
2758	Wood	bowl	4	94.55	55
2759		chest	4	94.74	57
2760		crafting_table	3	94.83	58
2761		ladder	5	94.74	57
2762		stick	3	95.08	61
2763		wooden_axe	5	94.64	56
2764		wooden_hoe	5	94.83	58
2765		wooden_pickaxe	5	98.33	60
2766		wooden_shovel	5	96.49	57
2767		wooden_sword	5	94.83	58
2768	Stone	charcoal	8	92.68	41
2769		furnace	7	90.00	40
2770		smoker	8	87.50	40
2771		stone_axe	7	95.12	41
2772		stone_hoe	7	94.87	39
2773		stone_pickaxe	7	94.87	39
2774		stone_shovel	7	94.87	39
2775		stone_sword	7	92.11	38
2776		torch	7	92.50	40
2777	Iron	blast_furnace	13	82.22	45
2778		bucket	11	89.47	38
2779		chain	12	83.33	36
2780		hopper	12	77.78	36
2781		iron_axe	11	82.50	40
2782		iron_bars	11	85.29	34
2783		iron_hoe	11	75.68	37
2784		iron_nugget	11	84.78	46
2785		iron_pickaxe	11	83.33	42
2786		iron_shovel	11	78.38	37
2787		iron_sword	11	85.42	48
2788		rail	11	80.56	36
2789		shears	11	82.05	39
2790		smithing_table	11	83.78	37
2791		stonecutter	12	86.84	38
2792		tripwire_hook	11	91.18	34

2795

2796

L THE USE OF LARGE LANGUAGE MODELS (LLMs)

2798

2799 In preparing this manuscript, we used an LLM as a writing assistant to improve the text. Its role
 2800 included refining grammar and phrasing, suggesting clearer sentence structures, and maintaining a
 2801 consistent academic tone. All technical contributions, experimental designs, and final claims were
 2802 developed by the human authors, who thoroughly reviewed and take full responsibility for the paper's
 2803 content.

2804

2805

2806

2807

2808
 2809
 2810
 2811
 2812
 2813
 2814
 2815
 2816
 2817
 2818

Table 21: The results of XENON* (with oracle dependency graph) on the Gold group, Diamond group, Redstone group, and Armor group. SR denotes success rate.

Group	Goal Item	Sub Goal Num.	SR	Eval Episodes
Gold	gold_ingot	13	78.38	37
	golden_axe	14	65.12	43
	golden_hoe	14	70.27	37
	golden_pickaxe	14	75.00	36
	golden_shovel	14	78.38	37
Diamond	diamond	12	71.79	39
	diamond_axe	13	70.00	40
	diamond_hoe	13	85.29	34
	diamond_pickaxe	13	72.09	43
	diamond_shovel	13	76.19	42
	diamond_sword	13	80.56	36
	jukebox	13	69.77	43
Redstone	activator_rail	14	67.39	46
	compass	13	70.00	40
	dropper	13	75.00	40
	note_block	13	89.19	37
	piston	13	65.79	38
	redstone_torch	13	84.85	33
Armor	diamond_boots	13	60.78	51
	diamond_chestplate	13	20.00	50
	diamond_helmet	13	71.79	39
	diamond_leggings	13	33.33	39
	golden_boots	14	75.00	40
	golden_chestplate	14	0.00	36
	golden_helmet	14	54.05	37
	golden_leggings	14	0.00	38
	iron_boots	11	93.62	47
	iron_chestplate	11	97.50	40
	iron_helmet	11	86.36	44
	iron_leggings	11	97.50	40
	shield	11	97.62	42

2854
 2855
 2856
 2857
 2858
 2859
 2860
 2861