EvoCurr: Self-evolving Curriculum with Behavior Code Generation for Complex Decision-making

Yang Cheng[3*†] **Weiyu Ma**[4*] **Zilai Wang**[5] **Wenhui Zhu**[1,2] **Yujing Hu**[6]
Tangjie Lv[6] **Mohamed Elhoseiny**[4‡] **Yue Deng**[1,2‡] **Jian Zhao**[1,2‡]

[1] Zhongguancun Academy [2] Zhongguancun Institute of Artificial Intelligence
[3] University of Science and Technology of China [5] Xi'an Jiaotong University
[4] King Abdullah University of Science and Technology [6] NetEase Fuxi AI Lab

Abstract

Complex decision-making often requires agents to progress through intermediate tasks rather than solve the final target directly. Existing LLM self-refinement methods typically iterate on a fixed target, while curriculum-learning methods often rely on hand-designed schedules, training-time optimization, or domain-specific difficulty metrics. To smooth the learning curve through adaptive curriculum design, we introduce EvoCurr, a general inference-time framework that co-evolves curricula and executable policies. A Designer proposes verifiable intermediate tasks from the latest accepted progress and recent failures, while a Solver generates or trains policies for these tasks. A task–policy pair is accepted only when it passes hard feasibility checks and reaches a specified performance threshold; meanwhile, an accepted-floor constraint preserves the latest verified checkpoint and prevents failed harder attempts from overwriting mastered behavior. We evaluate EvoCurr across code-as-policy and closed-loop MARL settings, using StarCraft II super-late-game micromanagement as our hardest stress test. EvoCurr demonstrates effective micromanagement across three LLM backbones without LLM parameter training. Improvements in additional experiments on Flatland, POGEMA, Overcooked, and LiveCodeBench Hard demonstrate the generality of EvoCurr.

1 Introduction

Complex decision-making tasks are rarely solved by attempting the final objective in one step. Agents often need to master simpler subproblems, preserve the successful behaviors, and gradually extend them toward harder goals. Curriculum learning formalizes this intuition, but practical curricula are still difficult to deploy broadly, as many methods require hand-designed task sequences, training-time schedules, or domain-specific difficulty metrics (Bengio et al., 2009; Wang et al., 2021; Soviany et al., 2022). These assumptions are restrictive for LLM-based agents, which are increasingly expected to solve new tasks at inference time by code generation, interpreting feedback, and interacting with external environments.

Most LLM self-improvement methods also do not explicitly construct curricula. They usually keep the target fixed: the model generates a solution, observes feedback, and then revises another solution for the same final task (Shinn et al., 2023). Such fixed-target refinement can repair local errors, but it is often insufficient when the final hard task combines executable validity, long-horizon control, multi-agent coordination, and environment-specific constraints. In these settings, progress may require changing the task itself. The system should identify intermediate tasks that are solvable, verify the corresponding policies, and use them as stepping stones toward the final objective.

We propose EvoCurr, an inference-time framework for curriculum-policy co-evolution. Given a final task and an acceptance criterion, a Designer proposes the next curriculum task using the final target, the latest accepted progress, and the history of recent successes and failures. A Solver then generates or trains a policy for the proposed task. The candidate is accepted only if it is feasible and reaches the required score. During the evolution process, feasibility is enforced through hard checks such as task-schema validation, code compilation when applicable, and runtime execution. Meanwhile, the accepted-floor constraint enables EvoCurr to advance only when a candidate task-policy pair is verified. The latest accepted curriculum is preserved, together with a rollback copy of its policy, when the Solver ultimately fails a harder task. This separates productive exploration from

*Equal contribution.
†Work done while interning at Zhongguancun Academy.
‡Correspondence: mohamed.elhoseiny@kaust.edu.sa; dengyue@zgci.ac.cn; jianzhao@zgci.ac.cn.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 35089–35124
October 24-29, 2026 ©2026 Association for Computational Linguistics

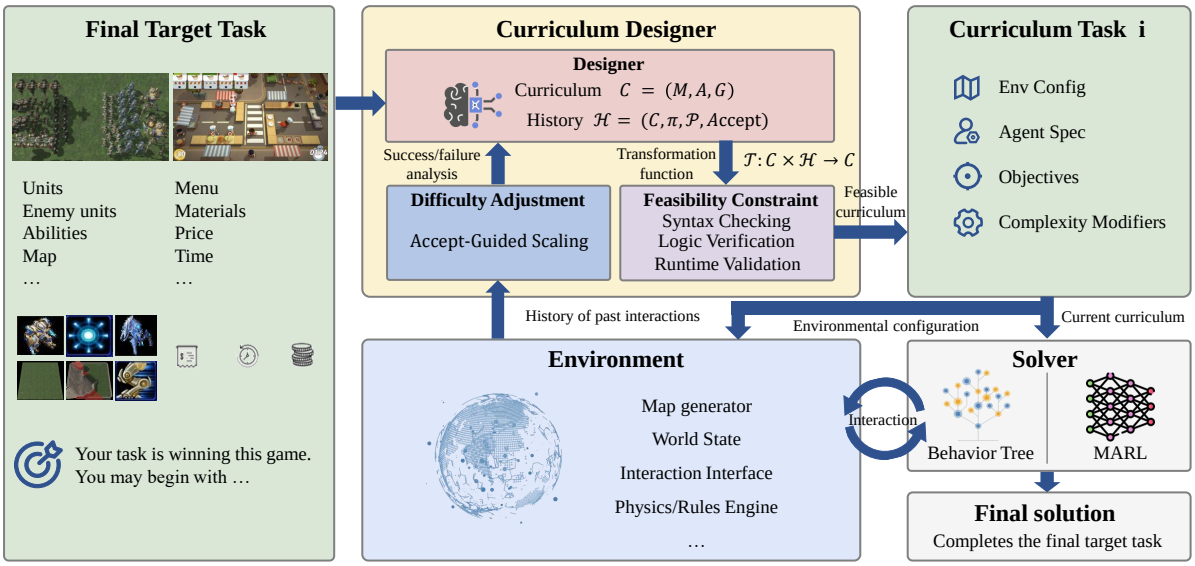


Figure 1: EvoCurr overview. A curriculum designer proposes the next curriculum C_{t+1}; a solver produces an executable policy π_{t+1} and evaluates it; the outcome feeds back to the designer. The loop starts from a simplified version of the final target T_f and proceeds until T_f is solved.

policy regression and allows the system to search over task difficulty at inference time.

We use StarCraft II super-late-game micromanagement as the hardest stress test for this idea. As shown in Figure 2, this setting sits between SMAC-style fixed skirmishes (Samvelyan et al., 2019) and AlphaStar-scale full-game training (Vinyals et al., 2019): economy, scouting, and production are removed, but the remaining battle still requires coordinated control over large heterogeneous armies under simulator feedback. Recent LLM-based StarCraft agents show that language models can reason about strategies and write game-playing code, while also exposing weaknesses in late-game control, long-horizon planning, and policy interpretability (Ma et al., 2024). This makes the domain a compact test of whether an LLM can turn local tactical knowledge into one coherent executable controller.

Across 12 StarCraft II late-game micro tasks, EvoCurr evolves interpretable behavior-tree policies with no parameter training for the LLM Solver and achieves high final-task win rates, while direct generation, fixed-target self-reflection, and autonomous coding-agent baselines are much less reliable. We further evaluate the same Designer–Solver mechanism beyond games: Flatland railway scheduling (Mohanty et al., 2020), POGEMA cooperative pathfinding (Skrynnik et al., 2025) , Overcooked MARL (Carroll et al., 2020), and LiveCodeBench (Jain et al., 2025) Hard show positive transfer across code-as-policy and closed-loop training settings, while pandapipes (Lohmeier et al., 2020) and JSSP (Lee et al., 2024) clarify cases where curriculum adds little because a simpler feedback loop or heuristic already saturates the task. These results support the method-level claim that self-evolving curricula help when direct final-task generation fails but meaningful, verifiable intermediate tasks exist.

Our contributions are fourfold: 1) We introduce EvoCurr, a general inference-time framework for co-evolving curricula and executable policies. 2) We propose an accepted-floor constraint that preserves verified progress while allowing failed harder tasks to guide future curriculum proposals. 3) We introduce a new StarCraft II super-late-game micromanagement benchmark that extends beyond SMAC-style skirmishes toward professional-level late-game control with large heterogeneous armies, diverse abilities, and terrain-dependent tactics. 4) We demonstrate the framework across code-as-policy and closed-loop MARL domains, using StarCraft II as the hardest stress test and additional benchmarks to establish method-level generality.

2 Related Work

Curriculum Learning. (Bengio et al., 2009) formalized curriculum learning, demonstrating that training on examples organized from easy to hard

improves generalization and convergence compared to random data shuffling. This paradigm has achieved success across computer vision, NLP, and reinforcement learning (Soviany et al., 2022; Wang et al., 2021). Various adaptive paradigms have been explored to eliminate predefined schedules. Self-paced learning (SPL) automatically determines the learning pace based on sample difficulty (Kumar et al., 2010), which was further extended with diversity constraints (Jiang et al., 2015). In reinforcement learning (RL), adaptive curriculum generation has been framed via comprehensive task frameworks (Narvekar et al., 2020), inference-based optimizations (Klink et al., 2020), teacher-student networks (Matiisen et al., 2019), and prioritized level replays (Jiang et al., 2021). However, these approaches primarily focus on training phase optimization and require either manual curriculum design or domain-specific difficulty metrics, leaving a gap for inference-time adaptive curriculum generation.

Environment Generation. Procedural content generation has evolved toward learning-based approaches, such as POET's co-evolution (Wang et al., 2019) and PAIRED's adversarial training (Dennis et al., 2020) for generating challenging, solvable environments. Recent LLM-driven methods like EnvGen (Zala et al., 2024) and Rainbow Teaming (Samvelyan et al., 2024) adaptively create training scenarios based on world knowledge. Other works explore evolution strategies and transferable curricul (Parker-Holder et al., 2022). However, these approaches primarily generate static training data prior to agent learning, rather than dynamically adapting during inference based on the solver's immediate capabilities.

LLM Self-Refinement and Executable Reasoning. A growing line of work improves LLM outputs through iterative feedback, self-reflection, or execution traces. Reflexion-style methods (Shinn et al., 2023) revise a solution after observing errors, while code-generation benchmarks such as LiveCodeBench (Jain et al., 2025) emphasize contamination-controlled evaluation of executable programs. These approaches are complementary to EvoCurr but operate on a fixed target: each iteration still asks the model to solve the same full problem. EvoCurr instead changes the task distribution during inference, using accepted intermediate tasks as checkpoints so that feedback controls both policy repair and difficulty progression.

Executable Policy Generation. Games such as StarCraft II have become standard benchmarks for complex decision-making, established by PySC2 (Li et al., 2026) and AlphaStar (Vinyals et al., 2019). Recent LLM integration includes TextStarCraft II (Ma et al., 2024, 2025a,b; Li et al., 2026) and behavior tree approaches (Deng et al., 2025, 2024). For code generation, Code as Policies (Liang et al., 2023) targets robot control, while behavior tree synthesis (Colledanchise and Ögren, 2018; Lykov and Tsetserukou, 2024) demonstrates that LLMs can produce structurally correct trees. These methods successfully generate executable policies but operate on fixed tasks without adaptive difficulty progression.

3 Method

EvoCurr frames inference-time policy generation as an iterative curriculum loop (Figure 1). Rather than asking the Solver to attempt the final target directly, a Designer proposes a sequence of progressively harder tasks; an *accepted floor* anchors progress to the most recently mastered task, preventing regression when a harder candidate fails.

3.1 Two-Agent Cooperative Framework

Let T_f be the final task and $\mathcal{H}_t$ the history of previous attempts. The *accepted floor* C_t^+ is the latest curriculum on which the Solver met the acceptance criterion. A rollback copy of the policy associated with C_t^+ is retained separately. At round t, the Designer proposes a candidate task. Here S denotes the complete task-level solving process, which may include multiple policy-generation or repair attempts for the same task before it returns a policy for evaluation:

$$\begin{aligned} \widehat{C}_{t+1} &= D(\mathcal{H}_t, C_t^+, T_f), \\ \widehat{\pi}_{t+1} &= S(\widehat{C}_{t+1}, \mathcal{H}_t), \\ m_{t+1}^{(i)} &= \mathcal{P}^{(i)}(\widehat{\pi}_{t+1}, \widehat{C}_{t+1}), \quad i = 1, \ldots, n. \end{aligned} \tag{1}$$

Here D (the Designer) and S (the Solver) may be implemented by LLM prompting, code generation, or RL training depending on the domain; $\mathcal{P}$ is the task-specific rollout metric, such as SC2 game success or Overcooked order completion. Accordingly, $m_{t+1}^{(i)}$ is the metric value from the i-th of n evaluation rollouts for the candidate task–policy pair, where n is the total number of rollouts at that stage; $\mathbb{1}[\cdot]$ denotes the indicator function.

Let $a_{t+1} \in \{0, 1\}$ be the accept flag for round $t + 1$. The empirical rollout-pass fraction $\widehat{r}_{t+1}$ is

computed by thresholding each rollout metric at τ, and ρ is the required fraction of passing rollouts in a stage:

$$\widehat{r}_{t+1} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[m_{t+1}^{(i)} \geq \tau]. \tag{2}$$

The candidate is accepted only if it is feasible and its pass fraction reaches ρ:

$$a_{t+1} = \mathbb{1}\Big[F_{\text{feas}}(\widehat{C}_{t+1}, \widehat{\pi}_{t+1}) = 1 \ \wedge \ \widehat{r}_{t+1} \geq \rho\Big]. \tag{3}$$

Here F_{feas} is a hard validity check covering task-schema validation, code compilation, and at least one runtime execution. The accepted-floor update is then:

$$C_{t+1}^{+} = \begin{cases} \widehat{C}_{t+1}, & a_{t+1} = 1, \\ C_t^{+}, & a_{t+1} = 0. \end{cases} \tag{4}$$

This is the main anti-regression mechanism. A failed harder task may inform the next Designer prompt, but it never replaces the most recently accepted task. A failed policy attempt within S does not itself trigger rollback; the Solver may continue revising the policy for the same task. Only when that task is declared unsuccessful after the task-level solving process is the policy restored to the copy associated with C_t^{+} before the next task proposal.

3.2 Curriculum Generation and Feasibility Constraints

A curriculum is a task configuration. In StarCraft II, for example, $C = (M, A, G)$ encodes the map, army specification , and goal; the Designer changes unit counts, unit types, abilities, and spawn regions to move from a simple fight toward T_f. In other domains, the same slot C denotes the corresponding problem instance: a railway layout in Flatland, a partially observable grid in POGEMA, an Overcooked layout and order schedule, or a code-generation subgoal.

The Designer is guided by a simple pacing rule rather than by a universal closed-form difficulty metric. After success, it proposes a task closer to T_f. After failure, it proposes an intermediate task that is harder than the accepted floor but easier than the failed candidate. This keeps progress monotone with respect to solved tasks while still allowing the curriculum to back off when the Solver is overchallenged.

Algorithm 1 EvoCurr Loop with Accepted-Floor Constraint

Require: Final task T_f, starting task C_0, rollout threshold τ, pass fraction ρ, and search budget B (the maximum number of curriculum rounds)
Objects: D is the Designer, S is the Solver, $\mathcal{P}$ is the evaluator, and F_{feas} checks compile/runtime validity.
State: $\mathcal{H}_t$ stores tried tasks, policies, rollout metrics, and accept flags; C_t^{+} is the accepted floor. A rollback copy of its associated policy is stored separately.
Variables: $\widehat{C}_{t+1}$ is a candidate task, $\widehat{\pi}_{t+1}$ is its policy, $\widehat{r}_{t+1}$ is its rollout-pass fraction, and a_{t+1} is the accept flag.
Initialize history $\mathcal{H}_0 \leftarrow \emptyset$ and accepted floor $C_0^{+} \leftarrow C_0$
for $t = 0, 1, \ldots, B-1$ **do**
  Designer proposes a candidate task
  $\widehat{C}_{t+1} \leftarrow D(\mathcal{H}_t, C_t^{+}, T_f)$
  Solver generates a policy for the candidate
  $\widehat{\pi}_{t+1} \leftarrow S(\widehat{C}_{t+1}, \mathcal{H}_t)$
  S may revise multiple policies before declaring this task solved or failed
  Evaluate and decide whether to accept
  Evaluate $m_{t+1}^{(1)}, \ldots, m_{t+1}^{(n)}$ and compute $\widehat{r}_{t+1}$
  $a_{t+1} \leftarrow \mathbb{1}[F_{\text{feas}}(\widehat{C}_{t+1}, \widehat{\pi}_{t+1}) = 1 \wedge \widehat{r}_{t+1} \geq \rho]$
  if $a_{t+1} = 1$ **then**
    $C_{t+1}^{+} \leftarrow \widehat{C}_{t+1}$ and save $\widehat{\pi}_{t+1}$ for rollback
  else
    $C_{t+1}^{+} \leftarrow C_t^{+}$ and restore its saved policy
  end if
  $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t \cup \{(\widehat{C}_{t+1}, \widehat{\pi}_{t+1}, \{m_{t+1}^{(i)}\}_{i=1}^{n}, a_{t+1})\}$
  if $a_{t+1} = 1$ and $\widehat{C}_{t+1} = T_f$ **then**
    return success with $\widehat{\pi}_{t+1}$
  end if
end for
return failure with the last accepted floor C_B^{+}

The feasibility gate F_{feas} is a hard validity check, not a soft reward. It includes task-schema validation, constraint checking, code compilation when the Solver emits code, and at least one runtime execution. Invalid tasks and non-executable policies receive $a_{t+1} = 0$ even if their textual plan appears plausible.

3.3 Solver Implementation Across Domains

The Solver can instantiate $S(\cdot)$ in two ways. For open-loop code-as-policy tasks such as StarCraft II, Flatland, POGEMA, and LiveCodeBench, it generates executable programs. In StarCraft II these programs are behavior trees: the Solver first writes structured control code, then repairs syntax/runtime errors, and finally refines strategic failures using rollout feedback. For closed-loop domains such as Overcooked, $S(\cdot)$ is an RL trainer; π_{t+1} denotes the trained policy parameters after a fixed environment-step budget on $\widehat{C}_{t+1}$.

Thus the policy representation changes across domains, but the outer loop does not: propose a task, solve it, evaluate it, and update the accepted

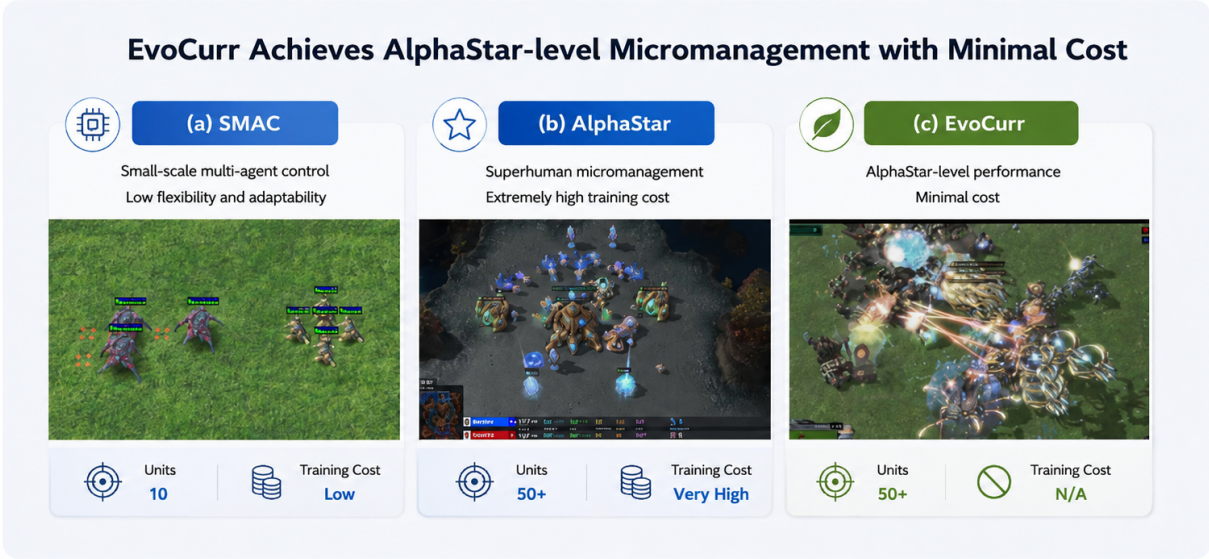


Figure 2: Positioning EvoCurr against existing StarCraft II paradigms. (a) SMAC targets small-scale skirmishes with limited flexibility. (b) AlphaStar handles large-scale, full-game complexity but requires prohibitive training compute. (c) EvoCurr tackles complex, super-late-game micromanagement (50+ units) at inference time, achieving high-level coordination with zero parameter training cost.

floor only after executable success.

Feedback semantics. The loop distinguishes feasibility feedback from performance feedback. A schema, compilation, or runtime failure means that a candidate lies outside the executable search space and therefore cannot advance the curriculum, regardless of how plausible its textual plan appears. An executable candidate with $\widehat{r}_{t+1} < \rho$ provides a different signal: the proposed task is valid, but the current policy is not yet reliable enough to serve as a checkpoint. Both outcomes remain in $\mathcal{H}_{t+1}$, so the Designer can use not only the latest success but also the location and type of recent failures when choosing the next difficulty increment. Conversely, acceptance promotes the task and its evaluated policy together. This coupling prevents task progression from running ahead of demonstrated solver capability.

Initialization and termination. Search starts from a seed C_0 that is simpler than T_f and serves as the initial accepted floor. EvoCurr succeeds only when the final task passes both feasibility and rollout criteria. If budget B is exhausted, the method returns the last accepted floor and saved policy rather than an unverified attempt. Thus, an unsuccessful search still returns an executable artifact and auditable failure history. Across domains, only the task representation, Solver, evaluator, and acceptance thresholds differ.

4 Experiments

We evaluate EvoCurr across multiple domains, with StarCraft II micro-management as the flagship stress test. The Solver generates complete behavior-tree scripts for fixed-army late-game engagements, turning a traditionally closed-loop control problem into open-loop code-as-policy synthesis. Further details on the proposed StarCraft II super-late-game micromanagement benchmark, including its distinction from SMAC-style skirmishes and its scope relative to AlphaStar-scale full-game play, are provided in Appendix B. Overcooked (Carroll et al., 2020) then tests the same Designer–Solver loop in a closed-loop MARL setting, where the Solver trains neural policies that continuously adapt to observations.

After these two primary experiments, Section 5 broadens the evaluation to Flatland railway scheduling, POGEMA cooperative pathfinding, LiveCodeBench Hard code generation, and two boundary controls (pandapipes and JSSP). Across these settings, the policy representation changes, but F_{feas}, the accepted-floor constraint, and progressive task generation remain the same. Implementation details are in Appendices G and E.1.

Evaluation questions. The experiments are organized around three questions. First, does

curriculum–policy co-evolution improve final-task reliability over direct generation and fixed-target revision? Second, does the same outer loop remain useful when the Solver changes from executable code generation to closed-loop policy training? Third, which task properties determine whether an explicit curriculum provides useful leverage? We therefore report both positive domains and boundary controls, retain executable failures in the reported outcomes, and separate task success from graded quality metrics whenever the environment supports both. Because metrics have different semantics across domains, we report them per domain rather than forming an uninterpretable aggregate score.

4.1 StarCraft II Micro-Management

Experiment Setup The Solver generates `python-sc2` behavior trees that act at the unit-action level. Generation occurs offline, whereas execution is observation-conditioned. To validate the framework's effectiveness across different architectures, we instantiate EvoCurr using three distinct LLM backbones: DeepSeek-V3.1, GPT-5, and Claude-4. We test on six newly designed micro maps against two opponent races (Terran vs Protoss (TvP) and Terran vs Zerg (TvZ)). Each of the 12 map–matchup configurations has its own final target T_f, specifying unit sets, technologies, and spawn regions; *compile-and-run* serves as a hard feasibility check in line with F_{feas}. Table 22 in Appendix E.3 reports the Bush–TvP target used in the expert-player pilot. For SC2, a rollout passes when it is won ($\tau = 1$), and a stage is accepted when at least 9 of 10 rollouts pass ($\rho = 0.9$). The primary baseline, *Direct Code*, attempts the same final tasks with the same environment and control interface; the methods' measured LLM-call budgets differ and are reported in Appendix B. Per-curriculum compositions, complete evolution traces, and the matched-game-conditions expert-player pilot are summarized in the appendix (Appendix B).

Statistical protocol. For direct baselines, we evaluate 10 independent target-task attempts per backbone to capture variability, executable rates, and peak performance. Conversely, an EvoCurr run spans a full curriculum-search trajectory with feasibility filtering and accepted-floor updates. We evaluate robustness via three views: reproducible task-level final-policy success over 10 game rollouts (Table 1), combat-score stability across backbones and attempts (Table 2), and direct-attempt distributions (Appendix C). A separate 120-game aggregate audit is retained in Appendix Table 7. Non-executable attempts score 0. The damage-cost-aware combat score $\mathcal{S}_{combat}$ is defined as:

$$\mathcal{S}_{combat} = \frac{1}{2}\left(1 + \frac{R_{\text{agent\_final}}}{R_{\text{agent\_init}}} - \frac{R_{\text{enemy\_final}}}{R_{\text{enemy\_init}}}\right), \tag{5}$$

where $R_{\text{agent\_final}}$, $R_{\text{agent\_init}}$, $R_{\text{enemy\_final}}$, and $R_{\text{enemy\_init}}$ are instantiated from the following side-level combat-power function:

$$R_{\text{side}} = \sum_i (m_i + \alpha v_i + \beta b_i)\,\frac{h_i + s_i}{h_i^{\max} + s_i^{\max}}. \tag{6}$$

where m_i, v_i, and b_i denote the mineral cost, vespene gas cost, and supply cost of unit i, respectively, and α, β are balancing coefficients for resource normalization. The health ratio $\frac{h_i+s_i}{h_i^{\max}+s_i^{\max}}$ measures the remaining combat effectiveness of each unit based on current and maximum health/shield values.

This metric aggregates resource cost and remaining health/shields; scores range from 0 to 1. A combat score $\mathcal{S}_{combat} > 0.5$ indicates successful annihilation of the majority of enemy forces while preserving our own, providing a finer-grained assessment than binary win/loss especially for failed code executions where the first term becomes 0.

Experimental Results We evaluate 12 complex micro-management tasks (2 matchups $\times$ 6 maps), with the reproducible task-level results retained in Table 1. In the separate 120-game aggregate audit reported in Appendix Table 7, EvoCurr obtains 99/120 wins (82.5%) with DeepSeek, 108/120 (90.0%) with Claude-4, and 113/120 (94.2%) with GPT-5. The combat-score statistics in Table 2 give the corresponding graded view: EvoCurr's curriculum-selected policies remain consistently strong across backbones, while direct generation is much less stable across its 10 one-shot attempts, even when its selected-best policy is competitive on some maps.

In contrast, baseline performance varies by specialization. Direct-10 ranges from 42.5% to 70.0% across the completed backbones, Reflexion-20 from 23.3% to 62.5%, and the code-agent baselines from 0.0% to 25.8%. Thus, while coding agents excel syntactically, they still struggle with strategic behavior-tree generation, a gap EvoCurr bridges through curriculum-driven evolution.

Table 1: Final-policy win rate (%) on StarCraft II micro tasks. Each selected policy is evaluated over 10 SC2 rollouts. This table provides the task-level success view; Table 2 reports the compact combat-score robustness summary.

Task	**EvoCurr** (DeepSeek)	**EvoCurr** (Claude)	**EvoCurr** (GPT)	DeepSeek-V3.1	GPT-5	Claude-4	Gemini-2.5	Claude_code	Codex	Cursor
bush (TVP)	**100**	**100**	**100**	0	0	10	0	0	0	0
corridor (TVP)	**100**	70	90	80	80	**100**	0	0	0	0
corner (TVP)	90	**100**	**100**	**100**	90	**100**	30	0	0	0
flat (TVP)	**100**	**100**	**100**	0	**100**	40	0	0	0	0
main (TVP)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	0	0	0
ramp (TVP)	**100**	**100**	**100**	10	0	**100**	**100**	0	0	0
bush (TVZ)	**100**	90	**100**	**100**	0	**100**	0	40	0	0
corridor (TVZ)	**100**	**100**	90	90	0	**100**	**100**	0	0	0
corner (TVZ)	**100**	**100**	**100**	30	**100**	**100**	**100**	**100**	**100**	0
flat (TVZ)	**100**	**100**	90	0	0	0	0	0	0	0
main (TVZ)	**100**	**100**	**100**	**100**	0	0	**100**	10	20	0
ramp (TVZ)	**100**	**100**	**100**	50	60	**100**	0	20	0	0

Table 2: Compact SC2 combat-score robustness summary. EvoCurr selected reports across-backbone mean ± std over three full curriculum-search outputs, one per backbone. Direct selected reports mean ± std over the best executable policy found by each of four direct backbones after 10 one-shot attempts. Direct all attempts pools all 40 one-shot attempts; non-executable attempts are counted as score 0. Full selected-score and direct-attempt tables are in the appendix.

Task	**EvoCurr selected**	**Direct selected**	**Direct all attempts**	**Direct exec.**
Bush (TvP)	0.677 ± 0.036	0.471 ± 0.100	0.167 ± 0.191	19/40
Corridor (TvP)	0.648 ± 0.058	0.557 ± 0.145	0.209 ± 0.220	22/40
Corner (TvP)	0.616 ± 0.057	0.636 ± 0.061	0.220 ± 0.252	20/40
Flat (TvP)	0.645 ± 0.048	0.434 ± 0.154	0.103 ± 0.188	10/40
Main (TvP)	0.684 ± 0.038	0.745 ± 0.076	0.239 ± 0.312	16/40
Ramp (TvP)	0.679 ± 0.044	0.651 ± 0.089	0.218 ± 0.273	17/40
Bush (TvZ)	0.683 ± 0.044	0.524 ± 0.153	0.163 ± 0.225	15/40
Corridor (TvZ)	0.641 ± 0.062	0.640 ± 0.066	0.173 ± 0.235	16/40
Corner (TvZ)	0.734 ± 0.110	0.698 ± 0.136	0.257 ± 0.318	17/40
Flat (TvZ)	0.590 ± 0.051	0.395 ± 0.104	0.113 ± 0.161	14/40
Main (TvZ)	0.738 ± 0.032	0.660 ± 0.103	0.283 ± 0.291	21/40
Ramp (TvZ)	0.702 ± 0.047	0.462 ± 0.152	0.173 ± 0.196	19/40

Scope and compute. EvoCurr targets a deliberately scoped fixed-army micro mini-game, not full StarCraft II: there is no economy, scouting, production, or build-order search. This scope makes the result complementary to AlphaStar-scale full-game training and substantially harder per engagement than SMAC-style small skirmishes. EvoCurr costs roughly 21–26 audited LLM calls per final SC2 task, compared with 10 for Direct-10 and 39 for Reflexion-20. We do not claim equal budgets or wall-clock efficiency. Detailed scope comparison, environment-interaction accounting, and iterative-baseline budgets are reported in Appendix B.

4.2 Overcooked

Experiment Setup We instantiate EvoCurr in a closed-loop Overcooked setting, with curricula parameterized by layouts, ingredient placements, and timing constraints. Unlike SC2, the Solver is a MARL trainer optimizing decentralized policies under a fixed $B = 10^7$ timestep budget per curriculum. Acceptance requires completing all mandatory orders ($\mathcal{P}(\pi|C) = 1$). The framework also yields greater rewards for fulfilling subsequent bonus orders (contributing to total deliveries in Table 3), isolating the impact of curriculum evolution from standard policy optimization. Detailed curriculum specifications and complete performance metrics for both map configurations are provided in Appendix H.

Experimental Results As shown in Table 3, EvoCurr outperforms direct E3T training under matched total environment-step budgets. On Coord. Ring with Multi-Recipe, EvoCurr achieves 29.10 effective deliveries on average versus 24.11 for direct training; on Counter Circuit with Multi-Recipe, EvoCurr reaches 18.64 versus 16.36 (Wang et al., 2024). The higher delivery counts include both required and bonus orders, showing that the progres-

Table 3: Curriculum progression and performance metrics for Overcooked maps.

Map	Task	Orders	Agent0 Delivery	Agent1 Delivery	Total Delivery	Total Reward
Map 1	EvoCurr	5	14.74	14.36	29.10	290.9
Map 1	Direct Training	5	11.93	12.18	24.11	240.5
Map 2	EvoCurr	5	7.82	10.82	18.64	186.2
Map 2	Direct Training	5	7.40	8.96	16.36	163.4

sive curriculum not only reaches the prescribed objectives but also enables more efficient exploration of bonus rewards. Performance drops at curriculum transitions reflect distribution shift: policies specialized to one curriculum must re-explore when difficulty increases, yet prior experience accelerates re-convergence, consistent with the monotone advancement prescribed by EvoCurr.

5 Cross-Domain Generality

Beyond StarCraft II, we evaluate EvoCurr across multi-agent pathfinding, operations research, closed-loop MARL, and code generation. We posit that autonomous curricula provide maximal leverage when a domain exposes a meaningful difficulty axis, executable feedback, and reusable intermediate solutions. Flatland and POGEMA illustrate this directly: despite distinct failure modes—direction-aware rail transitions in Flatland versus partially observable coordination in POGEMA—both benefit from evolutionary progression.

Flatland and POGEMA illustrate this most directly. Both are pathfinding domains with different failure modes: Flatland requires direction-aware rail transitions and train reservations, while POGEMA requires coordination under partial observations. In both cases, direct generation and fixed-target Reflexion remain at zero, whereas EvoCurr reaches non-trivial final-task performance across three seeds. The gain therefore comes not from additional iterations on the same prompt, but from changing the task distribution so that executable intermediate policies can be reused.

The other positive domains show that the mechanism is not tied to SC2 code-as-policy. In Overcooked, the accepted-floor curriculum improves total deliveries for a closed-loop MARL trainer under matched budgets. In LiveCodeBench Hard, securing runnable scaffolds before full hidden-test evaluation raises Pass@1 from 8/20 to 11/20.

However, boundary cases clarify EvoCurr's scope: it requires room for an acceptance ladder. In pandapipes, direct diagnostic feedback suffices, and in JSSP, a simple SPT-like heuristic already saturates the metric. Thus, EvoCurr excels when direct generation fails on the final task but verifiable intermediate steps exist.

6 Analysis

Why the Designer matters. The central ablation asks whether Solver-side self-correction alone can replace the Designer. It cannot. Reflection-enabled variants of the same LLM backbones still collapse on many final SC2 tasks, while EvoCurr raises those backbones to 90–100% win rates across almost all scenarios. This shows that the key gain is not merely code repair or extra attempts, but the accepted-floor curriculum that lets the Solver accumulate tactical skills without regressing. Full ablation tables are provided in Appendix C and Appendix D.

Reliability beyond peak performance. Although the best direct policy competes on some SC2 maps, this selected-best view hides failed and non-executable attempts. Across attempts, Table 2 shows lower mean combat scores and incomplete executability across tasks. EvoCurr's advantage lies in reliably constructing executable, high-performing policies across tasks and backbones.

How policies evolve. Qualitatively, EvoCurr grows behavior-tree policies by preserving solved micro-skills and adding new unit-specific control functions as the curriculum becomes harder. Early scripts learn basic focus fire; later scripts add Ghost, Viking, Siege Tank, Raven, Medivac, and terrain-specific control, while refining existing functions for area-of-effect avoidance and target prioritization. Detailed generated code examples, curriculum trajectories, and layered-critic analysis are deferred to Appendix C and Appendix F.

Table 4: Cross-domain target-task results (higher is better). Direct is one-shot generation, except direct E3T training for Overcooked; fixed-target iteration is Reflexion-style self-correction. SC2 shows aggregate ranges, Overcooked shows Map 1 / Map 2 deliveries, and Flatland/POGEMA show saved-policy metrics rather than acceptance rates. ISR = individual success rate; Pass@1 = one final submission per problem; SPT = shortest processing time; gen./iter. = generations/iterations.

Domain	**Final-task structure**	**Metric**	**Direct**	**Fixed-target iteration**	**EvoCurr**
StarCraft II	Heterogeneous fixed-army micro with terrain effects and 10+ active abilities	win rate	42.5–70.0%	23.3–62.5%	**82.5–94.2%**
Flatland	Direction-conditioned rail routing with multi-train reservations	completion	0.000	0.000 (20 gen.)	**0.710 ± 0.195**
POGEMA	Partially observable multi-agent pathfinding with time-indexed reservations	ISR	0.000	0.000 (20 gen.)	**0.693 ± 0.108**
Overcooked	Closed-loop coordination across staged layouts and order schedules	deliveries	24.11 / 16.36	–	**29.10 / 18.64**
LiveCodeBench Hard	Executable program synthesis under contest test constraints	Pass@1	8/20	8/20 (3 iter.)	**11/20**
pandapipes	Low-dimensional gas-network control with direct diagnostic feedback	feasibility	0.00	**1.00** (6 iter.)	0.33
JSSP	Dispatching benchmark where SPT already reaches the comparison ceiling	vs. SPT	1.00	1.00 → 0.91	1.00

7 Discussion, Future Work, and Conclusion

EvoCurr demonstrates that self-evolving curricula can extend frozen LLMs from local task knowledge to reliable executable policies without a hand-written curriculum sequence. The strongest narrative evidence comes from fixed-army StarCraft II super-late-game micro-management: EvoCurr solves a difficult middle ground between SMAC-scale skirmishes and AlphaStar-scale full-game training, while direct generation and autonomous coding agents fail. The additional experiments show that this is not merely a StarCraft II artifact. The same accepted-floor curriculum mechanism improves Flatland, POGEMA, Overcooked, and LiveCodeBench Hard, while pandapipes and JSSP mark cases where curriculum is unnecessary or the domain ceiling is already saturated.

Future Work Future directions include hierarchical Designer–Solver architectures for finer task decomposition, adaptive difficulty scaling based on acceptance patterns and solver feedback, and hybrid methods that distill interpretable behavior-tree policies into more reactive neural policies when closed-loop robustness is required.

Conclusion Overall, EvoCurr offers a general inference-time curriculum mechanism for structured executable reasoning. StarCraft II provides the central story and hardest stress test; the added OR, MARL, and NLP-code experiments show that the same mechanism transfers beyond games while exposing its limits.

Limitations Primary constraints include sensitivity to difficulty pacing, which may stall progress if the Designer proposes steps that are too small or too large, LLM context-window limits that restrict behavior-tree complexity in long-horizon scenarios, and cumulative overhead from maintaining historical trajectories $\mathcal{H}_t$. The boundary controls also clarify scope: EvoCurr is most useful when direct LLM generation cannot self-bootstrap and the solution requires structured multi-step reasoning; it is less useful when Reflexion already solves the task or when a simple heuristic saturates the benchmark.

Acknowledgments

We thank the anonymous reviewers and the Area Chair for constructive feedback. Their comments improved the clarity of the presentation and experimental discussion. This work was supported by the CCF-NetEase ThunderFire Innovation Research Funding (NO.202507).

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. 2020. On the utility of learning about humans for human-ai coordination. *Preprint*, arXiv:1910.05789.

Michele Colledanchise and Petter Ögren. 2018. *Behavior trees in robotics and AI: An introduction*. CRC Press.

Yue Deng, Weiyu Ma, Yuxin Fan, Ruyi Song, Yin Zhang, Haifeng Zhang, and Jian Zhao. 2025. Smac-r1: The emergence of intelligence in decision-making tasks. *Preprint*, arXiv:2410.16024.

Yue Deng, Yan Yu, Weiyu Ma, Zirui Wang, Wenhui Zhu, Jian Zhao, and Yin Zhang. 2024. Smac-hard: Enabling mixed opponent strategy script and self-play on smac. *Preprint*, arXiv:2412.17707.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. 2020. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.

Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander Hauptmann. 2015. Self-paced curriculum learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. 2021. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR.

Pascal Klink, Carlo D'Eramo, Jan R Peters, and Joni Pajarinen. 2020. Self-paced deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:9216–9227.

M Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23.

Jaejin Lee, Seho Kee, Mani Janakiram, and George Runger. 2024. Attention-based reinforcement learning for combinatorial optimization: Application to job shop scheduling problem. *arXiv preprint arXiv:2401.16580*.

Zongyuan Li, Yanan Ni, Runnan Qi, Chang Lu, Lumin Jiang, Xu Xiaojie, Xiangbei Liu, Pengfei Li, Yunzheng Guo, Zhe Ma, and 1 others. 2026. Llm-pysc2: Starcraft ii learning environment for large language models. *Advances in Neural Information Processing Systems*, 38:127595–127657.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE.

Daniel Lohmeier, Dennis Cronbach, Simon Ruben Drauz, Martin Braun, and Tanja Manuela Kneiske. 2020. Pandapipes: An open-source piping grid calculation package for multi-energy grid simulations. *Sustainability*, 12(23):9899.

Artem Lykov and Dzmitry Tsetserukou. 2024. Llm-brain: Ai-driven fast generation of robot behaviour tree based on large language model. In *2024 2nd international conference on foundation and large language models (FLLM)*, pages 392–397. IEEE.

Weiyu Ma, Yuqian Fu, Zecheng Zhang, Bernard Ghanem, and Guohao Li. 2025a. Ava: Attentive vlm agent for mastering starcraft ii. *Preprint*, arXiv:2503.05383.

Weiyu Ma, Jiwen Jiang, Haobo Fu, and Haifeng Zhang. 2025b. Tacticcraft: Natural language-driven tactical adaptation for starcraft ii. *Preprint*, arXiv:2507.15618.

Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Yuqiao Wu, Runji Lin, Haifeng Zhang, and Jun Wang. 2024. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. *Preprint*, arXiv:2312.11865.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. 2019. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740.

Sharada Mohanty, Erik Nygren, Florian Laurent, Manuel Schneider, Christian Scheller, Nilabha Bhattacharya, Jeremy Watson, Adrian Egli, Christian Eichenberger, Christian Baumberger, and 1 others. 2020. Flatland-rl: Multi-agent reinforcement learning on trains. *arXiv preprint arXiv:2012.05893*.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. 2022. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR.

Mikayel Samvelyan, Sharath C Raparthy, Andrei Lupu, Eric Hambro, Aram H Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, and 1 others. 2024. Rainbow teaming: Open-ended generation of diverse adversarial prompts. *Advances in Neural Information Processing Systems*, 37:69747–69786.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Alexey Skrynnik, Anton Andreychuk, Anatolii Borzilov, Alexander Chernyavskiy, Konstantin Yakovlev, and Aleksandr Panov. 2025. Pogema: A benchmark platform for cooperative multi-agent pathfinding. In *International Conference on Learning Representations*, volume 2025, pages 5576–5606.

Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. 2022. Curriculum learning: A survey. *International Journal of Computer Vision*, 130(6):1526–1565.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, and 1 others. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. 2019. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*.

Xihuai Wang, Shao Zhang, Wenhao Zhang, Wentao Dong, Jingxiao Chen, Ying Wen, and Weinan Zhang. 2024. Zsc-eval: An evaluation toolkit and benchmark for multi-agent zero-shot coordination. *Preprint*, arXiv:2310.05208.

Xin Wang, Yudong Chen, and Wenwu Zhu. 2021. A survey on curriculum learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):4555–4576.

Xue Yan, Jiaxian Guo, Xingzhou Lou, Jun Wang, Haifeng Zhang, and Yali Du. 2023. An efficient end-to-end training approach for zero-shot human-ai coordination. *Advances in neural information processing systems*, 36:2636–2658.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. 2024. Envgen: Generating and adapting environments via llms for training embodied agents. *arXiv preprint arXiv:2403.12014*.

A StarCraft II Mini-Game Maps

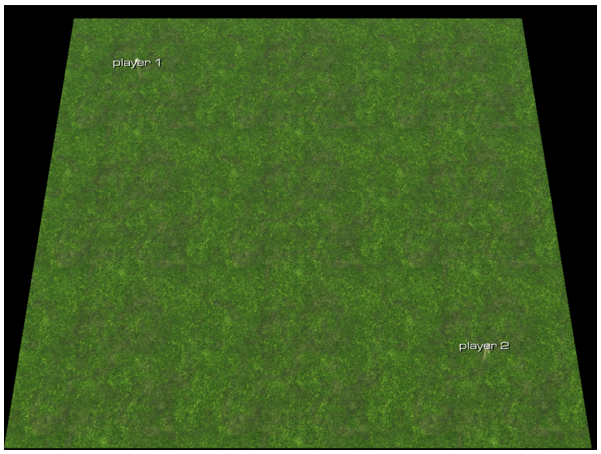


Figure 3: Flat map layout

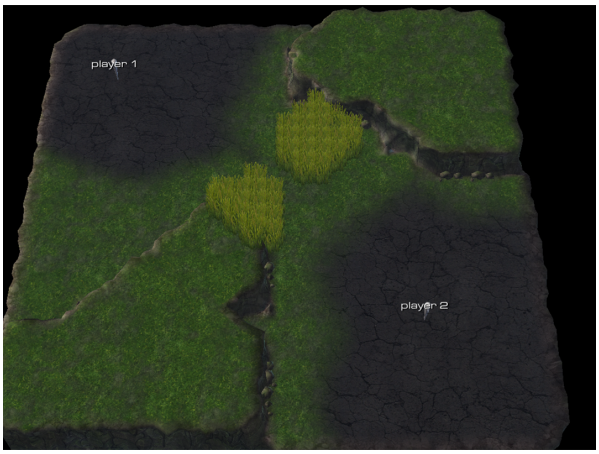


Figure 4: Bush map layout

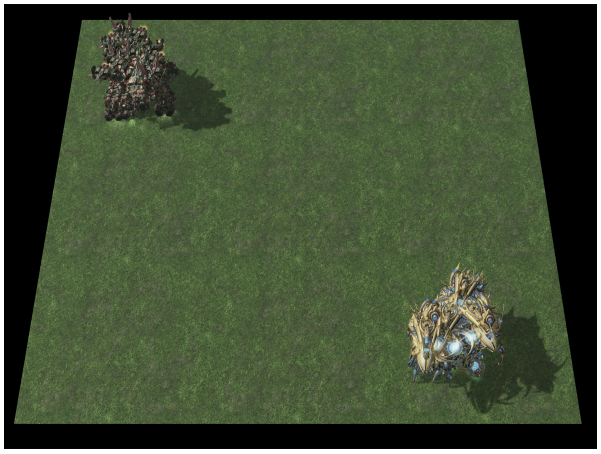


Figure 5: Terran vs. Protoss on Flat

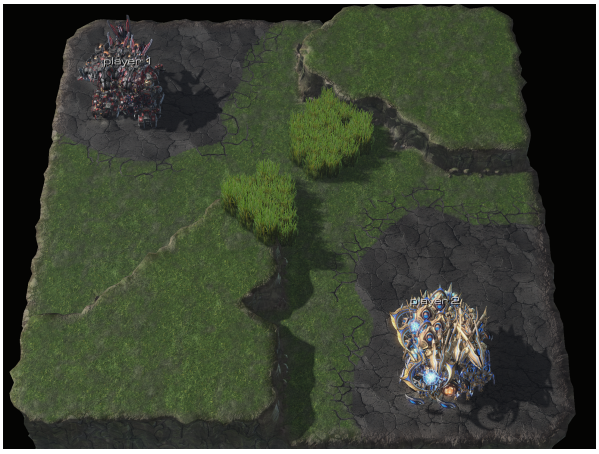


Figure 6: Terran vs. Protoss on Bush

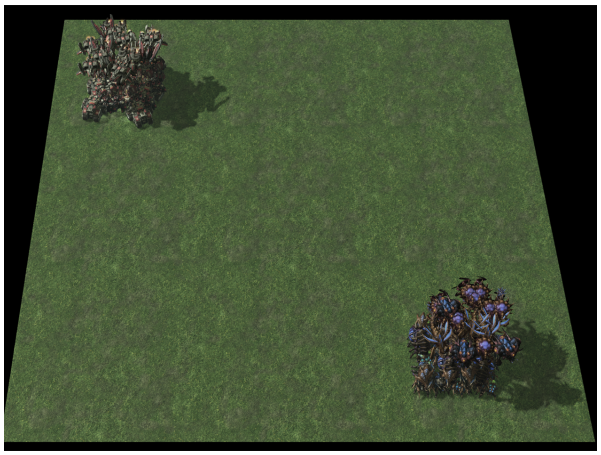


Figure 7: Terran vs. Zerg on Flat

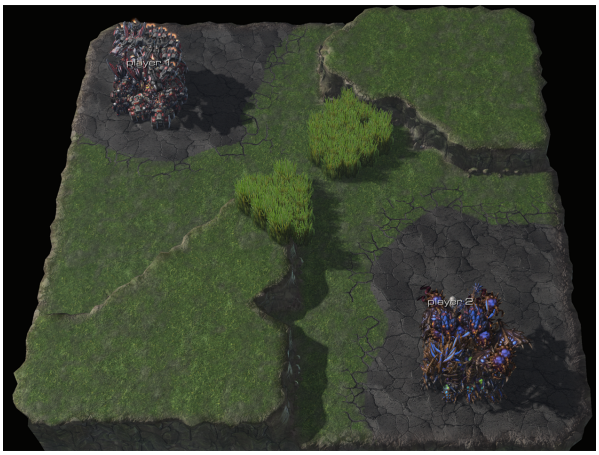


Figure 8: Terran vs. Zerg on Bush

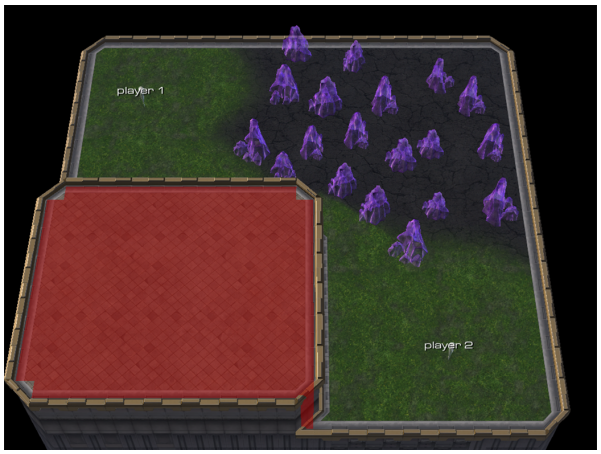


Figure 9: Corner map layout



Figure 10: Main map layout



Figure 11: Terran vs. Protoss on Corner



Figure 12: Terran vs. Protoss on Main



Figure 13: Terran vs. Zerg on Corner



Figure 14: Terran vs. Zerg on Main

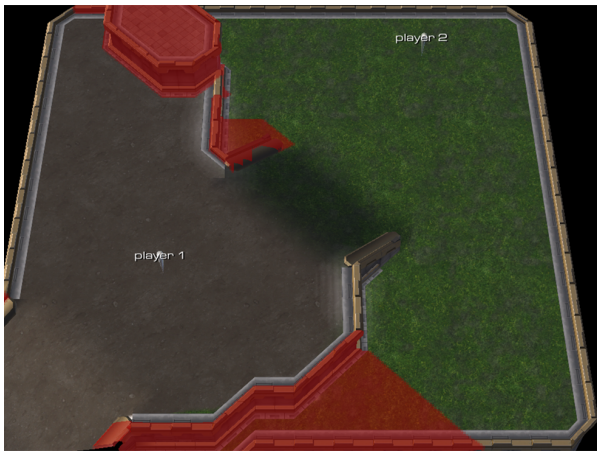


Figure 15: Ramp map layout

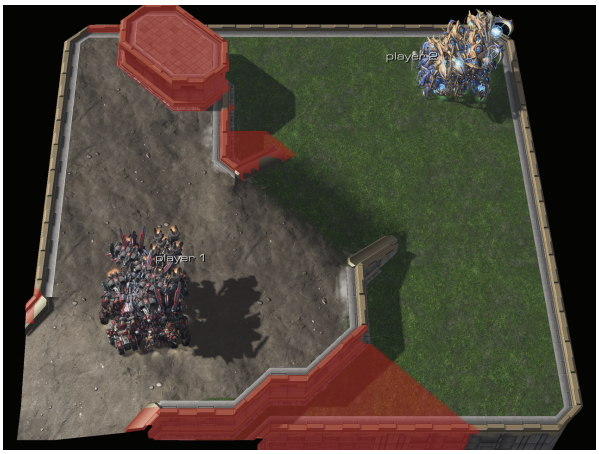


Figure 16: Terran vs. Protoss on Ramp

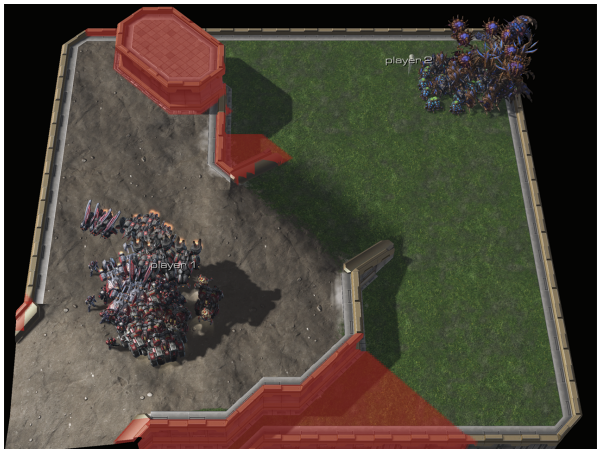


Figure 17: Terran vs. Zerg on Ramp

B Additional StarCraft II Analyses

This appendix provides robustness checks, scope comparisons, and compute accounting for the StarCraft II experiments.

Scope and positioning: a super-late-game micro mini-game. We emphasise upfront that EvoCurr solves a deliberately scoped *mini-game*: a fixed-army, super-late-game micro-management engagement with no economy, no build order, and no scouting. Both sides spawn with a full late-game tech tree: our agent commands 80+ Terran units across 10+ unit types (Marine, Marauder, Medivac, Ghost, Siege Tank, Viking, Cyclone, Widow Mine, Raven, Liberator) with all 10+ active abilities researched, and faces an equally late-game Protoss or Zerg opposition (Colossus, High Templar, Disruptor, Carrier, or Brood Lord, Lurker, Viper, Infestor, etc.) on a single tactical map. The single decision the agent must make is *how to fight*, not *what to build*.

This narrow scope is what makes the comparison in Table 5 both fair and informative. SMAC-style benchmarks (Samvelyan et al., 2019) target a similar regime—fixed armies, no economy—but only at very small scale (3–30 homogeneous units, 0–2 active abilities). AlphaStar (Vinyals et al., 2019) solves the full game including economy and macro, at the cost of approximately 200,000 TPU-hours of training. EvoCurr sits between them: it tackles the late-game micro complexity that SMAC scenarios omit, but does so without leaving the mini-game frame and without any parameter training. With roughly 1–3 hours of inference-time LLM calls and $1.5 in API cost, the system produces behavior-tree code that coordinates Stimpack, Siege Mode, Personal Cloaking, EMP, Snipe, AoE avoidance and Medivac healing in a single engagement, while exploiting map terrain (Bush concealment, Ramp choke-points, Corridor funnels) explicitly via readable Python.

We make two explicit non-claims. First, EvoCurr does *not* play full StarCraft II: there is no build order, no scouting, no production decision in our scenario. Second, we do *not* claim parity with AlphaStar on full-game generality; AlphaStar operates on raw camera and spatial actions while we use python-sc2's unit-level command API, so our result speaks to micro proficiency on this mini-game alone. What the comparison does establish is that, on the super-late-game micro mini-game that classical RL micro-benchmarks have underrepresented, an inference-time curriculum over a frozen LLM can produce readable policies with substantial multi-ability tactical coordination.

Compute budget analysis. A natural concern is whether EvoCurr's gains stem simply from spending more compute than the baselines. Table 6 disaggregates this. Per final task, EvoCurr-DeepSeek consumes roughly $0.2 of LLM API calls and 1–3 hours of wall-clock; the GPT-5 / Claude-4 backbones cost about $1.5 each. This is, in absolute terms, an order of magnitude more than direct one-shot $\times 10$ sampling ($0.02–$0.30 depending on backbone), but is comparable to or cheaper than the autonomous-agent baselines we contrast against: Claude Code consumes a similar 90-minute wall-clock budget and still achieves 0% on the final task (Appendix I), while Codex / Cursor at single-shot pricing also fail.

The decisive evidence that EvoCurr's advantage is mechanism rather than budget comes from our cross-domain ablations (Appendix I and the Flatland / POGEMA equal-budget experiments in Section 5): extending Reflexion to 20 iterations—roughly $4\times$ the per-attempt budget of Self-reflection and within an order of magnitude of EvoCurr's total spend—still yields 0% across 40 / 40 generations on the multi-agent pathfinding domains where direct LLM code generation fundamentally fails. The gap is therefore not closed by additional inference-time compute in the absence of a curriculum: EvoCurr's accepted-floor mechanism enables incremental progress through cheap simple stages, whereas budget-matched baselines repeatedly burn compute on the hardest task and never converge.

On compute-matched iterative baselines. A natural concern is whether EvoCurr's gain over direct generation reflects the curriculum mechanism itself or simply additional inference-time compute. We address this with three complementary lines of evidence.

(i) Reflection-enabled backbones at the same SC2 scale. Table 16 reports reflection-enabled variants of all four LLM backbones evaluated in Section 4.1. Reflection yields only marginal gains over the one-shot baseline and remains far below EvoCurr on the same final tasks; the curriculum gap is robust to this iteration scaling.

(ii) An autonomous Reflexion-style agent at $\sim 3\times$ the budget on the same SC2 task. Appendix I reports **Claude Code**, a state-of-the-art au-

Table 5: Positioning EvoCurr against SMAC-style closed-loop RL micro benchmarks and AlphaStar (Vinyals et al., 2019) full-game RL+imitation. **Scope:** EvoCurr targets a single super-late-game micro *mini-game*—fixed armies, full tech tree, no economy, no build order, no scouting—i.e., it is *not* a full-game agent. SMAC sits in the same mini-game frame but at very small scale (3–30 homogeneous units, 0–2 abilities); AlphaStar solves the full game at industrial training cost. Numbers for AlphaStar are from (Vinyals et al., 2019); SMAC characteristics aggregate canonical scenarios (3m, 8m_vs_9m, MMM2, 27m_vs_30m). * *Action-space caveat:* EvoCurr uses python-sc2 unit-level commands while AlphaStar uses raw camera + spatial actions; we therefore claim only that EvoCurr addresses only this fixed-army micro mini-game, not AlphaStar's full-game setting or action space.

Dimension	**SMAC (RL micro benchmarks)**	**AlphaStar**	**EvoCurr (ours)**
Scenario scale	3–30 units; 1–3 unit types	Full game (200+ units, all races)	**81 units, 10+ types, 10+ abilities**
Game phase	Isolated micro skirmish	Full game (build + macro + micro)	**Super-late-game micro mini-game**
Control paradigm	Closed-loop RL (per-step)	Closed-loop RL+imitation	**Open-loop code-as-policy**
Action space	Per-unit discrete (move/attack/...)	Raw camera + spatial actions	python-sc2 unit-action API*
Training compute	∼1–10 GPU-days	∼200,000 TPU-hours	**0 (no parameters trained)**
Inference cost	GPU forward pass	GPU forward pass	**≈1–3 hr LLM calls, ≈$1.5**
Solution interpretability	Low (neural policy)	Low (neural policy)	**High (readable Python behavior tree)**
Terrain utilisation	Minimal (open maps)	Strong (full map awareness)	**Strong (Bush concealment, Ramp choke, Corridor funnel explicit in code)**
Multi-ability coordination	0–2 abilities	50+ abilities	**Stimpack, SiegeMode, Cloak, EMP, Snipe, AoE-avoid, Heal, ... all in one engagement**
Reproducibility cost	Re-train each scenario	Train once at industrial scale	**Re-run inference, hours not months**

tonomous coding agent that implements an aggressive Reflexion-style execute-feedback-revise loop with full tool use, running for **90 minutes / 100+ simulated matches** against our final task. This budget exceeds EvoCurr's end-to-end LLM-call cost on every task in Section 4.1, yet Claude Code achieves a 0% win rate on the final task. The same agent's performance on its self-designed intermediate tasks is substantially higher (cf. Appendix I for the trajectory), confirming that the bottleneck is *curriculum structure* rather than per-iteration compute.

(iii) Direct 20-iteration Reflexion ablations in two mechanism-orthogonal cross-domain settings. On both Flatland (Section D.1) and POGEMA (Section D.2) we explicitly run Reflexion at a budget of **20 generations** (approximately $3\times$ and $1.6\times$ EvoCurr's training-time budget, respectively); both yield **0/20 generations with non-zero performance**. The POGEMA failure mode is particularly diagnostic: $17/20$ generations cannot even execute (e.g. `NameError: heapq` appears on 6 consecutive generations 12–19, indicating that Summarizer feedback fails to teach the model to add a single `import` statement over 20 rounds). On LiveCodeBench Hard (Section D.3) we further isolate

Table 6: Compute budget per single final SC2 task (e.g., Bush TvP). All EvoCurr and Claude Code numbers are as reported in Appendix J and Appendix I. Direct one-shot and Self-reflection rows are estimated from API pricing (DeepSeek-V3.1 input $0.27/M, output $1.10/M; GPT-5 input $1.25/M, output $10/M; Claude-4 Sonnet input $3/M, output $15/M; as of 2026-04) multiplied by the attempt counts already committed to in the paper (10 target-task attempts for direct, 10 attempts $\times$ one refinement round for self-reflection). **Win rate** is the average over the 12 final tasks of Table 1. AlphaStar (Vinyals et al., 2019) is included for reference only; it solves a different scope (full game vs. mini-game).

Method	**LLM calls**	**Wall-clock**	**Cost (USD)**	**Outcome**
Direct one-shot $\times 10$ (DeepSeek)	10	$\sim$10 min	$\approx$\$0.02 (est.)	55.0% macro
Direct one-shot $\times 10$ (GPT-5)	10	$\sim$10 min	$\approx$\$0.15 (est.)	44.2% macro
Direct one-shot $\times 10$ (Claude-4)	10	$\sim$10 min	$\approx$\$0.30 (est.)	70.8% macro
Self-reflection (1 round $\times 10$, DeepSeek)	20	$\sim$20 min	$\approx$\$0.04 (est.)	below EvoCurr
Self-reflection (1 round $\times 10$, GPT-5)	20	$\sim$20 min	$\approx$\$0.30 (est.)	below EvoCurr
Self-reflection (1 round $\times 10$, Claude-4)	20	$\sim$20 min	$\approx$\$0.60 (est.)	below EvoCurr
Claude Code (autonomous coding agent)	100+ matches	$\sim$90 min	comparable to EvoCurr	**0%**
Codex / Cursor (single-attempt agents)	1–10	$<$30 min	\$0.5–\$2 (est.)	0–40%
EvoCurr-DeepSeek (ours)	4–6 stages	1–3 hr	$\approx$**\$0.2**	**99.2% macro**
EvoCurr-Claude-4 (ours)	4–6 stages	1–3 hr	$\approx$**\$1.5**	**96.7% macro**
EvoCurr-GPT-5 (ours)	4–6 stages	1–3 hr	$\approx$**\$1.5**	**97.5% macro**
AlphaStar (full game, reference)	—	$\approx$200,000 TPU-hours	not applicable	full game (different scope)

this effect at the algorithmic-code-generation level: Reflexion-3 finishes at exactly the same Pass@1 as Direct one-shot (40%) while spending 2.5$\times$ the cost, and on one problem (`abc388_e`) the Summarizer's feedback *regresses* a 100%-correct solution to 0%, the exact failure mode (Summarizer drift) that the accepted-floor mechanism is designed to prevent.

Taken together, these three lines establish that scaling iterative inference compute on the SC2 final task, whether via plain reflection (Table 16), via a state-of-the-art autonomous agent (Appendix I), or via 20-iteration Reflexion in two structurally distinct domains and one NLP code-generation

domain (Section 5), does not close the gap. The curriculum mechanism, specifically Designer-driven progressive difficulty, accepted-floor anti-regression, and feasibility checking, contributes the gain, not the inference budget.

Aggregate 120-Game Audit Table 7 retains the separate aggregate audit used for the headline SC2 ranges. It is reported alongside, rather than used to overwrite, the reproducible task-level results in Table 1.

Table 7: Separate aggregate final-policy audit over the 12 StarCraft II micro tasks (10 evaluation games per task; 120 games per backbone). CI denotes confidence interval; all reported CIs are Wilson 95% intervals.

Method	Backbone	Wins	Win rate	Wilson 95% CI
EvoCurr	Claude-4	108/120	90.0%	83.3–94.2%
EvoCurr	DeepSeek-V3.1	99/120	82.5%	74.7–88.3%
EvoCurr	GPT-5	113/120	94.2%	88.4–97.1%
Direct-10	Claude-4	84/120	70.0%	61.3–77.5%
Direct-10	DeepSeek-V3.1	83/120	69.2%	60.4–76.7%
Direct-10	GPT-5	51/120	42.5%	34.0–51.4%
Direct-10	Gemini-2.5	56/120	46.7%	38.0–55.6%
Reflexion-20	Claude-4	75/120	62.5%	53.6–70.6%
Reflexion-20	DeepSeek-V3.1	41/120	34.2%	26.3–43.0%
Reflexion-20	GPT-5	30/120	25.0%	18.1–33.4%
Reflexion-20	Gemini-2.5	28/120	23.3%	16.7–31.7%
Code agent	Claude Code	31/120	25.8%	18.8–34.3%
Code agent	Codex	16/120	13.3%	8.4–20.6%
Code agent	Cursor	0/120	0.0%	0.0–3.1%

Actual Search Budgets Direct-10 generates ten independent policies per task and backbone. Reflexion-20 makes 20 policy-generation or revision calls and 19 reflection calls. EvoCurr counts all policy/Solver and critic/reflection calls up to the first accepted final-task checkpoint. A curriculum stage is not treated as an LLM call.

Table 8: Mean LLM calls per SC2 task. Search statistics are truncated at the first accepted final-task checkpoint.

Method	Policy/Solver	Critic/Reflection	Total
Direct-10	10.00	0.00	10.00
Reflexion-20	20.00	19.00	39.00
EvoCurr Claude-4	8.42	12.67	21.09
EvoCurr DeepSeek	8.25	17.83	26.08
EvoCurr GPT-5	9.00	12.00	21.00

The audited budgets show that EvoCurr uses 21–26 directly comparable LLM calls per task, between Direct-10's 10 calls and Reflexion-20's 39 calls. Accordingly, we compare methods under their measured budgets rather than describing them as exactly budget matched.

Paired Statistical Evidence We use the 36 common task–backbone configurations as paired statistical units. The primary analysis is a two-sided Wilcoxon signed-rank test with Holm correction over the three pairwise comparisons. We additionally report exact McNemar tests over matched rollout seeds. The two tests answer different questions; the configuration-level Wilcoxon test is primary.

Exact McNemar tests with Holm correction yield $p < 10^{-18}$ for EvoCurr vs Direct-10, $p < 10^{-40}$ for EvoCurr vs Reflexion-20, and $p < 10^{-8}$ for Direct-10 vs Reflexion-20. The first two comparisons agree across tests; the Direct–Reflexion comparison is not significant under the primary configuration-level Wilcoxon analysis.

Candidate Rejection and Curriculum Stages A rollout-evaluated candidate is a generated policy that passes formatting, compilation, and feasibility checks and reaches SC2 rollout evaluation. It is rejected if it fails at runtime or does not satisfy the stage-acceptance rule. Policies rejected before rollout evaluation are not included. Counts stop at the first accepted final-task checkpoint.

All 36 completed task–backbone trajectories eventually reach an accepted final checkpoint, giving a terminal trajectory failure rate of 0/36. The attempted Gemini-2.5 EvoCurr runs are excluded from this completed-backbone denominator and reported separately below.

Independent Claude-4 Curriculum Trajectories For each of the 12 SC2 map–matchup configurations, we run three independent Claude-4 curriculum trajectories and evaluate each final policy over 50 games. Across 36 independently generated policies and 1,800 games, EvoCurr obtains 1,630/1,800 wins (90.6%). The aggregate Wilson 95% interval is 89.1–91.8%. Because games are clustered within policies and tasks, this interval is a descriptive summary rather than a complete estimate of between- task and between-trajectory uncertainty. This repeated-trajectory analysis is limited to Claude-4.

Across these trajectories, the mean number of stages to the first accepted final task is 5.17 ± 0.70, with a range of 4–7.

Bush–TvP Expert-Player Pilot We evaluate the Bush–TvP final task with one professional player and three players who reported reaching Grandmaster on the Korean server. All participants control Terran regardless of their primary competitive race. The fixed army is given in Table 22. Bush–TvP was selected because it required the longest curriculum

Table 9: Corrected SC2 environment interactions. Search interactions stop at the first accepted final-task checkpoint. The standard 10-game final-policy evaluation is reported separately; the additional 50-game repeated- trajectory evaluations are excluded. Search/task and Total/task are means over the 12 tasks; ± reports the corresponding standard deviation where available.

Method	Backbone	Search/task	Final eval./task	Total/task	Total/12 tasks
EvoCurr	Claude-4	53.33	10	63.33 ± 6.50	760
EvoCurr	DeepSeek	59.17	10	69.17 ± 12.40	830
EvoCurr	GPT-5	53.33	10	63.33 ± 6.50	760
Direct-10	Each backbone	100	10	110.00	1,320
Reflexion-20	Each backbone	200	10	210.00	2,520
Code agent	Each backbone	100	10	110.00	1,320

Table 10: Paired comparisons over the 36 common SC2 task–backbone configurations. Differences are percentage points.

Comparison	Configurations	Mean rates	Difference	Adjusted p-value
EvoCurr vs Direct-10	36	88.9% vs 60.6%	+28.3	0.0039
EvoCurr vs Reflexion-20	36	88.9% vs 40.6%	+48.3	0.000034
Direct-10 vs Reflexion-20	36	60.6% vs 40.6%	+20.0	0.0678

Table 11: Rollout-evaluated candidate rejection during SC2 curriculum search.

Backbone	Evaluated	Rejected	Rate (Wilson 95% CI)
Claude-4	64	5	7.8% (3.4–17.0%)
DeepSeek	71	7	9.9% (4.9–19.0%)
GPT-5	64	3	4.7% (1.6–12.9%)
Overall	199	15	7.5% (4.6–12.1%)

Table 12: Stages to the first accepted SC2 final task in the original task–backbone evaluation. Here s.d. denotes standard deviation.

Backbone	Mean ± s.d.	Range
Claude-4	4.92 ± 0.29	4–5
DeepSeek	5.33 ± 0.89	4–7
GPT-5	5.08 ± 0.51	4–6

trajectory among the 12 SC2 configurations.

All games use the StarCraft II "Faster" speed and maps implemented with the StarCraft II Map Editor. Human participants use the standard graphical interface, mouse, keyboard, and their usual hotkeys and display settings; the agent uses the programmatic interface. Unit selection, movement, attacks, target selection, and command execution follow the standard game rules. Matches run locally with default local-game latency and no added delay or latency compensation. Fog of war is disabled for both sides. Thus, game-level conditions, mechanics, and information availability are matched, but the comparison is not paired at the low-level input modality. Each participant is compensated 50 Chinese yuan (RMB).

The pilot indicates that Bush–TvP remains challenging for strong players and supports qualitative comparison with human-recognizable tactics such as formation dispersion and priority targeting. It does not establish general human-level or superhuman performance.

Attempted Gemini-2.5 EvoCurr Runs Gemini-2.5 was attempted as an EvoCurr backbone, but no run completed a valid curriculum trajectory. Saved logs show hallucinated or incompatible `python-sc2` calls (e.g., `of_type` on a Python set and nonexistent methods such as `Point2.randomize`, `Units.in_range`, `Units.ground`, `BattleBot.do_actions`, and `Unit.ability_is_ready`); malformed identifiers; residual Markdown fences; syntax and type errors; corrupted Unicode feedback; and outputs without a valid `BattleBot.on_step`. Candidates repeatedly failed initial executable- policy validation, so no final EvoCurr policy was available for evaluation. This incomplete backbone is distinct from the completed Gemini Direct-10 and Reflexion-20 baselines in Table 1.

Preliminary SC2 Accepted-Checkpoint Ablation We retain easy-to-hard task generation, iterative policy refinement, execution feedback, and stage evaluation but remove restoration of the policy copy associated with the latest accepted floor. Five intermediate stages each achieve 5/5 wins, yet the final attempt obtains 0 wins, 1 draw, and 9 losses. The complete search and debugging pro-

Table 13: Three independent Claude-4 trajectories per SC2 configuration. Stage triples give stages to the first accepted final task; win triples give wins in the corresponding 50-game evaluations.

Configuration	Stages	Wins/50	Mean wins/50	Mean rate ± s.d.
TvP Bush	5, 7, 6	44, 42, 45	43.67	87.3% ± 3.1%
TvP Corner	5, 6, 4	45, 47, 45	45.67	91.3% ± 2.3%
TvP Corridor	5, 5, 5	46, 48, 46	46.67	93.3% ± 2.3%
TvP Flat	5, 6, 5	41, 43, 41	41.67	83.3% ± 2.3%
TvP Main	5, 7, 5	48, 48, 47	47.67	95.3% ± 1.2%
TvP Ramp	5, 4, 5	41, 46, 46	44.33	88.7% ± 5.8%
TvZ Bush	5, 4, 5	43, 44, 42	43.00	86.0% ± 2.0%
TvZ Corner	4, 5, 6	46, 48, 46	46.67	93.3% ± 2.3%
TvZ Corridor	5, 5, 5	48, 48, 49	48.33	96.7% ± 1.2%
TvZ Flat	5, 5, 5	44, 41, 42	42.33	84.7% ± 3.1%
TvZ Main	5, 5, 6	47, 45, 48	46.67	93.3% ± 3.1%
TvZ Ramp	5, 6, 5	47, 46, 47	46.67	93.3% ± 1.2%

Table 14: Expert-player pilot on Bush–TvP. Peak matchmaking rating (MMR) is self-reported.

ID	Primary race	Peak MMR	Wins/Games	Wilson 95% CI
P1 (professional)	Protoss	6000	6/10 (60%)	31.3–83.2%
P2 (Grandmaster)	Zerg	≈5500	2/10 (20%)	5.7–51.0%
P3 (Grandmaster)	Zerg	≈5500	3/10 (30%)	10.8–60.3%
P4 (Grandmaster)	Terran	≈5500	5/10 (50%)	23.7–76.3%
Aggregate	–	–	16/40 (40%)	26.3–55.4%

cess lasts approximately 90 minutes and uses more than 100 SC2 games.

This is a single-run preliminary ablation. Its final army uses 8 Cyclones and 4 Liberators, whereas the Bush–TvP pilot in Table 22 uses 7 Cyclones and 2 Liberators; other opposing-unit counts also differ. Accordingly, we do not present it as a strictly matched ablation of the pilot configuration. It provides qualitative evidence that later policy modifications can overwrite successful behavior when checkpoint restoration is removed, while the configuration mismatch and final-stage difficulty gap limit causal interpretation.

SC2 Acceptance-Threshold Sensitivity Because changing ρ can alter subsequent curriculum decisions, we run separate end-to-end searches rather than post-hoc re-scoring a fixed trajectory. Each stage uses ten games and the same final task and search budget.

Table 15: Independent end-to-end SC2 runs under different pass fractions.

ρ	Required wins/stage	Accepted stages	Final reached
0.50	at least 5/10	6	Yes
0.67	at least 7/10	5	Yes
1.00	10/10	6	Yes

All three runs reach an accepted final task. The stage counts need not vary monotonically because these are independent stochastic end-to-end searches.

C Designer Ablation and Code-Evolution Details

Component ablations. We separate adaptive task generation from rollback preservation. Fixed Curriculum uses an informed deterministic progression while retaining the Solver, accepted floor, and its rollback policy copy. The w/o Accepted Floor variant retains adaptive task generation but does not restore the latest verified task and policy after task-level failure. Each cross-domain cell below summarizes three independent DeepSeek trajectories, with each final policy evaluated on the same 10 seeds; parentheses give accepted trajectories.

The fixed curriculum is specified before evaluation and is not copied from an EvoCurr trajectory. Flatland first increases agents from 2 to 4 on a 30×30 grid with two cities, then grows the grid to 35×35, and finally increases cities from 2 to 3. POGEMA increases agents from 2 to 8 on an 8×8 grid, increases the horizon from 32 to 128, grows the grid to 20×20, and finally raises obstacle density from 0.15 to 0.25. Flatland keeps one train speed and no malfunctions; POGEMA keeps observation radius 3. The fixed baseline uses the same final objective, Solver prompt, evaluator, accepted-floor and rollback preservation, and per-stage Solver budget as EvoCurr, but makes no Designer calls and ignores Solver feedback when selecting tasks.

The informed fixed schedule matches or exceeds EvoCurr in both structured pathfinding domains,

Table 16: Controlled component ablations. Means and standard deviations are over independent curriculum trajectories, not environment seeds for one saved policy.

Domain	EvoCurr	Fixed Curriculum	w/o Accepted Floor
Flatland	0.792 ± 0.177 (2/3)	$\mathbf{0.825 \pm 0.125}$ **(2/3)**	0.617 ± 0.535 (2/3)
POGEMA	0.842 ± 0.137 (1/3)	$\mathbf{0.992 \pm 0.007}$ **(3/3)**	0.763 ± 0.000 (0/3)

whereas removing accepted-floor rollback lowers mean performance and causes a catastrophic Flatland failure. In SC2, EvoCurr's 88.9% mean across 36 configurations exceeds Reflexion-20's 40.6%, but this comparison combines adaptive task progression and checkpoint preservation. The single-run SC2 checkpoint ablation in Appendix B uses a different final configuration and is therefore only preliminary. The evidence supports a domain-dependent role for adaptive design and a clearer anti-regression role for checkpoint preservation; it does not establish that an adaptive Designer is universally necessary.

Behavior code generation. When facing a new curriculum with more units or new unit types, the behavior coder generates a new script by extending the script that solved the previous curriculum. The first refinement phase adds new unit-specific control functions, such as `control_vikings` and `control_ghosts`; the second phase refines existing control functions to improve coordination among units. For example, Marines initially learn focus fire, and later learn to split against area-of-effect damage before resuming focus fire.

For further insights into the specific curriculum trajectories, see Appendix F. Table 30 presents the step-by-step sub-task generation across all 12 scenarios, while Table 31 gives a representative full path showing how the framework advances through successful nodes and backs off from failed proposals.

Layered critic refinement. Although LLMs learn extensive coding resources during pre-training, their ability to generate scripts for the `python-sc2` package depends on the amount and quality of community-written examples they have seen. We therefore use a two-layer critic. The first layer is a sanity-check module that corrects grammar bugs, API misuse, and runtime exceptions. The second layer refines strategy-level logic in the behavior-tree script. This two-layer critic improves the rate of successful behavior-tree generation.

D Cross-Domain Experiment Details

This appendix gives the experimental details behind Table 4. The main text uses these results only to support the method-level generality claim; the implementation settings, failure modes, and auxiliary figures are kept here to avoid interrupting the SC2-led narrative.

D.1 Flatland

Flatland (Mohanty et al., 2020) is a 2D-grid multi-train pathfinding benchmark. Our final task is a 35×35 grid with 3 cities and 4 trains under `single_speed` and `no_malfunctions`; a run is counted as successful when mean completion is at least 0.90. Direct LLM generation fails because the API requires direction-conditioned transitions through `env.rail.get_transitions(r,c,d)`, and valid routing must search over (cell, direction) rather than cells alone. Multi-train collision avoidance further requires priority-aware reservations.

Across three random seeds, EvoCurr reaches 0.710 ± 0.195 mean completion, while Direct and Reflexion-5 remain at 0.000 on all seeds. A more adversarial Reflexion-20 ablation on seed 42 also produces 0/20 generations with non-zero completion despite spending roughly $3\times$ EvoCurr's training budget.

D.2 POGEMA

POGEMA (Skrynnik et al., 2025) is a partially observable multi-agent pathfinding benchmark. Our final task is a 20×20 grid with 8 agents, obstacle density 0.25, and observation radius 3. The metric is individual success rate (ISR), with a success threshold of 0.90. Unlike Flatland, POGEMA requires time-indexed reservations and global belief accumulation from local 7×7 observations.

The submitted 0.693 ± 0.108 value likewise evaluates one saved policy on three environment seeds. In a ten-seed re-evaluation, the old checkpoint passes 2/10 rollouts with mean ISR 0.7625 and is not accepted. A new final- stage adaptation rerun passes the original 3/3 acceptance test with mean

ISR 1.0 and then passes 9/10 expanded trials with mean ISR 0.9875. Direct and Reflexion-6 remain at 0.000. In the Reflexion-20 check, 17/20 generations crash and the remaining 3/20 execute but deadlock at ISR$= 0$.

Table 17: Multi-seed verification on the two main supplementary domains. Direct one-shot and short-horizon Reflexion are both 0.000 across all 3 seeds; EvoCurr is the only non-zero method. Reflexion-20 numbers are seed-42 only.

Method	seed 42	seed 43	seed 44	mean $\pm$ s.d.
Flatland – completion				
hand_rule	0.17	0.17	0.00	0.113 ± 0.098
direct_oneshot	0.000	0.000	0.000	0.000
Reflexion-5	0.000	0.000	0.000	0.000
Reflexion-20	0.000	–	–	0.000
EvoCurr	**0.880**	**0.750**	**0.500**	$\mathbf{0.710 \pm 0.195}$
POGEMA – ISR				
hand_rule	0.04	–	0.08	0.060
direct_oneshot	0.000	–	–	0.000
Reflexion-6	0.000	0.000	0.000	0.000
Reflexion-20	0.000	–	–	0.000 (17/20 crash)
EvoCurr	**0.710**	**0.580**	**0.790**	$\mathbf{0.693 \pm 0.108}$

Unified acceptance rule. For both domains, $\tau = 0.90$ is the per-rollout metric threshold and $\rho = 0.67$ is the required pass fraction. Using $\lceil \rho n \rceil$, Flatland requires 2/2 passing rollouts and POGEMA requires 3/3. The submitted saved policies in Table 17 were not accepted; the subsequent adaptation reruns above generated new accepted policies rather than relabeling the old checkpoints.

Independent component trajectories. The controlled component study uses the same DeepSeek backbone and current domain-grounded Solver protocol for every variant. Flatland adds explicit time-expanded reservations, reverse-edge conflict handling, bounded replanning, and safe fallback behavior identically to all variants. Each cell summarizes three independent curriculum trajectories, with every final policy evaluated on the same ten seeds.

The independent-trajectory component results are reported once in Table 16; parentheses there give accepted trajectories out of three.

The informed Fixed Curriculum is deterministic, fixed before evaluation, and does not use Solver feedback at runtime. It uses the same final objective, Solver prompt, evaluator, accepted-checkpoint preservation, and per-stage Solver budget as EvoCurr, but makes no Designer calls.

Flatland keeps a single train speed and no malfunctions throughout; POGEMA keeps observation radius 3. The results show that a strong informed schedule can suffice in these structured domains, while checkpoint removal lowers mean performance and produces a catastrophic Flatland failure.

Post-hoc threshold sensitivity. For the following analysis, curricula, candidate policies, rollout seeds, and final policies are held fixed and acceptance labels are recomputed. This is post-hoc rescoring, not an end-to-end rerun in which changed early decisions could alter later curricula.

D.3 LiveCodeBench Hard

We evaluate 20 *Hard*-tier problems from LiveCodeBench version v6 (Jain et al., 2025), restricted to contest dates after 2024-08 to mitigate contamination relative to the DeepSeek-V4-Pro training cutoff. Direct one-shot makes one Coder call. Reflexion-3 uses up to three Coder + Summarizer rounds against the full test suite. EvoCurr uses the same backbone and three-round budget, but decomposes each problem into a three-stage constraint-relaxation curriculum: produce runnable code, pass at least one test case, then pass all test cases. The accepted-floor mechanism prevents later feedback from destroying an already-working stage.

EvoCurr solves 11/20 problems versus 8/20 for Direct and Reflexion-3. Iteration alone does not close the gap: Reflexion-3 spends $2.5\times$ the cost of Direct yet obtains the same Pass@1, gaining one problem while regressing on another (`abc388_e`: $100\% \rightarrow 0\%$). EvoCurr's gains concentrate on problems where the runnable-code warm-up stage creates a scaffold that direct generation never reaches.

D.4 Boundary Controls

pandapipes. On an 18-junction gas-network control benchmark, 6-iteration Reflexion already reaches 1.00 feasibility and 0.009 MW power consumption, outperforming EvoCurr's 0.33 feasibility. This is a deliberate negative control: the control space is low-dimensional, the feedback is direct, and PI-style code templates are within the LLM's one-shot reach.

JSSP. On Taillard 10×5 Job-Shop Scheduling, all three methods tie SPT (Shortest Processing Time) at score 1.00. The LLM's first attempt is already SPT-like, so the curriculum cannot exceed an already-saturated ceiling. Reflexion can even degrade the working solution, consistent with the Summarizer-drift failure mode observed in LiveCodeBench.

Table 18: Construction of the informed Fixed Curriculum baseline.

Domain	Stages	Fixed progression
Flatland	1–3	30×30, 2 cities; agents $2 \rightarrow 3 \rightarrow 4$
Flatland	4–5	4 agents, 2 cities; grid $30 \times 30 \rightarrow 35 \times 30 \rightarrow 35 \times 35$
Flatland	6	cities $2 \rightarrow 3$, reaching 35×35 with 4 agents
POGEMA	1–4	8×8, density 0.15, horizon 32; agents $2 \rightarrow 4 \rightarrow 6 \rightarrow 8$
POGEMA	5	horizon $32 \rightarrow 128$
POGEMA	6–9	grid $8 \times 8 \rightarrow 11 \times 11 \rightarrow 14 \times 14 \rightarrow 17 \times 17 \rightarrow 20 \times 20$
POGEMA	10	density $0.15 \rightarrow 0.25$, reaching the final task

Table 19: Per-rollout threshold sensitivity with the saved control trajectories.

Domain	τ	Accepted stages	Final result	Mean metric	Final accepted
POGEMA	0.80	10/10	10/10	1.000	Yes
POGEMA	0.90	10/10	10/10	1.000	Yes
POGEMA	0.95	10/10	10/10	1.000	Yes
Flatland	0.80	7/10	9/10	0.950	Yes
Flatland	0.90	7/10	9/10	0.950	Yes
Flatland	0.95	7/10	9/10	0.950	Yes

Table 20: Pass-fraction sensitivity at $\tau = 0.90$ under fixed-trajectory post-hoc re-scoring.

Outcome	$\rho = 0.50$	$\rho = 0.67$	$\rho = 1.00$
POGEMA accepted stages	10/10	10/10	10/10
POGEMA final accepted	Yes	Yes	Yes
Flatland accepted stages	10/10	7/10	7/10
Flatland final accepted	Yes	Yes	No

Table 21: LiveCodeBench Hard (post-2024-08, contamination-controlled): per-method Pass@1 over 20 problems. All methods use the same DeepSeek-V4-Pro backbone.

Method	**Solved (Pass@1)**	**Avg pass rate**	**LLM calls**	**Cost (USD)**
Direct one-shot	8/20 (40%)	40.0%	20	$0.30
Reflexion-3	8/20 (40%)	40.0%	44	$0.74
EvoCurr	**11/20 (55%)**	**58.3%**	69	$1.03

E StarCraft II Micro-Management Details

E.1 Introduction to StarCraft II

StarCraft II is a real-time strategy game developed by Blizzard Entertainment. It has three asymmetric factions, Terran, Protoss, and Zerg, each with distinct units, technologies, and tactical interactions. Players must simultaneously manage economy, production, technology, army composition, and real-time combat control. This makes the game a long-standing benchmark for studying decision-making under partial observability, large action spaces, and severe time pressure.

For AI research, StarCraft II is valuable because it combines long-horizon strategic planning with millisecond-level tactical execution. AlphaStar demonstrated that large-scale learning systems can reach Grandmaster-level full-game play, while PySC2 and SMAC-style benchmarks made controlled micro-management evaluation more accessible. Our work focuses on a narrower but difficult slice of this space: fixed-army late-game engagements where the economy and production systems are removed, leaving the agent to solve the tactical question of how to fight with a large heterogeneous army.

E.2 StarCraft II API and Python Interfaces

The technical foundation enabling AI research in StarCraft II rests on Blizzard Entertainment's official StarCraft II Machine Learning API, which provides programmatic access to the game's complete state information and action execution capabilities. This API exposes the game engine through a protocol buffer-based interface that delivers real-time observations including unit positions, resource states, map geometry, and tactical information while accepting high-level commands for unit control, building construction, and technology research. The official **s2client-proto** defines the core communication protocol between external programs and the StarCraft II executable, establishing standardized data structures for observations, actions, and game configuration. This low-level interface handles the complex details of game state serialization, network communication, and command validation, but requires substantial boilerplate code and deep understanding of the underlying protocol specifications to implement effective AI agents.

Building upon this foundation, the research community has developed higher-level abstractions that significantly simplify AI development while preserving the full functionality of the underlying API. **PySC2**, developed by DeepMind, transforms the raw API into a structured reinforcement learning environment that follows standard RL conventions with observation spaces, action spaces, and reward functions. This environment emphasizes feature-layer representations and provides built-in mini-games for curriculum learning, making it particularly suitable for deep reinforcement learning approaches.

Complementing PySC2, the **python-sc2** library offers a more direct and intuitive interface focused on scripted bot development, where complex strategic behaviors can be implemented using straightforward Python code with minimal boilerplate. The python-sc2 library abstracts away protocol buffer complexities while exposing high-level game objects such as units, abilities, and map structures through clean Python APIs, enabling researchers to focus on strategic logic rather than low-level implementation details. Our EvoCurr framework leverages python-sc2's accessibility and expressiveness to generate behavior tree scripts that can be easily interpreted, debugged, and modified, making it an ideal choice for our curriculum-based approach to complex tactical reasoning.

E.3 Final Task Setting

To rigorously evaluate the framework's capacity for handling high-dimensional state spaces and heterogeneous multi-agent coordination, we construct two terminal scenarios (T_f) representing late-game full-technology engagements. As delineated in Tables 22 and 23, these tasks simulate asymmetric warfare against Protoss and Zerg factions, respectively. The agent is tasked with controlling a complex Terran composition that integrates biological infantry (e.g., Marines, Ghosts) with mechanical support (e.g., Siege Tanks, Liberators). This configuration necessitates the execution of precise micro-management actions, including the timely activation of abilities such as *Stimpack*, *Personal Cloaking*, and *Siege Mode*. Conversely, the adversary configurations are initialized with high-threat counter-units possessing lethal Area-of-Effect (AoE) capabilities, specifically High Templars and Colossi in the Protoss matchup, and Banelings and Broodlords in the Zerg matchup. Consequently, successful task completion demands that the agent exhibit emergent tactical behaviors, such as formation dispersion to mitigate AoE damage and priority targeting to neutralize key enemy assets.

Table 22: Final Terran vs Protoss Task Specification

Unit Type	Quantity	Technology
AGENTS (Terran)		
Marine	20	Stimpack
Marauder	12	Stimpack
Medivac	4	Heal
Ghost	8	PersonalCloaking
SiegeTank	6	SiegeTech
VikingFighter	8	AssaultMode
Cyclone	7	LockOn
WidowMine	7	Burrow
Raven	3	HunterSeeker
Liberator	2	DefenderMode
ENEMIES (Protoss)		
Zealot	15	Charge
Stalker	14	BlinkTech
Sentry	10	ForceField
HighTemplar	8	PsiStormTech
Colossus	4	ExtendedThermalLance
Tempest	5	GroundAttack
Disruptor	4	PurificationNova
Carrier	4	InterceptorLaunch

E.4 StarCraft II Score Tables

Win rate captures final success but not the quality of partial or already-winning policies. We therefore also report a damage-cost-aware combat score $\mathcal{S}_{combat}$ for more nuanced evaluation. This metric compares EvoCurr with direct target-task implementations generated by closed-source LLMs (DeepSeek-V3.1, GPT-5, Claude-4, and Gemini-2.5), given by:

$$\mathcal{S}_{combat} = \frac{1}{2}\left(1 + \frac{R_{\text{agent\_final}}}{R_{\text{agent\_init}}} - \frac{R_{\text{enemy\_final}}}{R_{\text{enemy\_init}}}\right), \tag{7}$$

where the total combat power for one side is

$$R_{\text{side}} = \sum_i (m_i + \alpha v_i + \beta b_i)\,\frac{h_i + s_i}{h_i^{\max} + s_i^{\max}}. \tag{8}$$

This metric aggregates resource cost and remaining health/shields; scores range from 0 to 1. A combat score $\mathcal{S}_{combat} > 0.5$ indicates successful annihilation of the majority of enemy forces while

Table 23: Final Terran vs Zerg Task Specification

Unit Type	Quantity	Technology
AGENTS (Terran)		
Marine	20	Stimpack
Marauder	12	Stimpack
Medivac	4	Heal
Ghost	8	PersonalCloaking
SiegeTank	6	SiegeTech
VikingFighter	8	AssaultMode
Cyclone	7	LockOn
WidowMine	7	Burrow
Raven	3	HunterSeeker
Liberator	2	DefenderMode
ENEMIES (Zerg)		
Zergling	60	ZerglingMovementSpeed
Baneling	24	CentrificalHooks
Roach	15	GlialReconstitution
Hydralisk	10	HydraliskSpeed
Lurker	6	Burrow
Corruptor	10	FlyerWeaponsLevel1
Infestor	3	EnergyUpgrade
Viper	4	FlyerArmorsLevel1
Overseer	3	FlyerArmorsLevel1
Queen	4	MissileWeaponsLevel1
Broodlord	4	FlyerWeaponsLevel1

preserving our own, providing a finer-grained assessment than binary win/loss especially for failed code executions where the first term becomes 0.

Following the protocol in Section 4.1, we keep selected-policy comparison and cheap-sampling variability separate. EvoCurr cells below are the final policies selected by complete curriculum searches. Direct selected cells are the best executable policies found by repeated one-shot target-task sampling under the same per-policy rollout evaluation. We then report the direct-attempt distributions separately, since those 10 attempts estimate one-shot generation variability rather than full curriculum-search variability.

To demonstrate the effectiveness of EvoCurr, we compared it against direct behavior tree generation using LLMs and an approach that enables models to refine behavior trees through self-reflection. The results are presented in Tables 24 and 25.

Table 24: **Winning Score (Part 1: Ours).** Comparison of EvoCurr variants.

Task	Evo-DeepSeek	Evo-Claude	Evo-GPT
bush (TVP)	**0.6998**	0.6359	0.6963
corridor (TVP)	**0.7146**	0.6095	0.6200
corner (TVP)	0.5515	0.6367	0.6586
flat (TVP)	**0.6996**	0.6070	0.6295
main (TVP)	0.7277	0.6565	0.6669
ramp (TVP)	0.7226	0.6345	0.6811
bush (TVZ)	**0.7322**	0.6701	0.6467
corridor (TVZ)	**0.6980**	0.6497	0.5751
corner (TVZ)	**0.8405**	0.7420	0.6206
flat (TVZ)	**0.6263**	0.6125	0.5314
main (TVZ)	0.7017	**0.7603**	0.7531
ramp (TVZ)	**0.7466**	0.6519	0.7064

Part 2: Baselines Comparison.

Task	DeepSeek	GPT	Claude	Gemini	Claude code	Codex	Cursor
bush (P)	0.33	0.43	0.35	0.49	0.19	0.27	0
corr (P)	0.56	0.51	0.63	0.26	0.19	0.21	0
corn (P)	**0.66**	0.52	0.62	0.55	0.28	0.27	0
flat (P)	0.41	0.56	0.39	0.30	0.21	0.22	0
main (P)	0.69	0.63	0.67	**0.77**	0.34	0.40	0
ramp (P)	0.45	0.48	**0.74**	0.66	0.21	0.23	0
bush (Z)	0.61	0.34	0.64	0.36	0.46	0.31	0
corr (Z)	0.58	0.56	0.65	0.59	0.34	0.31	0
corn (Z)	0.41	0.69	0.77	0.80	0.74	0.62	0
flat (Z)	0.36	0.29	0.49	0.37	0.31	0.27	0
main (Z)	0.68	0.43	0.43	0.66	0.45	0.45	0
ramp (Z)	0.45	0.45	0.68	0.29	0.45	0.29	0

Table 25: **Reflect Score (Part 1: Ours).** Comparison of EvoCurr variants with self-reflection.

Task	Evo-DeepSeek	Evo-Claude	Evo-GPT
pvt+bush	**0.6998**	0.6359	0.6963
pvt+corridor	**0.7146**	0.6095	0.6200
pvt+corner	0.5515	0.6367	0.6586
pvt+flat	**0.6996**	0.6070	0.6295
pvt+main	0.7277	0.6565	0.6669
pvt+ramp	**0.7226**	0.6345	0.6811
zvt+bush	**0.7322**	0.6701	0.6467
zvt+corridor	**0.6980**	0.6497	0.5751
zvt+corner	**0.8405**	0.7420	0.6206
zvt+flat	**0.6263**	0.6125	0.5314
zvt+main	0.7017	**0.7603**	0.7531
zvt+ramp	**0.7466**	0.6519	0.7064

Part 2: Baselines Comparison.

Task	DeepSeek	GPT-5	Claude-4	Gemini
pvt+bush	0.4605	0.1667	0.5208	0
pvt+corridor	0.3328	0.2649	0.4651	0
pvt+corner	**0.7332**	0.0899	0.6823	0
pvt+flat	0.5467	0.4502	0.6110	0
pvt+main	0.6518	0.4196	**0.7381**	0
pvt+ramp	0.4679	0.4142	0.5935	0.6649
zvt+bush	0.3189	0.2134	0.4271	0
zvt+corridor	0.5696	0.4746	0.2913	0.4614
zvt+corner	0.7043	0.7620	0.5773	0
zvt+flat	0.4520	0.2765	0.3424	0
zvt+main	0.3068	0.2982	0.5518	0.4567
zvt+ramp	0.2691	0.5488	0.3305	0

Table 26: Selected SC2 combat scores for Terran-vs-Protoss tasks. EvoCurr columns report the curriculum-selected final policy for each backbone. Direct columns report the best executable policy found by each direct-generation backbone within 10 target-task attempts.

	EvoCurr selected			Direct selected best			
Task	DS	Claude	GPT	DS	GPT	Claude	Gemini
Bush	0.700	0.636	0.696	0.373	0.406	0.590	0.514
Corridor	0.715	0.610	0.620	0.627	0.617	0.644	0.340
Corner	0.551	0.637	0.659	0.640	0.638	0.708	0.559
Flat	0.700	0.607	0.629	0.433	0.652	0.331	0.319
Main	0.728	0.657	0.667	0.666	0.706	0.766	0.841
Ramp	0.723	0.635	0.681	0.658	0.550	0.766	0.631

Table 27: Direct-generation combat-score distributions for Terran-vs-Protoss tasks. Cells report mean ± std over 10 independent target-task attempts; brackets give executable attempts out of 10.

Task	**DeepSeek-V3.1**	**GPT-5**	**Claude-4**	**Gemini-2.5**
Bush	0.098±0.159 [3]	0.261±0.147 [8]	0.097±0.210 [2]	0.212±0.210 [6]
Corridor	0.320±0.242 [7]	0.259±0.205 [7]	0.174±0.241 [4]	0.081±0.118 [4]
Corner	0.200±0.278 [4]	0.413±0.177 [9]	0.071±0.224 [1]	0.197±0.222 [6]
Flat	0.108±0.177 [3]	0.238±0.265 [5]	0.033±0.105 [1]	0.032±0.101 [1]
Main	0.247±0.279 [5]	0.476±0.276 [8]	0.150±0.316 [2]	0.084±0.266 [1]
Ramp	0.302±0.279 [6]	0.197±0.218 [5]	0.310±0.335 [5]	0.063±0.199 [1]

Table 28: Selected SC2 combat scores for Terran-vs-Zerg tasks. The reporting convention matches Table 26.

	EvoCurr selected			**Direct selected best**			
Task	DS	Claude	GPT	DS	GPT	Claude	Gemini
Bush	0.732	0.670	0.647	0.646	0.430	0.661	0.357
Corridor	0.698	0.650	0.575	0.668	0.541	0.676	0.675
Corner	0.840	0.742	0.621	0.495	0.745	0.764	0.789
Flat	0.626	0.612	0.531	0.362	0.318	0.548	0.353
Main	0.702	0.760	0.753	0.786	0.535	0.661	0.657
Ramp	0.747	0.652	0.706	0.428	0.411	0.681	0.329

Table 29: Direct-generation combat-score distributions for Terran-vs-Zerg tasks. The reporting convention matches Table 27.

Task	**DeepSeek-V3.1**	**GPT-5**	**Claude-4**	**Gemini-2.5**
Bush	0.255±0.283 [5]	0.226±0.196 [6]	0.100±0.224 [2]	0.071±0.149 [2]
Corridor	0.325±0.168 [9]	0.171±0.225 [4]	0.068±0.214 [1]	0.130±0.273 [2]
Corner	0.121±0.201 [3]	0.356±0.324 [6]	0.288±0.373 [4]	0.265±0.347 [4]
Flat	0.123±0.161 [4]	0.149±0.158 [5]	0.144±0.200 [4]	0.035±0.112 [1]
Main	0.534±0.240 [9]	0.297±0.213 [7]	0.066±0.209 [1]	0.234±0.309 [4]
Ramp	0.276±0.192 [7]	0.227±0.166 [7]	0.124±0.228 [3]	0.065±0.137 [2]

E.5 Generated Behavior Tree Code Examples

This section presents complete examples of behavior tree implementations generated by the EvoCurr framework at different curriculum stages, demonstrating the evolution of tactical complexity.

E.5.1 Early Stage: Basic Marine Micro-management

```python
from sc2 import maps
from sc2.bot_ai import BotAI
from sc2.data import Race, Difficulty
from sc2.ids.ability_id import AbilityId
from sc2.ids.unit_typeid import UnitTypeId
from sc2.main import run_game
from sc2.player import Bot, Computer

class BattleBot(BotAI):
    def __init__(self):
        super().__init__()
        self.stim_used = set()

    async def on_step(self, iteration: int):
        if iteration == 0:
            print("Marine Micro Bot - 5v2 Marines vs Zealots!")
        if self.units.exists:
            await self.marine_micro()

    async def marine_micro(self):
        marines = self.units(UnitTypeId.MARINE)
        zealots = self.enemy_units(UnitTypeId.ZEALOT)

        if not marines.exists or not zealots.exists:
            return

        close_zealots = zealots.filter(lambda z:
            marines.closest_to(z.position).distance_to(z) < 6)

        if close_zealots.exists:
            for marine in marines:
                if (marine.tag not in self.stim_used and
                    AbilityId.EFFECT_STIM_MARINE in
                    await self.get_available_abilities(marine)):
                    marine(AbilityId.EFFECT_STIM_MARINE)
                    self.stim_used.add(marine.tag)

        target = min(zealots, key=lambda z: z.health + z.shield)

        for marine in marines:
            dist = marine.distance_to(target)
            if dist < 1:
                retreat_pos = marine.position.towards(target.position, -3)
                marine.move(retreat_pos)
            elif dist <= 5:
                marine.attack(target)
            else:
                marine.move(target.position)
```

E.5.2 Intermediate Stage: Multi-unit Coordination

```python
class BattleBot(BotAI):
    async def on_step(self, iteration: int):
        if iteration == 0:
            print("Terran Battle Bot Activated!")

        if self.units.exists:
            await self.control_ghosts()
            await self.control_marines()
            await self.control_marauders()
            await self.control_medivacs()

    async def control_ghosts(self):
        ghosts = self.units(UnitTypeId.GHOST)
        if not ghosts.exists:
            return

        high_templars = self.enemy_units(UnitTypeId.HIGHTEMPLAR)
        stalkers = self.enemy_units(UnitTypeId.STALKER)

        for ghost in ghosts:
            if high_templars.exists:
                templar = high_templars.closest_to(ghost.position)
                if ghost.distance_to(templar) < 12:
                    if AbilityId.SNIPE_SNIPE in
                       await self.get_available_abilities(ghost):
                        ghost(AbilityId.SNIPE_SNIPE, templar)
                    elif AbilityId.EMP_EMP in
                         await self.get_available_abilities(ghost):
                        ghost(AbilityId.EMP_EMP, templar.position)
                    ghost.move(templar.position)
                    if AbilityId.BEHAVIOR_CLOAKON_GHOST in
                       await self.get_available_abilities(ghost):
                        ghost(AbilityId.BEHAVIOR_CLOAKON_GHOST)
            elif stalkers.exists:
                target = stalkers.closest_to(ghost.position)
                ghost.attack(target)

    async def control_medivacs(self):
        medivacs = self.units(UnitTypeId.MEDIVAC)
        if not medivacs.exists:
            return

        bio_units = self.units.filter(lambda unit:
                    unit.type_id in {UnitTypeId.MARINE,
                    UnitTypeId.MARAUDER})
        for medivac in medivacs:
            injured = bio_units.filter(lambda unit:
                      unit.health_percentage < 0.75)
            if injured.exists:
                target = injured.closest_to(medivac.position)
                medivac.move(target.position)
                if medivac.distance_to(target) < 5:
                    medivac(AbilityId.MEDIVACHEAL_HEAL, target)
            elif bio_units.exists:
                medivac.move(bio_units.center)
```

E.5.3 Advanced Stage: Complex Multi-unit Tactical Framework

```python

class BattleBot(BotAI):
    async def on_step(self, iteration: int):
        if iteration == 0:
            self.setup_complete = False
            await self.initial_positioning()
            self.setup_complete = True

        bio_units = self.units.of_type({UnitTypeId.MARINE, UnitTypeId.MARAUDER})
        medivacs = self.units(UnitTypeId.MEDIVAC)

        await self.avoid_aoe(bio_units + medivacs)
        await self.control_siege_tanks()
        await self.control_vikings()
        await self.control_liberators()
        await self.control_ghosts()
        await self.control_bio(UnitTypeId.MARINE)
        await self.control_bio(UnitTypeId.MARAUDER)
        await self.control_medivacs()

    async def avoid_aoe(self, units: Units):
        storms = [e for e in self.state.effects
                  if e.id == EffectId.PSISTORMPERSISTENT]
        disruptor_balls = self.enemy_units(UnitTypeId.DISRUPTORPHASED)

        threats = []
        for storm in storms:
            threats.append((storm.position, 2.5))
        for ball in disruptor_balls:
            threats.append((ball.position, 2.5))

        for unit in units:
            for pos, radius in threats:
                if unit.distance_to(pos) < radius:
                    away = unit.position.towards(pos, -3)
                    unit.move(away)
                    break

    async def control_siege_tanks(self):
        siege_tanks = self.units(UnitTypeId.SIEGETANKSIEGED)
        sieged_tanks = self.units(UnitTypeId.SIEGETANKSIEGED)

        for tank in siege_tanks:
            if tank.distance_to(Point2((15, 15))) > 2:
                continue
            abilities = await self.safe_get_abilities(tank)
            if AbilityId.SIEGEMODE_SIEGEMODE in abilities:
                tank(AbilityId.SIEGEMODE_SIEGEMODE)

        if sieged_tanks.exists:
            enemies = self.enemy_units
            if not enemies.exists:
                return
            priority_targets = enemies.of_type([UnitTypeId.COLOSSUS,
                                UnitTypeId.STALKER,
                                UnitTypeId.HIGHTEMPLAR,
                                UnitTypeId.ZEALOT])

            for tank in sieged_tanks:
                targets_in_range = priority_targets.in_attack_range_of(tank)
                if targets_in_range:
                    target = min(targets_in_range,
                          key=lambda t: (t.type_id not in
                            {UnitTypeId.COLOSSUS,
                             UnitTypeId.HIGHTEMPLAR},
                            t.distance_to(tank)))
                    tank.attack(target)
```

F Complete Curriculum Evolution Paths (StarCraft II)

F.1 Curriculum Design

We illustrate the curriculum-design process by listing the sub-tasks generated for the final task in Table 30. Each sub-task is generated based on the outcome of the previous curriculum. For StarCraft II tasks, we set the acceptance threshold to $\tau = 0.9$: the Designer increases difficulty when the win rate is above 90% and decreases difficulty otherwise. Among the 12 tasks, the Terran vs Protoss setting on the Bush map requires the longest curriculum trajectory because bush concealment makes enemy visibility and target selection harder; the Designer therefore lowers difficulty twice before the Solver finally masters the target task.

We omit the older evolution schematic here because it is visually noisy in the two-column appendix. The main paper keeps the clearer trajectory view in Figure 18; this appendix focuses on the exact curriculum tables needed for auditability.

F.2 Example EvoCurr Evolution Path

To provide a granular view of the curriculum evolution process, Table 31 traces a representative curriculum path in the Terran vs. Protoss scenario. The progression demonstrates the adaptive Designer–Solver dynamics: beginning with a simple engagement of 5 Marines against 2 Zealots, the curriculum escalates through increasingly complex multi-unit compositions incorporating specialized roles such as Ghosts for detection and Siege Tanks for area control. When the Solver fails to achieve the acceptance threshold at Stage 3 with an ambitious 5-unit-type composition, the framework retreats to a simplified 4-unit configuration in Stage 4, reestablishing a 90% win rate before advancing again. This adaptive adjustment enables the system to reach the final task at Stage 6, where the agent commands a diverse 10-unit-type army against an 8-unit-type enemy force.

G Overcooked Details

G.1 MARL Training Configuration

For the Overcooked experiments, we employ the E3T (Efficient End-to-End Training) algorithm (Yan et al., 2023) as our MARL training framework. The training configuration is detailed in Table 32.

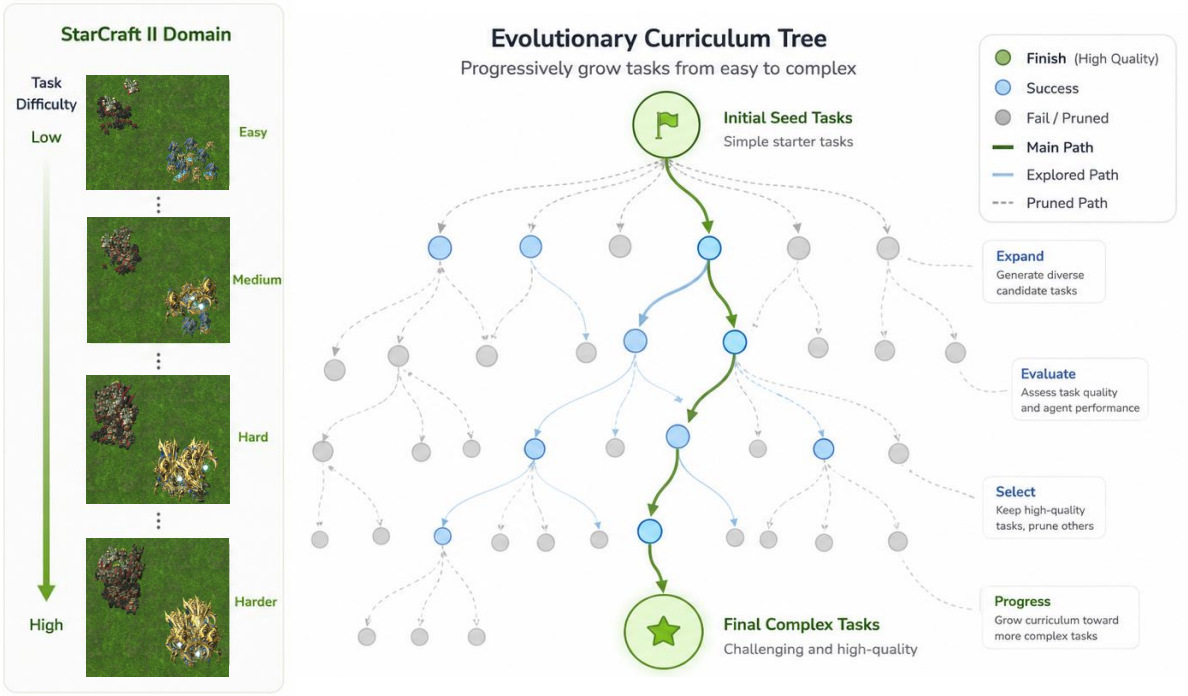


Figure 18: Conceptual illustration of the self-evolving curriculum tree in EvoCurr. The framework progressively grows task difficulty from simple starter seeds to the final high-quality complex tasks. By enforcing an accepted-floor constraint, the model efficiently prunes failed sub-paths and backs off to mastered checkpoints, preventing policy regression. Representative intermediate states from StarCraft II and Overcooked domains are illustrated on both sides.

G.2 Curriculum Design for Overcooked

The curriculum progression in Overcooked is structured around three key dimensions:

1. **Layout Complexity**: Starting from simple layouts (e.g., `small_corridor`) with direct paths between stations, progressing to complex layouts with obstacles and longer navigation requirements.

2. **Order Complexity**: Beginning with single-ingredient dishes, advancing to multi-ingredient recipes requiring precise coordination between agents.

3. **Temporal Constraints**: Initially allowing unlimited time for order completion, then introducing time pressure and simultaneous order requirements.

The acceptance criterion $\mathcal{P}(\pi|C) = 1$ is achieved when all required orders are completed within the episode. After meeting this criterion, agents can pursue bonus orders to maximize total deliveries. The entropy regularization schedule ensures exploration early in training (high entropy) while converging to more deterministic policies as training progresses.

For layouts with particular navigation challenges (e.g., `small_corridor`), we adjust the entropy coefficient horizons to $[0,\ 8 \times 10^6,\ 10^7]$ to allow for extended exploration before convergence. The shared policy architecture enables agents to learn cooperative behaviors more efficiently by sharing representations while maintaining individual action distributions.

H Overcooked Experimental Results Details

This appendix provides detailed experimental results for the Overcooked environment, showing the complete curriculum evolution and performance metrics for two different map configurations.

H.1 Map Configuration 1

H.1.1 Final Layout Specification

Table 30: Winning rates (%) of curricula designed for the 12 complex decision-making scenarios.

Task	Task1	Task2	Task3	Task4	Task5	Task6	Task7
Flat (TvP)	100	100	70	90	100	-	-
Flat (TvZ)	100	100	100	100	100	-	-
Bush (TvP)	100	100	40	90	50	100	100
Bush (TvZ)	100	90	60	100	100	-	-
Corridor (TvP)	100	90	100	100	90	-	-
Corridor (TvZ)	100	90	90	100	100	-	-
Corner (TvP)	90	90	90	100	-	-	-
Corner (TvZ)	100	100	90	90	100	-	-
Main (TvP)	100	100	100	100	-	-	-
Main (TvZ)	100	100	100	100	-	-	-
Ramp (TvP)	100	100	90	100	100	-	-
Ramp (TvZ)	100	100	100	90	100	-	-

```
{
  "grid": "XXXPX
          X 2 P
          D1X X
          O   X
          XTSXX",
  "start_all_orders": [
    {"ingredients": ["onion", "tomato"]},
    {"ingredients": ["onion", "onion"]},
    {"ingredients": ["onion", "tomato", "tomato
     "]},
    {"ingredients": ["onion", "onion", "tomato
     "]},
    {"ingredients": ["onion", "onion", "onion"]}
  ],
  "recipe_value": [10, 10, 20, 20, 20],
  "recipe_time": [10, 10, 20, 20, 20]
}
```

Grid Legend: X=Wall, P=Pot, D=Dish, O=Onion, T=Tomato, S=Service

H.1.2 Curriculum Evolution Results

The Overcooked Map 1 evolution path and result details are shown in Table 33.

H.1.3 Detailed Performance Metrics Comparison

The Overcooked Map 1 result details and metrics are shown in Table 34.

H.2 Map Configuration 2

H.2.1 Final Layout Specification

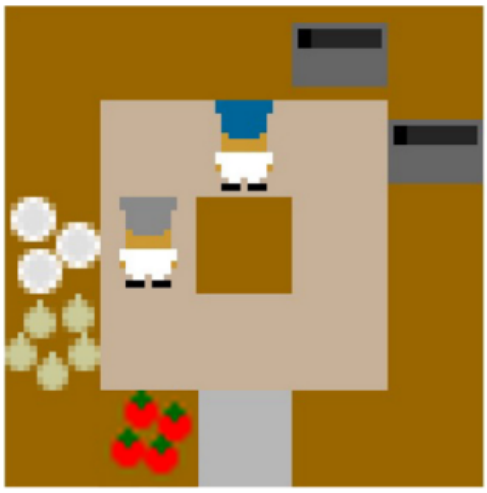


Figure 19: Overcooked Map 1.

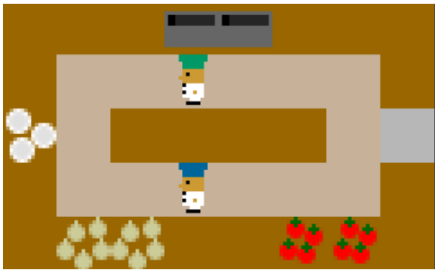


Figure 20: Overcooked Map 2.

Table 31: **Curriculum Evolution Trace (Path 1).** Detailed breakdown of unit compositions from the initial task to the final complex scenario.

Stage	Composition (Agent vs Enemy)	Win Rate
1	**Agent:** Marine (5) **Enemy:** Zealot (2, Charge)	90%
2	**Agent:** Marine (10), Marauder (5), Medivac (1) **Enemy:** Zealot (5, Charge), Stalker (5, Blink), HighTemplar (1, PsiStorm)	90%
3	**Agent:** Marine (15), Marauder (8), Ghost (2), Medivac (2), SiegeTank (1) **Enemy:** Zealot (10, Charge), Stalker (8, Blink), HighTemplar (2, PsiStorm), Colossus (1, ExtendedThermalLance)	Failed
4	**Agent:** Marine (12), Marauder (6), Ghost (1), Medivac (1) **Enemy:** Zealot (8, Charge), Stalker (6, Blink), HighTemplar (1, PsiStorm)	90%
5	**Agent:** Marine (18), Marauder (10), Ghost (2), Medivac (2), SiegeTank (1), Viking (2) **Enemy:** Zealot (12, Charge), Stalker (10, Blink), HighTemplar (2, PsiStorm), Colossus (1, ExtendedThermalLance)	90%
6	**Agent:** Marine (20), Marauder (12), Medivac (4), Ghost (8), SiegeTank (6), VikingFighter (8), Cyclone (7), WidowMine (7), Raven (3), Liberator (2) **Enemy:** Zealot (15, Charge), Stalker (14, BlinkTech), Sentry (10, ForceField), HighTemplar (8, PsiStormTech), Colossus (4, ExtendedThermalLance), Tempest (5, GroundAttack), Disruptor (4, PurificationNova), Carrier (4, InterceptorLaunch)	100%


```
{
  "grid": "XXXPDPXXX
          X   2   X
          X XXXXX X
          X   1   X
          XXXOSTXXX",
  "start_all_orders": [
    {"ingredients": ["onion", "tomato"]},
    {"ingredients": ["onion", "onion"]},
    {"ingredients": ["onion", "tomato", "tomato
     "]},
    {"ingredients": ["onion", "onion", "tomato
     "]},
    {"ingredients": ["onion", "onion", "onion"]}
  ],
  "recipe_value": [10, 10, 20, 20, 20],
  "recipe_time": [10, 10, 20, 20, 20]
}
```


Grid Legend: X=Wall, P=Pot, D=Dish, O=Onion, T=Tomato, S=Service

H.2.2 Curriculum Evolution Results

The Overcooked Map 2 evolution path and result details are shown in Table 35. Task 1 includes two single-ingredient recipes alongside complex recipes for an easier transition.

H.2.3 Detailed Performance Metrics Comparison

The Overcooked Map 2 result details and metrics are shown in Table 36.

I Claude Code Behavior-Tree Generation Details

I.1 Execution Results

Claude Code is evaluated as an autonomous coding-agent baseline. It generates its own intermediate curricula and iteratively edits the behavior-tree implementation using execution feedback.

Final winning rate: 0%
Total time: ∼90 min
Evaluation budget: 100+ simulated matches

Despite passing all self-designed intermediate curricula, Claude Code does not solve the final target task within this budget. Its final evaluation produces 0 wins, 1 draw, and 9 losses.

I.2 Curriculum Generation Process

The following trajectory illustrates the intermediate curricula generated by Claude Code, ordered from easier to harder settings.

Table 32: Overcooked MARL training hyperparameters

Parameter	Value
Algorithm	E3T
Number of agents	2
Episode length	400
Number of environment steps per curriculum	10^7
PPO epochs	15
Number of mini-batches	1
Rollout threads	100
Evaluation threads	10
Evaluation interval	20 episodes
Entropy Regularization	
Entropy coefficients	[0.2, 0.05, 0.01]
Entropy coefficient horizons	$[0, 6 \times 10^6, 10^7]$
Network Architecture	
CNN layers	[(32, 3×3, stride=1), (64, 3×3, stride=1), (32, 3×3, stride=1)]
Recurrent policy	LSTM
Shared policy	True
E3T Specific	
Epsilon (diversity bonus)	0.25
Weights copy factor	0.1
Random index	Enabled
Curriculum-specific	
Reward shaping horizon	10^8
Budget per curriculum	10^7 timesteps

Table 33: Map 1: Curriculum progression and performance metrics (A0 Del: Agent0 Delivery; A1 Del: Agent1 Delivery; Total: Total Delivery)

Task	Orders	A0 Del.	A1 Del.	Total	Reward
Task 0	3	11.75	12.33	24.08	239.2
Task 1	4	14.56	14.33	28.89	288.1
Task 2 (Final)	5	14.74	14.36	29.10	290.9
Direct Training	5	11.93	12.18	24.11	240.5

Table 34: Map 1: Key performance indicators for final task (A0: Agent0; A1: Agent1)

	EvoCurr		Direct	
Metric	**A0**	**A1**	**A0**	**A1**
Onion in Pot	15.77	15.94	13.38	13.57
Tomato in Pot	0.31	0.32	0.21	0.26
Dish Pickup	7.81	7.63	6.15	6.34
Soup Pickup	7.68	7.55	6.31	6.33
Cook Actions	7.44	8.67	6.34	7.30
Delivery	7.37	7.19	5.99	6.11
Idle Move	9.40	8.68	14.90	15.10

Table 35: Map 2: Curriculum progression and performance metrics (A0 Del: Agent0 Delivery; A1 Del: Agent1 Delivery; Total: Total Delivery)

Task	Ord.	A0 Del.	A1 Del.	Total	Reward
Task 0	3	7.28	7.25	14.53	290.3
Task 1	5*	7.32	7.28	14.60	291.6
Task 2 (Final)	5	7.82	10.82	18.64	186.2
Direct Training	5	7.40	8.96	16.36	163.4

Table 36: Map 2: Key performance indicators for final task (A0: Agent0; A1: Agent1)

	EvoCurr		Direct	
Metric	**A0**	**A1**	**A0**	**A1**
Onion in Pot	7.91	5.95	9.85	9.29
Tomato in Pot	3.17	3.87	0.08	0.07
Dish Pickup	5.45	4.20	4.32	4.81
Soup Pickup	4.12	5.62	4.24	4.79
Cook Actions	5.45	4.85	5.16	4.38
Delivery	3.92	5.42	3.71	4.49
Size-2 Del.	3.84	5.27	3.65	4.40
Size-3 Del.	0.07	0.14	0.05	0.08
Idle Move	14.42	14.21	10.49	11.28

Table 37: Claude Code execution outcomes on self-designed curricula and the held-out final task.

Task	Attempts	Outcome	Win rate
Curriculum 1	1	5W/0D/0L	**100%**
Curriculum 2	1	5W/0D/0L	**100%**
Curriculum 3	1	5W/0D/0L	**100%**
Curriculum 4	1	5W/0D/0L	**100%**
Curriculum 5	1	5W/0D/0L	**100%**
Final task	2	0W/1D/9L	**0%**

Table 38: Claude Code self-generated curriculum trajectory. Unit entries are written as *unit count (upgrade)*.

Step	Scenario Configuration
C1	**Friendly:** Marine 20 (Stimpack), Marauder 12 (Stimpack), Medivac 4 (Heal) **Enemy:** Zealot 6 (Charge), Stalker 4 (BlinkTech)
C2	**Friendly:** Marine 20 (Stimpack), Marauder 12 (Stimpack), Medivac 4 (Heal) **Enemy:** Zealot 10 (Charge), Stalker 8 (BlinkTech)
C3	**Friendly:** Marine 20 (Stimpack), Marauder 12 (Stimpack), Medivac 4 (Heal) **Enemy:** Zealot 12 (Charge), Stalker 10 (BlinkTech), Sentry 4 (ForceField)
C4	**Friendly:** Marine 20 (Stimpack), Marauder 12 (Stimpack), Medivac 4 (Heal), Ghost 4 (PersonalCloaking) **Enemy:** Zealot 12 (Charge), Stalker 10 (BlinkTech), HighTemplar 4 (PsiStormTech)
C5	**Friendly:** Marine 20 (Stimpack), Marauder 12 (Stimpack), Medivac 4 (Heal), Ghost 6 (PersonalCloaking), SiegeTank 4 (SiegeTech) **Enemy:** Zealot 15 (Charge), Stalker 12 (Blink), HighTemplar 6 (PsiStorm), Colossus 2 (ExtThermalLance)

I.3 Prompt Template Used for Coding Agents

Prompt Template

You have a sc2 control task as following:

AGENTS:

Marine: 20, (5,25), Stimpack
Marauder: 12, (5,25), Stimpack
Medivac: 4, (5,25), Heal
Ghost: 8, (5,25), PersonalCloaking
SiegeTank: 6, (5,25), SiegeTech
VikingFighter: 8, (5,25), AssaultMode
Cyclone: 8, (5,25), LockOn
WidowMine: 8, (5,25), Burrow
Raven: 3, (5,25), HunterSeeker
Liberator: 4, (5,25), DefenderMode

ENEMIES:

Zealot: 15, (25,5), Charge
Stalker: 15, (25,5), BlinkTech
Sentry: 10, (25,5), ForceField
HighTemplar: 9, (25,5), PsiStormTech
Colossus: 4, (25,5), ExtendedThermalLance
Tempest: 5, (25,5), GroundAttack
Disruptor: 4, (25,5), PurificationNova
Carrier: 4, (25,5), InterceptorLaunch

You should design the curricula **from easy to hard** to make sure the final control code can win the final task. You can change environments by assigning the scenario_type and map_name in rollout_config.py. Generate the control code and test it in res-temp.py. Prefer changing only the BattleBot class. Use the evaluation feedback to revise the policy.
When you pass the current curriculum with a high winning rate, you can design the next curriculum.

I.4 Generated Code Sample

```python
class BattleBot(BotAI):
    """
    Terran Auto-Micro Bot - Optimized Version
    Strategy: Immediate Stimpack + Focus Fire Nearest Enemy + Simple Kiting + Medivac Healing
    """

    def __init__(self):
        super().__init__()
        self.stim_used = set()
        self.started = False

    async def on_step(self, iteration: int):
        # Get units
        marines = self.units(UnitTypeId.MARINE)
        marauders = self.units(UnitTypeId.MARAUDER)
        medivacs = self.units(UnitTypeId.MEDIVAC)
        ghosts = self.units(UnitTypeId.GHOST)
        tanks = self.units(UnitTypeId.SIEGETANK) | self.units(UnitTypeId.SIEGETANKSIEGED)

        bio_units = marines | marauders
        all_combat = bio_units | ghosts | tanks
        enemies = self.enemy_units
        enemy_pos = Point2((27, 27))

        if not all_combat.exists:
            return

        # No enemies - quick attack
        if not enemies.exists:
            for unit in all_combat:
                unit.attack(enemy_pos)
            for medivac in medivacs:
                medivac.move(enemy_pos)
            return

        # === Optimization 1: Stimpack immediately upon first enemy sight ===
        if not self.started:
            self.started = True
            for unit in bio_units:
                if unit.health > 20:
                    if unit.type_id == UnitTypeId.MARINE:
                        unit(AbilityId.EFFECT_STIM_MARINE)
                    elif unit.type_id == UnitTypeId.MARAUDER:
                        unit(AbilityId.EFFECT_STIM_MARAUDER)
                    self.stim_used.add(unit.tag)

        # === Optimization 2: Simplified target selection ===
        low_health = enemies.filter(lambda u: u.health_percentage < 0.4)
        if low_health.exists:
            target = low_health.closest_to(bio_units.center)
        else:
            target = enemies.closest_to(bio_units.center)

        # === Optimization 3: Kiting tactics ===
        for unit in bio_units:
            closest = enemies.closest_to(unit.position)
            dist = unit.distance_to(closest)
            # Maintain distance from melee units
            if closest.type_id in [UnitTypeId.ZEALOT, UnitTypeId.ZERGLING]:
                if dist < 3:
                    retreat_pos = unit.position.towards(closest.position, -4)
                    unit.move(retreat_pos)
                    continue
            unit.attack(target)

        # === Ghost EMP ===
        for ghost in ghosts:
            shielded = enemies.filter(lambda u: u.shield > 20)
            if shielded.exists and ghost.energy >= 75:
                emp_target = max(shielded, key=lambda u: enemies.closer_than(3, u).amount)
                ghost(AbilityId.EMP_EMP, emp_target.position)
            else:
                ghost.attack(target)

        # === Tank Control ===
        for tank in tanks:
            if tank.type_id == UnitTypeId.SIEGETANK:
                closest = enemies.closest_to(tank.position)
                dist = tank.distance_to(closest)
                if 3 < dist < 13:
                    tank(AbilityId.SIEGEMODE_SIEGEMODE)
                else:
                    tank.attack(target)
            else:  # SIEGETANKSIEGED
                if tank.distance_to(enemies.closest_to(tank)) < 3:
                    tank(AbilityId.UNSIEGE_UNSIEGE)
                else:
                    tank.attack(target)

        # === Medivac Healing ===
        for medivac in medivacs:
            if bio_units.exists:
                injured = bio_units.filter(lambda u: u.health_percentage < 0.8)
                if injured.exists:
                    heal_target = min(injured, key=lambda u: u.health)
                    medivac(AbilityId.MEDIVACHEAL_HEAL, heal_target)
                else:
                    medivac.move(bio_units.center)
```

J Computational Experiments

We measure the approximate API cost and wall-clock time required to solve a final SC2 task such as Bush (TvP). These measurements use closed-source model APIs and include all curriculum stages needed to reach the final task.

For DeepSeek-V3.1, completing one final task costs approximately $0.2. For GPT-5 and Claude-4, the cost is approximately $1.5.

Each individual curriculum stage requires approximately 5–30 minutes. A final task typically requires 4–6 sequential stages, for a total wall-clock time of approximately 1–3 hours.