PRIME: PRIORITY SCHEDULING AND VALIDATED SKILL EVOLUTION FOR OPEN-WORLD AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

As autonomous agents move toward practical use, reliably handling long-horizon, complex tasks remains a key challenge. Minecraft provides a controllable open-world testbed, yet existing benchmarks largely focus on isolated goals such as finding, collecting, or crafting, leaving the changing priorities and capability gaps of complex execution underexplored. Open-world agents may fail because they execute the wrong available behavior at the wrong time, or because they lack a reliable capability for a necessary subgoal. To probe this gap, we introduce seven long-horizon tasks, such as preparing from scratch and defeating a zombie, or repeatedly chopping trees until 100 logs are collected, and evaluate agents by both task success and completion efficiency. We introduce **PRIME** (**P**riority-guided **R**untime Scheduling with **I**mportance-aware **M**issing-skill **E**volution), which couples runtime priority scheduling with selective skill evolution. PRIME maintains a priority-based schedule that adapts to task state and progress, while task-critical missing skills trigger validated evolution for future reuse. Across seven Minecraft tasks and three high-level model families, PRIME consistently improves task success and completion efficiency, with the largest gains on long-horizon and dynamically changing tasks.

1 INTRODUCTION

A plan is useful only if an agent knows when to follow or interrupt it and whether its actions can realize the named subgoals. Static task settings can obscure this distinction because a valid sequence may remain valid while it executes. Open worlds do not: resources become available out of order, targets move, partial products persist, and one observation can make a sensible action dangerous or wasteful. Long-horizon autonomy therefore requires both decomposition and control allocation over time.

We argue that two failure modes should be separated. In a *scheduling failure*, the required capability already exists, but the agent executes a lower-value subgoal. For example, continuing to collect wood while a nearby hostile makes survival urgent is not evidence that a new skill is needed. In a *capability-grounding failure*, the agent has identified a necessary and currently relevant subgoal, yet its action library contains no reliable executable grounding. Reordering the same library cannot solve this problem. The failures require different interventions: reallocate control in the first case and consider changing the action space in the second. Undifferentiated replanning can repeat an unexecutable plan; treating every failure as a missing skill invites unnecessary and potentially unsafe code generation.

This boundary is under-specified in current agents built on large language models (LLMs). ReAct and DEPS use interaction feedback to improve decisions; BAR reconstructs prerequisite structure; MindStores and M2PA retrieve and consolidate experience; Voyager and Odyssey expose reusable executable skills (Yao et al., 2023; Wang et al., 2023; Du et al., 2025; Chari et al., 2025; Zhou et al., 2025; Wang et al., 2024a; Liu et al., 2025). These advances improve reasoning, memory, or the behavior repertoire, but their system-level formulations generally do not isolate allocation among feasible subgoals from evidence-gated repertoire expansion. A larger library does not determine which skill should run now; a better planner does not make an absent behavior executable.

We introduce PRIME, an agent architecture organized around this separation. PRIME maintains a *stateful priority agenda*: a set of unfinished subgoals with prerequisites, expected products, state-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

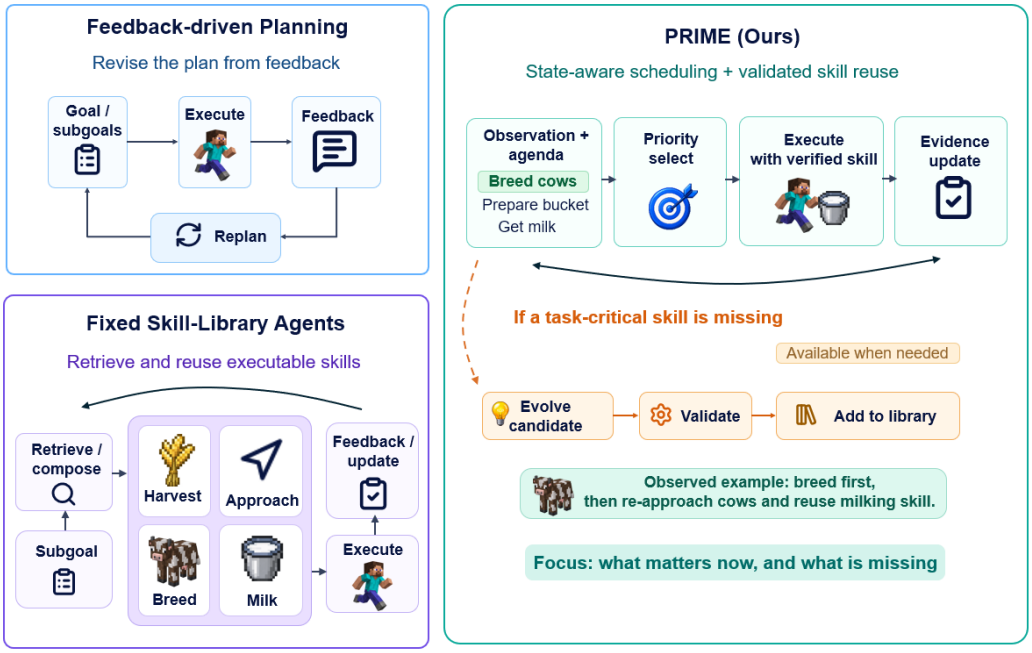


Figure 1: Comparison of three agent organizations. Feedback-driven planning revises the next subgoal, while a fixed skill-library agent retrieves reusable implementations. PRIME integrates state-aware scheduling with verified skill execution and conditionally evolves a candidate only when a task-critical capability is missing. Environment evidence closes both loops.

checkable completion criteria, executable groundings, and runtime status. At each decision boundary, it schedules one feasible grounded item using task priority together with risk, partial progress, dependency bottlenecks, local opportunity, and relevant failure history. Each call is bounded and returns control to the scheduler, which re-observes and rescores the agenda. Structural changes, risk transitions, and grounded failure evidence can revise unfinished work without erasing products already present in the persistent world.

Action-space expansion follows a narrower path. A failed call is not itself a capability diagnosis: the cause may be an unmet prerequisite, stale observation, transient geometry, or executor failure. PRIME first incorporates the evidence and revises the agenda. Skill generation becomes eligible only if the revised agenda still contains a necessary concrete subgoal, its prerequisites and infrastructure are not the blocker, and no available interface provides an executable grounding. Candidate Mineflayer programs are statically screened, tested in fresh worlds, and admitted to persistent memory only when independent state predicates pass. Thus language generation proposes a capability, whereas environment evidence controls whether the action space changes.

Figure 1 makes the architectural distinction explicit. Feedback-driven planning is strong at revising *what to do*, but it remains bounded by the capabilities already exposed to the executor. Fixed skill-library agents are strong at retrieving *how to execute* a named subgoal, but retrieval alone does not decide which unfinished branch deserves control under the current state. PRIME couples both decisions: an observation-grounded agenda selects the highest-priority feasible item, verified execution updates evidence, and only a task-critical grounding gap opens the evolve–validate–admit branch. The design therefore avoids generating a new skill when rescheduling is sufficient and avoids replanning indefinitely when the required behavior is absent.

The temporal distinction between *learning* and *reuse* is central to our evaluation. During earlier development, PRIME evolved and admitted skills under the same validation protocol. The formal PRIME_skill agent loads a frozen set of 20 such records. A framework-only PRIME condition would remove this accumulated seed while preserving scheduling, revision, gap detection, generation, and validation. That matched ablation remains to be measured. The available historical

comparison instead contrasts two already accumulated libraries under the same controller; it must not be interpreted as a no-skills comparison.

We evaluate seven agent designs on seven controlled Minecraft tasks using `gemini-3.8-flash`, `grok-4.7`, and `gpt-5.6-sol`, each at medium reasoning effort. Every model–method–task cell is attempted in five separately reset trials, and task-level success and timing are reported as five-run aggregates. The main comparison therefore schedules $3 \times 7 \times 7 \times 5 = 735$ episodes. Sequential, ReAct, DEPS, BAR, MindStores, and PRIME share task initialization, task contracts, and external success checkers. Odyssey-GPT is retained as a native-system baseline: its planner–action loop, prompts, skill retrieval, role structure, and Mineflayer execution stack remain intact, while all of its LLM-dependent roles are switched together to the tested model. We report hard-checker success, successful active and wall-clock time, and an explicitly labeled historical starting-library comparison.

Contributions. Our contributions are threefold. First, we formulate open-world execution around a distinction between *scheduling* among existing grounded behaviors and *capability grounding* for necessary but ungrounded subgoals. Second, we instantiate this distinction in PRIME, which combines stateful priority scheduling with an evidence-gated and independently validated mechanism for action-space expansion. Third, we operationalize the distinction in a controlled cross-model evaluation with five trials per task and model, supplemented by a provenance-tracked starting-library comparison and an environment-grounded execution example.

2 METHOD

PRIME couples three decisions that open-world agents must make repeatedly: *which unfinished subgoal deserves control now*, *whether existing skills can realize it*, and *what the environment confirms after execution*. These decisions define our three modules: dynamic priority scheduling, evidence-gated skill development and reuse, and environment-grounded feedback. Their connection is a shared subgoal contract—observable prerequisites, intended effects, and an unchanged success predicate—rather than a fixed task recipe.

Problem setting and notation. A task is $\mathcal{T} = (g, o_0, c_g, B)$: a natural-language goal, an initial observation, an external success predicate, and an execution budget. The world persists within the episode. At decision time t, o_t contains inventory, health, nearby entities, position, and other current sensor evidence; h_t retains execution and lifecycle observations. The library $\mathcal{S}_t$ exposes bounded executable Mineflayer programs. The LLM proposes subgoals and programs, but only the environment-grounded checker can establish $c_g(o_t, h_t) = 1$.

We use PRIME for the architecture without a previously accumulated learned-skill seed, and PRIME_skill for the same architecture initialized with skills developed and retained from earlier task attempts. Both retain the ordinary base action library and can develop skills online. Thus *without accumulated skills* does not mean without executable actions. Our main evaluation uses PRIME_skill with a fixed 20-record snapshot; these records are not learned anew in each test episode.

Figure 2 separates model-dependent proposal from the runtime control and verification path. The model under test may change the initial agenda and later revisions, but it does not directly declare task completion or library admission. The scheduler, mapper, executor, and checker repeatedly transform observations into bounded actions and hard evidence. This separation is also what allows the same protocol to compare different language models without changing the task contract.

2.1 MODULE I: DYNAMIC PRIORITY SCHEDULING OVER UNFINISHED WORK

Preserve progress, reconsider control. A sequence commits to an order; an agenda preserves what remains to be done while allowing that order to change. PRIME's planner constructs $\mathcal{A}_t = \{z_1, \ldots, z_m\}$, where

$$z_i = (u_i, s_i, p_i, n_i, \mathcal{R}_i, \mathcal{Q}_i, c_i, x_i). \tag{1}$$

Here u_i is a subgoal, s_i its skill grounding (or MISSING), p_i a model-assigned base priority, n_i a requested repeat count, $\mathcal{R}_i$ its prerequisites, $\mathcal{Q}_i$ its expected products, c_i an observable completion

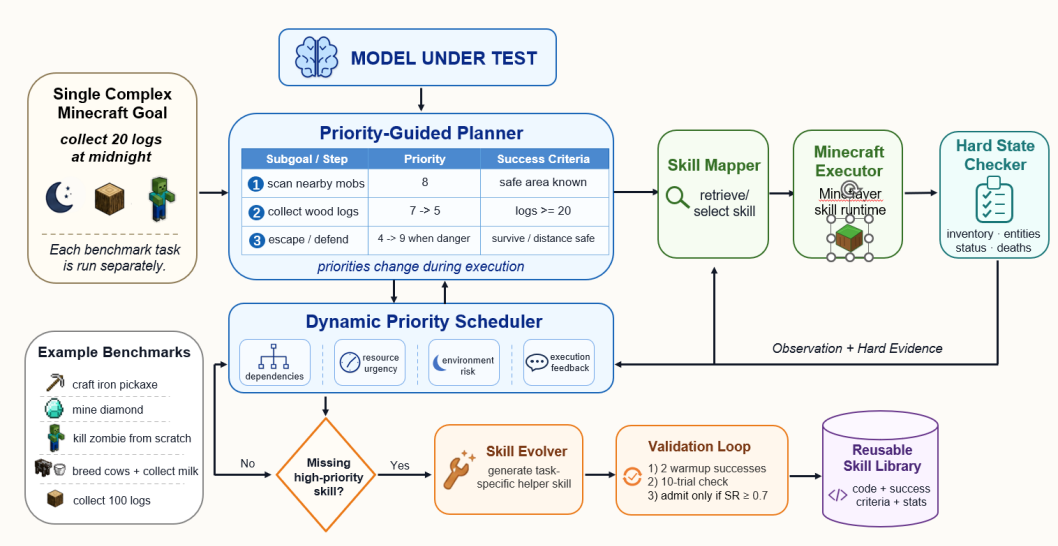


Figure 2: PRIME's full architecture. The evaluated model proposes and updates a priority-annotated agenda; the dynamic scheduler allocates control using dependencies, urgency, risk, and execution feedback. A skill mapper grounds the selected item, the Mineflayer executor acts, and a hard state checker returns observation-grounded evidence. A missing high-priority grounding alone opens the skill-evolution and independent-validation path before admission to the reusable library.

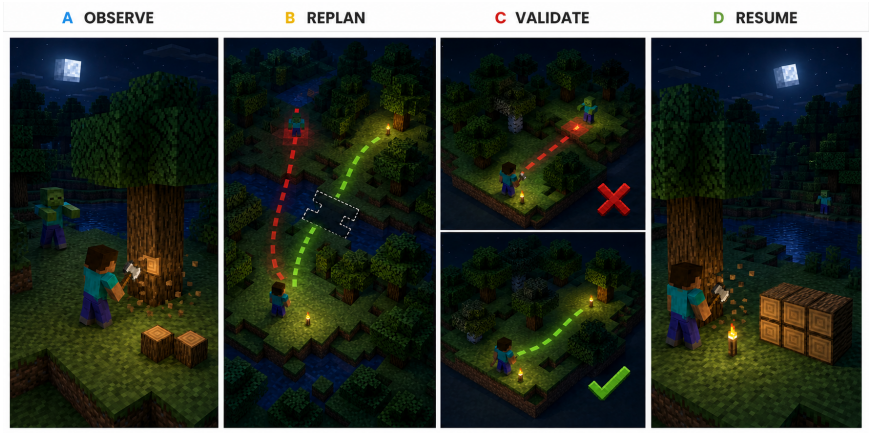


Figure 3: PRIME preserves collected logs, replans around new danger, validates the revised response, and resumes execution. This schematic illustrates the control loop rather than a measured trial.

predicate, and x_i its runtime status. Prerequisites and effects connect competing branches: collecting stone may enable both a tool and a furnace, while recovering solid footing may make several gathering actions executable again.

Feasibility is a hard constraint, not a soft preference. Ordinary execution selects only pending items whose declared inputs hold in the current observation and whose grounding resolves to an executable skill:

$$\mathcal{F}_t = \{z_i \in \mathcal{A}_t : x_i = \text{pending},\ \mathcal{R}_i(o_t) = 1,\ s_i \neq \textsc{Missing}\}. \tag{2}$$

The scheduler neither invents a recovery action nor supplies a task-specific route; the relevant item must already have been proposed and grounded.

Rank feasible work using current evidence. For $z_i \in \mathcal{F}_t$, the implemented score is

$$S_t(z_i) = p_i + \lambda_r U_i + \lambda_p P_i + \lambda_b D_i + \lambda_o O_i - \lambda_f F_i. \tag{3}$$

U_i indicates observed danger together with an unmet recovery target. P_i is the average capped fraction of an absolute inventory target already present; it is not a learned value function. D_i counts other unfinished agenda items with unmet inputs that this item can produce. O_i indicates a currently observed typed-entity opportunity aligned with the task's goal effects and the skill contract. Finally, F_i is the larger of the agenda-local failure count and the matching historical observed-failure count. Historical evidence penalizes repeating a failed capability in a matching context without being mistaken for a current observation. The fixed weights are $(60, 15, 25, 25, 20)$; the score is an operational scheduling heuristic, not a calibrated success probability or an optimality guarantee.

Short commitments, selective revision. The highest-scoring feasible item executes *one bounded skill invocation*, even if $n_i > 1$. PRIME then observes and rescores. This exposes decision boundaries during repetitive collection, so a newly relevant recovery or prerequisite can take control without discarding the inventory already acquired. Scheduling is repeated after every call; LLM agenda revision is not. Revision is triggered by evidence such as exhausted feasible work, changed risk or entity state, invalidated prerequisites, or failure without the expected progress. It modifies unfinished work using the original goal, current observation, previous agenda, and execution history. Appendix A specifies the bounded no-progress policy and the fast path for an already-grounded recovery item while a revision is pending.

2.2 Module II: Evidence-Gated Skill Development and Reuse

Separate a bad choice from a missing capability. Rescheduling helps only when the required behavior is available. Conversely, a failed invocation can reflect missing materials, stale geometry, or infrastructure failure, none of which alone justifies new code. PRIME therefore first incorporates execution evidence and attempts agenda revision and grounding with existing skills. Generation becomes eligible only for an unsatisfied, necessary subgoal retained by that revision, with declared prerequisites met but no executable grounding:

$$\begin{aligned} \mathrm{Gap}_t(u) = \mathrm{EvidenceRevised}_t \wedge \mathrm{RetainedNecessary}_t(u) \wedge \neg c_u(o_t) \\ \wedge \mathrm{Ready}_t(u) \wedge [m(u, \mathcal{S}_t) = \textsc{Missing}]. \end{aligned} \tag{4}$$

The necessity judgment is the reviser's operational diagnosis, not a proof that no alternative plan exists. The gate also requires an available evolver and independent validation resources, and excludes known infrastructure blockers and unchanged rejected contexts without new evidence. A mapper miss alone is insufficient.

Generate a candidate; admit an evidenced capability. The evolver produces a bounded asynchronous program using exposed interfaces and observed state. Static screening rejects privileged or result-forcing operations such as granting inventory, teleporting, or placing target resources. Screening is not a success test: a candidate must execute against the *original* subgoal predicate in separately initialized validation environments. Progress there never counts toward the live episode, whose world remains intact.

The implemented admission protocol requires two consecutive warm-up successes in at most six attempts, followed by ten fresh final trials with candidate code fixed. Admission requires at least seven final successes. Each success requires both normal candidate execution and the external predicate; inventory-increment predicates are instantiated against each trial's initial inventory. A revised candidate starts its trial credit from zero. These finite validation trials are an empirical admission screen, not an estimate of general reliability on arbitrary unseen maps.

Consolidate reusable behavior, not a successful task script. Admitted records retain executable code, descriptions, input/effect contracts, criteria, code identity, and validation provenance. They return to ordinary skill mapping and scheduling. PRIME does not insert an entire successful historical task route as a mandatory plan. Its storage separates an immutable initial seed from episode-local additions; formal parallel episodes do not silently exchange newly learned records.

For PRIME_skill, the initial seed is a selected, frozen subset of skills admitted during earlier development. The 20-record library combines gathering, crafting, recovery, search, and entity-interaction

behaviors (Appendix Table 4). Selection was curated after development, not an unfiltered autonomous accumulation process; held-out-task separation from skill development is not established. The distinction between development, consolidation, and later reuse is essential: successful reuse in a test episode need not involve new skill generation there.

2.3 Module III: Environment-Grounded Feedback and Progress Preservation

Use observed state to close both loops. The feedback module distinguishes fresh observations, historical observations, and model proposals. After a bounded invocation it checks subgoal predicates, records inventory changes and execution diagnostics, and updates agenda status. A call can report an error after obtaining an item; conversely, it can return normally without achieving the requested effect. PRIME retains both the actual partial gain and any new risk instead of collapsing them into a single textual success flag. Fresh pre-execution observations guard against changes during planning latency, while bounded helper calls and optional cooperative risk-yield checks expose control boundaries. Last-known entity positions remain historical evidence, not proof of current reachability.

Keep the goal and its judge outside the model. The same environment-derived predicates define prerequisites, subgoal completion, validation outcomes, and final task checks. Final success additionally uses the task's lifecycle constraints: breeding plus a milk bucket for Task 5, and no new death for Task 7. Agent declarations cannot relax these criteria. This checker is shared across evaluated methods; we do not attribute an advantage to a more lenient judge. Its methodological role is to make scheduling and library updates respond to evidence rather than self-reported progress.

Together, Module I selects what to do with available competence; Module II expands that competence only after a diagnosed gap and independent validation; Module III supplies the evidence that keeps both decisions aligned with the persistent world.

3 Experiments

We examine three questions: does the complete agent improve task completion across backbones (**RQ1**); what changes when its accumulated skill library is expanded (**RQ2**); and does the observed control flow match the proposed scheduling–feedback interaction (**RQ3**)? Every reported model–method–task cell averages five separately reset trials. No single episode is treated as the result of a cell.

3.1 Experimental Setup

Tasks and objective evidence. Table 6 lists seven controlled Minecraft tasks. The fixtures, reset procedures, and world seed (20260529) are fixed across conditions. Tasks 1–6 begin with an empty inventory. Task 7 provides an iron sword and six reachable logs in the world, not in inventory. Task 4 provides nearby resources, including diamond ore, but the agent must still construct the mining-tool chain. Task 5 provides three cows and four mature wheat plants. The same external checker evaluates every method.

Models and methods. We use `gemini-3.8-flash`, `grok-4.7`, and `gpt-5.6-sol`, each with requested reasoning effort `medium`. The main proposed condition is PRIME_skill, initialized with the same frozen 20-record library. Sequential, ReAct, DEPS, BAR, and MindStores are common-harness adapters of the corresponding decision protocols, not reproductions of their complete original systems (Yao et al., 2023; Wang et al., 2023; Du et al., 2025; Chari et al., 2025). They share the base executor interfaces, observations, initialization, and hard checker but do not receive PRIME's accumulated seed. Odyssey-GPT retains its native planner–action loop, retrieval, prompts, executor, and five LLM roles (Liu et al., 2025); all roles switch to the tested backbone.

Five-trial protocol. For every model–method–task cell, we run five independent episodes from the task-specific reset:

$$3 \text{ models} \times 7 \text{ methods} \times 7 \text{ tasks} \times 5 \text{ trials} = 735 \text{ episodes}. \tag{5}$$

Table 1: Five-trial main comparison. S/V is the number of majority-solved tasks out of seven valid task means. A and W are mean successful active and wall time (seconds), averaged from the five trials in each task cell and then across successful task cells. Odyssey's A uses native accounting.

	`gpt-5.6-sol`			`gemini-3.8-flash`			`grok-4.7`		
Method	S/V	A	W	S/V	A	W	S/V	A	W
Sequential	2/7	182.0	279.6	1/7	221.1	339.2	2/7	191.2	381.0
ReAct	4/7	270.7	394.2	3/7	164.6	357.5	3/7	170.9	402.0
DEPS	4/7	291.2	550.8	3/7	169.9	437.1	3/7	374.3	1486.0
BAR	2/7	150.7	244.5	1/7	118.1	433.8	2/7	200.0	1169.5
MindStores	2/7	308.2	500.3	3/7	271.4	464.9	3/7	505.4	1064.8
Odyssey-GPT	2/7	308.8	589.0	2/7	164.9	639.9	2/7	165.7	676.1
PRIME_skill	6/7	263.5	934.0	3/7	481.7	2745.5	5/7	321.7	1317.4

Administrative retries do not create extra observations and are never selected for a better outcome. A timeout, runtime error, or failed checker is a failed trial. Aggregation first forms the full expected index and then averages its five outcomes, so crashed or missing runs cannot disappear from the denominator.

Metrics. For model m, method a, and task k, task success rate is

$$\mathrm{SR}_{m,a,k} = \frac{1}{5}\sum_{r=1}^{5} \mathbf{1}\Big[c_k(o_{\mathrm{final}}^{(r)}, h^{(r)}) = 1\Big]. \tag{6}$$

Tables report this five-run mean. For the compact seven-task count, a task is *majority solved* when at least three of five trials pass ($\mathrm{SR}_{m,a,k} \geq 0.6$). Active execution time (A) and wall-clock time (W) are means over successful trials only. A is mapper plus live-skill time in the common runner and native nonempty action time in Odyssey; W includes planning, online evolution, and recorded overhead. Timing is interpreted only beside SR because success-conditioned means can cover different task sets.

3.2 Main Results: Repeated Collection Benefits Most

Cross-model outcomes. With GPT, PRIME_skill majority-solves 6/7 tasks, compared with 4/7 for ReAct and DEPS, the strongest baseline counts. With Gemini, PRIME_skill solves 3/7, tying ReAct, DEPS, and MindStores in count but not necessarily in the task-level mean SRs. With Grok, PRIME_skill solves 5/7 tasks, including the breeding-and-milk task, compared with at most 3/7 for any baseline. Figure 5 summarizes these thresholded task counts; the underlying endpoint remains the five-run SR.

On the four shared majority-solved tasks, active time changes from 641.3 to 646.6 seconds (T1), 21.2 to 37.4 (T2), 353.7 to 337.4 (T3), and 125.7 to 159.9 (T7). Only T3 is faster in active time. Consequently, this analysis supports a coverage hypothesis rather than a blanket speed claim.

3.3 Execution Example: Observe, Replan, Validate, Resume

Figure 3 illustrates the intended control interaction on the midnight collection task. A hostile observation changes what matters now; replanning preserves already collected logs while proposing a safer response; validation rejects an unsafe candidate and retains a grounded alternative; execution then resumes rather than restarting the task. The example is schematic, not one of the five measured trials. In the formal evaluation, only the external lifecycle checker can establish task success, and each task-level result is the mean of its five independently reset episodes.

4 LIMITATIONS

Controlled fixtures reduce variance but narrow external validity; the diamond task, for example, tests a mining-tool chain rather than natural cave exploration. Five trials per model–method–task cell reduce sensitivity to a single trajectory but remain a modest sample; matched no-seed and component ablations are absent, hosted models may drift, and provider-specific `medium` neither calibrates equal reasoning compute nor equalizes LLM calls. This is therefore an agent-design comparison, not a matched-token or matched-cost study. Active execution time excludes planning, initialization, recovery overhead, and online evolution. Because it conditions on success, its pooled value also depends on which tasks a method solves; Odyssey additionally uses distinct native-action accounting. It is therefore a secondary metric. Wall and component times are retained for audit.

The five common-harness baselines are adapters, not the original systems: MindStores has no cross-trial experience and cannot revise its one-shot plan from later tuples, whereas Odyssey-GPT retains a native, system-level stack. The within-domain, selected learned-skill snapshot establishes neither held-out-task transfer, out-of-distribution transfer, nor the yield of an unfiltered online accumulation process; the historical starting-library comparison isolates neither individual records nor separate score, revision, gap-gate, or admission effects. Hard checkers remain bounded proxies: rotten flesh stands in for a zombie kill, and `animals_bred` counts any bred animal, with cow attribution supplied by the fixture. Five sampled trials cannot exclude skill duplication, interference, or distribution shift. The ten-world admission gate is a coarse empirical screen, not a reliability guarantee. Code screening is not a security sandbox. PRIME also presumes a usable base executor: scheduling cannot repair widespread primitive failure, and bounded evolution is not a substitute for general low-level control.

5 RELATED WORK

5.1 OPEN-WORLD EMBODIED AGENTS

Minecraft has become a useful testbed for open-world agency because goals combine partial observability, long dependency chains, spatial interaction, and irreversible execution cost. MineDojo provides a broad environment and knowledge ecosystem, while VPT and STEVE-1 study learned low-level or instruction-conditioned control (Fan et al., 2022; Baker et al., 2022; Lifshitz et al., 2023). Language-model agents add higher-level decomposition and interaction: GITM and JARVIS-1 use structured reasoning and memory, Voyager grows an executable program library, and Odyssey combines planning, action, critique, retrieval, and a large open-world skill repertoire (Zhu et al., 2023; Wang et al., 2024b;a; Liu et al., 2025).

These systems broaden what an agent can attempt, but breadth alone leaves a runtime control problem. Once multiple subgoals are feasible, the agent must decide which one deserves control under the current state; once a desired subgoal lacks a grounding, the agent must decide whether the evidence justifies modifying its behavior repertoire. PRIME targets this boundary. Our Odyssey-GPT comparison preserves the native Odyssey stack rather than reducing it to a planner prompt, because Odyssey's capabilities arise from the interaction of its roles, retrieval, skills, and executor.

5.2 PLANNING, FEEDBACK, AND EXPERIENCE

Reactive deliberation predates language models: the Procedural Reasoning System executes partial intentions and interrupts or resumes them as conditions change (Georgeff & Lansky, 1987). PRIME's contribution is not reactivity itself, but an operational boundary between reallocating known behavior and changing the repertoire.

ReAct interleaves reasoning and action, and DEPS uses descriptions, explanations, planning, and selection to refine behavior from feedback (Yao et al., 2023; Wang et al., 2023). BAR addresses the perception gap between an initial state and a distant goal through backward prerequisite reasoning (Du et al., 2025). Other work improves the information available to the planner: AriGraph organizes spatial and episodic knowledge, MrSteve retrieves multimodal memories, M2PA integrates semantic, episodic, sensory, and working memory, and MindStores stores state–task–plan–outcome

experience for later review (Anokhin et al., 2025; Park et al., 2025; Zhou et al., 2025; Chari et al., 2025).

These approaches primarily study *what plan is proposed* or *what context supports it*. PRIME asks what happens after a plausible decomposition meets a changing world. An ordered plan collapses live alternatives into one cursor; wholesale regeneration can restore flexibility but obscures whether priority or the action space failed. A stateful agenda instead preserves completed products and blocked branches while using current state to reallocate control among remaining grounded items.

5.3 Executable Skills and Lifelong Acquisition

Executable abstractions shorten planning horizons and enable transfer. Code as Policies demonstrates language-model synthesis of embodied programs; Voyager accumulates reusable programs, Odyssey supplies primitive and compositional skills, and memory-based agents reuse plans or experience across tasks (Liang et al., 2023; Wang et al., 2024a; Liu et al., 2025; Zhou et al., 2025; Chari et al., 2025). Yet an executable library introduces a provenance and admission problem. A skill should not be generated merely because an attempt failed, and generated code should not become persistent merely because the model reports success. Failures may originate in the schedule, a missing prerequisite, observation error, transient geometry, or the executor itself.

PRIME couples library growth to an evidence gate: observation-grounded revision must first preserve a necessary subgoal and rule out ordinary blockers; only an explicit missing grounding can reach synthesis. Admission is then based on external predicates in independently initialized trials. This ordering differs from using reflection to rewrite a plan and from adding every useful-looking program to memory. It also makes accumulation experimentally visible: PRIME_skill reuses capabilities that previously passed the gate. A matched PRIME condition would retain the same online controller without that accumulated seed; that contrast remains unmeasured in this snapshot.

6 Conclusion

PRIME connects dynamic scheduling, evidence-gated skill development and reuse, and environment-grounded feedback through a shared subgoal contract. The resulting loop can change which unfinished behavior runs next without forgetting verified progress, and can distinguish a need for better control from a need for additional capability. Across five separately reset trials per model–method–task cell, PRIME_skill majority-solves all three 100-log task blocks, reaches 6/7 tasks with GPT and 5/7 with Grok, and is less uniform with Gemini. Grok also majority-solves the breeding-and-milk conjunction. A fixed-controller comparison finds 4/7 versus 6/7 majority-solved tasks with 16 versus 20 accumulated skills, without uniform timing improvements. These results support the coupled design but do not isolate the three modules causally. Matched-library controller comparisons, a genuine no-accumulated-seed condition, and broader seeds remain necessary.

AI use statement

We used generative AI tools to assist with literature search and summarization, software development and debugging, analysis workflow development, and language editing and drafting of the manuscript. AI tools were also used to provide feedback during the development and refinement of experimental analyses. All AI-assisted code, experimental procedures, analyses, citations, and written content were manually reviewed and verified by the authors against the original sources and experimental outputs. All experiments were executed by the authors, and all reported results were obtained from the resulting experimental runs. The authors determined the final methodology, interpretation of results, and conclusions, and take full responsibility for the content of this work.

Ethics statement

This work uses a simulated game and no human participants, private data, or user profiling. Generated code is restricted to a bounded Mineflayer interface, screened against privileged and result-forcing commands, and validated in isolated worlds with external state checks. These safeguards are not a security sandbox; deployment on untrusted systems requires stronger isolation and review.

Reproducibility statement

Section 3 specifies tasks, models, baselines, trials, and metrics. The supplementary artifact records task initializers and hard checkers, prompts and adapters, frozen PRIME source, the frozen learned-skill records and validation metadata, model identifiers and reasoning effort, runtime snapshots, embedding-model revision, budgets, run scripts, and file hashes. Appendices A and B document the scheduling, admission, comparison, and accounting controls needed to reconstruct the reported evaluation.

References

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. AriGraph: Learning knowledge graph world models with episodic memory for LLM agents. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 12–20, 2025. doi: 10.24963/ijcai.2025/2. URL https://www.ijcai.org/proceedings/2025/2.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching unlabeled online videos. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24639–24654, 2022. doi: 10.52202/068431-1789.

Anirudh Chari, Suraj Reddy, Aditya Tiwari, Richard Lian, and Brian Zhou. MINDSTORES: Memory-informed neural decision synthesis for task-oriented reinforcement in embodied systems. *arXiv preprint arXiv:2501.19318*, 2025. URL https://arxiv.org/abs/2501.19318. Presented at the ICLR 2025 Workshop on Reasoning and Planning for Large Language Models.

Weihong Du, Wenrui Liao, Binyu Yan, Hongru Liang, Anthony G. Cohn, and Wenqiang Lei. BAR: A backward reasoning based agent for complex Minecraft tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6126–6149, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.318. URL https://aclanthology.org/2025.findings-acl.318/.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-1333. URL https://openreview.net/forum?id=rc8o_j8I8PX.

Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 677–682, 1987. URL https://aaai.org/papers/00677-AAAI87-121-reactive-reasoning-and-planning/.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation*, pp. 9493–9500. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila A. McIlraith. STEVE-1: A generative model for text-to-behavior in Minecraft. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-3064.

Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. Odyssey: Empowering Minecraft agents with open-world skills. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 187–195, 2025. doi: 10.24963/ijcai.2025/22. URL https://www.ijcai.org/proceedings/2025/22.

Junyeong Park, Junmo Cho, and Sungjin Ahn. MrSteve: Instruction-following agents in Minecraft with what-where-when memory. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CjXaMI2kUH.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1480. URL https://openreview.net/forum?id=KtvPdGb31Z.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. JARVIS-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024b. doi: 10.1109/TPAMI.2024.3511593. URL https://doi.org/10.1109/TPAMI.2024.3511593.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yanfang Zhou, Xiaodong Li, Yuntao Liu, Yongqiang Zhao, Xintong Wang, Zhenyu Li, Jinlong Tian, and Xinhai Xu. M2PA: A multi-memory planning agent for open worlds inspired by cognitive theory. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 23204–23220, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1191. URL https://aclanthology.org/2025.findings-acl.1191/.

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

A Implementation Details

Agenda and scheduling. PRIME agenda records contain `subgoal`, `skill`, `priority`, `repeat`, `requires`, `produces`, `success_criteria`, and `reason`, plus runtime status. The configured score weights are 60 for risk/recovery urgency, 15 for observed partial progress, 25 for dependency bottlenecks, 25 for observation-grounded opportunity, and 20 for the failure penalty. The `max_revisions` threshold is set to 12 and is applied to the consecutive no-progress revision counter; measured task progress or a newly admitted capability resets that counter. A selected item executes one call before yielding to observation and scheduling, regardless of its remaining repeat count.

Revision evidence. The scheduler re-observes and rescores after every bounded call. The reviser receives the original goal and unchanged hard task criterion, the latest structured observation, available skill descriptions and interfaces, the previous agenda, recent execution feedback, historical failures, and stored spatial evidence. Agenda revision can be triggered by the absence of a feasible item; refreshed state making a selected item already satisfied or infeasible; entity lifecycle, danger, or health transitions; or failure without expected progress. Skill evolution additionally requires that the latest agenda is an evidence-informed revision and that the concrete missing item is neither satisfied nor blocked by an unmet prerequisite. The evolver must also be available, the item must have positive base priority, and an unchanged rejected context cannot be retried without new progress or input evidence. There is one latency-sensitive exception to revising first. When danger is already

PRIME execution with selective skill evolution

1. Initialize $\mathcal{A}_0 \leftarrow \textsc{Plan}(g, o_0, \mathcal{S}_0)$.
2. While budget remains and $c_g(o_t, h_t) = 0$:
   (a) Update agenda status from o_t and the hard checker.
   (b) If revision evidence holds and the pure-recovery fast path above does not apply, set $\mathcal{A}_t \leftarrow \textsc{Revise}(\mathcal{A}_{t-1}, o_t, h_t)$.
   (c) If the revised agenda exposes a confirmed gap u, generate and independently validate a candidate s; add s to $\mathcal{S}_t$ only if admitted, then revise the agenda.
   (d) Form feasible grounded set $\mathcal{F}_t$. If empty, record the blocker and return to revision.
   (e) Execute one bounded call of $\arg\max_{z_i \in \mathcal{F}_t} S_t(z_i)$; observe o_{t+1} and append feedback to h_t.
3. Return external decision $c_g(o_T, h_T)$ and measured execution time.

Figure 4: PRIME PRIME control flow. Revision and evolution are conditional; they are not mandatory stages between every pair of skill calls.

observed and a revision is pending, the scheduler may execute once the unchanged, highest-scoring agenda item only if it is an already grounded, direct-execution, pure status-recovery action whose declared inputs remain satisfied. This fast path cannot author an action or enter skill evolution, and the revision remains pending.

Execution bounds. The formal frozen PRIME launcher permits at most 400 scheduler steps and 240 ordinary skill calls in an episode. The episode timeout is 9,000 seconds. Risk-aware yielding is enabled at helper, query, navigation, and digging boundaries. These checks return control to the scheduler; they do not issue privileged Minecraft state changes.

Candidate screening and validation. Generated skills are complete asynchronous Mineflayer functions with bounded waits and retries. Static screening rejects result-forcing or server-administration behavior, including `/give`, `/tp`, `/summon`, and `/setblock`. A surviving candidate must obtain two consecutive warm-up successes within at most six attempts. It is then evaluated in ten independently reinitialized final trials. Admission under `prime-independent-v1` requires both executor completion and the subgoal-specific external criterion in at least seven trials (final success rate greater than 0.6). The validation path is bounded by 1,200 skill calls and 7,200 seconds. Candidate code is fixed across a final block. Evidence from a rejected block may condition a later bounded generation, but all warm-up and final-trial credit restarts at zero for the new candidate.

Persistent records and the reuse seed. An admitted record stores executable code, task and subgoal context, the success criterion, validation protocol, final trial count, measured success rate, validation contexts, and update metadata. Reuse is attempted through exact-name and normalized-subgoal matching before new generation. The frozen learned-skill artifact contains 20 unique admitted records, each with ten final validation trials. It includes `breedExactlyTwoNearbyCows` at 0.90 validation success. These records were produced during earlier PRIME skill development and subsequently frozen for the formal reuse condition. In addition to the previously resolved admitted set, the snapshot includes four explicitly curated admitted additions:

- `mineOneDryReachableWoodLog`,
- `mineIronInDifficultTerrainFallback`,
- `mineNineCobblestoneDry`, and
- `collectNearbyDroppedAcaciaLog`.

All 20 satisfy the same ten-final-trial admission requirement. The composition and its hash were fixed before the factorial campaign and do not vary by model, task, trial, or observed formal outcome. Appendix Table 4 reports the complete frozen inventory and final-trial pass counts.

Objective checkers. Inventory and health criteria are read from environment state. Task 5 additionally queries the server statistic `minecraft.custom:minecraft.animals_bred`; the

Table 2: Implemented decision protocols. "Shared base" denotes the same task-facing component interfaces in the common harness; it does not include PRIME's 20-skill accumulated seed. Odyssey-GPT is shown separately because its native skill and execution stack is part of that system baseline.

Method	Planning state	Observation and feedback use	Executable repertoire
Sequential	One ordered subgoal list and a cursor	Plans once from the initial observation; advances after the runner finishes the current item	Shared base
ReAct	One immediate subgoal per decision	Chooses again from the current observation, accumulated progress, and last execution feedback	Shared base
DEPS	Description, explanation, candidate plan, and one selected subgoal	Repeats the structured cycle at each decision using completed, failed, and last-step feedback	Shared base
BAR	Backward prerequisite decomposition with an execution cursor	Caches the decomposition and selects among remaining steps using current execution state; records last-step feedback	Shared base
MindStores	One initial ReviewExperience → PlanGeneration → PredictOutcome route, with conditional pre-execution revision	Initial observation conditions retrieval, prediction, and revision; formal trials retrieve no tuples, and later outcomes are recorded only for provenance	Shared base; empty per-trial experience store
PRIME_skill	Stateful feasible-item agenda	Re-observes and rescores after every bounded call; revises on evidence and evolves only at a confirmed grounding gap	Shared base plus frozen 20-skill seed and any newly admitted skills
Odyssey-GPT	Native planner–action loop with QA, Critic, and Comment roles	Native retrieval, critique, and interaction flow	Native Odyssey skill library and executor

judge creates and reads the objective outside model-controlled code. Task 7 uses a separate watcher: any new death fails the episode, and missing complete zero-death evidence cannot pass. Task completion never follows solely from an agent or skill return value.

B Additional Experimental Controls

Five-trial accounting. The expected index contains three models, seven methods, seven tasks, and five separately reset trials per cell, for 735 main-comparison episodes. Existing valid task failures and successes are retained, and administrative retries do not increase the sample size. Aggregation materializes the full index before computing each five-run task mean, so missing records cannot silently disappear. The supplementary data retain source paths, hashes, trial identifiers, and normalized checker outcomes.

Model control. All methods use `gpt-5.6-sol`, `gemini-3.8-flash`, or `grok-4.7` with reasoning effort fixed to `medium`. For the five planner-style baselines and PRIME, the selected model serves the method's configured high-level calls in the common campaign harness. No method receives the source fixture's static subgoal plan or author notes.

Frozen runtime. The artifact pins Minecraft 1.19.4 with Fabric 0.14.21, CPython 3.9.23, Node.js 20.20.2, and the exact Odyssey/Mineflayer source and dependency snapshot used by the campaign. Retrieval and mapping use a frozen `paraphrase-multilingual-MiniLM-L12-v2` embedding model; its exact repository revision and file hashes are recorded in the supplementary manifest. This embedding model is not replaced when the evaluated language model changes. Runtime sources, task artifacts, and the container image are likewise hash-pinned.

Adapter boundary. Sequential, ReAct, DEPS, BAR, and MindStores are in-repository implementations of the named decision protocols over the shared runner interfaces; they are not executions of the original projects' full software stacks. Each trial starts with fresh planner state. In particular, the formal launcher passes no external experience database to MindStores: retrieval is empty at trial start, the adapter may predict and revise its initial plan, and resulting tuples are recorded for provenance but are not transferred to another trial. Because this adapter generates its plan before execution, tuples recorded later in the same episode cannot alter that already generated plan. Odyssey-GPT is the sole native-system baseline.

Odyssey-GPT control. Odyssey-GPT retains the native Odyssey planner/action loop, prompts, skill implementation, skill retrieval, embedding model, role topology, and Minecraft/Mineflayer runtime. Within a model block, the Action Agent, Planner Agent, Planner QA, Critic Agent, and Comment Agent are all assigned the same tested model at medium effort. The historical "GPT" suffix is kept only as the baseline name. The external task initializer and hard checker are shared with the formal evaluation, but Odyssey's internal execution stack is not replaced by PRIME's. Results are therefore interpreted as a system-level comparison rather than a planner-component ablation.

Table 3: Five-trial task-level outcomes by backbone. P / seconds denotes a majority-solved task and its mean successful active execution time; F denotes fewer than three hard-checker passes. A dash indicates that the success-conditioned timing is not available for audit. Odyssey-GPT uses native action timing.

Method	T1	T2	T3	T4	T5	T6	T7
`gpt-5.6-sol`							
Sequential	F	F	P / 215.4	F	F	P / 148.6	F
ReAct	F	P / 114.8	P / 291.4	P / 354.5	F	P / 322.2	F
DEPS	F	P / 103.4	P / 447.8	P / 421.6	F	P / 191.8	F
BAR	F	P / 53.2	P / 248.1	F	F	F	F
MindStores	F	F	P / 283.2	P / 333.2	F	F	F
PRIME_skill	P / 646.6	P / 37.4	P / 337.4	P / 201.9	F	P / 197.8	P / 159.9
Odyssey-GPT	F	P / 98.7	F	P / 518.8	F	F	F
`gemini-3.8-flash`							
Sequential	F	F	P / 221.1	F	F	F	F
ReAct	F	P / 115.1	P / 261.8	F	F	P / 117.0	F
DEPS	F	P / 114.9	P / 320.2	F	F	P / 74.5	F
BAR	F	P / 118.1	F	F	F	F	F
MindStores	F	P / 70.2	P / 276.3	P / 467.6	F	F	F
PRIME_skill	P / 767.0	P / 47.1	P / 631.1	F	F	F	F
Odyssey-GPT	F	P / 233.6	F	F	F	P / 96.1	F
`grok-4.7`							
Sequential	F	P / 63.2	P / 319.2	F	F	F	F
ReAct	F	P / 113.8	P / 265.4	F	F	P / 133.6	F
DEPS	F	P / 200.0	P / 836.5	F	F	P / 86.3	F
BAR	F	P / 79.5	P / 320.5	F	F	F	F
MindStores	F	P / 69.5	P / 357.3	P / 1089.4	F	F	F
PRIME_skill	P / 712.9	P / 27.9	P / 285.8	P / 260.2	P / –	F	F
Odyssey-GPT	F	P / 109.5	P / 221.8	F	F	F	F

Task initialization. Every trial begins from its task-specific reset. Tasks 1–6 use an empty inventory. Task 7 starts with one iron sword, fixed midnight, easy difficulty, and mob spawning enabled after setup. The fixture places six reachable logs; the remaining logs must be obtained through ordinary world interaction. Task 5 supplies exactly three cows and four mature wheat plants. Task 4 supplies a controlled nearby resource field but no crafted tools or inventory.

Timing. The active timing column uses `effective_execution_time_sec`; wall time is reported separately using `wall_clock_time_sec`. All successful records in this snapshot provide both metrics. In the common runner the active measure is mapper plus live-skill time; in Odyssey it is elapsed native environment-step time for nonempty action submissions. It excludes planner LLM calls, initialization, reset/recovery overhead, and PRIME's generation and validation phase. The underlying exporter can fall back to wall time for some failed records, but this paper never treats that fallback as active completion time. Traces retain `wall_clock_time_sec`, planner, mapper, skill, and evolution timings for audit. Timing tables condition on hard-checker success and always appear beside SR; failure and timeout durations are retained but are not presented as successful completion times.

Uncertainty and unmeasured contrasts. Every model–method–task cell contains five separately reset trials. Task-level SR is their arithmetic mean, so one outcome changes a cell mean by 20 percentage points. The compact solved-task counts apply a predeclared three-of-five majority threshold; they do not replace the underlying SR. The available 16-versus-20-record comparison also uses five trials per task and condition. It compares two accumulated libraries and cannot isolate individual skills, online learning, or scheduler terms.

Where the difference occurs. PRIME_skill majority-solves the 100-log task under all three backbones, while none of the baseline model–method blocks reaches the same task-level criterion. This is the clearest recurring difference: preserving progress over many bounded collection calls remains difficult even when short crafting chains are solvable. Midnight collection remains stricter because every run must also preserve positive health and zero new deaths. The Grok result on Task 5 shows that PRIME can complete both the breeding and milk branches, but the lower and less uniform results on other backbones caution against treating the task as universally solved.

Table 4: Frozen accumulated skill library. Pass counts refer to independent skill-validation trials, not the five-run formal task outcomes.

Skill record	Pass/10	Skill record	Pass/10
`breedExactlyTwoNearbyCows`	9	`placeCraftWoodenPickaxe AndRecoverTable`	10
`collectNearbyDroppedAcaciaLog`	10	`placeFurnaceSmeltThreeRawIron`	10
`exitWaterAndStandOnSolidGround`	10	`searchAndApproachTwoCows`	9
`harvestOneAdditionalWheat`	10	`searchBeyondCoveredArea AndApproachTwoCows`	7
`harvestTwoMatureWheat`	10	`searchDryCoastBeyondAndMineOneLog`	7
`mineIronInDifficultTerrainFallback`	8	`searchFartherAndHarvestMatureWheat`	8
`mineNineCobblestoneDry`	10	`searchUnexploredTerrain AndApproachTwoCows`	10
`mineOneDryReachableWoodLog`	10	`searchUnexploredTerrainMineOneLog`	10
`mineReachableIronUntilThree`	10	`separateFromNearbyDrowned OnDryGround`	10
`mineReachableWoodLog`	10	`smeltThreeRawIronWithExactWoodFuel`	10

Every record uses `prime-independent-v1` with ten final trials; admission requires a final success rate greater than 0.6. Pass counts are validation provenance, not formal task results.

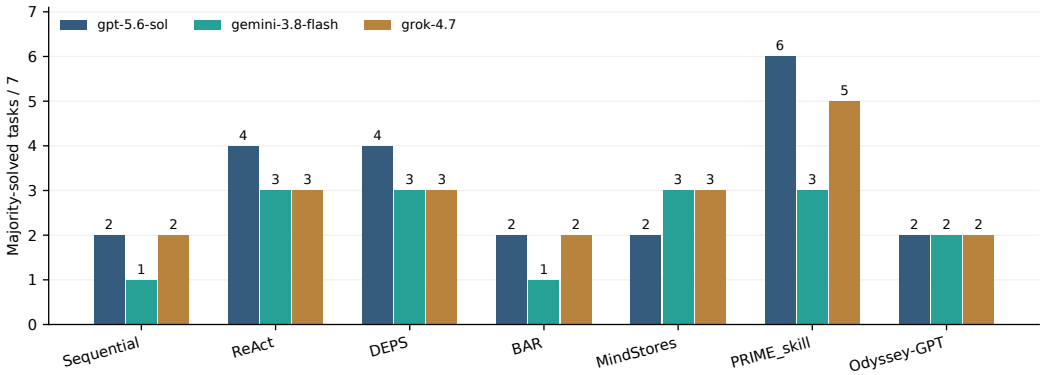


Figure 5: Majority-solved tasks after five independent trials per model–method–task cell. A bar ranges from zero to seven and counts tasks whose five-run mean SR is at least 0.6. The Grok PRIME_skill bar is 5/7.

Time must be interpreted at task level. For Task 2, PRIME_skill uses 37.4, 47.1, and 27.9 seconds of mean successful active execution under GPT, Gemini, and Grok, respectively; the fastest successful common-harness baseline in each block uses 53.2, 70.2, and 63.2 seconds. This is a local execution advantage, not evidence of a universal end-to-end speedup. Planning and capability development can dominate wall time, and Table 1 therefore reports A and W separately.

B.1 Skill Reuse Analysis: Larger Coverage, Not Uniform Acceleration

The fixed-controller comparison uses 16 versus 20 already accumulated skills with GPT at medium effort and the same task reset. Each task and library condition is run five times, for $2 \times 7 \times 5 = 70$ episodes. Both conditions permit online development. This is a starting-library contrast, not skills versus no skills and not online evolution on versus off.

The expanded seed increases majority-solved task coverage from 4/7 to 6/7. Tasks 4 and 6 cross the majority threshold, the four previous successes remain, and Task 5 does not cross it in the GPT block. This pattern is consistent with useful accumulated capability but does not isolate which added record causes each change.

Table 5: Starting-library comparison with five trials per task and condition. Passes counts majority-solved tasks; P/F in task columns applies the same three-of-five criterion. Times are five-run mean successful active seconds.

Method	Seed size	Passes	T1	T2	T3	T4	T5	T6	T7
PRIME_skill	16	4/7	P / 641.3	P / 21.2	P / 353.7	F	F	F	P / 125.7
PRIME_skill	20	6/7	P / 646.6	P / 37.4	P / 337.4	P / 201.9	F	P / 197.8	P / 159.9

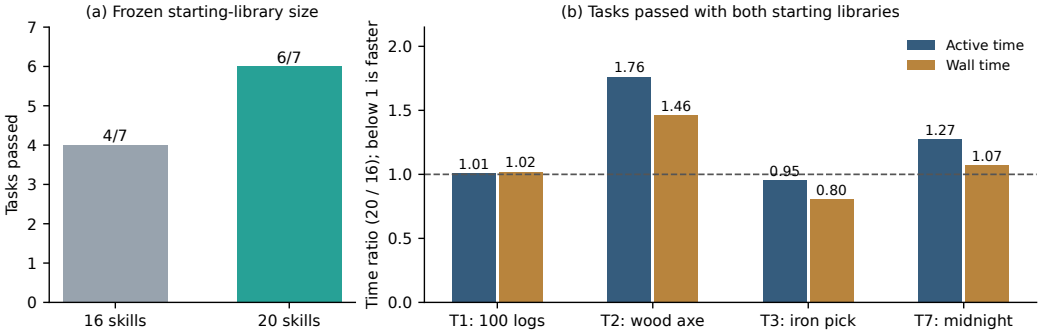


Figure 6: Starting-library analysis with GPT. (a) Majority-solved task coverage increases from 4/7 to 6/7. (b) Time ratios use only tasks majority-solved under both conditions. Values below one favor the 20-record seed; increased coverage does not imply uniform acceleration.

Table 6: Task goals and fixed hard-checker evidence. Success is established from environment state, never from an agent completion message.

ID	Goal	Required environment evidence
1	Collect 100 logs	Total log inventory ≥ 100
2	Craft a wooden axe	`wooden_axe` ≥ 1
3	Craft an iron pickaxe	`iron_pickaxe` ≥ 1
4	Obtain a diamond	`diamond` ≥ 1
5	Breed cows and obtain milk	`animals_bred` increases by ≥ 1 and `milk_bucket` ≥ 1
6	Prepare for zombie combat	`rotten_flesh` ≥ 1 (kill proxy)
7	Collect 20 logs at midnight	Logs ≥ 20, positive health, and complete evidence of zero new deaths