MoEC: A Memory-Routed Mixture-of-Experts Controller for Adaptive Minecraft Control

Hui Wu[2]*, Chao Xu[1], Jianghui Wang[1]†,
Ziqiong Liu[1], Dong Li[1], Yiwei Dai[3], Emad Barsoum[1]
[1]Advanced Micro Devices, Inc.
[2]Aerospace Information Research Institute, Chinese Academy of Sciences
[3]School of Artificial Intelligence, Jilin University
wuhui21@mails.ucas.ac.cn, jianghui.wang@amd.com

Abstract

Embodied agents in open-ended environments such as Minecraft increasingly adopt planner–controller architectures, with large language models acting as high-level planners. While planning has advanced rapidly, control remains underexplored. Existing systems commonly rely on a monolithic policy to execute subgoals across varying contexts, forcing incompatible behaviors into a shared parameter space and causing interference that scaling only partially mitigates. To address this, we propose **MoEC**, a Memory-Routed **M**ixture-**o**f-**E**xperts **C**ontroller for Adaptive Minecraft Control. MoEC routes via a subgoal-indexed, nonparametric expert memory and regulates capacity through failure-triggered expert growth and redundancy-aware consolidation. This design enables continual adaptation without full retraining, while maintaining parameter efficiency and with bounded inference cost. We evaluate MoEC on diverse and compositional Minecraft tasks, demonstrating significant gains in adaptability, robustness, and execution consistency over strong baselines, yielding a scalable and efficient alternative for open-ended control.

1 Introduction

Embodied AI in complex open-ended environments such as Minecraft requires agents to handle combinatorial variability of tasks and contexts while executing long-horizon behaviors. Earlier approaches often relied on end-to-end policies trained via behavior cloning or reinforcement learning, typically tailored to narrowly scoped tasks (Levine et al., 2016; Codevilla et al., 2018; Rahmatizadeh et al., 2018). Recent work has shifted toward modular architectures that decouple high-level planning from low-level control, improving compositionality and adaptability (Zitkovich et al., 2023; Park et al., 2025). Building on this paradigm, large language models (LLMs) now serve as planners, leveraging reasoning capabilities and prior knowledge to translate free-form instructions into structured subgoals (Huang et al., 2023; Shinn et al., 2023; O'Neill et al., 2024). While these advances have substantially improved planning in open-ended domains, the control layer responsible for executing subgoals has received considerably less attention.

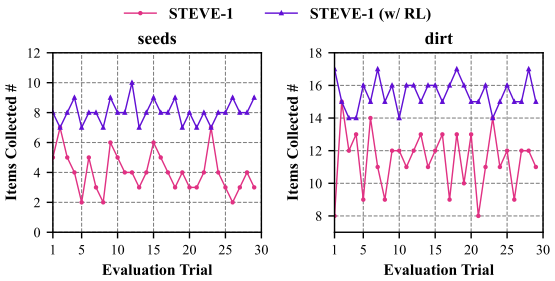

Figure 1: Performance variance of the STEVE-1 controller across independent evaluation trials under the same fixed subgoal. Each point shows the number of target items collected in one trial (higher is better). The figure illustrates that a monolithic controller can produce inconsistent outcomes even when executing the same subgoal in similar settings, motivating the need for adaptive modular control.

Most existing systems adopt a monolithic controller, forcing all subgoals to share a single policy and overlooking contextual variation. Such fixed designs struggle in open-ended settings, where executing an identical subgoal may require divergent behaviors across contexts. We probe this issue by evaluating STEVE-1 (Lifshitz et al., 2023), a widely adopted controller with instruction-following fine-tuning, under identical planner-generated subgoals for short-horizon tasks such as collecting seeds and digging dirt. Empirical results reveal substantial variability, with seed collection ranging from 2 to 7 items and dirt digging from 8

*Work done during an internship at AMD Inc.
†Corresponding Author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 22444–22459

to 13 across trials despite identical plans, exposing a structural weakness in handling context diversity.

Monolithic controllers force incompatible behaviors into a shared parameter space, causing interference and instability as context diversity grows. The conflict stems from reconciling competing behavioral gradients within a single set of parameters. Scaling or adapter tuning partially mitigates it, but lacks explicit mechanisms for specialization or isolation and is computationally costly. Addressing this structural limitation is critical for building scalable, long-horizon embodied agents. This raises a fundamental question: *How can control architectures preserve generality while enabling adaptive modularity to accommodate diverse subgoal-context realizations?*

To address this challenge, we propose **MoEC**, a Memory-Routed **M**ixture-**o**f-**E**xperts **C**ontroller for Adaptive Minecraft Control. It replaces monolithic control with adaptive modularity via dynamic expert evolution rather than a fixed modular design. As shown in Figure 2, MoEC augments the planner–controller paradigm with an evolving control layer. First, a subgoal-indexed, non-parametric expert memory performs two-stage routing: hard subgoal alignment followed by context-conditioned similarity search, without retraining the backbone. Second, on execution failure, an LLM judge either triggers replanning or targeted expert growth, and merges redundant experts to keep capacity bounded, maintaining scalability alongside adaptability. Third, each expert is a lightweight policy head trained online with reinforcement learning on a frozen backbone, enabling fast adaptation at low computational cost. Unlike static modular or fixed-capacity MoE designs, MoEC treats modularity as a continually evolving structure, reducing behavioral interference and improving long-horizon consistency under open-ended variability. The main contributions of our work are:

- We identify the lack of *adaptive modularity* in current embodied controllers as a structural bottleneck to scalability in open-ended environments.
- We introduce **MoEC**, a Memory-Routed **M**ixture-**o**f-**E**xperts **C**ontroller for Adaptive Minecraft Control, which replaces monolithic control with *dynamic expert evolution* rather than fixed-capacity modular designs.
- Our design employs a subgoal-indexed, non-parametric expert memory for two-stage, context-conditioned routing; *failure-triggered* expert growth and *redundancy-aware* consolidation enables continual adaptation *without* backbone retraining.
- Extensive experiments on diverse and compositional Minecraft tasks show significant gains in adaptability, robustness, and execution consistency over strong baselines, with *bounded inference cost*.

2 Related Work

Open-World Embodied Agents. Minecraft has become a primary testbed for open-ended embodied agents, enabled by platforms such as Malmo (Johnson et al., 2016), MineRL (Guss et al., 2019), and MineDojo (Fan et al., 2022). These benchmarks have supported progress in hierarchical RL (Milani et al., 2020; Lin et al., 2022; Mao et al., 2022), imitation learning from large-scale video datasets (Amiranashvili et al., 2020; Guss et al., 2021), and instruction following via video-pretrained policies such as VPT (Baker et al., 2022). Recent systems adopt planner–controller architectures and integrate large language models to decompose free-form goals into subgoals (Huang et al., 2023; Wang et al., 2024; Qin et al., 2024), often combining planning with skill reuse or multimodal grounding for long-horizon tasks (Wang et al., 2024; Li et al., 2024b). Recent embodied-agent research has also explored open-world capability beyond Minecraft, including long-horizon spatio-temporal memory for robot navigation (Anwar et al., 2025) and LLM-guided symbolic planning in interactive visual environments (Zhu et al., 2025). Despite these advances in planning, most controllers remain fixed and general-purpose, and degrade under contextual diversity and long-horizon dependencies (Lifshitz et al., 2023; Park et al., 2025), motivating control with structural adaptation rather than static designs.

Task Adaptation in RL. Many embodied agents employ monolithic controllers, a single shared policy intended to generalize across tasks and contexts (Baker et al., 2022; Lifshitz et al., 2023), but performance often degrades under unseen conditions or context shifts (Park et al., 2025; Li et al., 2024a). To improve adaptability, meta-RL methods infer latent context for fast parameter updates (e.g., PEARL (Rakelly et al., 2019)), option frameworks

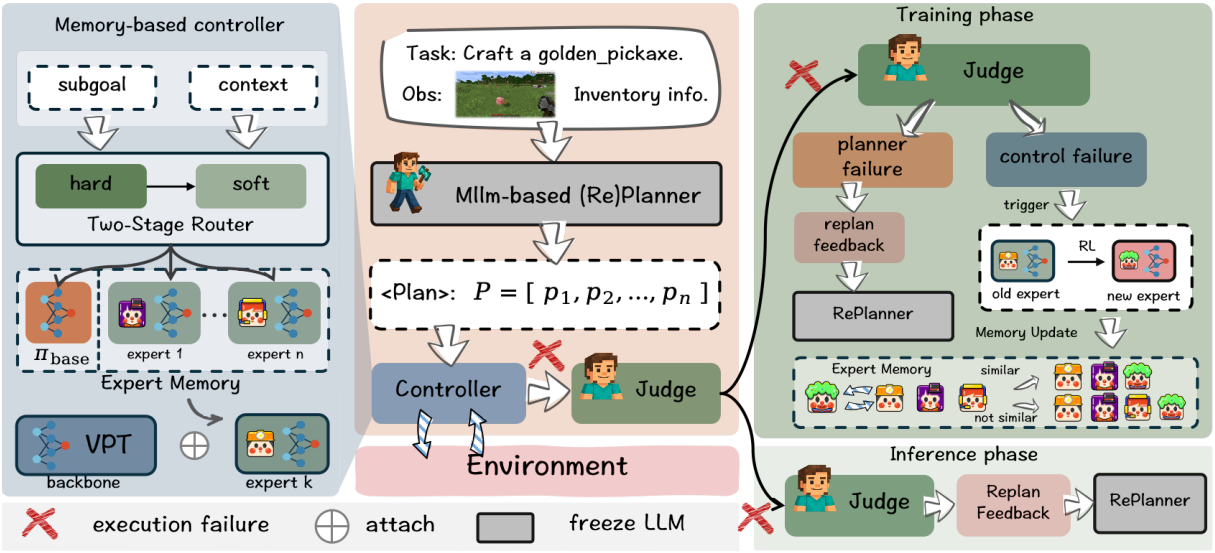

Figure 2: Framework overview of MoEC.

factor policies into reusable subskills (Bacon et al., 2017), and continual-learning approaches reduce forgetting via selective updates or retrieval (e.g., LoTUS (Wan et al., 2024), IsCiL (Lee et al., 2024)); others pair skill libraries with meta-learning for faster adaptation (Nagabandi et al., 2018). Related parameter-efficient adaptation methods have also explored how to improve fine-tuning efficiency under strict parameter budgets (Zhao et al., 2025). These paradigms are parameter-centric, adapting weights within fixed architectures and incurring interference and costly retraining as diversity grows. MoEC instead enables structural adaptation via expert evolution, supporting scalable modularity without global weight updates.

Mixture-of-Experts for Control. Mixture-of-Experts (MoE) enables conditional computation by routing inputs to specialized sub-networks, from early gating (Jacobs et al., 1991) to large-scale LLMs such as GShard, Switch, and GLaM (Lepikhin et al., 2020; Fedus et al., 2022; Du et al., 2022). In control, MoE-style policies scale to multi-task behaviors (e.g., quadruped expert routing (Song et al., 2024)), and Minecraft systems similarly compose task experts (e.g., Optimus-3 (Li et al., 2025)). However, most approaches assume fixed expert sets with parametric gating, requiring global updates to add behaviors. MoEC instead adopts a subgoal-indexed, non-parametric expert memory with online expert evolution, avoiding backbone retraining and fixed-capacity limits.

Memory in Embodied Agents. Memory modules in LLM-driven agents primarily support high-level planning and long-horizon reasoning (Wang et al., 2023a, 2024; Zhang et al., 2023). Voyager maintains a skill library for subgoal reuse (Wang et al., 2023a); Jarvis-1 and Optimus-1 leverage multimodal memory for contextual grounding (Wang et al., 2024; Li et al., 2024b); episodic recall further improves long-horizon decision making (Shinn et al., 2023). These memories are typically passive and confined to the planning layer. MrSTEVE explores memory for low-level control but remains tied to static structures (Park et al., 2025). MoEC instead uses an active, non-parametric memory to route subgoals to context-appropriate experts and to drive controller evolution over time.

3 Method

3.1 MoEC Overview

We introduce **MoEC**, a memory-routed mixture-of-experts controller that replaces monolithic control with adaptive modularity. MoEC augments the standard planner–controller pipeline with a control layer that evolves over time while keeping a *frozen backbone*, enabling scalable specialization in open-ended environments (Figure 2). Throughout, *controller* denotes the module comprising the frozen backbone, expert memory, and router; *policy* refers to an individual expert head.

Problem Definition. For a given instruction–observation pair (I, O), the planner outputs a

subgoal sequence:

$$P = [g_1, \ldots, g_m] \tag{1}$$

At execution step i, the agent observes context c_i (environment geometry and affordance state) and must realize subgoal g_i. MoEC maintains an expert library $\mathcal{E} = \{e_k\}$ with heads $\{\pi_k\}$ on a frozen backbone ϕ, and a router $r(g_i, c_i)$ that selects which expert to invoke. The objective is to complete P with high success and low variance across contexts while keeping per-step inference cost bounded.

High-Level Design. MoEC is built around two mechanisms (detailed next): (i) a **subgoal-indexed, non-parametric expert memory** enabling **two-stage routing** $r(g_i, c_i)$ (Section 3.2); and (ii) **capacity regulation** via **failure-triggered expert growth** and **redundancy-aware consolidation** (Sections 3.3–3.4). If no suitable expert is retrieved, the controller falls back to a base policy π_{base} on the frozen backbone ϕ. Training updates are localized to expert heads; the backbone and memory remain fixed during evaluation.

Positioning. MoEC contrasts with planner-centric memory, meta-RL, and fixed-capacity MoE paradigms by shifting adaptability to the control layer and using non-parametric, context-conditioned routing with expert growth and consolidation, without retraining the backbone.

3.2 Expert Memory and Routing

Conventional MoE controllers use parametric gating, tying expert selection to global weights and requiring retraining to add or reweight experts. This couples routing with learning, amplifies gradient interference, and incurs substantial compute. MoEC decouples the two by adopting a non-parametric expert memory, so routing becomes retrieval rather than a learned gate.

Expert Memory. We represent the control layer with a non-parametric memory

$$\mathcal{M} = \{(g_k, z_k, \pi_k, \text{stats}_k)\}_{k=1}^{K} \tag{2}$$

where g_k is a subgoal key, z_k is a context embedding, $\pi_k(a \mid s) = \pi(a \mid s; \theta_k)$ is an expert head on the frozen backbone ϕ, and stats_k stores routing metadata (e.g., usage counts and an uncertainty score u_k computed from recent episodic-return variance; details in Appendix A.1). Experts are trained locally and inserted into $\mathcal{M}$ without any global parameter update, isolating specialists and reducing interference.

Two-Stage Routing. We denote the context at step i as c_i, the local situation when subgoal g_i begins. In general, c_i may include recent visual observations and agent state (e.g., inventory). In this work, we *instantiate* c_i with a short stack of frames and encode it using a frozen multimodal encoder (details in Appendix A.2).

Given subgoal g_i and context c_i at step i, the router selects an expert via two stages.

Stage 1 (Subgoal filter). Candidate set by hard goal alignment:

$$\mathcal{K}(g_i) = \{\, k \;:\; g_k = g_i \,\} \tag{3}$$

Stage 2 (Context rank). Encode the current context and score candidates:

$$z_i = f_{\text{enc}}(c_i) \tag{4}$$

$$s_k = \text{sim}(z_i, z_k) - \beta\, u_k \quad \text{for } k \in \mathcal{K}(g_i) \tag{5}$$

$$\alpha_k = \frac{\exp(s_k/\tau)}{\sum_{j \in \mathcal{K}(g_i)} \exp(s_j/\tau)} \tag{6}$$

$$k^* = \arg\max_{k \in \mathcal{K}(g_i)} \alpha_k, \qquad \pi^* = \pi_{k^*} \tag{7}$$

Here $\text{sim}(\cdot,\cdot)$ is cosine similarity; $\beta \geq 0$ controls the influence of u_k, and τ is a temperature. If $\max_{k \in \mathcal{K}(g_i)} s_k < \gamma$ for a threshold γ, the controller falls back to π_{base} on ϕ. This separates *semantic intent* (Stage 1) from *context variation* (Stage 2), enabling retrieval-based, context-conditioned selection without retraining the backbone.

Complexity and Benefits. Routing over $\mathcal{M}$ costs $O(|\mathcal{K}(g_i)|)$ per step (or sublinear with an ANN index), plus a forward pass for f_{enc}. Because candidates are filtered by subgoal and routing is non-parametric, inference cost remains bounded as K grows, while the decoupling of routing from learning yields interpretable, plug-and-play modularity.

3.3 Judge-Driven Expert Growth

Failure Diagnosis. MoEC uses a judge module $\mathcal{F}_{\text{judge}}$ to decide whether an execution failure arises from planning or control. Given a multimodal failure context C_{fail} (recent observations, environment state, subgoal metadata, diagnostic cues), the judge outputs

$$(\psi, \Sigma) \leftarrow \mathcal{F}_{\text{judge}}(C_{\text{fail}}) \tag{8}$$

where $\psi \in 0, 1$ indicates failure type (0: planning, 1: control) and Σ is a summary used for plan repair. If ψ=0, the system triggers replanning; otherwise, MoEC performs targeted control adaptation.

Expert Creation. Upon a control failure at the current subgoal–context pair (g_i, c_i), MoEC instantiates a lightweight expert head $\pi_k(\cdot;\theta_k)$ on the frozen backbone ϕ. Creation is gated by (i) low routing confidence and (ii) novelty:

$$\max_{k \in \mathcal{K}(g_i)} \alpha_k < \gamma \tag{9}$$

$$\Delta(c_i, c_k) > \delta \quad \forall (g_k, c_k, \pi_k) \in \mathcal{M}(g_i) \tag{10}$$

where α_k and $\mathcal{K}(g_i)$ come from Sec. 3.2, and $\Delta(\cdot)$ is cosine distance between context embeddings z produced by the same encoder as in Sec. 3.2. This ensures the new expert covers an under-served context region for g_i.

Expert Training. The new head is initialized from the base head (or the best candidate for g_i) and trained *locally* with PPO while the backbone remains frozen. We encode observations and the subgoal as

$$s_t = z_{o_t} + W_\theta z_{g_i} + b_\theta \tag{11}$$

and optimize the clipped objective

$$\mathcal{L}_{\text{PPO}} = -\mathbb{E}_t \left[\min \left(\rho_t A_t,\ \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon) A_t \right) \right] \tag{12}$$

with sparse, completion-based rewards

$$r_t = \mathbb{I}\big(\mathcal{C}(s_t, a_t)\big) \tag{13}$$

where ρ_t is the importance ratio, A_t the advantage, and $\mathcal{C}$ denotes subgoal success. Training starts immediately after failure, resumes from the failure point, and runs for a capped horizon H (Appendix A.3). These interactions are excluded from evaluation metrics. After training, we insert $(g_i, z_i, \pi_k, \text{stats}_k)$ into memory $\mathcal{M}$, where $z_i = f\text{enc}(c_i)$ (Sec. 3.2).

3.4 Scalability via Expert Consolidation

Uncontrolled expert growth risks memory bloat and degrades routing efficiency over long horizons. MoEC performs similarity-driven consolidation to reduce redundancy while preserving behavioral diversity.

Trigger Condition. Consolidation applies to experts linked to the same subgoal. Two experts π_i and π_j are eligible if

$$\cos(z^{(i)}, z^{(j)}) > \delta \quad \text{and} \quad |\hat{s}_i - \hat{s}_j| < \gamma \tag{14}$$

where $z^{(\cdot)}$ are their context keys (embeddings) and $\hat{s}$ are recent success rates. The dual criterion enforces both contextual proximity and comparable reliability.

Merge Strategy. We initialize the merged expert via confidence-weighted interpolation:

$$\theta_m = \lambda \theta_i + (1-\lambda)\theta_j, \qquad \lambda = \frac{\hat{s}_i}{\hat{s}_i + \hat{s}_j} \tag{15}$$

To avoid forgetting, π_m is refined with lightweight behavior cloning on trajectories from both parents (details in Appendix A.4), preserving competence across their joint context region.

Execution Mode. Consolidation is triggered periodically during idle phases, so it does not interfere with online routing or growth. Together with novelty-gated creation, consolidation bounds memory size and sustains modularity, keeping selection overhead small as diversity increases.

4 Experiment

4.1 Experiment Setup

Environment and Tasks. We evaluate our **MoEC** on the JARVIS-1 benchmark (Wang et al., 2024), which is built upon MineDojo (Fan et al., 2022) and uses Minecraft as the primary testbed for open-ended embodied control. This benchmark features a large compositional action space, long-horizon dependencies, and diverse task structures, and is widely regarded as a representative environment for studying scalability and adaptability. All agents operate in survival mode with randomized spawn positions and empty inventories under standard Minecraft survival settings (e.g., natural resource gathering and crafting rules).

Our evaluation covers more than 70 tasks organized into nine functional categories of increasing complexity (e.g., *wooden_pickaxe*, *iron_pickaxe*, *diamond_pickaxe*), providing both broad coverage and a structured progression. Each task is assessed over multiple randomized trials to account for environment stochasticity. Full task details, category definitions, and trial counts are in Appendix B.

Baselines. We compare **MoEC** against a set of strong baselines for instruction execution. *Instruct-GPT* (Huang et al., 2022) employs a base LLM with chain-of-thought prompting for subgoal generation. *ReAct* (Yao et al., 2022) enhances reasoning by interleaving language inference with environment interaction. *DEPS* (Wang et al., 2023b) dynamically decomposes tasks and incorporates self-corrective feedback. *STEVE-1* (Lifshitz et al., 2023) uses a frozen pretrained controller without further adaptation. *JARVIS-1* (Wang et al., 2024)

Group	Task	GPT	ReAct	DEPS	JARVIS-1	Optimus-1	MoEC
Wooden		26.67	45.00	75.00	91.55	**100.00**	**100.00**
	AVG	27.30±14.86	40.31±13.30	80.23±17.32	88.84±16.82	**98.67±1.95**	96.08±3.53
Stone		20.00	20.00	75.00	94.20	93.57	**95.65**
	AVG	20.21±12.32	39.00±12.15	69.27±7.78	88.69±4.87	92.61±4.02	**92.91±6.15**
Iron		0.00	0.00	20.00	33.82	**59.42**	36.76
	AVG	3.27±2.85	4.61±3.63	16.92±4.69	34.63±10.61	**48.29±7.69**	46.28±9.78
Redstone		0.00	2.00	10.00	22.78	**37.50**	26.58
	AVG	1.04±1.30	1.14±1.18	6.02±3.61	17.51±9.34	25.12±8.69	**25.24±6.06**
Golden		0.00	2.00	6.00	14.49	16.42	**17.14**
	AVG	0.00±0.00	0.45±0.60	2.20±1.55	6.85±4.71	8.51±5.69	**8.75±4.53**
Diamond		0.00	0.00	2.00	9.20	9.09	**9.62**
	AVG	0.00±0.00	0.35±0.48	2.42±1.01	8.99±2.68	**11.61±5.75**	10.04±2.58
Armor		6.67	0.00	10.00	30.30	58.14	**60.61**
	AVG	1.36±2.25	0.50±0.88	3.71±3.78	13.44±14.62	23.12±22.46	**23.79±25.78**
Decoration		15.00	15.00	25.00	50.00	-	**55.56**
	AVG	17.12±11.59	17.13±9.19	29.59±15.94	46.67±23.39	-	**56.48±16.56**
Food		13.33	16.67	16.67	43.55	-	**56.45**
	AVG	9.40±4.29	15.56±6.83	22.85±8.15	46.75±11.16	-	**50.75±6.09**

Table 1: Success rate (%) comparison across methods and task groups. For each group, the first row reports the success rate (%) on a single task within that group, while the second row (AVG) reports the mean ± standard deviation of success rates (%) across all tasks in that group, evaluated over multiple randomized trials. Higher values indicate better performance.

combines a language-based planner with a general-purpose controller and serves as the standard benchmark on MineDojo. *Optimus-1* (Li et al., 2024b) integrates hierarchical knowledge graphs and multimodal memory, achieving SOTA performance.

Evaluation Metrics. We report the average success rate across multiple randomized trials for each task. A trial is considered successful if the agent acquires the specified target items within the allotted time budget. The success rate is calculated as the proportion of successful trials out of the total number, using different random seeds for environment initialization. For each task group, we additionally provide the mean and standard deviation across all tasks to reflect both aggregate performance and consistency.

Implementation Details. **MoEC** employs GPT-4o (via the OpenAI API) for the Planner and Judge, and uses STEVE-1 (Lifshitz et al., 2023) as a *frozen policy backbone* for expert heads. All experiments are performed using eight NVIDIA V100 GPUs. Full reinforcement learning configurations, including hyperparameters and training schedules, are provided in Appendix B.1. All reported metrics are computed on *frozen, no-learning* rollouts: any online interactions used to train new expert heads (failure-triggered expert growth) are executed in separate adaptation rollouts and *excluded from evaluation*; after adaptation, we reset the environment and re-run the episode with the updated expert library, with fixed seeds/trials detailed in Appendix B.1.

4.2 Performance Comparison

Table 1 summarizes success rates across nine functional groups. **MoEC** attains the top or second-best average in every group, consistently outperforming JARVIS-1 and matching Optimus-1 despite a stricter evaluation protocol and no planner-level enhancements. Overall, MoEC lifts the global average by a large margin over STEVE-1 and other strong baselines, indicating that adaptive modular control improves execution robustness.

Gains on the most long-horizon categories (e.g., *Diamond*, *Golden*) are smaller. This aligns with *planner–side* limitations: these groups exhibit higher planner-attributed failure rates and lower subgoal validity (Appendix B.2). Even so, MoEC still provides substantial execution-level gains over

Group	Task	w/o memory	w/ log	w/ stone	w/ iron	w/ redstone	w/ gold	w/ diamond	MoEC
Wooden	composter	0.8298	0.8936	-	-	-	-	-	**0.8936**
	bowl	0.9362	0.9574	-	-	-	-	-	**0.9574**
Stone	stone_shovel	0.8649	0.8919	0.9459	-	-	-	-	**0.9459**
	stone	0.8413	0.8413	0.8730	-	-	-	-	**0.8730**
Iron	chain	0.3390	0.3729	0.4746	0.4576	-	-	-	**0.4576**
	shears	0.3281	0.3281	0.5781	0.5781	-	-	-	**0.5781**
Armor	shield	0.4242	0.4091	0.4394	0.5303	-	-	-	**0.5303**
	golden_helmet	0.0667	0.0667	0.0800	0.0800	-	0.1067	-	**0.1067**
Redstone	piston	0.2532	0.2658	0.2532	0.2785	0.3038	-	-	**0.3038**
	redstone_torch	0.2247	0.2472	0.2472	0.2697	0.2809	-	-	**0.2809**
Golden	golden_hoe	0.0290	0.0145	0.0435	0.0580	-	0.0580	-	**0.0580**
	golden_shovel	0.0548	0.0548	0.0548	0.0685	-	0.0882	-	**0.0882**
Diamond	diamond_pickaxe	0.0665	0.0679	0.0665	0.0780	-	-	0.0882	**0.0882**
	diamond_axe	0.0423	0.0423	0.0282	0.0563	-	-	0.0845	**0.0845**

Table 2: Ablation study on expert memory and expert group composition. Best results are bold and "–" denotes that the task does not involve the corresponding expert group.

monolithic controllers (e.g., *Diamond*: 10.04% vs. 0.35% for ReAct).

Finally, MoEC is complementary to planner improvements. With a stronger planner, pairing keeps the controller unchanged yet further boosts long-horizon groups, consistent with planning quality being the primary bottleneck, while MoEC strengthens execution under fixed plans.

4.3 Ablation Study

Expert memory and specialist groups. Table 2 quantifies the impact of expert memory and specialized expert groups. Removing the memory module (*w/o memory*) causes substantial drops on complex tasks: e.g., *chain* falls from 0.4576 to 0.339 (-26%), *shears* from 0.5781 to 0.3281 (-43%), and long-horizon *diamond_axe* from 0.0845 to 0.0423 (-50%). In contrast, short-horizon tasks (e.g., *bowl*) show smaller differences, indicating planner-generated subgoals are often sufficient in simpler contexts without additional specialization.

Enabling expert groups further underscores the value of specialization: *chain* improves from 0.339 (*w/o memory*) to 0.4746 (*w/ iron*); *golden_shovel* and *diamond_axe* exhibit gains exceeding 60% when their corresponding experts are available. These results confirm that context-sensitive experts are crucial for complex or rare subgoals, and demonstrate that MoEC effectively scales specialization with task diversity.

Expert memory size. Figure 3 examines the effect of limiting expert memory. We vary the retained proportion of experts from 0% to 100%, keeping lower-level experts fixed and subsampling higher-level ones. Performance improves with memory size but saturates beyond 60% for most tasks, indicating diminishing returns from additional experts. For example, *white_carpet* reaches 0.82 at 20% memory and only marginally increases to 0.86 at full capacity, while *chain* grows from 0.35 (0%) to 0.47 (100%). This suggests robust execution does not require exhaustive expert retention when experts are compositional and context-aware. MoEC thus maintains scalability: it supports continual expert growth without unbounded inference cost, since memory can be pruned or partially loaded with minimal performance impact.

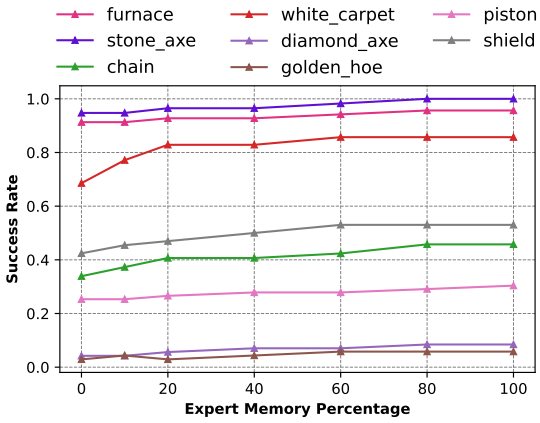

Figure 3: Ablation study on expert memory size.

Model	Seed	Dirt	Flower	Log
P_{base}	4.3	12.2	0.4	**11.8**
P_1 (Seed)	7.8	–	–	–
P_2 (+Dirt)	6.7	8.0	–	–
P_3 (+Flower)	6.2	6.3	0.8	–
P_4 (+Log)	6.5	7.8	0.5	11.2
MoEC	**7.8**	**15.4**	**1.8**	11.6

Model	Log	Flower	Dirt	Seed
P_{base}	**11.8**	0.4	12.2	4.3
P_1 (Log)	11.6	–	–	–
P_2 (+Flower)	10.3	0.7	–	–
P_3 (+Dirt)	9.7	0.5	6.3	–
P_4 (+Seed)	10.5	0.3	6.2	7.3
MoEC	11.6	**1.8**	**15.4**	**7.8**

Table 3: Performance comparison under sequential finetuning. Each model P_i is trained cumulatively on one additional task. P_{base} corresponds to the frozen STEVE-1 model. "–" indicates the model was not trained on that task. Best results are bold.

4.4 Further Analysis

Judge-Driven Failure Attribution over Training. We assess the judge (Sec. 3.3; Appx. A.3) by tracking failure attribution across three training stages defined by expert-memory size (0–30%, 30–70%, 70–100%). In each stage, we uniformly sample 50 failures and categorize them as planning or *control* using the judge's labels. As shown in Fig. 4, early failures are predominantly control-related ($\geq 65\%$), while later stages shift toward planning ($\geq 90\%$). This pattern indicates that judge-driven adaptation progressively stabilizes execution and shifts the bottleneck to planning quality, validating the effectiveness of the judge in guiding control adaptation.

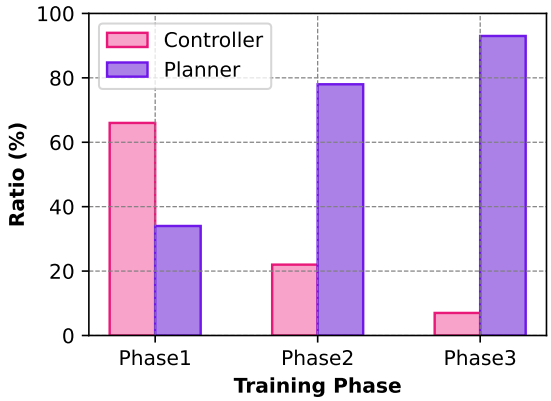

Figure 4: Failure attribution shifts from control to planning as training progresses.

Sequential Monolithic Fine-Tuning: Interference vs. Forgetting. To probe non-modular controllers, we construct a sequential fine-tuning setup where each shared policy P_i is trained cumulatively on the first i tasks. Table 3 shows degradation on both previously seen and newly added tasks; e.g., P_4 underperforms P_2 on *Dirt* despite more data. This suggests that degradation stems not only from forgetting but also from representational interference inherent to shared-parameter policies. In contrast, **MoEC** maintains balanced performance, supporting continual specialization without sacrificing prior competence, highlighting the benefit of modular expert isolation.

Parameter-Efficient Adaptation (Head-Only) vs. Full Fine-Tuning. We compare no adaptation, full parameter fine-tuning, and head-only fine-tuning as used by **MoEC**. As shown in Fig. 5, head-only adaptation matches or exceeds full fine-tuning on *seed* and *dirt*, while updating only a small number of parameters. On average, head-only runs complete in 28.4 s over 3 epochs, enabling rapid and low-overhead specialization compared to full-parameter fine-tuning.

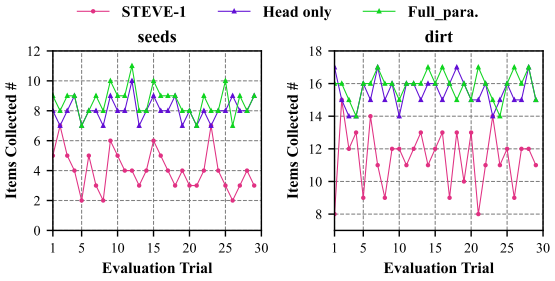

Figure 5: Comparison on *seed* and *dirt*. 'STEVE-1' is the base policy; 'Head-only' is our partial fine-tuning; 'Full para.' denotes full-parameter adaptation.

Additional consolidation results are provided in Appendix A.4, confirming its effectiveness in bounding memory growth.

Generalization to Open-Weight LLMs. To assess whether **MoEC** depends on proprietary LLM backbones, we replace GPT-4o with an open-weight instruction-tuned model, **LLaMA2-70B**, for both the Planner and Judge. Our main experiments use GPT-4o to remain consistent with prior work and ensure controlled comparisons.

Table 4: Additional evaluation of MoEC with different LLM backbones.

Task	GPT-4o	LLaMA2-70B
wooden_pickaxe	96.77%	92.52%
stone_pickaxe	94.52%	90.84%

Running full evaluations across all tasks with multiple LLM backbones is computationally expensive, so we report results on two representative tasks, `wooden_pickaxe` and `stone_pickaxe`. As shown in Table 4, while LLaMA2-70B yields slightly lower absolute performance, **MoEC** exhibits consistent performance trends across both backbones. These results suggest that the effectiveness of **MoEC** is not tied to a specific proprietary model and generalizes to open-weight LLMs.

5 Conclusion

We introduced **MoEC**, a memory-routed mixture-of-experts controller that integrates routing, judge-driven growth, and redundancy-aware consolidation. MoEC routes via a subgoal-indexed, non-parametric expert memory, triggers targeted expert growth upon failure, and consolidates redundancy on a frozen backbone. This design enables interpretable specialization while maintaining bounded inference cost. On JARVIS-1, MoEC attains top or second-best success across task groups, surpasses strong language and action baselines, and closely matches Optimus-1 under a stricter evaluation protocol without planner-level enhancements. Overall, adaptive modularity through non-parametric routing, failure-triggered local growth, and consolidation provides a scalable path to robust and consistent instruction execution in open-ended environments.

Limitations

We have not evaluated MoEC beyond Minecraft. Extending it to other embodied domains would require adapting the perceptual encoder and action space, defining a compatible subgoal ontology, and re-indexing the expert memory. We leave these directions for future work.

References

Artemij Amiranashvili, Nicolai Dorka, Wolfram Burgard, Vladlen Koltun, and Thomas Brox. 2020. Scaling imitation learning in minecraft. *arXiv preprint arXiv:2007.02701*.

Abrar Anwar, John Welsh, Joydeep Biswas, Soha Pouya, and Yan Chang. 2025. Remembr: Building and reasoning over long-horizon spatio-temporal memory for robot navigation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2838–2845. IEEE.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 1726–1734. AAAI Press.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. 2022. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. 2018. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, and 1 others. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. 2019. Minerl: A large-scale dataset of minecraft demonstrations. *Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI*.

William Hebgen Guss, Stephanie Milani, Nicholay Topin, Brandon Houghton, Sharada Mohanty, Andrew Melnik, Augustin Harter, Benoit Buschmaas, Bjarne Jaster, Christoph Berganski, and 1 others. 2021. Towards robust and domain agnostic reinforcement learning competitions: Minerl 2020. In *NeurIPS 2020 Competition and Demonstration Track*, pages 233–252. PMLR.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, and 1 others. 2023. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, pages 1769–1782. PMLR.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. 2016. The malmo platform for artificial intelligence experimentation. In *Ijcai*, volume 16, pages 4246–4247.

Daehee Lee, Minjong Yoo, Woo Kyung Kim, Wonje Choi, and Honguk Woo. 2024. Incremental learning of retrievable skills for efficient continual task adaptation. *Advances in Neural Information Processing Systems*, 37:17286–17312.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

Hao Li, Xue Yang, Zhaokai Wang, Xizhou Zhu, Jie Zhou, Yu Qiao, Xiaogang Wang, Hongsheng Li, Lewei Lu, and Jifeng Dai. 2024a. Auto mc-reward: Automated dense reward design with large language models for minecraft. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16426–16435.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Weili Guan, Dongmei Jiang, and Liqiang Nie. 2025. Optimus-3: Towards generalist multimodal minecraft agents with scalable task experts. *arXiv preprint arXiv:2506.10357*.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. 2024b. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *arXiv preprint arXiv:2408.03615*.

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

Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. 2022. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. In *In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3257–3263.

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

Stephanie Milani, Nicholay Topin, Brandon Houghton, William H Guss, Sharada P Mohanty, Keisuke Nakata, Oriol Vinyals, and Noboru Sean Kuno. 2020. Retrospective analysis of the 2019 minerl competition on sample efficient reinforcement learning. In *NeurIPS 2019 competition and demonstration track*, pages 203–214. PMLR.

Anusha Nagabandi, Chelsea Finn, and Sergey Levine. 2018. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, and 1 others. 2024. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE.

Junyeong Park, Junmo Cho, and Sungjin Ahn. 2025. Mrsteve: Instruction-following agents in minecraft with what-where-when memory. In *The Thirteenth International Conference on Learning Representations*.

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

Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. 2018. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3758–3765. IEEE.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. 2019. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR.

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

Wenxuan Song, Han Zhao, Pengxiang Ding, Can Cui, Shangke Lyu, Yaning Fan, and Donglin Wang. 2024.

Germ: A generalist robotic model with mixture-of-experts for quadruped robot. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11879–11886. IEEE.

Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. 2024. Lotus: Continual imitation learning for robot manipulation through unsupervised skill discovery. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 537–544. IEEE.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *Comment: Project website and open-source codebase: https://voyager. minedojo. org/Cited on*, page 33.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. 2023b. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 34153–34189.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, and 1 others. 2024. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Jesse Zhang, Jiahui Zhang, Karl Pertsch, Ziyi Liu, Xiang Ren, Minsuk Chang, Shao-Hua Sun, and Joseph J Lim. 2023. Bootstrap your own skills: Learning to solve new tasks with large language model guidance. *arXiv preprint arXiv:2310.10021*.

Jinman Zhao, Xueyan Zhang, Jiaru Li, Jingcheng Niu, Yulan Hu, Erxue Min, and Gerald Penn. 2025. Tiny budgets, big gains: Parameter placement strategy in parameter super-efficient fine-tuning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6326–6344.

Wang Bill Zhu, Miaosen Chai, Ishika Singh, Robin Jia, and Jesse Thomason. 2025. Psalm-v: Automating symbolic planning in interactive visual environments with large language models. *arXiv preprint arXiv:2506.20097*.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, and 1 others. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR.

A Additional Method Details

A.1 Uncertainty Estimation for Routing

To quantify the reliability of each expert, MoEC computes an uncertainty estimate σ_k from the stability of recent episodic returns. For each expert k, we maintain a fixed-size *FIFO* queue that stores the most recent N execution episodes (N=20 in all experiments). Each entry is the return $R_k^{(i)}$ obtained when attempting the associated subgoal–context pair.

Formally, let $\{R_k^{(i)}\}_{i=1}^{n_k}$ denote the most recent n_k returns for expert k, where $n_k = \min(N, \text{history length})$. We compute the sample mean and (biased) variance:

$$\bar{R}_k = \frac{1}{n_k}\sum_{i=1}^{n_k} R_k^{(i)}, \qquad v_k = \frac{1}{n_k}\sum_{i=1}^{n_k}\big(R_k^{(i)} - \bar{R}_k\big)^2 \tag{16}$$

The uncertainty estimate is then normalized as

$$\sigma_k = \frac{\sqrt{v_k} + \epsilon}{\bar{R}_k + \epsilon}, \qquad \epsilon = 10^{-6} \tag{17}$$

so lower σ_k indicates higher and more consistent returns, while volatile or poorly performing experts yield larger σ_k. For newly created experts with no recorded episodes, we assign a default high uncertainty (e.g., σ_k=1.0) to prevent premature selection. When partial history is available (n_k<N), the above statistics are computed from available samples without padding.

Finally, we integrate the uncertainty into routing via an uncertainty-adjusted softmax:

$$\alpha_k = \frac{\exp\big(s_k/\sigma_k\big)}{\sum_j \exp\big(s_j/\sigma_j\big)} \tag{18}$$

where s_k denotes the cosine similarity between the current context embedding z_C and the stored embedding $Z_C^{(k)}$. This mechanism prioritizes experts that are both semantically aligned and empirically reliable, improving robustness under contextual diversity.

A.2 MineCLIP Encoder

We adopt the pretrained MineCLIP video-text model as a frozen context encoder. Given a recent 16-frame RGB clip $X_t = \{x_{t-15}, \ldots, x_t\}$ and a subgoal text g_i, MineCLIP's visual backbone produces a video embedding $v_t = f_v(X_t) \in \mathbb{R}^D$ and its text branch yields $z_{g_i} = f_\ell(g_i) \in \mathbb{R}^D$. We use the video `[CLS]` token as the context key and L^2-normalize it for cosine retrieval:

$$z \;=\; \frac{f_v(X_t)}{\|f_v(X_t)\|_2} \tag{19}$$

Unless otherwise stated, MineCLIP weights are kept frozen and no task-specific fine-tuning is applied; frames are resized and normalized with CLIP statistics, and shorter clips at episode start are padded by repeating the earliest frame. This section only reuses MineCLIP "as is" to obtain robust video features; all routing and learning mechanisms are ours (Secs. 3.2–3.4).

A.3 Judge-Driven Expert Growth

Judge Input and Failure Context. The failure context C_{fail} aggregates multimodal signals from: (i) the last visual observations as a 16-frame RGB clip, (ii) the environment state (e.g., inventory and spatial features), and (iii) the current subgoal g_i issued by the planner. These cues suffice to distinguish planning inconsistencies from control execution errors.

Failure Classification Details. The judge $\mathcal{F}_{\text{judge}}$ is implemented with GPT-4o using a fixed prompt enumerating common planning/control failure patterns. Given a structured textual description of C_{fail}, it returns a binary decision and a diagnostic summary:

$$(\psi, \Sigma) \leftarrow \mathcal{F}_{\text{judge}}(C_{\text{fail}}), \quad \psi \in \{0, 1\} \tag{20}$$

Here, ψ=0 denotes a planning error (triggering replanning) and ψ=1 a control failure (triggering expert growth). To ensure determinism, the temperature is set to 0, and the judge is invoked only upon explicit failure signals (e.g., timeouts or repeated no-progress steps).

Expert Growth Trigger and Gating. When ψ=1, a new expert for subgoal g_i is considered only if the current context is novel relative to memory $\mathcal{M}(g_i)$. We compute the context embedding $z_i = f_{\text{enc}}(C_i)$ using the MineCLIP-based encoder (Appx. A.2) and assess novelty by cosine distance:

$$\Delta(z_i, z_k) \;=\; 1 - \cos(z_i, z_k) \tag{21}$$

$$\Delta(z_i, z_k) > \delta \;\; \forall\, (g_i, z_k, \pi_k) \in \mathcal{M}(g_i) \tag{22}$$

with δ=0.25 in all experiments. In a 512-D MineCLIP space, semantically similar contexts typically yield distances < 0.1, whereas unrelated

states exceed 0.3; thus δ=0.25 filters minor variations while admitting meaningful shifts. This choice yielded stable, sublinear expert growth across benchmarks.

Implementation Notes. Each newly created expert attaches a lightweight head $\pi_k(\cdot;\theta_k)$ to the frozen backbone ϕ and is adapted locally with PPO over a capped rollout horizon H=128 steps per update (distinct from the task-level limit of 1500 steps for short tasks). Only head parameters are updated to avoid interference with previously learned behaviors. These adaptation episodes are excluded from evaluation metrics for fairness.

A.4 Expert Merge Strategy

Merge Trigger Details. To prevent uncontrolled memory growth, MoEC periodically consolidates experts associated with the same subgoal. Two experts π_i and π_j are eligible for merging if

$$\cos\big(\bar{z}^{(i)},\, \bar{z}^{(j)}\big) > \delta_{\text{merge}} \quad \text{and} \quad \big|\hat{s}_i - \hat{s}_j\big| < \gamma \tag{23}$$

where $\bar{z}^{(\cdot)}$ is the averaged MineCLIP embedding (Appx. A.2) over the most recent K_o frames for that expert, and $\hat{s}$ is the rolling success rate over the last W_s episodes. In all experiments we set δ_{merge}=0.85 and γ=0.05, ensuring high similarity in both context and performance before merging. This is consistent with the novelty gate used for expert growth (Sec. 3.3): new experts require cosine similarity < 0.75 (distance > 0.25), leaving a safe margin that avoids oscillation between creation and consolidation.

Parameter Interpolation and Refinement. When merging is triggered, a new expert π_m is initialized via confidence-weighted interpolation:

$$\theta_m \;=\; \lambda\,\theta_i + (1-\lambda)\,\theta_j, \qquad \lambda \;=\; \frac{\hat{s}_i}{\hat{s}_i + \hat{s}_j} \tag{24}$$

This biases π_m toward the more reliable parent. To mitigate forgetting, we refine π_m with lightweight behavior cloning on trajectories from both parents for up to E=3 epochs using a small learning rate (1×10^{-4}), lower than the PPO learning rate (3×10^{-4}), which corrects interpolation errors with negligible overhead relative to RL updates.

Execution Mode and Efficiency. Consolidation runs during idle phases at fixed intervals (T_c=5000 environment steps), so it does not interfere with online routing or growth. Similarity checks and merges are performed within each subgoal group, where expert counts remain moderate due to novelty gating, keeping the computational cost minimal. Together with growth control, this achieves sublinear memory expansion while preserving behavioral coverage.

A.5 Complexity and Stability Discussion

Routing operates over the per-subgoal candidate set with time complexity $O\big(|\mathcal{K}(g_i)|\big)$ per decision (Sec. 3.2); with ANN indexing it can be sublinear in $|\mathcal{M}(g_i)|$. The number of experts remains bounded through novelty-gated creation and periodic consolidation. Merge checks occur only within a subgoal and at fixed intervals T_c, preventing disruption of online interaction. Stability is maintained by confidence-weighted interpolation followed by behavior cloning, ensuring the merged expert retains competence across the combined context region.

B Experiment Details

B.1 Experiment Setting

PPO hyperparameters for expert adaptation are summarized in Table 5. These values remain consistent across all benchmarks unless otherwise noted.

Setting	Value
Epochs per Update	3
Discount Factor (γ)	0.99
GAE Lambda (λ)	0.95
Clip Ratio (ϵ_{clip})	0.2
Learning Rate	3×10^{-4}
Value Loss Coefficient (c_{vf})	0.5
Entropy Coefficient (c_{ent})	0.01
Optimizer	Adam
Rollout Length per PPO Update (H)	128
Task-Level Step Limit (Short Task)	1500
Task-Level Step Limit (Long Task)	12000

Table 5: PPO hyperparameters for **MoEC** expert adaptation.

In addition, **MoEC** uses the following key hyperparameters: uncertainty window size $N = 20$, novelty threshold $\delta_{\text{novel}} = 0.25$, merge similarity threshold $\delta_{\text{merge}} = 0.85$, success-rate tolerance $\gamma_{\text{merge}} = 0.05$, merge check interval $T_c = 5000$ environment steps, and behavior cloning refinement with a learning rate of 1×10^{-4} for $E = 3$ epochs.

B.2 Case Study: Long Horizon Task

We analyze a representative long horizon task, *golden_shovel*, where the agent must obtain gold

ingots from raw world conditions in survival mode. The pipeline spans multiple resource gathering stages and deep cave exploration. We sample 100 failures at three training stages defined by expert memory size and attribute them with the judge (planning vs. control).

Failures are mainly planner related and the share rises over training. Early, mid, and late stages show planner attribution of 72%, 84%, and 91%, respectively. Control errors are mostly local collisions and brief stuck states that MoEC reduces through specialist growth.

Table 6: Judge attribution across stages.

	Early	Mid	Late
Planner	72%	84%	91%
Control	28%	16%	9%

B.3 Task Descriptions and Per-Task Results

We summarize detailed task specifications and per-task evaluation outcomes across all functional categories. For each task, we report the natural language instruction, success rate, and the number of evaluation trials. Tasks are organized by semantic category (e.g., *Wooden*, *Stone*) to capture increasing diversity in procedural requirements and behavioral complexity. Success rate is defined as the proportion of trials in which the subgoal is fully completed within the step limit. All evaluations use fixed random seeds and consistent episode lengths to ensure comparability.

B.4 Further Analysis

Effect of Expert Merging. We evaluate the impact of merging on scalability by tracking the number of experts during training. Figure 6 illustrates expert growth with and without consolidation. Without merging, the number of experts grows nearly linearly, exceeding 400 by 6×10^5 steps as novel contexts accumulate. In contrast, periodic similarity-driven merging introduces visible reductions at fixed intervals, flattening the overall growth curve and reducing the final expert pool by approximately 50%. This demonstrates that merging effectively constrains memory growth with no noticeable impact on task performance.

Efficacy Analysis. At inference time, **MoEC** uses the same backbone–head architecture as STEVE-1, so forward-pass latency is unaffected.

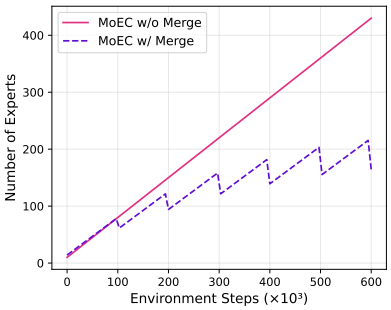

Figure 6: Expert count vs. interaction steps for **MoEC** with and without merging.

The only additional cost comes from expert selection, which computes similarities over the per-subgoal candidate set with time complexity $O(|\mathcal{K}(g_i)|)$ (sublinear in $|\mathcal{M}(g_i)|$ with an ANN index). In practice, $|\mathcal{K}(g_i)|$ remains small (typically under 200 experts after consolidation), making selection overhead negligible relative to backbone inference. In storage, each expert head occupies 71.8 MB, yielding roughly 14 GB for 200 experts; consolidation further constrains growth, preserving scalability.

Table 7: Evaluation results in the *Wooden* category.

Task	Success Rate	Evaluation Times	Language Instruction
chest	1.0000	71	Pick up a chest given nothing.
oak_fence	0.9667	60	Pick up a chest given nothing.
oak_boat	0.9259	54	Pick up a oak_boat in Forest.
wooden_shovel	0.9722	72	Pick up a wooden_shovel given nothing.
wooden_pickaxe	0.9677	62	Pick up a wooden_pickaxe given nothing.
stick	0.9767	86	Pick up a stick given nothing.
crafting_table	0.9853	68	Pick up a crafting_table given nothing.
wooden_sword	1.0000	66	Pick up a wooden_sword given nothing.
bowl	0.9574	47	Pick up a bowl given nothing.
ladder	0.9868	76	Pick up a ladder given nothing.
barrel	0.8947	57	Pick up a barrel given nothing.
composter	0.8936	47	Pick up a composter given nothing.
wooden_axe	0.9636	55	Pick up a wooden_axe given nothing.

Table 8: Evaluation results in the *Stone* category.

Task	Success Rate	Evaluation Times	Language Instruction
furnace	0.9565	69	Craft a furnace given an iron_axe.
smoker	0.8000	75	Craft a smoker given an iron_axe.
stone_shovel	0.9459	74	Craft a stone_shovel given an iron_axe.
stone_pickaxe	0.9452	73	Craft a stone_pickaxe given an iron_axe.
charcoal	0.9079	76	Craft a charcoal given an iron_axe.
stone	0.8730	63	Craft a stone given an iron_axe.
stone_hoe	0.9459	74	Craft a stone_hoe given an iron_axe.
stone_sword	0.9877	81	Craft a stone_sword given an iron_axe.
stone_axe	1.0000	57	Craft a stone_axe given an iron_axe.

Table 9: Evaluation results in the *Iron* category.

Task	Success Rate	Evaluation Times	Language Instruction
iron_pickaxe	0.3676	68	Smelt and craft an iron_pickaxe.
bucket	0.5476	42	Smelt and craft a bucket.
hopper	0.4918	65	Smelt and craft a hopper.
iron_sword	0.5753	73	Smelt and craft an iron_sword.
smithing_table	0.4306	72	Smelt and craft a smithing_table.
chain	0.4576	59	Smelt and craft a chain.
rail	0.5000	62	Smelt and craft a rail.
shears	0.5781	64	Smelt and craft a shears.
iron_shovel	0.5634	71	Smelt and craft an iron_shovel.
tripwire_hook	0.4833	60	Smelt and craft a tripwire_hook.
iron_bars	0.4340	53	Smelt and craft an iron_bars.
iron_nugget	0.4627	67	Smelt and craft an iron_nugget.
iron_door	0.4478	67	Smelt and craft an iron_door.
iron_trapdoor	0.4110	73	Smelt and craft an iron_trapdoor.
crossbow	0.1905	63	Smelt and craft a crossbow.

Table 10: Evaluation results in the *Redstone* category.

Task	Success Rate	Evaluation Times	Language Instruction
dropper	0.2658	79	Mine redstone and make dropper.
note_block	0.3086	81	Mine redstone and make note_block.
compass	0.1646	79	Mine redstone and make compass.
activator_rail	0.1905	63	Mine redstone and make activator_rail.
piston	0.3038	79	Mine redstone and make piston.
redstone_torch	0.2809	89	Mine redstone and make redstone_torch.

Table 11: Evaluation results in the *Golden* category.

Task	Success Rate	Evaluation Times	Language Instruction
gold_ingot	0.1714	70	Smelt and craft a gold_ingot.
golden_axe	0.0625	64	Smelt and craft a golden_axe .
golden_hoe	0.0580	69	Smelt and craft a golden_hoe.
golden_sword	0.0471	85	Smelt and craft a golden_sword .
golden_shovel	0.0822	73	Smelt and craft a golden_shovel.
golden_pickaxe	0.1039	77	Smelt and craft a golden_pickaxe.

Table 12: Evaluation results in the *Diomand* category.

Task	Success Rate	Evaluation Times	Language Instruction
diamond_pickaxe	0.0882	692	Dig down to mine diamond and craft diamond_pickaxe.
diamond_shovel	0.1250	88	Dig down to mine diamond and craft diamond_shovel.
diamond_sword	0.1237	97	Dig down to mine diamond and craft diamond_sword.
diamond_hoe	0.0588	68	Dig down to mine diamond and craft diamond_hoe.
diamond	0.0962	728	Dig down to mine diamond and craft diamond.
jukebox	0.1266	79	Dig down to mine diamond and craft jukebox.
diamond_axe	0.0845	71	Dig down to mine diamond and craft diamond_axe.

Table 13: Evaluation results in the *Armor* category.

Task	Success Rate	Evaluation Times	Language Instruction
shield	0.5303	66	Craft shield and equip it.
iron_helmet	0.6061	33	Craft iron_helmet and equip it.
golden_boots	0.0617	81	Craft golden_boots and equip it.
golden_helmet	0.1067	75	Craft golden_helmet and equip it.
diamond_helmet	0.0429	70	Craft diamond_helmet and equip it.
diamond_boots	0.0800	75	Craft diamond_boots and equip it.

Table 14: Evaluation results in the *Decoration* category.

Task	Success Rate	Evaluation Times	Language Instruction
yellow_dye	0.5000	30	Obtain the yellow_dye.
white_dye	0.3824	34	Obtain the white_dye.
item_frame	0.4286	28	Obtain the item_frame.
white_carpet	0.8571	35	Obtain the white_carpet .
white_banner	0.7097	31	Obtain the white_banner.
orange_wool	0.5556	36	Obtain the orange_wool.
pink_wool	0.5200	25	Obtain the pink_wool.

Table 15: Evaluation results in the *Food* category.

Task	Success Rate	Evaluation Times	Language Instruction
cooked_chicken	0.5448	63	Kill chicken to obtain chicken and cook it.
cooked_porkchop	0.4286	63	Kill pig to obtain porkchop and cook it.
cooked_mutton	0.5645	62	Kill sheep to obtain mutton and cook it.
cooked_beef	0.4921	63	Kill cow to obtain beef and cook it.