HARNESS-MAP: BUDGET-EFFICIENT HARNESS SEARCH VIA STRUCTURE-TOKEN FACTORIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Automating harness evolution via naive meta-agent search is prohibitively expensive, requiring hundreds of meta-agent calls and minutes-to-hours per evaluation. *How can we efficiently search for a diverse set of high-quality harnesses under a tight evaluation budget?* To address this, we introduce **Harness-MAP** to structure the search space, factorizing graph-level program structures from token-level implementations. Bilevel search over this structure-token representation space illuminates an archive of diverse, high-quality harnesses. The map also enables a resolution-adaptive search strategy that navigates this space efficiently, sub-splitting partitions with high fitness variance while retracting ineffective refinements. Across three interactive spatial navigation benchmarks with the frozen Qwen3-VL-30B-A3B-Instruct backbone model, our search approach rapidly achieves performance leaps under minimal evaluation budgets: $10.7\% \rightarrow 100.0\%$ on SpatialWorld-3dgames (39 evals), $10.0\% \rightarrow 90.0\%$ on BALROG-MiniHack (41 evals), and $32.5\% \rightarrow 46.5\%$ on SimWorld-Robotics (6 evals). Harness-MAP outperforms prior harness evolution methods across both search and held-out tasks, establishing factorized representations as an effective Quality-Diversity paradigm for budget-aware agent harness search.

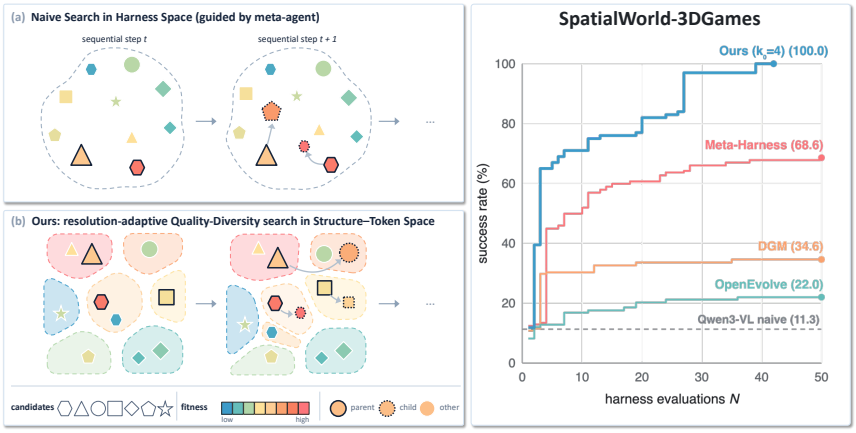

Figure 1: **Conceptual overview and benchmark performance of Harness-MAP. (a) Naive search** rapidly collapses into narrow local lineages over a metric-free search space. **(b) Harness-MAP (Ours)** factorizes candidates into a structure-token space and dynamically sub-splits high-variance cells (at $t+1$) with resolution-adaptive search. **(c) Performance growth** on SpatialWorld-3DGames shows that our method achieves a state-of-the-art Pareto frontier, outperforming Meta-Harness (Lee et al., 2026) and Map-Elites (Mouret & Clune, 2015)-based OpenEvolve (Sharma, 2025).

1 INTRODUCTION

What turns a foundation model into an agent is the harness: a set of components (e.g., the prompts it reads, the tools it may call, the memory it maintains) together with the control flow that connects them. Beyond model scale, improving the harness offers a practical, cost-effective paradigm to significantly boost agent performance (Luo et al., 2026). Harness optimization is therefore essential

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.

for building effective agents. To automate this process, recent paradigms have shifted toward harness evolution, progressing from manual engineering to iterative search by an LLM proposer. Emerging work on Recursive Self-Improvement (RSI) (Lin et al., 2026; Chen et al., 2026a; Lee et al., 2026; Zhang et al., 2026c) has increasingly framed harness search as a gradient-free code optimization problem, where an LLM proposer repeatedly rewrites the harness using execution feedback to guide revisions and task performance as its optimization signal. However, automating harness evolution via naive meta-agent search is prohibitively expensive. A full search loop may require hundreds of meta-agent calls and a single candidate evaluation can take minutes to hours, which remains highly sample-inefficient and wasteful (e.g. numbers).

Under a fixed evaluation budget, practical success of evolving effective harnesses is highly bound to search efficiency. We therefore ask: *How can we **efficiently** search for a **diverse** set of high-**quality** harnesses under a tight **evaluation budget**?* This poses an inherent trade-off between search efficiency and diversity, yet prior works have either focused on diversity-oriented (Zhang et al., 2026b; Weng et al.; Luo et al., 2026) or efficiency-oriented (Lange et al., 2026) methods separately, rather than co-optimizing both. To better navigate this trade-off, a core issue in prior works is the unstructured space, where *a harness has no coordinates, so a search step has no size and a search trajectory has no coverage*. Such limitations tend to result in blind search without a broader landscape of potential harnesses, and thus cause inefficient compute allocation.

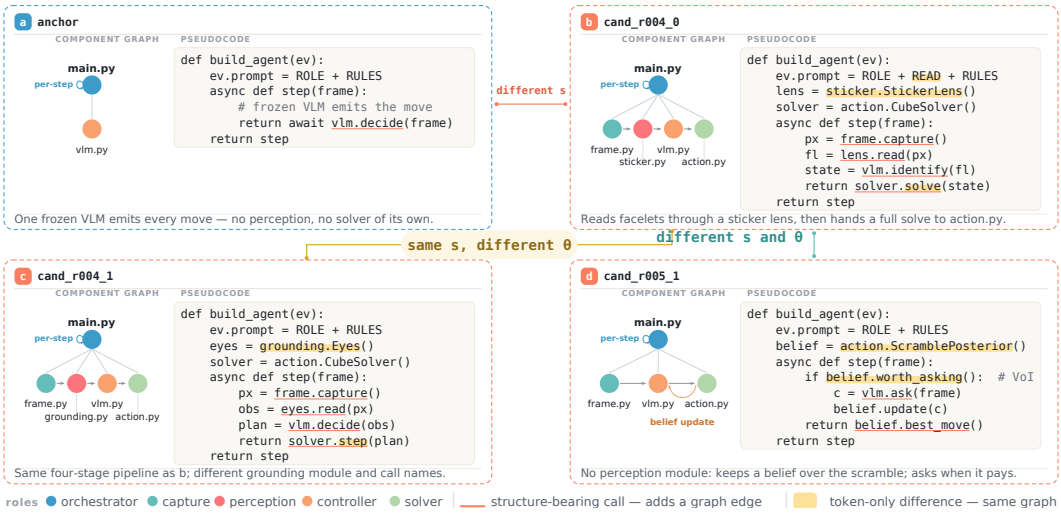

Figure 2: **Concrete harness examples illustrating structure-token factorization.** Candidates from a Rubik's cube corpus exemplify cell placement in CVT (Centroidal Voronoi Tessellation) clustering. ***a* vs. *b* (Δs):** Structural module additions lead to reassign the anchor harness a from Cell A to Cell B. ***b* vs. *c* ($\Delta\theta$ only):** Same-step candidates with identical structures (s) but distinct token-level implementations (θ) stay in Cell B. ***b* vs. *d* ($\Delta s, \Delta\theta$):** A subsequent candidate altering both coordinates still maps to Cell B due to structure-token proximity.

To close this gap, we propose **Harness-MAP**, structuring harness candidates into a factorized representation space, and searching across two levels. High-level steps refine the structural skeleton shared across a family of harnesses, while low-level steps flesh out that skeleton with token-level implementations. Specifically, this structured coordinate space unlocks two key capabilities: *(i) The search loop achieves quantifiable search progress towards diverse, high-quality harness candidates.* Factorizing graph-level structures from token-level implementations provides a structured coordinate space for harnesses. This allows the search loop to explicitly measure search step sizes and quantify space coverage, transforming evolution from localized drift into diverse exploration. *(ii) The map enables adaptive budget allocation, focusing search effort on high-potential regions.* Instead of searching at fixed step sizes, the map dynamically adjusts its resolution. Regions exhibiting high performance variance are sub-divided for deeper exploration, while ineffective refinements are retracted otherwise. Evaluation budget is thus concentrated where deeper search yields performance gains. In this way, Harness-MAP breaks the fundamental trade-off between search efficiency and diversity, discovering a diverse set of high-quality harnesses under tight evaluation budgets.

Our contributions are summarized as follows:

(1) We introduce a *factorized representation space for harnesses* that decouples graph-level program structures from token-level implementations, forming a coordinate space where search step sizes and exploration coverage become explicitly quantifiable.
(2) This structured map enables an efficient *resolution-adaptive search framework* that dynamically optimizes search resolution based on local performance variance.
(3) Driven by this search paradigm, our method establishes a *state-of-the-art Pareto frontier* across three interactive spatial navigation benchmarks with the frozen Qwen3-VL-30B-A3B-Instruct model: $10.7\% \rightarrow 100.0\%$ on SpatialWorld-3dgames (39 evals), $10.0\% \rightarrow 90.0\%$ on BALROG-MiniHack (41 evals), and $32.5\% \rightarrow 46.5\%$ on SimWorld-Robotics (6 evals).

2 Related Work

Harness Evolution and LLM-Guided Evolutionary Search. An agent harness builds executable program around foundation models (Luo et al., 2026; Lin et al., 2026). Recent Recursive Self-Improvement (RSI) paradigms automate harness optimization via execution-trace feedback (Lee et al., 2026; Chen et al., 2026b; Agrawal et al., 2026b). Because harness code spaces are discrete, non-differentiable, and prone to local optimization collapse (Ursekar et al., 2026), gradient-free evolutionary algorithms offer an ideal solution to search for diverse harnesses by using LLMs as context-aware mutation operators (Santos et al., 2025; Lehman et al., 2023; Zhang et al., 2026b). Existing works mitigate premature convergence through code archiving or failure tracking (Luo et al., 2026; Zhang et al., 2026b; Weng et al.; Zhang et al., 2026a), but they rely on localized metrics that lack coordinates to quantify coverage. Harness-MAP addresses this gap by projecting candidates onto a factorized representation space, establishing explicit coordinates for evolutionary exploration.

Quality-Diversity Search and CVT-MAP-Elites. Quality-Diversity (QD) algorithms maintain archives of diverse, high-performing solutions across behavior space descriptors (Pugh et al., 2016; Cully & Demiris, 2017). As a foundational QD technique, MAP-Elites (Mouret & Clune, 2015) discretizes the behavior space into a fixed grid, retaining only the highest-performing candidate within each bin. However, this rigid grid structure cause severe spatial sparsity and intractable archive sizes in high-dimensional spaces. To address this bottleneck, CVT-MAP-Elites (Vassiliades et al., 2017) replaces fixed grids with Centroidal Voronoi Tessellations (CVT) fitted via k-means clustering, ensuring uniform region density across behavior spaces. Although LLMs have been integrated into QD pipelines (Cully, 2019; Guo et al., 2026; Lehman et al., 2023; Bradley et al., 2024), existing code-level methods still rely on surface-level text heuristics (Sharma, 2025) and static partitions that waste rollout budget on saturated regions. Harness-MAP overcomes both limitations. Specifically, it replaces surface-level text heuristics via factorized structure-token descriptors, while improving search efficiency through an adaptive-resolution mechanism at zero rollout cost (see the conceptual comparison of quality-diversity archive partitioning in Fig. 5).

3 Method

3.1 Overview

Problem Formulation of LLM-Guided Harness Search. Given a frozen foundation model M and a task distribution $\mathcal{X}$, a harness $H \in \mathcal{H}$ is the executable program that turns M into a functional agent. Executing H on $x \sim \mathcal{X}$ yields a trajectory $\tau \sim p_M(H, x)$ scored by reward $r(\tau, x)$. Harness search seeks the optimal harness program $H^\star$ (Lee et al., 2026):

$$H^\star \;=\; \arg\max_{H \in \mathcal{H}} \; \mathbb{E}_{x \sim \mathcal{X},\, \tau \sim p_M(H,x)} \big[\, r(\tau, x) \,\big]. \tag{1}$$

Existing harness evolution paradigms conduct search directly over the *raw, token-level space* $\mathcal{H}$ of possible harnesses, relying on an LLM proposer to iteratively edit low-level programs ($H \rightarrow H'$). However, since this unstructured token space lacks coordinate metrics, these methods search blindly without insight into unexplored regions or search coverage. This limitation motivates our structured coordinate space, providing a multi-resolution view of the harness landscape.

Search on the Structured Harness-MAP. To this end, we introduce Harness-MAP, a factorized representation space that restructures the raw space $\mathcal{H}$ into a map of cell partitions $\mathcal{C}$ (§3.2). Each cell $c \in \mathcal{C}$ defines a distinct sub-space $c \subset \mathcal{H}$. During search, the framework maintains an archive $\mathcal{A} = \{H_c\}_{c \in \mathcal{C}}$ storing the *elite candidate* H_c with best-so-far performance for each cell c:

$$H_c = \arg\max_{H \in c} \mathbb{E}_{x \sim \mathcal{X},\, \tau \sim p_M(H,x)} \big[\, r(\tau, x) \,\big]. \tag{2}$$

Searching over this structured map directly overcomes the limitations of evolving in a metric-free space. First, occupied cells provide an explicit view for search coverage and fitness distribution (Tab. 2), revealing unexplored regions and preserving population diversity. Second, mapping harness candidates to this coordinate space makes search step sizes measurable along visible trajectories (e.g., intra-cell movement for minor edits and inter-cell for major structural shifts; Figs. 1 and 2). Therefore, this multi-resolution view converts blind re-writing into a traceable search process.

3.2 Harness-MAP Construction

Searching for a diverse set of high-quality harnesses requires illuminating the global search space rather than tracking a single lineage. Quality-Diversity (QD) methods provide this global view by archiving elites over a factorized coordinate space φ. Built on CVT-MAP-Elites (Vassiliades et al., 2017), our Harness-MAP constructs this representation space directly by factorizing each harness into a structural skeleton and token-level implementation.

Factorizing a Harness. We formalize this factorization as $H = (s, \theta)$, decoupling each candidate into a structural graph s and token-level content θ. The joint descriptor is:

$$\varphi(H) \;=\; \big(\varphi_S(s);\, \varphi_T(\theta)\big) \;\in\; \mathbb{R}^{d_S} \times \mathbb{R}^{d_T}, \tag{3}$$

allowing structural and token moves to be measured independently. Along non-redundant dimensions, this design disentangles high-level structural mutations from low-level code edits (§ A.8).

(a) Structural descriptor $\varphi_S(s)$. The skeleton of harness s represents its top-level architectural wiring, independent of internal execution. We encode this wiring as a directed graph $G(s) = (V, E)$, where vertices V are top-level components (e.g., functions, classes, modules) and edges E denote call, reference, or import dependencies extracted via Abstract Syntax Tree (AST) parsing. Crucially, programs inside components are parsed solely for inter-component references, delegating internal logic to the token descriptor. Vertices $v \in V$ receive initial functional role labels $\ell_0(v)$ (e.g., perception, policy, controller; § A.6). To capture role-aware multi-hop topology, we apply L-round Weisfeiler–Lehman (WL) refinement method (Shervashidze et al., 2011), as it provides a deterministic, training-free metric invariant to code refactoring (§ A.6):

$$\ell_{t+1}(v) \;=\; \mathrm{hash}\big(\ell_t(v),\, \{\!\{\, \ell_t(u) : u \in \mathcal{N}(v) \,\}\!\}\big), \tag{4}$$

where $\{\!\{\cdot\}\!\}$ is a multiset over neighborhood $\mathcal{N}(v)$. After t rounds, $\ell_t(v)$ forms a hash encoding the t-hop role-topology surrounding component v. We define the structural descriptor $\varphi_S(s) \in \mathbb{R}^{d_S}$ as a normalized histogram over the codebase-wide pattern space $\Lambda = \bigcup_{t=0}^{L} \Lambda_t$:

$$\varphi_S(s) \;=\; \left[\, \frac{\#_t(\lambda)}{|V|} \,\right]_{t \le L,\ \lambda \in \Lambda_t}, \tag{5}$$

where $\#_t(\lambda)$ counts vertices carrying pattern λ in harness s, and $d_S = |\Lambda|$ is the total number of unique structural patterns observed across the harness codebase. In this structural subspace $\mathbb{R}^{d_S}$, each dimension represents a specific t-hop component connectivity pattern, and its value denotes the proportion of components in the harness exhibiting that pattern. At $t = 0$, φ_S reduces to base component proportions for guiding structural-directive edits (§3.3).

(b) Token-level behavior descriptor $\varphi_T(\theta)$. To capture internal implementation logic, we extract three feature blocks, including AST for control-flow statistics f_{AST}, numeric constant parameters f_{num}, and documentation embeddings f_{doc} (empirical analysis in § A.8). Each block is z-score

Algorithm 1 Comparison between standard CVT-MAP-Elites and Harness MAP.

(a) CVT-MAP-Elites

procedure CVT-MAP-ELITES(k)
  $C \longleftarrow$ CVT(k) ▷ *CVT centroids*
  $(\mathcal{X}, \mathcal{P}) \longleftarrow$ create_archive(k)
  for $i = 1 \rightarrow G$ **do** ▷ *Initialization*
    $\mathbf{x} =$ random_solution()
    ADD_TO_ARCHIVE($\mathbf{x}, \mathcal{X}, \mathcal{P}$)
  end for
  for $i = 1 \rightarrow I$ **do** ▷ *Main loop, I iterations*
    $\mathbf{x} =$ selection($\mathcal{X}$)
    $\mathbf{x}' =$ variation($\mathbf{x}$)
    ADD_TO_ARCHIVE($\mathbf{x}', \mathcal{X}, \mathcal{P}$)
  end for
  return archive $(\mathcal{X}, \mathcal{P})$
end procedure

procedure ADD_TO_ARCHIVE($\mathbf{x}, \mathcal{X}, \mathcal{P}$)
  $(p, \mathbf{b}) \longleftarrow$ evaluate($\mathbf{x}$)
  $c \longleftarrow$ closest_centroid($\mathbf{b}, C$)
  if $\mathcal{P}(c) =$ null or $\mathcal{P}(c) < p$ **then**
    $\mathcal{P}(c) \longleftarrow p, \mathcal{X}(c) \longleftarrow \mathbf{x}$
  end if
end procedure

(b) Harness MAP (Ours)

procedure HARNESSMAP(k_0)
  $C \longleftarrow$ CVT(k_0) ▷ *Base Voronoi grid*
  $(\mathcal{H}, \mathcal{P}) \longleftarrow$ create_archive(k_0)
  while count_occupied($\mathcal{H}$) $< \lceil m\% \cdot k_0 \rceil$ **do** ▷ *Coverage init*
    $H =$ meta_agent_propose($\mathcal{H}_0$)
    if is_valid(H) **then** ▷ *Filter invalid harness*
      ADD_TO_ARCHIVE($H, \mathcal{H}, \mathcal{P}$) ▷ *Update cell count*
    end if
  end while
  for $t = 1 \rightarrow T$ **do** ▷ *Main loop, T search steps*
    $\mathcal{H}_{\text{batch}} \longleftarrow$ selection($\mathcal{H}, B$) ▷ *Batched Selection*
    for each $H \in \mathcal{H}_{\text{batch}}$ **do**
      diag $\longleftarrow$ meta_agent.diagnosis($H, \mathcal{H}$)
      for $j = 1 \rightarrow b$ **do** ▷ *Breed b children*
        directive$_j \longleftarrow$ meta_agent.get_directive(H, diag, j)
        $H' =$ meta_agent.variation(H, diag, directive$_j$)
        if is_valid(H') **then**
          ADD_TO_ARCHIVE($H', \mathcal{H}, \mathcal{P}$)
        end if
      end for
    end for
    ADAPT_RESOLUTION($\mathcal{H}, \mathcal{P}, C$) ▷ *Dynamic sub-split*
  end for
  return archive $(\mathcal{H}, \mathcal{P})$
end procedure

standardized, denoted as $\hat{f}$, and rescaled by its feature dimension:

$$\varphi_T(\theta) \;=\; \left[\sqrt{\alpha}\,\frac{\hat{f}_{\text{AST}}}{\sqrt{d_{\text{AST}}}};\; \sqrt{\beta}\,\frac{\hat{f}_{\text{num}}}{\sqrt{d_{\text{num}}}};\; \sqrt{\gamma}\,\frac{\hat{f}_{\text{doc}}}{\sqrt{d_{\text{doc}}}} \right], \tag{6}$$

where $\alpha = \beta = \gamma = 1$. Dimension rescaling prevents high-dimensional embeddings from dominating distance metrics, while component-wise normalization ensures φ_T invariant to codebase size.

Partitioning harness candidates into cells. To prevent high descriptor dimensions from causing severe parition sparsity, we partition the space using Centroidal Voronoi Tessellation (CVT) (Vassiliades et al., 2017). Fitting k centroids $Z = \{z_1, \ldots, z_k\}$ via k-means over sampled descriptors defines each Voronoi cell C_i,

$$C_i \;=\; \left\{ H \in \mathcal{H} : \|\varphi(H) - z_i\| \le \|\varphi(H) - z_j\| \quad \forall j \right\}, \tag{7}$$

enabling candidate assignment via nearest-centroid lookup. Fitting centroids relies on static descriptors and consumes zero evaluation budget. The resolution parameter k controls archive granularity for search step size and is optimized dynamically during search (§3.4).

3.3 Searching over the Harness-MAP

We construct the map built on standard CVT-MAP-Elites (Vassiliades et al., 2017), which maintains quality-diversity in a coordinate space, but it lacks semantic guidance (Alg. 1(a)). Conversely, typical harness evolution methods operate over an unstructured loop (i.e., propose $\rightarrow$ evaluate $\rightarrow$ diagnose $\rightarrow$ refine) driven by LLM feedback. Lacking structured spatial coverage, these unstructured loops suffer from redundant local exploration, wasting expensive rollouts on candidates that yield marginal gains or homogeneous behaviors.

Harness-MAP (Alg. 1(b)) bridges this gap by unifying QD archiving with LLM-guided evolutionary search. Instead of tracking a single lineage or mutating blindly, our loop continuously selects elites across the harness map we built in §3.2, diagnoses local failure modes, and generates targeted harness candidates to systematically expand coverage and optimize performance.

Coverage Initialization. Before main exploration begins, the meta-agent proposes candidate root harnesses to populate the base cells. To avoid unfaithful code before evaluation, each proposal undergoes a post-verifier of validity filtering to avoid cheating shortcuts prior to rollout. Sampling

continues until occupied cells satisfy a target coverage percentage of initial k_0 cells ($m\%k_0$), establishing structurally diverse anchors that prevent initial search steps collapse onto narrow regions.

Directive-Steered Variation. Feedback-conditioned code mutations often suffer from unguided behavior drift. To systematically steer evolution toward targeted map cells, candidate generation is jointly conditioned on the parent harness, an accumulated diagnosis, and a component-aware specific directives (e.g. more or less perception proportions for structural edits, similar implementations of a specific module for token-level mutations). Thus, the textual directives act as an inverse structural or token-level mapping, translating target coordinates into explicit structural (e.g., component organization) or token-level (e.g., control flow) refinements. This targeted generation avoids blind mutation and concentrates compute strictly on functional candidates filling distinct behavioral cells.

3.4 Adapting Map Resolution

Standard MAP-Elites uses static search granularity, which wastes evaluations on uniform cells while ignoring distinct candidates in high-variance ones (see conceptual comparison of QD archive partitioning in §A.6). To address this, at each search step t, Harness-MAP dynamically adapts cell resolution at zero evaluation cost by treating sub-splitting as a localized data-driven hypothesis (Alg. 2).

Sub-splitting. At each step, we select the single occupied cell with the largest internal fitness range and divide it via local clustering CVT. Otherwise, the other cells remain unchanged, avoiding partitioning on flat and saturated task landscapes.

Reversion and Memory. To save budget on valuable subdivisions, a sub-split is retained only if the child cells outperform the original parent elite. Otherwise, the subdivision will be reversed. Additionally, the parent cell logs a memory of its dynamic fitness distribution, avoiding redundant re-splitting until a novel candidate alters its fitness distribution. In this way, the search with adaptive resolution concentrates the evaluation budget strictly where finer resolution yields quality gains.

4 Experimental Setup

Baseline Methods. We evaluate Harness-MAP against representative baselines covering complementary paradigms of automated harness evolution. (i) ***General**-purpose optimization.* Meta-Harness (Lee et al., 2026) and GEPA with optimize anything (Agrawal et al., 2026b;a) test whether agentic code search and reflective optimization can improve a harness without an explicit quality-diversity objective. (ii) ***Diversity**-oriented search.* DGM (Zhang et al., 2026b) evaluates whether open ended evolution with an archive of alternative solutions discovers effective harnesses. (iii) ***Sample**-efficient search.* ShinkaEvolve (Lange et al., 2026) targets sample-efficient program evolution through exploration-aware parent selection and novelty-based rejection. (iv) *Fixed-resolution **QD** search.* OpenEvolve (Sharma, 2025) serves as the *MAP-Elites* baseline, providing a comparison with fixed-resolution QD search using different descriptors. (v) ***Direct inference***. We evaluate default, unevolved harnesses powered by Qwen3-VL-30B-A3B-Instruct (Bai et al., 2025) to isolate the gains by harness optimization from the capabilities of the underlying backbone model.

Benchmarks and Metrics. We evaluate our method across three interactive reasoning benchmarks: (i) SpatialWorld-3DGames (Gao et al., 2026) (110 instances across 5 games) tests spatial reasoning, evaluated by Task Success Rate (SR) and normalized scores; (ii) SimWorld-Robotics (Zhuang et al., 2026) (single-agent subset, $50/10$ search/val split) assesses embodied navigation and decision-making via SR, Subtask SR, Distance Progress, and safety metrics; (iii) BALROG-MiniHack (Paglieri et al., 2025) (40 episodes across 8 games) tests long-horizon grid-world planning using SR and normalized reward. Full statistics and metrics appear in § A.1.

Backbones and Hyperparameters. We use `Qwen3-VL-30B-A3B-Instruct` as the frozen backbone for the base agent, served locally via vLLM on $2\times$ NVIDIA H100 GPUs. For all main results, the search meta-agent driving harness evolution is `claude-opus-4-6`, invoked in headless mode via the Claude Code CLI. To verify backbone robustness, we perform ablation experiments using `gpt-5.6-sol` via the Codex CLI under high reasoning effort. Across all Harness-MAP configurations, we set the coverage-initialization ratio to 50%. The search budget is set to $N = 50$ candidate evaluations for SpatialWorld-3DGames and BALROG-MiniHack, and reduced to $N = 16$ for easy tasks and $N = 6$ for hard tasks of SimWorld-Robotics to accommodate its substantially

Table 1: Results across three interactive benchmarks under the budget-matched evaluation protocol.

Method	SpatialWorld (SR%)	SimWorld-MMNav (SubSR%)		BALROG (SR%)
	3DGames	easy	hard	MiniHack
Direct inference				
GPT-4o	-	33.1	34.4	10.0±4.7
GPT-5	36.4	-	-	20.3±7.3
Qwen2.5-VL-72B-Instruct	7.6	23.8	23.9	5.0±3.4
Qwen3-VL-30B-Instruct	10.7	32.0	29.0	10.0±4.7
Harness search at $N = 50, 16, 8, 50$				
Meta-Harness	68.6	39.0	36.0	75.0±6.8
GEPA	18.6	41.0	33.3	30.0±7.2
DGM	34.6	40.0	38.5	72.5±7.1
ShinkaEvolve	60.6	40.5	34.0	20.0±6.3
OpenEvolve	21.2	36.5	37.0	27.5±7.1
Ours				
Harness-MAP	**100.0**	**46.5**	**41.0**	**90.0±4.7**

slower rollouts. We sweep the initial resolution $k_0 \in \{2, 4, 8, 16, 32\}$ on SpatialWorld-3DGames and and MiniHack, and test a sparser range $k_0 \in \{2, 4, 8\}$ for SimWorld-Robotics (§ A.10).

Budget-Aware Harness Evolution Evaluation. *(i) Budget-matched evaluation protocol.* Following recent evaluation methods for agent harness evolution under unified budget (Wang et al., 2026), we benchmark search efficiency under tight compute constraints using two complementary metrics (§ A.5): *Candidate Evaluation Budget (N)* caps the total task-level executions in a search run. Given task rollouts form the primary computational bottleneck, fixing N ensures budget-matched fairness across both initialization and iterative search. *Sequential Steps (T)* counts the synchronous proposal batches generated from a shared state. Because the number of per-step parallel proposals varies across search loops, an equal budget N yields distinct step counts T, capturing trade-offs between parallel exploration breadth and sequential refinement depth. *(ii) Quality-diversity evaluation.* To evaluate the quality-diversity of the evolved harnesses across the behavior space, we report two standard QD metrics. *Coverage* indicates the fraction of occupied cells relative to total capacity, which measures the spatial exploration breadth. *QD-Score* is the cumulative fitness aggregated across all occupied elites, providing a holistic measure of joint population quality and diversity.

5 Quantitative results

Main Results. On three interactive spatial reasoning testbeds, Harness-MAP achieves superior performance over both direct inference baselines and frontier harness search algorithms under a strictly matched evaluation budget (Tab. 1). On SpatialWorld-3DGames, it achieves a perfect 100.0% task success rate, outperforming the top search baseline Meta-Harness (68.6%) and substantially surpassing direct inference with frontier models, including GPT-5 (36.4%). On BALROG-MiniHack, it hits a 90.0% success rate, exceeding strong LLM-guided search baselines such as Meta-Harness (75.0%) and DGM (72.5%) by a clear margin. On SimWorld-MMNav, it achieves top-tier performance across both easy (46.5% SubSR) and hard (41.0% SubSR) settings. These results highlight that our search approach unlocks hidden potential of smaller frozen models, effectively bridging the performance gap and outperforming unevolved frontier backbones.

Search Efficiency. *(1) Budget-wise efficiency.* Under budget-matched evaluation (Fig. 3, Tab. 7), Harness-MAP exhibits superior sample efficiency across all benchmarks, rapidly attaining top-tier performance as candidate budget N grows. On SpatialWorld-3DGames (Fig. 6, Tabs. 8, 9; §A.7), Harness-MAP ($k_0 = 4$) achieves 71.0% at $N = 10$, surpassing Meta-Harness's full-budget score (68.6% at $N = 50$), and reaches 100.0% by $N = 40$. On BALROG-MiniHack (Fig. 7, Tab. 10; §A.7), Harness-MAP ($k_0 = 2$) attains 75.0% at $N = 20$, matching Meta-Harness's full-budget performance in under half the evaluations, and scales to 90.0% at $N = 50$ ($k_0 = 16$). Unlike baselines that plateau early, Harness-MAP maintains steady gains across budget increments, proving that adaptive resolution efficiently converts tight compute into continuous performance growth.

(2) Sequential Step-wise Efficiency. Beyond total budgets N, we evaluate search dynamics over sequential steps T, which reflect optimization depth (Figs. 3, 6, Tab. 8). These step-wise results reveal three key insights. First, Harness-MAP achieves peak task performance within fewer search steps.

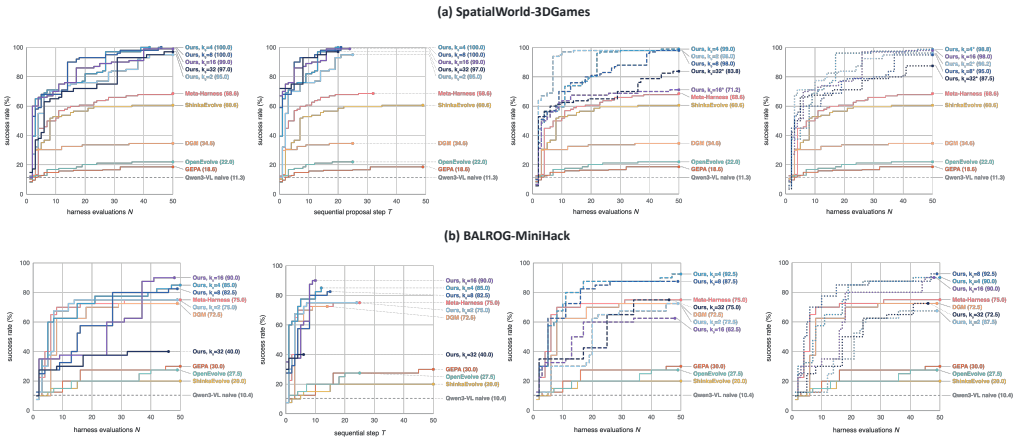

Figure 3: **Search performance and resolution comparison on (a) SpatialWorld-3DGames and (b) BALROG-MiniHack.** From left to right in each panel: SR versus evaluation budget N under adaptive resolution, sequential steps T, and N for fixed and hierarchical resolution variants.

Second, coverage-gated initialization exhibits high diversity-driven efficiency, frequently discovering optimal harnesses prior to the main search loop (e.g., Snake, Maze). Third, adaptive resolution incrementally refines the map with incrementally finer resolution rather than exponential explosion (§A.3), sustaining overall growth while preventing over-partitioning and wasted rollouts.

Quality-Diversity Performance. To fairly evaluate baselines under QD metrics, taking SpatialWorld-3dgames as an example, all candidate harnesses generated during their search runs are offline embedded into our structure-token descriptor space and mapped onto the k_0 Voronoi cells. As shown in Tab. 2, while baselines achieve high coverage at coarse grids, their local edits suffer severe coverage collapse as resolution increases. In contrast, Harness-MAP maintains robust, stable coverage (over 50%) across all resolutions, enabling its QD-scores to scale incrementally. Furthermore, Ours with adaptive resolution achieves superior QD-scores across intermediate resolutions ($k_0 \in \{4, 8, 16\}$), confirming that dynamically adjusting map granularity effectively concentrates budget where deeper behavioral exploration is effective.

Table 2: QD-score (coverage%) across k_0.

Method	Initial Map Resolution (k_0)				
	$k_0 = 2$	$k_0 = 4$	$k_0 = 8$	$k_0 = 16$	$k_0 = 32$
Baselines					
Meta-Harness	0.93 (100%)	1.30 (70%)	1.46 (45%)	2.27 (32%)	2.20 (16%)
DGM	0.96 (100%)	1.48 (70%)	1.62 (45%)	2.14 (29%)	1.97 (12%)
GEPA	0.43 (100%)	0.49 (70%)	0.70 (57%)	0.83 (32%)	1.15 (21%)
ShinkaEvolve	0.36 (90%)	0.36 (50%)	0.46 (30%)	0.46 (20%)	0.46 (8%)
OpenEvolve	0.27 (90%)	0.29 (55%)	0.29 (25%)	0.29 (15%)	0.29 (9%)
Ours (Harness-MAP)					
Ours (adaptive)	1.16 (100%)	**1.48 (70%)**	**2.43 (64%)**	**3.51 (58%)**	7.25 (**57%**)
Ours (fixed)	1.15 (100%)	1.40 (65%)	2.38 (55%)	3.23 (52%)	**7.89** (52%)
Ours (hier)	**1.25** (100%)	1.22 (65%)	2.30 (57%)	2.71 (42%)	6.68 (49%)

6 Qualitative Results

We analyze the discovered harnesses by our approach on BALROG-MiniHack (see cases in § A.9):

(1) *Performance improves consistently via modular edits.* Success rates rise from 0.275 to 0.850 through incremental additions of deterministic code owner modules (e.g., spatial pathfinding fixes, Sokoban BFS solvers, and interaction protocols). Performance growth is primarily driven by MAP archive recombination, combining dispatch skeletons with byte-identical modules from disparate cell elites rather than direct linear mutations.
(2) *Optimal harnesses exhibit high structural and algorithmic diversity.* They show high design diversity (textual similarity < 0.093) and complementary task coverage, achieving 100% success via an oracle selector. Perception converges on shared parsers, but control paradigms diverge. Performance correlates inversely with token cost. The top-performing harness ($k_0 = 16$, 0.900 score) even uses traditional algorithm without any LLM calls during evaluation.
(3) *Search failures stem from misallocation and flawed rules.* Uniform selection repeatedly samples low-scoring cells, wasting compute and capping $k_0 = 32$ performance at 0.400.

(4) *Dynamic map partitioning focuses search on high-potential regions.* Breakthroughs originate from newly subdivided cells (e.g., Cell 23 yielding the 0.900 champion). The system self-corrects by merging uninformative sub-trees while expanding resolution in high-variance regions.

7 Ablation Study

Resolution Control. Cell count k decides budget allocation: under-partitioning (k too low) reduces archive diversity, whereas over-partitioning (k too high) slows cell rotation and dilutes per-cell evaluation with fewer mutation chances. Rather than fixing k, Harness-MAP dynamically adjusts resolution during search. We evaluate this design by sweeping initial resolution k_0 and benchmarking our adaptive mechanism against fixed and hierarchical baselines.

(i) Initial resolution k_0. We sweep $k_0 \in \{2, 4, 8, 16, 32\}$ from coarse to fine initialization under budget $N = 50$ (cost details in Tabs. 8, 10; §A.7). On SpatialWorld-3DGames, moderate initial resolutions ($k_0 \in \{4, 8\}$) achieve a perfect 100.0% success rate, maintaining strong performance (95.0% to 99.0%) across all k_0. On BALROG-MiniHack, performance peaks at $k_0 = 16$ (90.0%) and $k_0 = 4$ (85.0%), but sharply degrades to 40.0% at $k_0 = 32$, confirming that excessive partitioning severely dilutes search budget under tight constraints. *(ii) Adaptive resolution.* We compare our dynamic mechanism (k_{adaptive}) against static (k_{fixed}) and scheduled (k_{hier}) resolution baselines (§A.4). Across all k_0 configurations, k_{adaptive} consistently matches or outperforms both baselines on SpatialWorld-3DGames (Fig. 3, Tabs. 8, 10), reaching a Pareto frontier that prevents over-partitioning while sustaining active exploration.

Structure-Token Descriptor. To verify the contributions of our designed descriptor disentangled from our search loop, we perform a descriptor-wise ablation on SpatialWorld-3DGames ($k_0 = 8$) crossing the search loop with different descriptors. We compare two hybrid frameworks (§A.4): OpenEvolve with our CVT descriptor (OpenEvolve$_{\text{st-cvt}}$) and our loop with OpenEvolve's 2D descriptor (Harness-MAP$_{\text{cd}}$). The two descriptors are complementary for representations and search pipeline, where our descriptor builds a structure-token representation space and partitions this space via CVT (§3.2), whereas OpenEvolve maps program texts to a 2D space of complexity and diversity. Results in Tab. 3 demonstrate that our structure-token CVT space is an inherently effective descriptor design for guiding harness search. On the one hand, Harness-MAP$_{\text{cd}}$ degrades overall success ($100.0\% \rightarrow 96.2\%$) and search efficiency N (e.g., requiring 37 vs. 16 evals on Rubik, alongside coverage failure on Maze3D). On the other hand, OpenEvolve$_{\text{st-cvt}}$ yields a massive performance jump ($21.9\% \rightarrow 71.4\%$) while reducing required budget (e.g., hitting 100% on Maze and Block3D in 6 evals).

Table 3: Descriptor ablation results with SR first reaching at N/T.

Method	Maze	Maze3D	Block3D	Rubik	Snake	Wtd. Avg.
Harness-MAP Loop						
Harness-MAP (Full)	100.0/10/1	100.0 (46/7)	100.0 (10/1)	100.0/16/2	100.0/10/1	**100.0**
Harness-MAP$_{\text{cd}}$	100.0/14/4	84.0/23/4†	100.0/18/3	100.0/37/7	100.0/8/1	96.2
OpenEvolve Loop						
OpenEvolve	60.0/37/36	20.0/50/49	5.0/50/49	15.0/50/49	10.0/50/49	21.9
OpenEvolve$_{\text{st-cvt}}$	100.0/6/5	48.0/50/49	100.0/6/5	15.0/50/49	100.0/26/25	**71.4**

†CovErr: OpenEvolve's descriptor could not fill 50% of K_0 on Maze3D, so this cell ran at a 25% target.

8 Conclusion and Limitations

Conclusion. We present Harness-MAP, a sample-efficient quality-diversity framework for agent harness search under tight evaluation budgets. By mapping candidates onto a structure-token space, Harness-MAP renders evolutionary step sizes and archive coverage explicitly measurable. It dynamically adjusts archive resolution during search, sub-splitting high-variance cells at zero evaluation cost via cached descriptors while reverting uneffective subdivisions. Extensive evaluations demonstrate that Harness-MAP advances the Pareto frontier of success rate and efficiency, which provides a new Quality-Diversity paradigm for budget-aware agent harness search.

Limitations. While Harness-MAP advances budget-efficient harness evolution, several limitations remain in the air. First, setting a large initial resolution (k_0) under tight evaluation budgets consumes substantial compute during initialization, leaving fewer sequential steps for iterative refinement than fast greedy methods. Second, required evaluation budget and convergence rate remain unpredictable on unsaturated tasks. Therefore, Harness-MAP is best suited for complex tasks requiring high structure-token diversity rather than simple problems solvable by shallow greedy search.

AI USE STATEMENT

We integrated generative AI into our *AutoResearch* framework to drive the meta-agent for recursive self-improvement. Beyond this core method, we used AI to polish the writing, generate figures, and refine technical details of our theoretical models. All hypotheses, experimental designs, data, and analyses were completed independently. We led every technical discussion, carefully audited all theoretical results across multiple manual reviews, and take full responsibility for the final manuscript.

REFERENCES

Lakshya A Agrawal, Donghyun Lee, Shangyin Tan, Wenjie Ma, Karim Elmaaroufi, Sanjit A Seshia, Koushik Sen, Dan Klein, Ion Stoica, Joseph E Gonzalez, et al. optimize_anything: A universal api for optimizing any text parameter. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 1300–1304, 2026a.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026b.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Herbie Bradley, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Grégory Schott, and Joel Lehman. Quality-diversity through ai feedback. In *International Conference on Learning Representations*, volume 2024, pp. 21036–21147, 2024.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026a.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026b.

Antoine Cully. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 81–89, 2019.

Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2017.

Hongcheng Gao, Hailong Qu, Jingyi Tang, Jiahao Wang, Zihao Huang, Hengkang Qiao, Shihong Huang, Junming Yang, Yi Li, Hongyixuan Yuan, et al. Spatialworld: Benchmarking interactive spatial reasoning of multimodal agents in real-world tasks. *arXiv preprint arXiv:2606.09669*, 2026.

Dongxin Guo, Jikun Wu, and Siu Ming Yiu. Parameter-efficient neuroevolution for diverse llm generation: Quality-diversity optimization via prompt embedding evolution. *arXiv preprint arXiv:2605.09781*, 2026.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, volume 2026, pp. 74026–74078, 2026.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of evolutionary machine learning*, pp. 331–366. Springer, 2023.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Xiaotian Luo, Fengxingyu Wang, Chuanrui Hu, Dizhan Xue, and Yafeng Deng. Self-evolving agent harnesses via gated semantic quality-diversity. *arXiv preprint arXiv:2607.13683*, 2026.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wołczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. In *International Conference on Learning Representations*, volume 2025, pp. 96666–96702, 2025.

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

Gabriel Machado Santos, Rita Maria Da Silva Julia, and Marcelo Zanchetta Do Nascimento. Diverse prompts: Illuminating the prompt space of large language models with map-elites. In *2025 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2025.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. Vero: A harness for agents to optimize agents. *arXiv preprint arXiv:2602.22480*, 2026.

Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630, 2017.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026.

Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing, 2026. *URL https://arxiv. org/abs/2602.04837*.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026b.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026c.

Yan Zhuang, Jiawei Ren, Xiaokang Ye, Jianzhi Shen, Tianai Yue, Muhammad Faayez, Xuhong He, Xiyan Zhang, Ziqiao Ma, Lianhui Qin, et al. Simworld-robotics: Synthesizing photorealistic and dynamic urban environments for multimodal robot navigation and collaboration. *Advances in Neural Information Processing Systems*, 38:51854–51895, 2026.

A Appendix

A.1 More experiment details.

Details of Benchmark Subsets and Metrics.

We evaluate on three spatial reasoning benchmarks, and . Tab. 4 summarizes the task specifications, dataset search/validation splits, and primary evaluation metrics across all environments.

(1) The SpatialWorld-3DGames (Gao et al., 2026) subset comprises 110 total instances across five abstract spatial environments designed to test 3D topological reasoning, geometry manipulation, and path planning. Only *Snake* adopts a search/validation holdout split, while all other environments are evaluated on full fixed level sets.

(2) The SimWorld-Robotics (Single-Agent Simworld-MMNAV) (Zhuang et al., 2026) subset evaluates multimodal visual navigation in complex synthetic urban environments. The benchmark is structured into two dynamic modes with an identical search/validation task split.

(3) BALROG-MiniHack (Paglieri et al., 2025) evaluates long-horizon dynamic planning and spatial reasoning within procedural grid-world environments. We search candidates on the official `minihack_tasks` split (40 episodes across 8 tasks, 5 episodes each) with fixed deterministic seeds (b=0). To assess generalization without search leak, performance is evaluated on two disjoint holdouts: *Holdout A (unseen levels)* re-runs the 8 search tasks with fresh level seeds (b=1), while *Holdout B (unseen tasks)* evaluates 8 tasks outside the search set (comprising 4 unseen task variants and 4 new task families). Performance is reported using Task Success Rate (SR) based on the environment's terminal success status with standard error $\sqrt{p(1-p)/N}$.

A.2 Generalization to held-out tasks.

To verify the generalization capabilities of the best evolved harnesses by each search method, Tab. 5 compares performance between search splits and validation tasks. While evolutionary search baselines like GEPA, ShinkaEvolve, and OpenEvolve exhibit noticeable generalization gaps, Harness-MAP, Meta-Harness, and DGM consistently retain 100.0% success rates across both search and validation tasks. This proves that descriptor-space diversity prevents search overfitting, producing robust harnesses that transfer effectively to unseen tasks.

A.3 Resolution Dynamics.

Fig. 4 and 8 tracks archive resolution trajectories of SpatialWorld-3dgames and MiniHack across search steps T. While fixed resolution remain static, hierarchical expansion leads to excessive resolution growth in complex environments (e.g. Maze3D, Rubik). In contrast, the adaptive resolution exhibits responsive dynamics, where it rapidly suppresses unhelpful sub-splits on simpler landscapes (e.g. Blockz3D), and selectively deepens granularity in complex environments where quality gains are verified. Crucially, adaptive resolution caps overall growth rather than expotential explosion, which proves that our adaptive mechanism effectively prevents over partitioning.

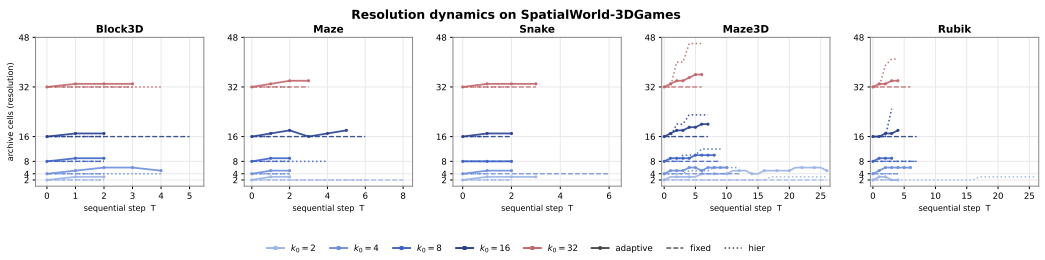

Figure 4: Resolution dynamics on SpatialWorld-3DGames.

Table 4: Overview of benchmark specifications, dataset splits, and evaluation metrics across SpatialWorld-3DGames, SimWorld-Robotics, and BALROG-MiniHack.

Env.	Task Goal	Search Split	Val Split	Evaluation Metrics
SpatialWorld (3D Games)				
Maze	Single-floor 3D grid exit-finding	Lvls 1–20	None	Binary Success ($\{0, 1\}$)
Snake	$4 \times 4 \times 4$ 3D grid food collection	Seeds 1–20	Seeds 21–25	$min(score/L, 1.0)$, $L = 4$
Block3D	$6 \times 6 \times 6$ structure reconstruction	Lvls 1–20	None	Binary Success ($\{0, 1\}$)
Maze3D-Pro	Multi-floor 3D maze (horizon $\leq$ 2000)	Lvls 1–25	None	Binary Success ($\{0, 1\}$)
Rubik	3D Rubik's cube state alignment	Lvls 1–20	None	Binary Success ($\{0, 1\}$)
SimWorld-Robotics (Single-Agent MMNAV)				
Easy	Clean environment stripped of static obstacles, pedestrians, and background vehicles.	Maps 20_0–20_24	Maps 20_25–20_29	SR, Subtask SR, Distance Progress (%)
Hard	Includes static obstacles (barriers, trees) and dynamic agents (pedestrians, background traffic).	Maps 20_0–20_24	Maps 20_25–20_29	SR, Subtask SR, Distance Progress (%), Stat/Dyn Collisions, Red Light Viol.
BALROG-MiniHack — official `minihack_tasks`; every level fixed by a common random number seed $\mathrm{sha256}(b\|\tau\|i)[:8]$				
Search set (b=0; $8 \times 5 = 40$ episodes; the split the search optimizes)				
Boxoban-Medium	Sokoban-style boulder pushing onto goal squares	5 eps	–	Task SR, ±SE
Boxoban-Hard	As above, harder boulder/goal configurations	5 eps	–	Task SR, ±SE
MazeWalk-9x9	Reach the exit of a procedural 9×9 maze	5 eps	–	Task SR, ±SE
MazeWalk-15x15	Reach the exit of a procedural 15×15 maze	5 eps	–	Task SR, ±SE
Corridor-R3	Three rooms joined by corridors; hidden passages must be searched for	5 eps	–	Task SR, ±SE
CorridorBattle-Dark	Survive and clear a pack of hostiles in an unlit corridor	5 eps	–	Task SR, ±SE
Quest-Easy	Cross a lava barrier to the staircase; requires acquiring and applying a level item	5 eps	–	Task SR, ±SE
Quest-Medium	As above with hostiles and a larger liquid barrier	5 eps	–	Task SR, ±SE
Holdout A — unseen levels (b=1; the same eight tasks, zero seed overlap with the search set)				
Boxoban-Medium	Same goal, new level instances	–	5 eps	Task SR, ±SE
Boxoban-Hard	Same goal, new level instances	–	5 eps	Task SR, ±SE
MazeWalk-9x9	Same goal, new level instances	–	5 eps	Task SR, ±SE
MazeWalk-15x15	Same goal, new level instances	–	5 eps	Task SR, ±SE
Corridor-R3	Same goal, new level instances	–	5 eps	Task SR, ±SE
CorridorBattle-Dark	Same goal, new level instances	–	5 eps	Task SR, ±SE
Quest-Easy	Same goal, new level instances	–	5 eps	Task SR, ±SE
Quest-Medium	Same goal, new level instances	–	5 eps	Task SR, ±SE
Holdout B — unseen tasks (b=0; eight tasks *outside* the official eight, disjoint since τ enters the hash)				
Boxoban-Unfiltered	Sokoban on the unfiltered level pool	–	5 eps	Task SR, ±SE
Corridor-R5	Five rooms joined by corridors (unseen variant of Corridor-R3)	–	5 eps	Task SR, ±SE
CorridorBattle	Corridor combat with the corridor *lit* (unseen variant of CB-Dark)	–	5 eps	Task SR, ±SE
Quest-Hard	Longer multi-stage quest (unseen variant of Quest-Easy/Medium)	–	5 eps	Task SR, ±SE
Labyrinth-Small	Navigate a small labyrinth to the staircase (new family)	–	5 eps	Task SR, ±SE
MultiRoom-N4	Traverse four connected rooms through doors (new family)	–	5 eps	Task SR, ±SE
River-Narrow	Cross a river at a narrow crossing (new family)	–	5 eps	Task SR, ±SE
LavaCrossingS9N1	Cross a lava field on a 9×9 grid (new family)	–	5 eps	Task SR, ±SE

Table 5: Generalization performance across search and held-out validation tasks.

Method	SpatialWorld: Snake (SR%)		SimWorld: Easy (SubSR%)		SimWorld: Hard (SubSR%)		BALROG-MiniHack (SR%)		
	Search	Val	Search	Val	Search	Val	Search	Holdout A	Holdout B
Direct inference									
Qwen3-VL-30B	10.7	0.0	32.0	32.5	29.0	27.5	10.0 ± 4.7	10.0 ± 4.7	0.0 ± 0.0
Harness search									
Meta-Harness	100.0	**100.0**	39.0	40.0	36.0	37.5	75.0 ± 6.8	67.5 ± 7.4	$\mathbf{72.5 \pm 7.1}$
GEPA	10.0	5.0	41.0	**52.5**	33.3	34.0	30.0 ± 7.2	20.0 ± 6.3	30.0 ± 7.2
DGM	100.0	**100.0**	40.0	40.0	38.5	45.0	72.5 ± 7.1	55.0 ± 7.9	65.0 ± 7.5
ShinkaEvolve	8.0	5.0	40.5	40.0	34.0	42.5	20.0 ± 6.3	10.0 ± 4.7	27.5 ± 7.1
OpenEvolve	10.0	5.0	36.5	30.0	37.0	35.0	27.5 ± 7.1	17.5 ± 6.0	15.0 ± 5.6
Ours									
Harness-MAP$_{k_0=2}$	100.0	**100.0**	**46.5**	35.0	37.0	**45.00**	75.0 ± 6.8	57.5 ± 7.8	62.5 ± 7.7
Harness-MAP$_{k_0=4}$	100.0	**100.0**	44.0	50.0	37.0	30.0	75.0 ± 6.8	55.0 ± 7.9	62.5 ± 7.7
Harness-MAP$_{k_0=8)}$	100.0	**100.0**	45.5	45.0	**41.0**	25.0	$\mathbf{87.5 \pm 5.2}$	$\mathbf{75.0 \pm 6.8}$	57.5 ± 7.8
Harness-MAP$_{k_0=16}$	100.0	**100.0**	-	-	-	-	62.5 ± 7.7	45.0 ± 7.9	65.0 ± 7.5
Harness-MAP$_{k_0=32}$	100.0	**100.0**	-	-	-	-	62.5 ± 7.7	50.0 ± 7.9	55.0 ± 7.9

A.4 More details of ablation study

Fixed and hierarchical resolution.

K_{fixed} is predefined and stays unchanged during search, which is the classic MAP-Elites on a CVT grid as ordinarily applied (Mouret & Clune, 2015; Vassiliades et al., 2017). K_{hier} climbs an increasing ladder $k_0 \to 2k_0 \to 4k_0$ on the coarse-to-fine schedule of Mouret & Clune (2015, §9.2),

with rung ℓ of L going live at step

$$t_\ell \;=\; 1 + \big\lfloor \ell T / L \big\rfloor, \qquad \ell = 0, \ldots, L-1. \tag{8}$$

For instance, $L = 3$ rungs over $T = 20$ steps activate at steps 1, 7, and 14, an even split of the budget. An advance fires at the top of a step whenever the program crosses a rung boundary, moving every stored harness onto the finer rung from its cached descriptor at no evaluation cost. It fires whether or not the archive warrants a finer view, and never reverses. $K_{adaptive}$ is our design stated in §3.4. It divides one cell with biggest fitness range at a time, and merges the division back when it fails to produce anything better than the undivided cell already held.

Descriptors of OpenEvolve (Sharma, 2025).

OpenEvolve maps program text to two scalar values, measuring character count and mean surface distance to a live reference set through differences in length, line count, and character sets. Discretizing this space into axis aligned bins introduces two structural instabilities during search. First, because the reference set is resampled from the live population, candidate features depend on population state rather than the code alone. Second, min max normalization against running statistics shifts bin boundaries whenever new extremes appear, causing candidates to change cells without modifying their source code.

For *OpenEvolve*$_{st-CVT}$, we eliminate both instabilities while preserving OpenEvolve feature content. We freeze the reference set to the bootstrap corpus to make feature computation fully deterministic, and replace running normalization with z-score standardization using corpus statistics. Additionally, we replace the axis aligned grid with a Voronoi partition fitted via k-means. This Voronoi structure is necessary for our adaptive mode to refine individual cells through splitting, which a fixed grid cannot support. Consequently, *open_harness* transplants OpenEvolve feature content into our partition geometry rather than importing its descriptor framework entire.

A.5 Budget-matched evaluations

Budget Counting.

By our Harness-MAP, the budget of N could be formulated as:

$$N \;=\; \underbrace{\lceil m\% \, k_0 \rceil}_{\text{initialization}} \;+\; \underbrace{G \cdot P \cdot C}_{\text{search}}, \tag{9}$$

in which k_0 is the initial cell count (resolution), $m\%$ the fraction of cells the founders must occupy before evolution, G the number of generations, P the parents selected per generation (batch size) and C the children bred by per parent. The budget N, spent on evaluations of initialization and search, is therefore fixed for fair comparison and scaling analysis. The coverage-gated initialization evaluates one harness per newly covered cell until it reaches $\lceil m\% k_0 \rceil$ occupied cells. The search proper spends PC evaluations in each of G generations.

Sequential steps.

Tab. 6 gives what one step costs in each loop. Every method is initialized from the same reference harness H_0, and that initialization is step $T = 0$ throughout. The initialization phase of our method is coverage-gated, evaluating $\lceil mk_0 \rceil$ roots rather than one starter to increase the width.

The self-improvement loops differ in where the width of a step comes from. Meta-Harness Lee et al. (2026) appends to a single log and proposes $k = 2$ edits from its latest entry, so both candidates share one parent. DGM Zhang et al. (2026b) draws one parent from the archive with replacement and self-modifies each twice, so its two candidates are usually distinct lineages but may collapse onto one node when the same parent is drawn twice. ShinkaEvolve (Lange et al., 2026) and OpenEvolve (Sharma, 2025) both advance one candidate at a time, the former from a parent sampled within an island and the latter from an occupied MAP-Elites bin. GEPA (Agrawal et al., 2026b) is the one loop that N does not determine, since it spends its budget on minibatch rollouts rather than whole-harness evaluations and its Pareto front advances per instance. We therefore report it against N and T in Tab. 1 and Tab. 8. Ours breeds C children from each of $P = \log_2 k_0$ cell elites in rotation, so a step spans $P \times C$ times evaluations.

We run every baseline serially, one proposal per step, as its published default specifies. ShinkaEvolve and OpenEvolve both ship parallel controllers that would let several candidates be proposed from

Table 6: How a budget of N harness evaluations converts into *sequential steps*. Each baseline executes its published search loop at its default setting. Initialization is step 0 for every method where baselines evaluate H_0 alone, ours evaluates $\lceil mk_0 \rceil$ roots, which is costlier in N but equally shallow in T.

Method	Selection unit for each step	One step evaluates	T at budget N
Meta-Harness (Lee et al., 2026)	latest log entry	1 parent $\times$ 2 edits $= 2$	$(N-1)/2$
GEPA (Agrawal et al., 2026a)	Pareto front member	1 parent $\times$ 1 mutation $= 1^*$	$*$
DGM (Zhang et al., 2026b)	any archive node	1 parent $\times$ 2 self-improve attempts	$(N-1)/2$
ShinkaEvolve (Lange et al., 2026)	sampled island parent	1 parent $\times$ 1 offspring $= 1^\ddagger$	$N-1$
OpenEvolve (Sharma, 2025)	occupied MAP-Elites bin	1 parent $\times$ 1 candidate $= 1$	$N-1$
Ours	P cells in rotation	P elite parents $\times$ C children $= PC$	$(N - \lceil mk_0 \rceil)/(PC)$

* GEPA evaluates one candidate per step but spends its budget on minibatch rollouts rather than whole-harness evaluations, so T is not a fixed function of N.
† Parents are drawn with replacement, so a step may branch twice from the same node; $k = 2$ is the SWE-bench default ($k = 4$ for Polyglot).
‡ One primary parent under diff and rewrite operators, two under crossover. ShinkaEvolve and OpenEvolve additionally condition each proposal on inspiration programs drawn from the archive, which cost no evaluations and are not counted here.

one snapshot of the archive. Therefore, enabling them would widen a step and shorten T at the same budget N.

A.6 More details of our method.

Conceptual comparison of QD archive partitioning. See the conceptual comparison in Fig. 5 and § 2.

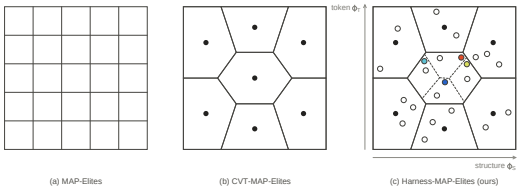

Figure 5: Conceptual of quality-diversity archive partitioning. Unlike (a) standard MAP-Elites with fixed grid and (b) CVT-MAP-Elites with fixed Voronoi regions, **(c) Harness-MAP (ours)** operates over a structure-token space with adaptive resolution.

Construction of Role Labels. Before forming the subsequent Weisfeiler–Lehman (WL) structural descriptor $\varphi_S(s)$, we have to construct the initial role labels $\ell_0(v)$ first. The component graph $G(s) = (V, E)$ contains two types of vertices: top-level component definitions (functions and classes) and module files. Initial labels $\ell_0(v)$ are assigned deterministically without learned embeddings using an ordered vocabulary of 11 functional agent roles:

$$\mathcal{R} = \{\text{controller}, \text{memory}, \text{prompt}, \text{perception}, \text{tool}, \text{policy}, \text{validity}, \text{recovery}, \text{parse}, \text{spatial}, \text{data}\}. \tag{10}$$

For each top-level definition, its lowercased identifier name is matched against pre-defined keyword lists associated with each role in $\mathcal{R}$. The assignment follows a strict first-match-wins rule based on the order of $\mathcal{R}$. Any component that fails to match any role keyword is assigned the generic fallback label `logic`. For module nodes, the module containing the largest component count is treated as the primary file, while all remaining secondary modules are assigned the label `tool`.

L-round Weisfeiler–Lehman (WL) Refinement. We adopt WL (Shervashidze et al., 2011) refinement mechanism for capturing harness structure, specifically for three reasons: (i) *Role-aware topology.* Unlike naive metrics like node/edge counts, WL distinguishes different agent workflows (e.g., a harness where *Perception* directly calls *Controller* from one where it first routes through a *Planner*); (ii) *Robustness to refactoring*: Harness coordinates remain invariant to refactoring choices that do not alter architecture, such as renaming functions, reordering code blocks, or splitting modules into separate files; (iii) *Deterministic and zero-cost Evaluation*. Because WL relies purely

on CPU string hashing, it introduces zero model inference overhead and enables deterministic cell placement across thousands of evolutionary search steps.

Edges E in $G(s)$ comprise directed call dependencies and name references extracted from the source AST between top-level components, as well as import relationships between module nodes. Using $\ell_0(v)$ as the initial seeds, we execute $L = 2$ rounds of Weisfeiler–Lehman refinement over undirected edges. In each round t, the updated label for node v is computed via a canonical hash:

$$\ell_{t+1}(v) = \mathrm{MD5}\Big(\ell_t(v) \;\|\; \mathrm{sort}\big(\{\{\ell_t(u) : u \in \mathcal{N}(v)\}\}\big)\Big)[:10], \tag{11}$$

where $\|$ denotes string concatenation, and the MD5 hash is truncated to the first 10 characters. After $L = 2$ rounds of hashing, the structural descriptor $\varphi_S(s) \in \mathbb{R}^{d_S}$ is compiled by computing a histogram of label frequencies across all rounds $t \in \{0, 1, 2\}$, normalized by the total node count $|V|$.

Algorithm of Adaptive-Resolution Search. See Alg 2.

Algorithm 2 Adaptive Map Resolution Procedure

procedure ADAPT_RESOLUTION($\mathcal{H}, \mathcal{P}, C$)
  $c^* \longleftarrow \arg\max_{c \in C} \mathrm{range}(c, \mathcal{H})$ ▷ *Cell with largest range*
  if can_split(c^*, `memory`) **then**
    $C \longleftarrow$ sub_split_cvt(c^*, ϕ) ▷ *Sub-split via local CVT*
  end if
  for each sub-split cell c **do**
    if $\max_{c' \in \mathrm{children}(c)} \mathcal{P}(c') \le \mathcal{P}(c)$ **then** ▷ *No improvement*
      $C \longleftarrow$ revert_split(c) ▷ *Revert and restore members*
      `memory`(c) $\longleftarrow$ search_history(c) ▷ *Log distribution memory*
    end if
  end for
end procedure

A.7 Detailed per-benchmark results

Budget scaling results. See Tab. 7.

Table 7: Evaluation budget scaling results. Harness search across three benchmarks under an increasing outer-loop evaluation budget N.

Method	SpatialWorld-3DGames					SimWorld-Robotics								BALROG-MiniHack				
						easy				hard								
Direct inference																		
Qwen3-VL-30B-Instruct	10.7					32.5				27.5				10.0 ± 4.7				
Harness search at $N = 50, 16, 8, 50$																		
	10	20	30	40	50	4	8	12	16	2	4	6	8	10	20	30	40	50
Meta-Harness	51.9	60.7	66.0	67.8	68.6	39.0	39.0	39.0	39.0	35.0	35.0	36.0	36.0	70.0 ± 7.2	72.5 ± 7.1	72.5 ± 7.1	$\mathbf{75.0 \pm 6.8}$	$\mathbf{75.0 \pm 6.8}$
GEPA	14.9	16.3	16.6	18.6	18.6	41.0	41.0	41.0	41.0	34.5	34.5	34.5	34.5	20.0 ± 6.3	20.0 ± 6.3	20.0 ± 6.3	20.0 ± 6.3	30.0 ± 7.2
DGM	30.3	33.6	33.6	34.6	34.6	40.0	40.0	40.0	40.0	36.0	37.0	38.5	38.5	62.5 ± 7.7	70.0 ± 7.2	72.5 ± 7.1	72.5 ± 7.1	72.5 ± 7.1
ShinkaEvolve	52.8	56.6	59.7	60.6	60.6	37.5	40.5	40.5	40.5	34.0	34.0	34.0	34.0	15.0 ± 5.6	20.0 ± 6.3	20.0 ± 6.3	20.0 ± 6.3	20.0 ± 6.3
OpenEvolve	16.8	20.2	21.2	22.0	22.0	36.5	36.5	36.5	36.5	35.5	35.5	35.5	37.0	15.0 ± 5.6	20.0 ± 6.3	20.0 ± 6.3	27.5 ± 7.1	27.5 ± 7.1
Ours																		
Harness-MAP$_{k_0=2}$	70.0	76.0	84.0	93.0	95.0	41.5	**44.0**	**44.0**	**46.5**	36.0	36.0	37.0	37.0	67.5 ± 7.4	$\mathbf{75.0 \pm 6.8}$	$\mathbf{75.0 \pm 6.8}$	$\mathbf{75.0 \pm 6.8}$	$\mathbf{75.0 \pm 6.8}$
Harness-MAP$_{k_0=4}$	71.0	82.0	97.0	**100.0**	**100.0**	41.0	41.0	41.5	**44.0**	36.0	36.0	36.0	37.0	62.5 ± 7.7	72.5 ± 7.1	$\mathbf{77.5 \pm 6.6}$	$\mathbf{80.0 \pm 6.3}$	$\mathbf{85.0 \pm 5.6}$
Harness-MAP$_{k_0=8}$	68.0	93.0	95.0	98.0	**100.0**	39.0	**44.5**	**44.5**	**45.5**	33.5	35.5	**40.0**	**40.0**	32.5 ± 7.4	57.5 ± 7.8	$\mathbf{80.0 \pm 6.3}$	$\mathbf{80.0 \pm 6.3}$	$\mathbf{82.5 \pm 6.0}$
Harness-MAP$_{k_0=16}$	70.0	86.0	90.0	99.0	99.0	-	-	-	-	-	-	-	-	37.5 ± 7.7	37.5 ± 7.7	60.0 ± 7.7	$\mathbf{87.5 \pm 5.2}$	$\mathbf{90.0 \pm 4.7}$
Harness-MAP$_{k_0=32}$	65.0	72.0	75.0	93.0	97.0	-	-	-	-	-	-	-	-	30.0 ± 7.2	37.5 ± 7.7	37.5 ± 7.7	40.0 ± 7.7	40.0 ± 7.7

A.8 Empirical analysis of Structure-Token representation

Concrete Instantiation on BALROG. On the BALROG benchmark, harness candidates are mapped into an 802-dimensional joint representation space $\varphi(H) \in \mathbb{R}^{802}$ composed of four distinct feature blocks. The structural descriptor $\varphi_S(s) \in \mathbb{R}^{16}$ forms the first 16 dimensions, comprising 4 base role proportions at $h = 0$ (`controller`, `logic`, `perception`, `tool`) and 12 multi-hop neighborhood hash signatures (6 for $h = 1$ and 6 for $h = 2$). The remaining 786 dimensions constitute the token descriptor $\varphi_T(\theta) \in \mathbb{R}^{786}$, which concatenates 15 AST control-flow metrics (e.g., maximum control-flow count and branching ratios), 3 numeric scale statistics, and 768-dimensional natural language embeddings using`nomic-ai/CodeRankEmbed` extracted from documentation,

Table 8: Per-environment results on SpatialWorld-3DGames with Codex CLI as the meta-agent. For each environment we report the task success rate in percent, $N^\star$, the number of evaluations at which that score was first reached, and $T^\star$, the sequential steps those evaluations bought.

Method	Maze			Maze3D			Block3D			Rubik			Snake			Wtd. Avg.
	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	
Direct inference																
Qwen3-VL-30B-Instruct	45.0	–	–	0.0	–	–	5.0	–	–	0.0	–	–	6.3	–	–	10.7
Harness search																
Meta-Harness	80.0	34	17	48.0	39	26	100.0	13	6	15.0	24	12	100.0	27	13	68.6
GEPA	45.0	11	10	8.0	4	3	10.0	2	1	20.0	32	31	10.0	26	25	18.6
DGM	55.0	35	17	8.0	1	0	5.0	1	0	5.0	1	0	100.0	4	2	34.6
ShinkaEvolve	85.0	3	2	12.0	15	14	95.0	36	35	35.0	21	20	76.0	6	5	60.6
OpenEvolve	60.0	24	12	20.0	36	18	5.0	1	0	15.0	2	1	10.0	3	1	22.2
Ours$_{(K_{adaptive})}$																
Harness-MAP$_{k_0=2}$	100.0	17	8	90.0	34	17	100.0	4	2	85.0	42	21	100.0	4	2	95.0
Harness-MAP$_{k_0=4}$	**100.0**	**3**	**1**	**100.0**	**39**	**19**	100.0	13	6	100.0	27	14	100.0	3	1	**100.0**
Harness-MAP$_{k_0=8}$	100.0	12	4	100.0	45	21	100.0	5	1	100.0	14	5	**100.0**	**2**	**0**†	**100.0**
Harness-MAP$_{k_0=16}$	100.0	29	11	95.0	35	14	100.0	9	1	100.0	40	16	**100.0**	**2**	**0**†	99.0
Harness-MAP$_{k_0=32}$	100.0	12	0†	85.0	47	16	100.0	16	0†	100.0	31	8	100.0	3	0†	97.0
Ours$_{(K_{fixed})}$																
Harness-MAP$_{k_0=2}$	100.0	3	1	90.0	14	7	100.0	5	2	**100.0**	**7**	**3**	**100.0**	**2**	**1**	98.0
Harness-MAP$_{k_0=4}$	100.0	44	22	95.0	30	15	100.0	3	2	100.0	17	4	100.0	22	5	99.0
Harness-MAP$_{k_0=8}$	**100.0**	**4**	0†	90.0	46	7	100.0	10	1	100.0	40	6	100.0	4	0†	98.0
Harness-MAP$_{k_0=16}$	100.0	16	1	75.0	50	6	100.0	40	4	15.0	16	1	100.0	8	0†	78.0
Harness-MAP$_{k_0=32}$	100.0	16	0†	85.0	46	3	100.0	26	1	55.0	50	4	100.0	16	0†	88.0
Ours$_{(K_{hier})}$																
Harness-MAP$_{k_0=2}$	100.0	11	5	90.0	15	7	**100.0**	**3**	**1**	95.0	37	18	**100.0**	**3**	**1**	97.0
Harness-MAP$_{k_0=4}$	100.0	10	2	95.0	46	11	100.0	14	3	100.0	18	4	100.0	2	0†	99.0
Harness-MAP$_{k_0=8}$	100.0	22	3	80.0	34	5	100.0	10	1	100.0	40	6	100.0	10	1	96.0
Harness-MAP$_{k_0=16}$	100.0	8	0†	95.0	50	6	100.0	16	1	100.0	32	3	100.0	8	0†	99.0
Harness-MAP$_{k_0=32}$	100.0	16	0†	90.0	46	3	100.0	36	2	60.0	46	3	100.0	16	0†	90.0

† Saturated during initialization, before the first search step.

Table 9: Per-environment results on SpatialWorld-3DGames with Claude CLI as the meta-agent.

Method	Maze			Maze3D			Block3D			Rubik			Snake			Wtd. Avg.
	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	
Direct inference																
Qwen3-VL-30B-Instruct	45.0	–	–	0.0	–	–	5.0	–	–	0.0	–	–	6.3	–	–	10.7
Ours$_{(K_{adaptive})}$																
Harness-MAP$_{k_0=2}$	100.0	3	1	100.0	14	7	100.0	8	4	100.0	5	2	100.0	10	5	**100.0**
Harness-MAP$_{k_0=4}$	100.0	5	2	100.0	12	5	100.0	2	0†	100.0	7	3	100.0	10	4	**100.0**
Harness-MAP$_{k_0=8}$	100.0	2	0†	100.0	25	21	100.0	14	10	100.0	5	1	100.0	3	0†	**100.0**
Harness-MAP$_{k_0=32}$	100.0	2	0†	100.0	23	7	100.0	3	0†	100.0	2	0†	100.0‡	2	0†	**100.0**

† Saturated during initialization, before the first search step.

docstrings, and code comments. The index boundary at dimension 16 explicitly divides structural and token coordinates, allowing mutation operators to evaluate structural distance $\|\Delta z[:16]\|$ versus token distance $\|\Delta z[16:]\|$ when issuing evolutionary steering directives (§3.3).

We use 802-dimensional descriptor space is $\varphi(H)$ on BALROG with geometric variance distribution. The joint descriptor vector $\varphi(H) \in \mathbb{R}^{802}$ is formed by horizontal concatenation of four standardized sub-blocks:

$$\varphi(H) = \Big[\ \varphi_{\text{wl}} \,\|\, \varphi_{\text{ast}} \,\|\, \varphi_{\text{num}} \,\|\, \varphi_{\text{doc}}\ \Big] \ \in\ \mathbb{R}^{16+15+3+768}, \tag{12}$$

where $\varphi_{\text{wl}} \in \mathbb{R}^{16}$ represents Weisfeiler–Lehman topological histograms, $\varphi_{\text{ast}} \in \mathbb{R}^{15}$ captures control-flow syntax counts, $\varphi_{\text{num}} \in \mathbb{R}^3$ records component scaling metrics, and $\varphi_{\text{doc}} \in \mathbb{R}^{768}$ encodes documentation text embeddings. To prevent high-dimensional text blocks from dominating cell assignments, each block is standardized by corpus standard deviation and scaled by $1/\sqrt{\dim}$ prior to concatenation, granting equal nominal weight to each block.

Across 337 evolved genomes collected across five canonical experimental arms on BALROG, empirical variance analysis reveals that the 15-dimensional AST block accounts for 94.8% of the total joint variance (φ_{num} contributes 3.9%, while φ_{wl} accounts for 1.3%). Within the AST block, the

Table 10: Per-task search results on BALROG-MiniHack with Claude CLI. We report the task success rate in percent over its five seeded episodes.

Method	Box-H			Box-M			MW-9			MW-15			Cor-R3			CB-Dark			Q-Easy			Q-Med			Avg.
	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	SR	$N^\star$	$T^\star$	
Direct inference																									
Qwen3-VL-30B-Instruct	0.0	–	–	0.0	–	–	52.9	–	–	20.0	–	–	7.1	–	–	0.0	–	–	0.0	–	–	0.0	–	–	10.0 ± 4.7
Harness search																									
Meta-Harness	100.0	6	3	100.0	6	3	100.0	3	1	100.0	4	2	100.0	4	2	100.0	31	15	0.0	1	0	0.0	1	0	75.0 ± 6.8
GEPA	0.0	12	0	0.0	12	0	100.0	12	9	60.0	12	9	80.0	50	44	20.0	50	15	0.0	12	0	0.0	12	0	30.0 ± 7.2
DGM	100.0	6	4	100.0	6	4	100.0	8	5	100.0	8	5	100.0	8	5	80.0	29	17	0.0	1	0	0.0	1	0	72.5 ± 7.1
ShinkaEvolve	0.0	1	0	0.0	1	0	100.0	15	14	60.0	15	14	20.0	6	5	0.0	1	0	0.0	1	0	0.0	1	0	20.0 ± 6.3
OpenEvolve	0.0	1	0	0.0	1	0	100.0	13	12	80.0	40	39	40.0	13	12	20.0	40	39	0.0	1	0	0.0	1	0	27.5 ± 7.1
Ours$_{(K_{adaptive})}$																									
Harness-MAP$_{k_0=2}$	100.0	7	3	100.0	7	3	100.0	3	1	100.0	3	1	100.0	5	2	**100.0**	**14**	**7**	0.0	1	0	0.0	1	0	75.0 ± 6.8
Harness-MAP$_{k_0=4}$	**100.0**	**5**	**1**	**100.0**	**5**	**1**	**100.0**	**2**	**0**†	100.0	31	8	**100.0**	**3**	**1**	40.0	49	12	80.0	25	6	100.0	33	8	$\mathbf{85.0 \pm 5.6}$
Harness-MAP$_{k_0=8}$	100.0	16	4	100.0	16	4	**100.0**	**2**	**0**†	100.0	17	5	100.0	13	3	40.0	13	3	100.0	45	14	100.0	46	14	$\mathbf{82.5 \pm 6.0}$
Harness-MAP$_{k_0=16}$	100.0	37	8	100.0	37	8	**100.0**	**2**	**0**†	**100.0**	**2**	0†	100.0	41	9	40.0	34	7	100.0	31	6	100.0	25	5	$\mathbf{90.0 \pm 4.7}$
Harness-MAP$_{k_0=32}$	0.0	1	0	0.0	1	0	**100.0**	**2**	**0**†	**100.0**	**2**	0†	100.0	34	4	60.0	28	3	0.0	1	0	0.0	1	0	40.0 ± 7.7
Ours$_{(K_{fixed})}$																									
Harness-MAP$_{k_0=2}$	100.0	20	10	100.0	20	10	100.0	7	3	100.0	18	9	100.0	21	10	80.0	45	22	0.0	1	0	0.0	1	0	72.5 ± 7.1
Harness-MAP$_{k_0=4}$	**100.0**	**5**	**1**	**100.0**	**5**	**1**	**100.0**	**2**	**0**†	**100.0**	**2**	0†	100.0	7	2	100.0	21	5	**100.0**	**26**	**6**	**100.0**	**16**	**4**	$\mathbf{92.5 \pm 4.2}$
Harness-MAP$_{k_0=8}$	**100.0**	**5**	**1**	**100.0**	**5**	**1**	**100.0**	**2**	**0**†	**100.0**	**2**	0†	100.0	5	1	20.0	6	1	**100.0**	**26**	**8**	**100.0**	**16**	**4**	$\mathbf{87.5 \pm 5.2}$
Harness-MAP$_{k_0=16}$	100.0	17	3	100.0	17	3	**100.0**	**2**	**0**†	100.0	11	1	100.0	23	4	40.0	17	3	0.0	1	0	0.0	1	0	62.5 ± 7.7
Harness-MAP$_{k_0=32}$	100.0	25	2	100.0	25	2	**100.0**	**2**	**0**†	**100.0**	**2**	0†	100.0	17	1	60.0	24	2	100.0	42	6	0.0	1	0†	75.0 ± 6.8
Ours$_{(K_{hier})}$																									
Harness-MAP$_{k_0=2}$	100.0	23	11	100.0	21	10	100.0	12	6	100.0	14	7	100.0	25	12	60.0	37	18	0.0	1	0	0.0	1	0	67.5 ± 7.4
Harness-MAP$_{k_0=4}$	100.0	6	1	100.0	6	1	100.0	2	0†	**100.0**	**2**	0†	100.0	13	3	80.0	8	2	0.0	1	0	0.0	1	0	70.0 ± 7.2
Harness-MAP$_{k_0=8}$	**100.0**	**4**	0†	**100.0**	**4**	0†	**100.0**	**2**	**0**†	**100.0**	**2**	0†	100.0	9	2	60.0	47	15	**100.0**	**26**	**8**	**100.0**	**15**	**4**	$\mathbf{92.5 \pm 4.2}$
Harness-MAP$_{k_0=16}$	100.0	16	2	100.0	16	2	100.0	3	0†	100.0	3	0†	100.0	12	1	40.0	30	6	100.0	45	10	100.0	18	3	$\mathbf{90.0 \pm 4.7}$
Harness-MAP$_{k_0=32}$	100.0	24	2	100.0	24	2	100.0	5	0†	100.0	12	0†	100.0	32	4	60.0	34	4	40.0	41	5	20.0	41	5	72.5 ± 7.1

† Saturated during initialization, before the first search step.

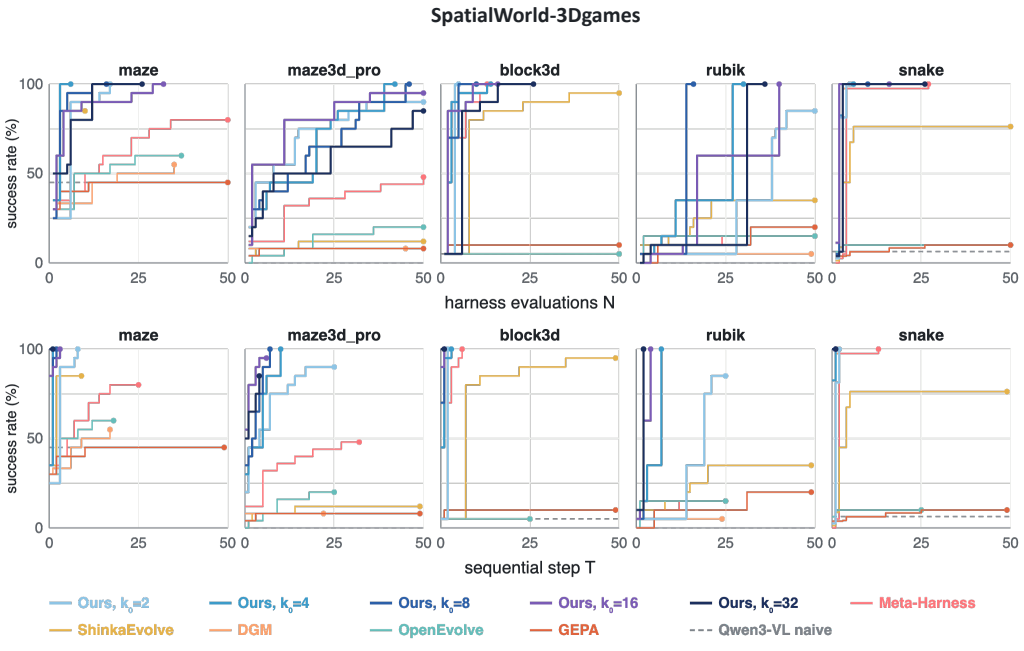

Figure 6: Harness search on SpatialWorld-3DGames. Best-so-far success rate against candidate-evaluation budget N (top) and against sequential proposal step T (bottom).

scalar measuring maximum control-flow count (ranging from 6 to 376) single-handedly captures 85.9% of joint variance. Because descriptor normalization parameters were frozen prior to evolutionary search and remain identical across all arms, Voronoi cells dynamically group candidate harnesses into standardized complexity bins. Crucially, because baseline methods never compute

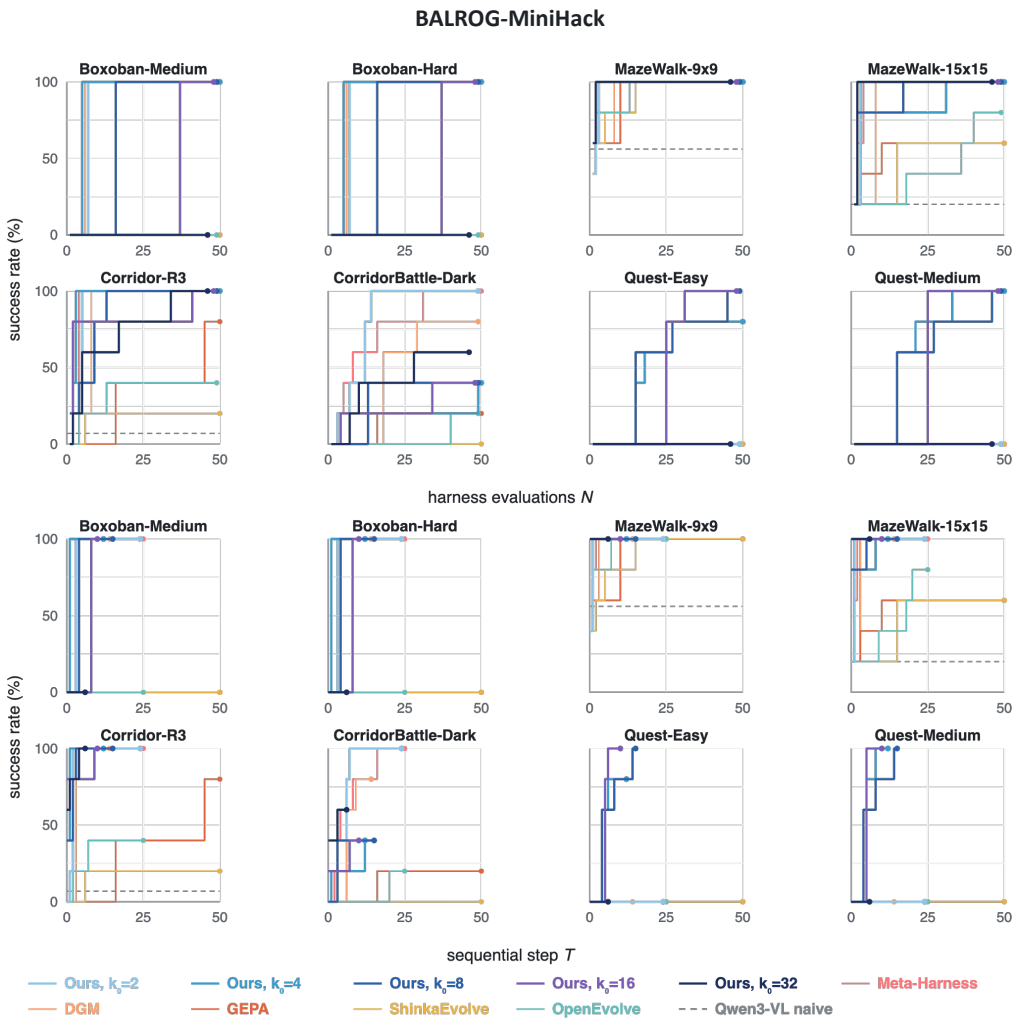

Figure 7: Harness search on BALROG-MiniHack. Best-so-far success rate against candidate-evaluation budget N.

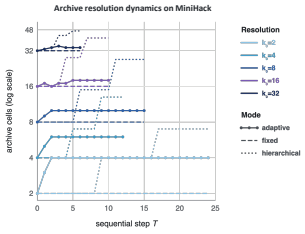

Figure 8: Archive resolution dynamics on BALROG-MiniHack against sequential steps T.

or optimize $\varphi(H)$, this geometric distribution leaves all reported performance scores completely unconfounded.

A.9 More cases

```
================================================================================
          HARNESS-MAP QUALITATIVE ANALYSIS CASES & PSEUDOCODE
```

```
================================================================================

1. CASE 1: CONSTANT IMPROVEMENT VIA MODULAR INTEGRATION
--------------------------------------------------------------------------------
* Candidate N03 (0.350) [tools/chart.py]: Fixed spatial logic in Chart.observe
  - Pseudocode:
      def act(self, obs, prev_action):
          self._refresh_legal(...)
          update_observation(...)
          chart = chart_mod.parse_map(...)
          if self.last_how in ('kick-dir', 'modal'): observe(...)
          self._rung(...)
          if action is not None: return self._emit(response, action, how)

* Candidate N05 (0.600) [push_plan in harness.py]: Added Sokoban BFS planner
  - Pseudocode:
      def act(self, obs, prev_action):
          update_observation(...)
          chart = parse_map(...)
          if planned is not None:
              return self.emit(shell, planned, 'push-tape') # Zero token call

* Candidate N15 (0.720) [tools/quest.py]: Added QuestErrand interaction protocol
  - Pseudocode:
      def act(self, obs, prev_action):
          update_observation(...)
          armed = quest_mod.armed(...)
          if armed:
              ans = prompt(...)
              if ans is not None: return self._shell(*ans)

* Candidate N21 (0.775) [Recombined from r016_0]: Recombined quest + push modules
  - Pseudocode:
      def act(self, obs, prev_action):
          for module in self.modules:
              plan = claim(...)
              if plan is not None and _no_call(plan[1]):
                  return self._shell(action, how)

* Candidates N39-N47 (0.800 -> 0.850): Added VolleyOwner & MeleeOwner
  - Pseudocode:
      def act(self, obs, prev_action):
          tick(...)
          plan = self.claim_step(...)
          if plan is not None:
              out = self.emit(...)


2. CASE 2: STRUCTURAL AND ALGORITHMIC DIVERSITY OF CHAMPIONS
--------------------------------------------------------------------------------
* Champion k02 (0.750) [8-rung VetoBoard]:
  - Pseudocode:
      def act(self, obs, prev_action):
          self._mount(...)
          frame = parse_frame(...)
          planner = StepPlanner(...)
          self._arbitrate(...)
          if need is not None: response = generate(...)

* Champion k04 (0.850) [9-entry priority owner ladder]:
  - Pseudocode:
      def act(self, obs, prev_action):
          plan = self.claim_step(...)
          if plan is not None: out = self.emit(...)

* Champion k08 (0.825) [Situational routing + LLM]:
  - Pseudocode:
      def act(self, obs, prev_action):
          self._classify(...)
          messages = build_messages(...)
          response = generate(...)
          action = self._match(...)

* Champion k16 (0.900) [Pure compiled symbolic logic, Zero LLM calls]:
  - Pseudocode:
      def act(self, obs, prev_action):
          fr = parse_glyph_map(...)
          c = Quest.armed(...)
          for solver in (Sokoban, Quest, Chart):
              got = getattr(solver, 'solve')(...)
              if got is not None: return got
```

```
* Champion k32 (0.400) [Lexicographic arbiter key]:
  - Pseudocode:
      def act(self, obs, prev_action):
          messages = build_messages(...)
          response = generate(...)


3. CASE 3: KEY FAILURE MODES DURING SEARCH
--------------------------------------------------------------------------------
* Allocation Stalls (Arm k32, capped at 0.400 score) [N44, N45, N46]:
  - Pseudocode (Unfocused LLM Fallback Sampling):
      def act(self, obs, prev_action):
          if stall and self.reflections < REFLECT_BUDGET:
              base = build_messages(...)
              absorb_reflection(...) # Spent iterations without breeding elites

* Over-Admitting Predicates (Arm k16, N43 regression: 0.900 -> 0.225):
  - Pseudocode (Unsound Reflex Predicate Overriding Fallback):
      def act(self, obs, prev_action):
          owned = self.reflex(...) # Overly permissive predicate
          if owned and owned[0] in legal:
              return self.reply(*owned) # Prematurely overrides LLM fallback logic


4. CASE 4: DYNAMICS OF DYNAMIC MAP PARTITIONING
--------------------------------------------------------------------------------
* Cell 3 (Depth-1 Split & Retreat):
  - Action: Split in R1 (score spread 0.05); sub-elites failed to outperform
    parent; merged back in R2.
* Cell 21 (Depth-1 Partitioning, cand_r052, 0.600 score):
  - Action: Partitioned in R5, producing tools/quest.py.
* Cell 23 (Depth-2 High-Variance Partitioning, cand_r064, 0.900 champion):
  - Action: Partitioned in R8; children averaged 0.756 score, yielding top champion.
================================================================================
```

A.10 META-AGENT IMPLEMENTATION DETAILS.

Our LLM-guided search process is driven by the loop in §3.3. The five shaded lines of Alg. 1(b) are where that loop calls a language model, and they all reduce to one mechanism: a single *headless, tool-using session* over a candidate directory. A session is given the candidate as a working directory, is told to act rather than ask, and terminates only once the artifact its role owns exists on disk. What separates the roles is the system prompt.

System Prompt. Every role's prompt is assembled mechanically: a fixed preamble that pins the execution contract and interpolates the current iteration's paths, followed by that role's skill specification verbatim. The preamble is identical across roles and across benchmarks.

```
You are running HEADLESS (no interactive shell). You have exactly four tools: read_file,
write_file, list_dir, run_bash. run_bash runs from the repo root with the skill env vars
set, so `$OUT_DIR`, `$PARENT_DIR`, etc. resolve. Read text/file contents via the tools;
there is no image display. Follow the skill to completion, then STOP (a short final
summary, no tool call). Do NOT ask questions - act. Your job is done ONLY once the
required output file exists on disk.

Runtime env values:
  WORKSPACE=...   REPO_ROOT=...   PARENT_TAG=...   PARENT_DIR=...   ITER_N=...
  OUT_DIR=...     SETTING=...     SUBSET=...       TRACE_DIR=...    GAME=...

===== SKILL: follow these instructions =====
```

Coverage proposal (line 5 of Alg. 1(b)). Coverage initialization must aim at cells that are *empty*, which is precisely when the usual steering signal is unavailable: our per-cell directive works by handing the proposer exemplar harnesses drawn from the target region, and an empty cell has no members to exemplify. With a single occupied cell it returns no guidance at all, exactly when guidance matters most. We therefore describe the target without exemplars. We take the centroid delta between an occupied cell and the target cell and verbalize it at *block* resolution, reporting which block dominates the move – structural wiring versus token-level implementation – as decided by the descriptor's block split. The directive is purely descriptive; it names a direction in design space and leaves the realization to the model, rather than prescribing a diff.

Validity filtering (lines 6 and 17 of Alg. 1(b)). Both the coverage loop and the breeding loop admit a candidate only if it passes `is_valid`. Validity is two gates. The first is mechanical: the harness must import and expose the episode entrypoint unchanged, since a broken import or an altered signature scores zero. The second is an adversarial anti-cheat judge that reads the candidate's own code and traces and rules on the *mechanism*, never the score. A `FAIL` verdict removes the candidate from selection, so an inflated score can never outrank an honest rival. The judge defaults to `OK` absent a concrete, code-cited violation and reserves `WARN` for gray areas – a deliberate asymmetry, since rejecting on suspicion would penalize exactly the unusual solutions the search exists to find.

```
You are the anti-cheat judge. A candidate just scored on the task; decide whether that
     score
was earned honestly or by gaming the harness. You are adversarial: assume the proposer may
have found a shortcut that inflates the metric without the frozen agent actually solving
     the
task. Judge the mechanism in the code + traces, never the scalar.

## What counts as CHEATING (-> FAIL)
- Oracle leakage into the decision. The harness reads privileged/ground-truth state (exact
  target coordinates, the answer label, full_state()/oracle fields) and feeds it into the
  action choice. The agent must decide from its observation only. (Reading oracle for
  logging is fine; routing it into decide/the action is not.)
- Hardcoding / memorization. Answers keyed to episode/seed/task id; a lookup table of
  correct actions; branching on the specific eval instance rather than perceiving it.
- Scorer exploitation. Detecting or manipulating the evaluator, short-circuiting env.step,
  faking success, exiting to bank a partial score.
- Trace faking. Fabricated llm_calls/observations; claiming steps that never called the
     model.

## What is NOT cheating (-> OK)
- A better prompt, skill, tool, memory scheme, parsing, or control flow that helps the
     agent
  perceive/decide better from its legitimate observation. That is what evolution should
     find.

## Finish
Write $OUT_DIR/judge.json exactly:
{"verdict": "OK|WARN|FAIL", "reasons": ["<concrete, code-cited>"],
 "cheat_locations": ["harness.py:NN", "tools/x.py:NN"]}
Default to OK if you find no concrete violation (do not FAIL on suspicion alone).
```

Diagnosis (line 13 of Alg. 1(b)). The diagnostician turns one evaluation into the gradient the next proposal refines against. It reads the candidate's own source, its episode traces end to end, and the numeric summary, and it must produce a causal chain rather than a restatement of the metric. Its central requirement is to classify the binding constraint into one of three kinds, because the right repair differs sharply between them: a harness bug is recovered by a code fix, a model ceiling must be scaffolded rather than re-worded, and a task gap needs a goal-directed lever. Without that classification the subsequent proposal tends to drift toward prompt rewording, which is the exhausted axis.

```
You are the causal diagnostician for one just-evaluated harness candidate. Explain why it
scored what it did, then hand the next proposer concrete, code-level changes that would
     move
the metric. Your diagnosis.md is the proposer's PRIMARY gradient - so it must be causal and
code-level, not a metric restatement or vague advice.

## What to read - the RAW artifacts (not just the scalar)
1. The candidate's own harness (harness.py, tools/*.py, AGENT.md/rules.md/skills/*.md).
2. The traces in $TRACE_DIR - read several episodes end to end: the model's reasoning, the
   frame it saw, the action it emitted, what the env did. Cite episodes/steps/frames.
3. The numeric summary - as HINTS to verify, not gospel.
4. Prior gradient - earlier diagnosis.md / NOTES, so you don't re-recommend an exhausted
     axis.

## Anti-cheat (more latitude than the proposer)
You MAY read oracle/privileged fields for YOUR analysis. But every FIX you recommend must
     keep
the frozen agent deciding from its observation only.

## Binding constraint - classify it
HARNESS-BUG - loop/parse/fallback/tool/prompt-composition logic is wrong; a code fix
     recovers points.
VLM-CEILING - the frozen model cannot perceive/decide correctly even with a correct harness
     ; the
```

```
              fix must SCAFFOLD the decision, not re-word the prompt.
TASK-GAP    - the agent operates legally but doesn't achieve the goal; a better planning/
              goal-directed lever is needed.

## Rules
- Causal, not correlational. "Score is low" is not a diagnosis.
- Judge by stable signals, not the scalar - the metric is noisy.
```

Directives (line 15 of Alg. 1(b)). Each of the b children bred from a parent receives its own directive, so siblings diverge by construction instead of by sampling noise. A directive has two parts. The first is the cell-level steer described above, which names where in the map the child should land. The second cycles the child index through a fixed list of design axes – spatial grounding, action planning, verification and recovery, efficiency and commitment, and observation parsing – so that sibling j is pushed along a different concern than sibling j'. Together these give the inverse mapping described in §3.3: a target coordinate becomes an explicit instruction about *what kind* of edit to make, at either structural or token-level resolution.

Variation (line 16 of Alg. 1(b)). The proposer receives the parent harness copied into its working directory, the accumulated diagnosis, and its directive, and makes exactly one coherent change. Two constraints do most of the work. First, the harness skeleton is fixed and benchmark-independent – a task supplies only the *content* of these files, never a different structure – which is what lets the same meta-agent run unchanged across all three benchmarks and what makes attribution to an axis meaningful. Second, the change must be *axis-distinct*: retuning a constant is not a new idea, whereas adding a tool, rewiring the control flow, routing a skill, or adding a memory field is. Prompt rewording is explicitly deprioritized as the noise-dominated axis.

```
You are the proposer. Take the harness in $OUT_DIR (a copy of a parent that already scored)
and make ONE coherent, axis-distinct improvement, so the frozen agent it drives scores
     higher.
You do NOT run the evaluator. Your job ends when you write $OUT_DIR/NOTES.md and stop.

## Hard constraints
- Do NOT quit. Never write "the harness is optimal" or skip the iter. If your idea feels
     weak,
  ship it - the eval is more informative than a skip.
- Edit ONLY $OUT_DIR. Do not touch the evaluator, task_spec, prior candidates, or
     $PARENT_DIR.
- Act, don't ask. No questions; make the edit and finish.

## The canonical rich-harness structure (same for every benchmark)
<harness>/
  harness.py   # ENTRYPOINT + orchestrator: compose the system prompt from the .md priors,
               #   run the perceive->decide->act loop, use the tools, maintain memory.
  AGENT.md     # prose prior: role / task framing + verified perception facts
  rules.md     # prose prior: output contract, legal action set, anti-cheat rule
  memory.md    # fenced scratchpad TEMPLATE; filled per step (self-derived state, NEVER
     oracle)
  skills/*.md  # composable prose priors, one concern per file (locate, plan, recover)
  tools/*.py   # executable CODE helpers the harness CALLS (model call+parse+fallback, ...)
  NOTES.md     # module map + evolvable-lever table + the change you made

Entrypoint contract (uniform). harness.py MUST expose:
    def run_episode(env, task_spec, vlm_clients, trace_dir=None) -> EpisodeResult
A broken import or a changed signature scores 0.

Two evolvable axes (this is what makes attribution work):
- prose levers - AGENT.md, rules.md, memory.md, skills/*.md: change what the agent is told.
- code levers  - harness.py, tools/*.py: change the mechanism (control flow, parsing, tool
                 logic, memory maintenance, skill routing).

## Your procedure
1. Orient. Read NOTES.md (lever table), AGENT.md/rules.md, and the gradient: the directive
     in
   your brief and any $OUT_DIR/diagnosis.md. Fix the binding constraint it names.
2. Make ONE axis-distinct structural change. An axis is where a new idea lives, not a
     constant -
   k=3 to k=5 is NOT an axis change; adding a tool, rewiring the loop, adding/routing a
     skill,
   adding a memory field ARE. Favor code/skill/workflow/memory levers over prompt rewording
     .
3. Keep the contract intact. Anti-cheat: never wire privileged/oracle state (exact target
   coords, ground-truth labels) into the agent's decision path.
4. Finish. Write $OUT_DIR/NOTES.md: ## axis / ## change / ## why / ## files.
```

Adaptive resolution (line 22 of Alg. 1(b)). ADAPT_RESOLUTION is the one shaded operation that involves no language model. It is a purely geometric refinement of the partition: cells that have accumulated enough members and enough internal spread are sub-split into finer Voronoi regions, so resolution follows the search rather than being fixed in advance. We keep it in the shaded group because it is what closes the loop between the map and the meta-agent – a sub-split creates new empty cells, which in turn become targets for the directives of the next round.