SC2FORGE: A CALIBRATED BENCHMARK AND A PROGRAM-WRITING AGENT FOR FULL-GAME STARCRAFT II

Anonymous authors
Paper under double-blind review

ABSTRACT

A language model can play a real-time strategy game from inside the control loop or write the program that plays it. The first needs a model call per decision and, on our benchmark, at least 5.5 wall-clock seconds per game second; the second delivers a compiled package that plays a typical game faster than real time, with no model call. We contribute **SC2FullGame-Bench**, which scores a StarCraft II bot package on 93 full-game cells, 90 against the built-in opponents (three maps, ten levels and three races) and 3 against a community bot. It also rates each package with four probe games (economy, tech, combat and defense). Before any submission is scored, the benchmark is calibrated: two ladders of known order and a package's agreement with itself on a repeated seed show that it separates agents of different strength. We also contribute **SC2Forge**, a framework that turns a minimal demonstration package, the runtime's documentation and a playbook of named openings into a compiled package through a write–test–revise loop with a verifier ladder, a tool-using rewrite arm and a capability-keyed archive. SC2Forge's package scores 0.943, its best of three runs, at the level of VibeCraft's expert-system Protoss package (0.932), the bot its playbook was written from. The package is separated above an iterated prompt (0.767) and an off-the-shelf coding agent (0.803); a coding agent given the playbook under the framework's own model budget reaches 0.900, within 0.05 per cell of the framework's package on sealed suite A's six seeds and separated below it on the released suite's six seeds. The framework also generates Terran and Zerg packages, each scoring above VibeCraft's package for that race; on connect-four the loop, its arms adapted to the game, delivers a policy scoring 0.977 against a scripted 0.611.

1 INTRODUCTION

A game agent can be built in three ways, two of which use a large language model (LLM) and are drawn in Fig. 1. The model can be placed in the control loop: at every decision step the game state is rendered as text and the reply is parsed into commands (Ma et al., 2024a; Shao et al., 2024; Li et al., 2024). Reinforcement learning has produced the strongest full-game agents (Vinyals et al., 2019; Han et al., 2020), at the cost of reward and action-space engineering per game, a training budget of millions of games, and a policy that cannot be read, edited or seeded with written expert knowledge. The third way puts the model outside the loop as an engineer: it writes, tests and revises a program from the runtime's documentation and a minimal demonstration package, and the program plays a typical game faster than real time.

Models that write the program are an established line, from scored functions and reward code to control policies and programs for small games (Romera-Paredes et al., 2024; Novikov et al., 2025; Ma et al., 2024b; Guo et al., 2026; Eberhardinger et al., 2024). SMAC-R1 and EvoCurr (Deng et al., 2024; Cheng et al., 2025) are the immediate predecessors of this work: they have a model write decision scripts for StarCraft micro-management and score each on a fixed task, and we take the same loop to the full game. In all of these systems, the loop takes its evaluator's score as exact and needs no estimate of its own noise. A full game is different: one evaluation is up to a half-hour match with a binary outcome, the same package on the same seed does not always end the same way, and a package that installs cleanly can still crash on its first tick. Two questions are therefore

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

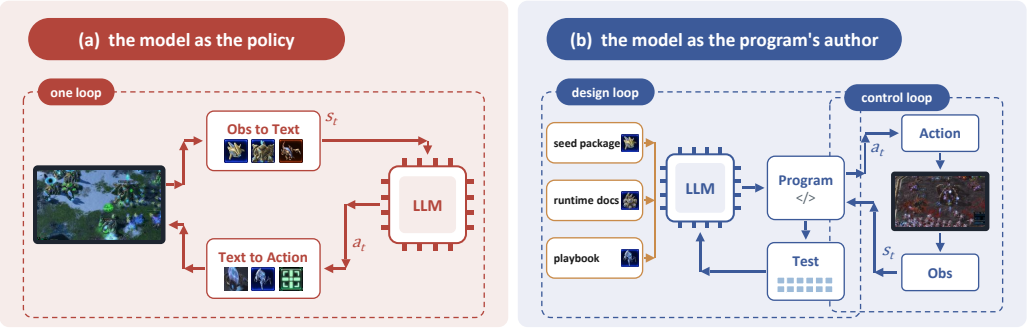

Figure 1: Two ways a language model plays. (a) The model is the policy: at every decision the state is rendered as text, the model replies, and the reply is parsed into commands. (b) The model is the program's author: it writes, tests and revises the program in a design loop, and the program alone plays the control loop.

open. The first is whether a difference between two packages' win rates is real: the suites that score agents on this game (Shen et al., 2025; Anonymous, 2025; Park et al., 2026; Tang et al., 2025) report win rates without showing that they separate agents whose order is known in advance. The second is whether a model outside the loop can reach the level of a package that a strong engineer writes by hand, and which of its inputs that level depends on, an expert's written openings or a loop that tests and revises; it has not been measured against such a package on the same cells and seeds.

We answer both questions on StarCraft II. **SC2FullGame-Bench** scores a package on 93 full-game cells, 90 against the game's built-in opponents and 3 against a community bot. Before scoring a submission it orders two ladders of known order and publishes its measured same-seed agreement, which sets every comparison's sample budget. **SC2Forge** takes a minimal demonstration package, the runtime's documentation and a playbook of named openings, and runs a write–test–revise loop (§4). On the benchmark it is compared on identical cells and seeds with the in-loop agent, the iterated prompt and the coding agent, the last two each with and without the playbook, and with VibeCraft's expert-system bots (catmaniii, 2026); one part of the loop is then removed at a time, and the frozen packages are played on unseen cells and maps. The loop delivers its package at a one-off cost of 3.6M billable tokens and 17.1 hours, and the package plays with no model call. The in-loop agent TextStarCraft2 (Ma et al., 2024a) needs a model call at every decision and at least 5.5 wall-clock seconds per game second behind DEEPSEEK-V4-FLASH.

Contributions. (i) A full-game benchmark whose discrimination is established before it is used: 93 cells, two ladders of known order that it reproduces, four capability probes, and per-cell same-seed agreement.

(ii) A framework that extends program optimization from micro-task scripts to full-game packages: a runtime API index, a minimal demonstration and optional expert knowledge become a compiled package. It ran for all three races of StarCraft II without a change to its runtime and three times over on Protoss: given a playbook written from an expert bot, it implements the builds the playbook describes at that bot's level, and without the playbook its three runs still average 0.838.

(iii) Paired comparisons on identical cells and seeds: the framework's package scores 0.943, its best of three runs, at the level of VibeCraft's expert-system Protoss package (0.932) and above most generated baselines. Its Terran and Zerg packages beat VibeCraft's bots of the same race, and its Protoss verdicts against VibeCraft and the coding agent hold on two sealed suites; the loop, its arms adapted to the game, scores 0.977 on connect-four against a scripted 0.611.

2 Related Work

Agents for the full game: a model in the loop, or a trained policy. TextStarCraft2 renders the game as text and parses the reply into macro actions (Ma et al., 2024a), SwarmBrain adds a tactical state machine (Shao et al., 2024), LLM-PySC2 exposes the PySC2 interface (Li et al., 2024), and

several suites benchmark models at play (Shen et al., 2025; Anonymous, 2025; Park et al., 2026; Tang et al., 2025). The line is still active: MASMP wraps the model in a memory-augmented state machine (Qi et al., 2025) and SEMA prunes observations to halve decision latency (Ma et al., 2026). Both still call a model per decision, so throughput is set by model latency rather than the game, the cost the compiled design removes (§5.2). Reinforcement learning is the other approach: SC2LE defined the full game as a learning problem (Vinyals et al., 2017), AlphaStar reached grandmaster level with league training (Vinyals et al., 2019), and TStarBots, TStarBot-X and SCC (Sun et al., 2018; Han et al., 2020; Wang et al., 2021) followed with hand-designed action spaces and shaped rewards; we compare cost profile, not strength.

Models that write the program. Code as Policies showed that a model can write the control program itself (Liang et al., 2023a), and Voyager grew a library of such programs from environment feedback (Wang et al., 2024). Both rely on the model's judgment of its own program, not a calibrated measurement. Pairing an LLM mutation operator with an evolutionary loop over code is established: ELM (evolution through large models) with MAP-Elites (Lehman et al., 2022; Mouret & Clune, 2015), FunSearch and AlphaEvolve (Romera-Paredes et al., 2024; Novikov et al., 2025), Eureka (Ma et al., 2024b), Code Evolution (Guo et al., 2026), co-evolution against generated opponents (Li et al., 2026; Ke et al., 2026), program search over small games (Eberhardinger et al., 2024), and the predecessors of this work, SMAC-R1 and EvoCurr (Deng et al., 2024; Cheng et al., 2025), which write LLM decision scripts for StarCraft micro-management. In each, the artifact is one function or one script, scored on a fixed task whose score the loop can take as exact, and SC2Forge extends that line to a full-game package: the artifact is a multi-module bot that can crash on its first tick or break its package contract, and the loop has to decide what to believe rather than what to maximize. Verifier-guided synthesis filters candidate programs before they are scored (Chen et al., 2023); the ladder does that here for a package that must survive a real game before anything is scored at all, and its refusals are of a kind a test suite does not see (§5.3). Coding agents produce the same artifact by a second method (Shi et al., 2026), and expert systems of hand-coded rules by a third, here VibeCraft's three race bots (catmaniii, 2026), whose Protoss bot supplied the framework's playbook. Code-Space Response Oracles (Hennes et al., 2026) replace PSRO's best-response oracle (Lanctot et al., 2017) with a model that writes the policy as code, against a generated population rather than a calibrated fixed set. Zhang et al. (2026) and Zhao et al. (2026) hold that an agent's reliability is set by its harness rather than its model, and ask that reliability be reported with gain. AutoHarness (Lou et al., 2026) has a model write a code harness that removes illegal actions from an in-loop agent over 145 text games and chess. There the harness wraps a model that acts every turn, whereas here the code is the agent.

Benchmark validity. Catalogs list the failure modes a benchmark must rule out (Zhu et al., 2025; Reuel et al., 2024; Liang et al., 2023b), EvalPlus and SWE-bench show the grader as the failure point (Liu et al., 2023; Jimenez et al., 2024), and SMACv2 tests its own non-degeneracy (Ellis et al., 2023); the full-game RL agents above were evaluated against human players (AlphaStar, SCC) or the built-in levels, cheating tiers included (TStarBots, TStarBot-X), without a per-cell noise floor or a paired design; paired-seed evaluation (Sharma, 2025), sequential testing (Wald, 1945; Hsu & Shekhar, 2026; Akter et al., 2025; Wu et al., 2026) and paired variance reduction (Deng et al., 2013) underlie §3.

3 SC2FullGame-Bench: a calibrated full-game benchmark

Game cells and outcome score. SC2FullGame-Bench has 93 full-game cells. Of these, 90 are built-in cells: the three maps Simple64, AbyssalReefLE and AcropolisLE crossed with the built-in opponent's ten levels (lv1–lv10) and its three races, with the opponent's build set to random. The remaining 3 are external cells, one per map, against *Chance* (lladdy, 2025), a random-race bot from the AI Arena competition that runs unmodified under its own interpreter in a game hosted by the submission; three further AI Arena bots, all Zerg and not cells, each played nine games against twelve of the paper's packages (Appendix A). A submission is a frozen package whose file digests are recorded before it is played. The released default is one game per cell on a fixed seed, and the 93 cells are the released suite. A win scores 1; a tie, a loss or a game reaching the 1,800 s cap scores 0. A crash is a loss, and the outcome score is the mean over cells. Every compiled package's body score is the mean over the 93 cells with three seeds per cell unless another basis is named beside it; the in-loop agent plays one seed and is labeled so; every other score is given as a per-cell difference

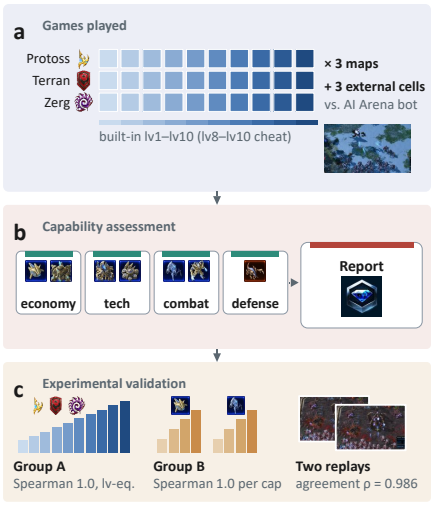

Figure 2: SC2FullGame-Bench. (a) 93 game cells per package. (b) The four capability probes, reported beside the outcome and never entering it. (c) Calibration: the two ladders of known order.

Table 1: Both calibration ladders in their known order, measured before any submission was scored: outcome on the 90 built-in cells; level-equivalent, the built-in level whose outcome matches; capability scores as defined in §3. Plotted ladders with intervals: Appendix D.

Subject	Outcome	Lv-eq.	Econ.	Tech	Comb.	Def.
Group A: built-in AI as subject						
lv1	0.015	1.0	0.49	0.15	0.49	0.52
lv2	0.148	2.0	0.52	0.21	0.49	1.00
lv3	0.256	3.0	0.54	0.58	0.48	1.00
lv4	0.359	4.0	0.57	0.59	0.49	1.00
lv5	0.433	5.0	0.57	0.63	0.50	0.67
lv6	0.574	6.0	1.00	0.82	0.53	1.00
lv7	0.644	7.0	1.00	0.82	0.54	1.00
lv8 (cheat)	0.600	6.4	0.84	0.67	0.51	1.00
lv9 (cheat)	0.744	9.0	0.84	0.84	0.51	1.00
lv10 (cheat)	0.900	10.0	0.98	1.00	0.52	1.00
Group B: one scripted package under a cap						
worker cap 12	0.000	0.0	0.67	0.00	0.71	0.00
worker cap 16	0.506	5.5	0.71	0.35	0.73	0.33
worker cap 22	0.567	5.9	0.75	0.35	0.70	0.24
worker cap 30	0.578	6.1	0.75	0.35	0.72	0.24
army cap 10	0.139	1.9	0.75	0.23	0.70	0.24
army cap 20	0.322	3.6	0.75	0.31	0.70	0.24
army cap 40	0.533	5.7	0.75	0.35	0.71	0.24
army cap 60	0.561	5.9	0.75	0.35	0.70	0.24

or a test statistic. Intervals are Wilson over cells at one game per cell and a cell bootstrap where a row plays several seeds.

Calibration ladders. Before any submission was scored, we measured two ladders whose strength order is fixed in advance (Table 1 and Fig. 2), over 7,893 games. *Group A* is the game's own built-in AI at lv1–lv10, played as the subject through an observer host, a client that hosts the game and watches. The game orders lv1–lv7. lv8–lv10 are cheating tiers with extra vision or resources, ordered among themselves but not against lv7. *Group B* is one scripted package: the benchmark's own basic Protoss build, played uncapped and then under one cap at a time, a worker cap or an army-supply cap. It is not the VibeCraft Protoss package of Table 2. Both ladders reproduce the registered order (Appendix I): Group A is strictly increasing from lv1 to lv7 with Spearman 1.0, lv8 to lv10 are ordered among themselves with lv10 at 0.900, and Group B is monotone in both cap families with Spearman 1.0; lv8, which adds map vision, scores below lv7 on the outcome and on the economy and tech probes. Above lv10 the comparisons of §5.2 rest on the two tests below.

Same-seed agreement and sample size. Every cell reports its measured same-seed agreement ρ, the probability that a second replay of a seed repeats the first replay's result. Pooled, ρ is 0.986: 1.000 for the built-in AI subjects over 2,700 pairs, and 0.946 for the ladder's package and its eight capped variants over 990 pairs. Every package comparison is registered with $\rho = 0.939$, below the measured 0.946, so the tests are conservative. Let $s_i \in \{-1, 0, 1\}$ be the sign of the outcome difference between two packages in one paired game, one cell and one seed, the first-named package minus the second, and let $S = \sum_i s_i$ over the n pairs. The framework's package is named first unless the text says otherwise, so a positive S, z or per-cell difference means the framework's package won more. Under the null that the two packages are behaviorally identical, $\Pr[s_i \neq 0] = 1 - \rho$, so

$$\mathrm{Var}[S \mid H_0] = n(1-\rho) \quad \text{and} \quad n_{\min}(\delta) = \frac{(1-\rho)(z_\alpha + z_\beta)^2}{\delta^2}, \tag{1}$$

where δ is the expected per-cell sign and z_α, z_β the normal quantiles of level and power; Appendix F derives it. Two packages that play different openings disagree on more cells than one package disagrees with itself, so every comparison also measures its own disagreement rate: ρ sets the sample budget before a comparison is run, and every pair is afterwards judged at the larger of $1 - \rho$ and its own Wilson upper bound. The sign test separates two packages when $|z| \geq 1.645$ in either direction, a two-sided level of 0.10, with the sign of S giving the direction. This *sign test* fixes the sample budget by Eq. 1, but it is not the test of record, because its identical-behavior null can undercount variance for two packages of equal strength but different behavior. Every decision is therefore

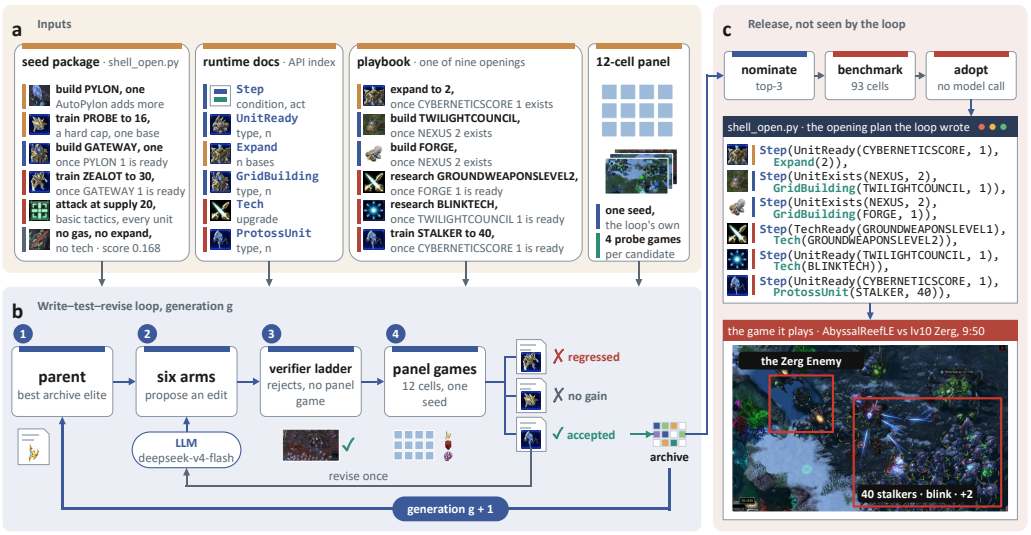

Figure 3: SC2Forge. (a) The three inputs and the development panel. (b) A generation: six arms propose edits to the parent, the ladder rejects what cannot run, survivors play the panel and take the judge's verdict, and every judged candidate enters the archive, whose best cell occupant becomes the next parent. (c) The benchmark chooses among the archive's three best entries, and the adopted package plays with no model call; beside it, the steps the loop wrote from the playbook's lines and one of its games.

also computed by a whole-cell randomization test, the *cell test*, two-sided at 0.05, since the games of one cell share its map and opponent, with a cell-bootstrap interval for the per-cell difference. We call a comparison separated only when the two tests agree. **Not separated is not the same as equivalent**: where a difference's cell-bootstrap interval lies within ± 0.05 per cell, the equivalence reading is reported beside its verdict (Appendix F). The rule's correction matters even where the measured disagreement is smallest: the framework's package and the VibeCraft Protoss package disagree on 0.082 of their 279 pairs, whose Wilson upper bound, 0.12, is above $1 - \rho = 0.061$, so the rule recomputes its own threshold from that bound in place of the registered rate: a threshold of 9.5 on S, against the 6.8 the registered ρ would have given. The observed S is 3, below either: on the comparison where it mattered most the rule raised its own threshold, and the difference did not clear it. In the other direction, on the framework's package and the coding agent, over the same 279 pairs, S is 39 at $z = 4.93$, and there the difference clears the threshold. On fresh seeds and $1{,}674$ games the two tests together falsely separate an identical pair in 0.0055 of $20{,}000$ subsets, and the near-null pair of the framework's package against VibeCraft's in 0.0087, which Appendix G reports.

Capability probes. Four probe games measure four capabilities and never enter the outcome score: *economy*, *tech and production*, *combat* and *defense*. Economy and tech are ratios of the measured quantity to the maximum over the two calibration ladders, so a package above that maximum scores above 1. Combat is the arena's value-differential share, with 0.5 as parity, and defense lies in $[0, 1]$. A score is comparable within a race, and across races it is indicative only, because the constants are the Protoss panel's. The first three order Group A lv1–lv7 with Spearman 0.982, 0.964 and 0.857. Defense does not order them, with Spearman 0.401, and is released as a diagnostic only. Appendix D gives the caps, the probe games and Table 16; Appendix K gives the symbols.

4 SC2Forge: The Generation Framework

What the loop is given. We give the loop three inputs and a development panel (Fig. 3a). The first is a seed package that builds workers and one unit type and nothing else, scoring 0.168 on the benchmark. The second is the API index: the runtime library every package here uses, sharpy-sc2 on python-sc2 (DrInfy et al., 2020; BurnySc2 et al., 2020), indexed into the calls and argument types a package may use. The third is a *playbook* of nine named openings written from VibeCraft's

Protoss bot (catmaniii, 2026): that bot's own openings as it plays them, taken as they stand rather than selected or filtered by measured strength. Each opening is a timed early sequence of structures, units and upgrades in prose, with supply-timed build orders, and with no code and no win rates. The panel is twelve development cells from the benchmark's grid, four per map, eleven at lv7–lv10 and one at lv4, on seeds the benchmark never uses.

What a generation does. A generation opens by picking a parent and a target. The parent is the archive's best entry, and the target is one of its weakest capabilities, ties rotating by generation. The loop keeps a five-number *card* for every package it has judged: the benchmark's four capabilities, measured by its own probe games, and a fifth, scouting, which records whether the package saw the opponent's base by 4:00 game time and is used only inside the loop.

The parent, the playbook, the API index and the parent's per-cell results go to six arms. Four of them are the patch-type arms: three anchored editors that each change one located region of one module, and a plan-module writer that writes one plan module from a specification. The other two rewrite a module end to end, and are the rewrite arms; the second of them, the *agentic* arm, gets a bounded tool loop. Within 24 replies and two trial games it reads files, searches symbols, runs the verifier on a staged copy, plays one game, and writes. Each arm returns a structured edit against the file's current digest. The arms differ in how reliably they propose: over the run the ladder refused 7 of the 77 proposals it faced, the two rewrite arms passed it 17 of 17 and 15 of 15 against 9 to 10 of 11 to 12 for the four patch-type arms, and 54 of the 77 edits landed in the opening plan (Appendix C).

Nothing reaches a game before the verifier ladder's six checks, which run least costly first: (1) the edit applies and parses; (2) every identifier, import and call exists in the game build; (3) the edit calls no API that crashes the game on its first tick; (4) the package still satisfies its contract, the interface the runner loads; (5) the entry module imports; and (6) a smoke game shows the edit changed the behavior, not only the code. A candidate that crashes when a panel game starts gets one repair attempt with the error text.

What survives the ladder plays the twelve development cells and the four probe games, three candidates at a time. A judge reads the new card against the parent's and returns one of three verdicts. *Accepted* means the target rose by at least 0.20 with nothing else falling more than 0.10, or two numbers rose by at least 0.20, or the twelve-cell outcome rose by at least 0.25. *Regressed* means that a card number fell by more than 0.10 or that the outcome fell by more than 0.25. The rest is *no gain*. **The search's terms are cards and outcomes**: the archive inserts every judged candidate and names an elite by quality, the card mean plus twice the outcome, so none of the judge's thresholds enters retention, parent selection or the final ranking (Appendix C). The panel's own composition does not decide an outcome verdict either: **no eleven-cell panel moves a generation-one rise across the 0.25 outcome threshold** (Appendix C). The archive is MAP-Elites (Mouret & Clune, 2015) keyed by the card, each of its five numbers binned at 0.5; every judged candidate stays in it as a lineage record, the best occupant of each capability cell is an elite whatever the verdict, and **the bin width is not one of the ranking's terms**, so no choice of grid can change which package is adopted (Appendix C). We return the generation's highest-quality rewrite to a rewrite arm once, with its per-cell results, and ask it to revise; the revision is judged against the candidate it revises, on the generation's target, and the next generation reads the archive again.

What the run delivers. The run stops at whichever of three registered caps is reached first: tokens, game seconds or wall-clock hours. When the run stops, we freeze the three highest-ranked archive entries, the nominees, ranked by quality. Each is played on the benchmark's released default, one game per cell, and the best outcome is the package delivered (§5.1). Choosing among three nominees is close to optimal on this suite: a choice made with knowledge of the measured half would have scored 0.005 higher here and at most 0.036 higher over seven runs, estimated by split halves.

5 EXPERIMENTS

5.1 THE GENERATION RUN AND ITS PACKAGE

Generation run. The framework ran from the seed package with DEEPSEEK-V4-FLASH alone, under registered caps, and stopped at the game-second check, the token cap also crossed, after five generations, 7.77M input-plus-output tokens, 497,756 game seconds and 17.1 hours. Generation

Table 2: Every agent on SC2FullGame-Bench: outcome over the 93 cells per race, three seeds per cell unless named; the Protoss package's level-equivalent, the built-in level its 90-cell outcome matches, extrapolated above lv10 (Appendix K); and its four capability scores (§3; defense is a diagnostic, marked †). The in-loop rows are TextStarCraft2 on one seed as Protoss; a dash marks a race not run or a probe not measured. The framework's Protoss row is its first run, the best of its three; every baseline row is one run, and one model, DEEPSEEK-V4-FLASH, wrote every generated package here. Games per row and the paired statistics are in Tables 3 and 5 and Fig. 4, which excludes the repaired in-loop row.

	Outcome				Capability (Protoss)			
Agent	Protoss	Terran	Zerg	Lv-eq.	Econ.	Tech	Comb.	Def.†
do-nothing control	0.000	—	—	0.0	0.41	0.00	0.71	0.00
seed package	0.168	0.147	0.211	2.2	0.46	0.20	0.71	0.00
VibeCraft	0.932	0.746	0.620	10.5	1.08	0.80	0.69	1.00
TextStarCraft2, in-loop (one seed)	0.441	—	—	5.2	0.86	0.06	—	0.00
TextStarCraft2, repaired (one seed)	0.376	—	—	4.4	—	—	—	—
iterated prompt	0.767	0.491	0.312	9.3	1.01	—	0.74	1.00
playbook-fed prompt	0.821	0.459	0.470	9.6	1.00	0.64	0.77	0.00
coding agent	0.803	0.681	0.828	9.5	1.00	0.73	0.68	1.00
playbook-fed coding agent	0.900	0.613	0.602	10.3	1.04	0.92	0.68	1.00
SC2Forge	0.943	0.853	0.857	10.6	1.02	0.72	0.74	1.00

one produced twelve proposals, two from each of the six arms; five of them passed the verifier ladder, and with the agentic arm's revision six played the panel; that revision won ten of the twelve panel games. Generations two to five accepted none as a gain: the best moved its target by 0.18 against the required 0.20. The delivered package is that generation-one revision, which the judge scored no gain, and in no run was the delivered package the archive's top entry; what the loop's search contributed is what it kept: the framework's runs are $+0.050$ per cell above the condition that archives only accepted candidates (three runs each, Table 14). Appendix B follows the run generation by generation, with generation one's cards and the per-generation counts in Tables 10 and 11.

Benchmark score. We compared the three nominees on the benchmark's released default, the basis of the adoption rule, and adopted the best; the other two score 0.925 on that basis, and the archive had ranked the adopted package third. It scores 0.943 over 279 games, 0.959 on the built-in cells, and wins 4 of 9 external games (Table 3). On the built-in cells its three maps score 0.978, 0.956 and 0.944, and its lowest level slice is 0.852, at the cheating levels lv8 and lv10 (Table 9).

Contents of the package. The adopted package differs from the seed in one file, the opening plan. The agentic arm replaced the seed's one-gateway opening with the two-base blink-stalker opening that the playbook describes (Appendix B): a second base, a Twilight Council and a Forge on two bases, blink and weapons $+2$, and stalker production to 40 before the attack. The package implements a build the playbook describes, not one the loop discovered, and the run measures what each input contributes rather than assuming the answer. The playbook supplies what to build and accounts for $+0.076$ per cell over three runs of each condition, while removing its build orders costs 0.072 per cell and inverting its timings 0.010 (§5.3). With no expert text at all the loop still reaches 0.838, against 0.914 with the playbook and the VibeCraft Protoss package's 0.932. Those runs had no opening to copy, so the loop is not confined to reproducing one it is handed.

5.2 COMPARISON WITH BASELINE AGENTS

The package is generated once and then plays with no model call: its cost is fixed, while the in-loop agent's rises with every game. The framework's Protoss run spent 3.6M billable tokens over 17.1 hours (Appendix A). The in-loop agent spends 581 model calls per game and needs 5.5–18 wall-clock seconds per game second, where a compiled package needs 0.27–1.50 seconds per game second, a median of 0.66, so that a typical game runs faster than real time. We ran TextStarCraft2 (Ma et al., 2024a) from its public repository behind DEEPSEEK-V4-FLASH, with two disclosed source patches (Appendix E); it is published for Protoss alone. It scores 0.441 over 93 cells on one seed, separated below the framework's package, and wins no game at lv8 or above. On three seeds the

verdict is the same: over 36 games on 12 cells, Simple64 at lv5–lv8 against all three races, the row scores 0.139 against the framework's 0.944, separated under both tests ($z = 5.09$) and short of 0.50. Its extractor reads an action from 33.5% of its replies; with the extractor repaired to 88%, the row scores 0.376, not separated from the score it replaces.

Prompt and coding-agent baselines. We prompted the same model for 150 rounds from the seed package, giving each round the contract, the source and the previous results, with no verifier; its best package scores 0.767, separated below the framework's package ($z = 5.89$). Given the framework's playbook, only one of ten prompting chains produced a package that starts; its first-round package scores 0.821, $+0.054$ per cell above the same prompt without the playbook, and it remains separated below the framework's package (Table 5). We ran DeepSeek Harness (Shi et al., 2026) from a clean workspace holding the seed package, a task statement and the runtime's documentation, with shell access to the game server and no playbook, verifier, benchmark cell or seed; told the framework's spend but given no budget of its own, it delivered after 11.35 hours. Its package scores 0.803, and the framework's package is separated above it ($z = 4.93$). With no playbook on either side, the loop's runs score $+0.035$ per cell above the agent's, not separated ($z = 1.32$, cell $p = 0.20$) and not equivalent (post hoc). With the playbook, under a guard at the framework's billable spend, the coding agent stopped after 4.60 hours and its package scores 0.900, $+0.097$ per cell above its version without the playbook, and it is not separated from the framework's package ($p = 0.12$, Appendix A), with the same verdict on sealed suite A; on the released suite's six seeds the framework's package is separated above it ($z = 3.80$, Table 5). The guard equalizes the model budget, and only that: the game budgets differ, the agent's 206 test games against the framework's $497{,}756$ game seconds.

VibeCraft bots. VibeCraft's Protoss bot, the source of the framework's playbook, scores 0.932, and the framework's package is not separated from it, with its point estimate $+0.011$ per cell ahead within $[-0.029, +0.050]$ ($z = 0.52$). That comparison is on the framework's best of three runs; pooled across all three, the framework's runs average 0.914 and lie 0.018 per cell below it at $z = -1.54$ with cell $p = 0.37$, so the two are not separated on the three-run basis (a sensitivity, not a registered test). Single-opening packages of this project, each opening in the seed package's skeleton, score 0.367 to 0.978 on the released default over the 90 built-in cells, one game per cell, and the framework's package is not separated from the four strongest (Appendix A). Holm's correction over Table 6 changes no verdict.

Other races. We ran the loop from a Terran and from a Zerg seed package, each with a playbook written from that race's VibeCraft bot. Changing race left the framework's runtime untouched: both runs used a runtime byte-identical to the Protoss run's, with a seed package generated from the Protoss one by literal substitution and the race's probes. The judge accepted no candidate in either race run, because the card clips each capability score at 1 on these races' probe quantities; both runs delivered through the archive's retention of every judged candidate, the mechanism that kept the Protoss package. The loop delivers 0.853 and 0.857, above its own starting point and above the VibeCraft bot of its own race (Tables 2 and 5). Both are below the framework's Protoss package, a cross-race comparison that is indicative only. The Protoss row's four baselines were also run for each race (Appendix C): the framework's package for that race is separated above seven of the eight under the adjusted selection rule, the exception being the coding agent's Zerg package (Table 5). The playbook on its own does not bring a baseline up to the framework's level: it raises the Zerg prompt by $+0.158$ per cell and lowers the Terran prompt and both coding agents, the Zerg coding agent separated. The playbook's contribution depends on how it is delivered: a baseline receives the playbook whole, at 48–70 kB, while the framework's arms receive it beside per-cell results and under the ladder, the panel and the judge.

5.3 Ablations and replicates

We removed three parts of the loop one at a time: the archive's retention, so that only accepted candidates become parents; the playbook's build orders, leaving its nine opening names and descriptions; and the playbook itself. We ran each ablated loop three times, and a further run swapped the model family; Appendix C gives every run. Each is compared as three runs against the framework's three, paired by cell, seed and run index and judged under the rule of §3 with Holm's correction over that block's four conditions, the three part-removals and the inversion. Pooled over cells each removal is separated below the framework's runs; only the build orders' loss excludes zero in the run-level t intervals of Appendix C. The run that swaps the model family reaches 0.885 with GLM-5.2 for

DEEPSEEK-V4-FLASH, separated above the coding agent at $z = 2.96$ with cell $p = 0.02$, and not separated from VibeCraft's Protoss at $z = -1.94$ with cell $p = 0.08$. The sign test alone would separate it, in VibeCraft's favor; the cell test does not, and the rule requires both. As a placebo, we inverted every timing in the playbook: each milestone moved one tier later, every wall-clock line was doubled, and the openings, actions and structures named were untouched (Appendix C). Three runs of the loop on it deliver 0.903, 0.893 and 0.918, mean 0.904, spread 0.025, their packages carrying no nominee selection, as Table 13 records: not separated from the framework's three ($+0.010$ per cell, $z = 0.82$, Holm $p = 0.49$) and separated above the runs without build orders by $+0.062$ per cell at $z = 5.03$. What the playbook supplies is what to build far more than exactly when: removing its build orders costs 0.072 per cell and getting every timing wrong 0.010. Of the 16 accepted candidates over the three runs, 7 rewrote the opening plan's gates rather than only its contents (Appendix C).

The verifier ladder's checks run without a model call; the reference check is deterministic name analysis (Khati et al., 2026). With the ladder off, 18 of the 74 candidates the judge saw crashed at their first panel game. Its refusals are of two kinds: the reference check caught edits importing a name the vendored runtime does not have, and the contract check caught an edit that breaks the interface the runner loads (Appendix B). Appendix H follows the ladder into six other environments.

5.4 GENERALIZATION TO SEALED CELLS AND OTHER GAMES

The frozen packages also played two sealed suites of 60 cells each, on maps this project had never played; the three-seed verdicts of §5.2 carry over. On both the framework's package is separated above the coding agent run without the playbook ($z = 5.25$ and 5.34 on six seeds), not separated from VibeCraft's Protoss ($z = 0.26$ and -1.34, both on six seeds), and on suite A not separated from the playbook-fed coding agent's either ($z = 1.02$ on three seeds, 0.89 on six). Every row's suite-B score stays within 0.025 of its suite-A score (Table 4); the playbook-fed row did not play suite B. **No map-dependent weakness appears on the sealed suites**: the framework's package scores 0.933 and 0.956 on suite A's two maps over three seeds each; the four rows keep one order on those two maps, and suite B reorders them (Appendix I). We ran the loop on two further games, both with references and benchmarks fixed before the runs (Appendix G). On *connect-four* the package is a single policy file, played on held-out seeds against three opponents of graded randomness; it scores 0.977, against 0.611 for the scripted reference and 0.058 for the seed. On *microRTS* (Huang et al., 2021) the package has the structure of §4, and the arms received a domain note and a four-plan playbook in prose, with no code from the reference; on three maps and the game's four opponents, at 30 games per cell, it scores 0.528 against 0.464 for the reference and 0.000 for the seed. Paired by cell and game index it is above the reference under the sign test ($S = +23$, $z = 3.00$); under the cell test over its 12 cells the difference is $+0.064$ per cell with $p = 0.09$, so separation is not claimed. The gain is concentrated on the lightRush opponent.

6 CONCLUSION

We measure one way of building a game agent with a language model: the model writes, tests and revises a program, and the program plays. SC2FullGame-Bench orders two ladders of known strength and publishes the same-seed agreement from which every comparison's sample size is planned. On it the loop's package scores 0.943 against the VibeCraft Protoss package's 0.932, the two not separated. Removing the archive's retention, the playbook's build orders or the playbook lowers the pooled score over three runs of each, and the build orders' loss holds run by run, while inverting every timing costs 0.010, the smallest of the four. With no expert text the loop's three runs still average 0.838, not separated from the coding agent's 0.803 under the same condition.

We do not establish an advantage over VibeCraft's Protoss package; the coding agent given the playbook is not separated from ours on three seeds (separated on the released suite's six seeds, equivalent on sealed suite A's six), and without the playbook it is separated below on three. The main result is on one game with two transfers, and the limitation we put first is what the package is, not how it scores: a build the playbook describes, re-timed and integrated by the loop, not a strategy the loop discovered. One thing remains untested: a population in place of the suite's fixed opponents.

AI USE STATEMENT

This work used generative AI as a tool. While writing the paper we discussed our ideas with it to settle them and to sharpen the approach; it collected the literature, checking every reference against its record except one anonymous preprint whose page could not be fetched; it gave writing and structural suggestions, made the figures and polished the writing; and we used it to review the paper for possible problems, including its numbers and its citations. We wrote the code by vibe coding: generative AI produced the harness and the analysis scripts under our direction, and we ran the experiments with them. It did not generate, impute or adjust any experimental number: every quantity is computed by released scripts from per-game records under protocols frozen before the corresponding games.

ETHICS STATEMENT

No human subjects or personal data are involved; the artifact is game records, code and protocols, and the optimized agent plays against a game's built-in AI and community bots.

REPRODUCIBILITY STATEMENT

Released: the frozen protocol of every study reported here, with its registered predictions, including the failed ones; per-game outcome and telemetry records for every benchmark, calibration and optimization game; the verifier ladder, the suite index with its 93 cell cards and four probe definitions; the frozen packages of every leaderboard row with their file digests; and every analysis script with fixed seeds. Same-seed agreement is below 1.0 for packages, so identical protocols reproduce distributions and not individual games; full-game replication requires the pinned game build and platform (StarCraft II 4.10, Base75689, Linux). The appendices hold the detail behind each section: A the comparison rows, B the run, C the ablation, replicate and race runs, D the calibration, E the in-loop agent, F the derivations, G the separation rule's validation and the two other games, H the verifier ladder in six other environments, I every registered prediction the body relies on with its outcome, J the release inventory and its status, K the symbols. Code, protocols and records are in the supplementary material. Figs. 1, 2 and 3 use unit icons from StarCraft II, © Blizzard Entertainment.

REFERENCES

Sanjeda Akter, Ibne Farabi Shihab, and Anuj Sharma. Anytime-valid answer sufficiency certificates for LLM generation via sequential information lift. *Transactions on Machine Learning Research (TMLR)*, 2025. URL https://arxiv.org/abs/2510.06478.

Anonymous. StarCraft II arena: evaluating LLMs in strategic planning, real-time decision making, and adaptability. OpenReview preprint o3V7OuPxu4, under double-blind review, 2025.

BurnySc2 et al. python-sc2: a StarCraft II bot API client library for Python 3. https://github.com/BurnySc2/python-sc2, 2020. Repository read 2026-09-17.

catmaniii. VibeCraft: voice- and text-commanded StarCraft II bots for the three races. https://github.com/catmaniii/vibecraft, 2026. MIT license; repository read 2026-09-17.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: code generation with generated tests. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2207.10397.

Yang Cheng, Weiyu Ma, Zilai Wang, Wenhui Zhu, Tangjie Lv, Yujing Hu, Mohamed Elhoseiny, Yue Deng, and Jian Zhao. EvoCurr: self-evolving curriculum with behavior code generation for complex decision-making. *arXiv preprint arXiv:2508.09586*, 2025. URL https://arxiv.org/abs/2508.09586.

Alex Deng, Ya Xu, Ron Kohavi, and Toby Walker. Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pp. 123–132, 2013.

Yue Deng, Weiyu Ma, Yuxin Fan, Ruyi Song, Yin Zhang, Haifeng Zhang, and Jian Zhao. SMAC-R1: the emergence of intelligence in decision-making tasks. *arXiv preprint arXiv:2410.16024*, 2024. URL https://arxiv.org/abs/2410.16024.

DrInfy et al. sharpy-sc2: Python framework for rapid development of StarCraft II AI bots. https://github.com/DrInfy/sharpy-sc2, 2020. Repository read 2026-09-17.

Manuel Eberhardinger, James Goodman, Alexander Dockhorn, Diego Perez-Liebana, Raluca D. Gaina, Duygu Çakmak, Setareh Maghsudi, and Simon Lucas. From code to play: benchmarking program search for games using large language models. *arXiv preprint arXiv:2412.04057*, 2024. URL https://arxiv.org/abs/2412.04057.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N. Foerster, and Shimon Whiteson. SMACv2: an improved benchmark for cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2212.07489.

Ping Guo, Chao Li, Yinglan Feng, and Chaoning Zhang. Code evolution for control: synthesizing policies via LLM-driven evolutionary search. *arXiv preprint arXiv:2601.06845*, 2026. URL https://arxiv.org/abs/2601.06845.

Lei Han, Jiechao Xiong, Peng Sun, Xinghai Sun, Meng Fang, Qingwei Guo, Qiaobo Chen, Tengfei Shi, Hongsheng Yu, Xipeng Wu, and Zhengyou Zhang. TStarBot-X: an open-sourced and comprehensive study for efficient league training in StarCraft II full game. *arXiv preprint arXiv:2011.13729*, 2020. URL https://arxiv.org/abs/2011.13729.

Daniel Hennes, Zun Li, John Schultz, and Marc Lanctot. Code-Space Response Oracles: generating interpretable multi-agent policies with large language models. arXiv preprint arXiv:2603.10098, 2026. URL https://arxiv.org/abs/2603.10098.

Chia-Yu Hsu and Shubhanshu Shekhar. Efficient sequential evaluation of large language models. *arXiv preprint arXiv:2607.17409*, 2026. URL https://arxiv.org/abs/2607.17409.

Shengyi Huang, Santiago Ontañón, Chris Bamford, and Lukasz Grela. Gym-μRTS: toward affordable full game real-time strategy games research with deep reinforcement learning. In *IEEE Conference on Games (CoG)*, 2021. URL https://arxiv.org/abs/2105.13807.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.06770.

Xinyi Ke, Kai Li, Junliang Xing, Yifan Zhang, and Jian Cheng. Game-theoretic co-evolution for LLM-based heuristic discovery. *arXiv preprint arXiv:2601.22896*, 2026. URL https://arxiv.org/abs/2601.22896.

Dipin Khati, Daniel Rodriguez-Cardenas, Paul Pantzer, and Denys Poshyvanyk. Detecting and correcting hallucinations in LLM-generated code via deterministic AST analysis. *arXiv preprint arXiv:2601.19106*, 2026. URL https://arxiv.org/abs/2601.19106.

Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. URL https://arxiv.org/abs/1711.00832.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022. URL https://arxiv.org/abs/2206.08896.

Haoran Li, Zengle Ge, Ziyang Zhang, Xiaomin Yuan, Yui Lo, Qianhui Liu, Bocheng An, Dongke Rong, Jiaqun Liu, Annan Li, Jianmin Wu, Dawei Yin, and Dou Shen. Beyond static evaluation: co-evolutionary mechanisms for LLM-driven strategy evolution in adversarial games. *arXiv preprint arXiv:2606.10389*, 2026. URL https://arxiv.org/abs/2606.10389.

Zongyuan Li, Yanan Ni, Runnan Qi, Lumin Jiang, Chang Lu, Xiaojie Xu, Xiangbei Liu, Pengfei Li, Yunzheng Guo, Zhe Ma, Huanyu Li, Hui Wu, Xian Guo, Kuihua Huang, and Xuebo Zhang. LLM-PySC2: StarCraft II learning environment for large language models. *arXiv preprint arXiv:2411.05348*, 2024. URL https://arxiv.org/abs/2411.05348.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023a. URL https://arxiv.org/abs/2209.07753.

Percy Liang, Rishi Bommasani, Tony Lee, et al. Holistic evaluation of language models. *Transactions on Machine Learning Research (TMLR)*, 2023b. URL https://arxiv.org/abs/2211.09110.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.01210.

lladdy. Chance: a StarCraft II bot that plays a random race and selects its strategy at random. AI Arena bot 117, https://aiarena.net/bots/117/; source https://github.com/lladdy/chance-sc2, 2025. AI Arena bot page, read 2026-09-15.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: improving LLM agents by automatically synthesizing a code harness. arXiv preprint arXiv:2603.03329, 2026. URL https://arxiv.org/abs/2603.03329.

Li Ma, Hao Peng, Yiming Wang, Hongbin Luo, Jie Liu, Kongjing Gu, Guanlin Wu, Hui Lin, and Lei Ren. Self-evolving multi-agent framework for efficient decision making in real-time strategy scenarios. *arXiv preprint arXiv:2603.23875*, 2026. URL https://arxiv.org/abs/2603.23875.

Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Yuqiao Wu, Runji Lin, Haifeng Zhang, and Jun Wang. Large language models play StarCraft II: benchmarks and a chain of summarization approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a. URL https://arxiv.org/abs/2312.11865.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024b. URL https://arxiv.org/abs/2310.12931.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. URL https://arxiv.org/abs/1504.04909.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: a coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu Park, Byeong-Uk Lee, Jaeyoung Hwang, Jaewoo Ahn, Ameya S. Mahabaleshwarkar, Bilal Kartal, Pritam Biswas, Yoshi Suhara, Kangwook Lee, and Jaewoong Cho. Orak: a foundational benchmark for training and evaluating LLM agents on diverse video games. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2506.03610.

Paul. Paul: a Zerg StarCraft II bot built on BurnySc2's fork of python-sc2 and sharpy-sc2. AI Arena bot 78 (Paul_2.1), https://aiarena.net/bots/78/; source https://github.com/august-k/sharpy-paul, 2020. AI Arena bot page, read 2026-09-15.

PlayerPBrief. Popo: a Zerg StarCraft II bot playing one of 15 builds per game. AI Arena bot 126, https://aiarena.net/bots/126/, 2021. AI Arena bot page, read 2026-09-15.

Runnan Qi, Yanan Ni, Lumin Jiang, Zongyuan Li, Kuihua Huang, and Xian Guo. Memory-Augmented State Machine Prompting: a novel LLM agent framework for real-time strategy games. *arXiv preprint arXiv:2510.18395*, 2025. URL https://arxiv.org/abs/2510.18395.

Anka Reuel, Amelia Hardy, Chandler Smith, Max Lamparth, Malcolm Hardy, and Mykel J. Kochenderfer. BetterBench: assessing AI benchmarks, uncovering issues, and establishing best practices. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2411.12990.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Xiao Shao, Weifu Jiang, Fei Zuo, and Mengqing Liu. SwarmBrain: embodied agent for real-time strategy game StarCraft II via large language models. *arXiv preprint arXiv:2401.17749*, 2024. URL https://arxiv.org/abs/2401.17749.

Udit Sharma. When does pairing seeds reduce variance? evidence from a multi-agent economic simulation. *arXiv preprint arXiv:2512.24145*, 2025. URL https://arxiv.org/abs/2512.24145.

Pengbo Shen, Yaqing Wang, Ni Mu, Yao Luan, Runpeng Xie, Senhao Yang, Lexiang Wang, Hao Hu, Shuang Xu, Yiqin Yang, and Bo Xu. SC2Arena and StarEvolve: benchmark and self-improvement framework for LLMs in complex decision-making tasks. *arXiv preprint arXiv:2508.10428*, 2025. URL https://arxiv.org/abs/2508.10428.

Yifan Shi, Wei Zhang, and Tianyi Cui. A programming paradigm for spatiotemporal composability. *arXiv preprint arXiv:2608.25512*, 2026. URL https://arxiv.org/abs/2608.25512. Accompanies DeepSeek Harness, https://github.com/deepseek-ai/deepseek-harness.

Sir Slimestone. Zerglord: a Zerg StarCraft II bot with editable strategies (Roach–Hydra), version 1.8.3. AI Arena bot 152, https://aiarena.net/bots/152/, 2020. AI Arena bot page, read 2026-09-15.

Peng Sun, Xinghai Sun, Lei Han, Jiechao Xiong, Qing Wang, Bo Li, Yang Zheng, Ji Liu, Yongsheng Liu, Han Liu, and Tong Zhang. TStarBots: defeating the cheating level builtin AI in StarCraft II in the full game. *arXiv preprint arXiv:1809.07193*, 2018. URL https://arxiv.org/abs/1809.07193.

Wenjie Tang, Yuan Zhou, Erqiang Xu, Keyan Cheng, Minne Li, and Liquan Xiao. DSGBench: a diverse strategic game benchmark for evaluating LLM-based agents in complex decision-making environments. *arXiv preprint arXiv:2503.06047*, 2025. URL https://arxiv.org/abs/2503.06047.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. StarCraft II: a new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017. URL https://arxiv.org/abs/1708.04782.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z.

Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16 (2):117–186, 1945.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: an open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2024. URL https://arxiv.org/abs/2305.16291.

Xiangjun Wang, Junxiao Song, Penghui Qi, Peng Peng, Zhenkun Tang, Wei Zhang, Weimin Li, Xiongjun Pi, Jujie He, Chao Gao, Haitao Long, and Quan Yuan. SCC: an efficient deep reinforcement learning agent mastering the game of StarCraft II. In *International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10905–10915, 2021.

Skyler Wu, Yash Nair, and Emmanuel J. Candès. Efficient evaluation of LLM performance with statistical guarantees. *arXiv preprint arXiv:2601.20251*, 2026. URL https://arxiv.org/abs/2601.20251.

Xuanliang Zhang, Dingzirui Wang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. Scaling laws for agent harnesses via effective feedback compute. arXiv preprint arXiv:2605.29682, 2026. URL https://arxiv.org/abs/2605.29682.

Cen Mia Zhao, Haibo Ruan, Wenjie Chen, Pei-fen Tu, Usman Abbasi, and Joel Hesch. Beyond prompts: measuring and optimizing LLM tool-agent harnesses. arXiv preprint arXiv:2609.05736, 2026. URL https://arxiv.org/abs/2609.05736.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, et al. Establishing best practices for building rigorous agentic benchmarks. *arXiv preprint arXiv:2507.02825*, 2025. URL https://arxiv.org/abs/2507.02825.

A Detail on the comparison rows

This appendix holds the comparison rows in full: the leaderboard's per-cell detail, the same comparisons restated with the cell as the resampling unit, the three further community opponents of the nine-game study, the six-seed scores, the construction of the sealed suites, the per-game cost of both kinds of agent, and the protocols of the iterated prompt and the coding agent.

Cluster-aware inference and the test seeds. Table 6 restates, with a whole cell as the resampling unit, the comparisons of seven of Table 2's packages with the framework's package: the do-nothing control, the three VibeCraft bots counted separately, the iterated prompt, the coding agent and the seed package, so the three games a cell contributes are never treated as three independent observations, with 20,000 permutations, a cell bootstrap and Holm's correction over the seven comparisons reported together. The framework's package is above the coding agent's by 0.140 per cell ($p < 0.001$, $[0.075, 0.208]$) and above the iterated prompt's by 0.176 ($p < 0.001$); its interval for the difference from the VibeCraft Protoss package, $+0.011$ with $[-0.029, +0.050]$, contains zero. The table also gives every row's score on the two seeds played after the packages were frozen: adoption used the released default seed alone, so those two seeds entered no selection, and the three nominees adoption chose between were not separated from one another ($S = +1$ each).

Other community opponents. Three further AI Arena bots, all Zerg: *Paul* (Paul, 2020), *Popo* (PlayerPBrief, 2021) and *Zerglord* (Sir Slimestone, 2020). Each played nine games against twelve of the paper's packages: every compiled package of Table 2 except the do-nothing control and the two in-loop rows, an earlier package of the playbook-fed coding-agent protocol among them, and the ablation, replicate and model-family packages behind Appendix C. The orderings below cover the eight of those twelve that also played the released suite's external cell, and the order they are compared against is their outcome there: Popo's order agrees with it at Spearman 0.80 (framework 7 of 9, VibeCraft Protoss package 8, coding agent 4, seed 3); Zerglord loses 9 of 9 to every row of Table 2 it met except the seed, which won 1 of 9, and the playbook-fed prompt, which won 6

Table 3: The table of §5.2 in full: the same outcome scores with the games per row and the split into the 90 built-in and the 3 external cells. Three seeds per cell except the in-loop rows, which play one seed as labeled and only as Protoss, the only race the published system supports; the framework's Protoss row is its first run, the best of its three runs (§5.3); every baseline row is one run, the playbook-fed coding agent's under the framework's billable spend; the Terran and Zerg packages were selected as Appendix C states, and the seed rows are the packages the three runs started from.

Agent	Race	Outcome	Games	Built-in	Ext.
do-nothing control	Protoss	0.000	279	0.00	0.00
seed package	Protoss	0.168	279	0.17	0.00
	Terran	0.147	279	0.15	0.00
	Zerg	0.211	279	0.22	0.00
VibeCraft	Protoss	0.932	279	0.95	0.44
	Terran	0.746	279	0.76	0.33
	Zerg	0.620	279	0.63	0.22
TextStarCraft2, in-loop (one seed)	Protoss	0.441	93	0.46	0.00
TextStarCraft2, repaired (one seed)	Protoss	0.376	93	0.39	0.00
iterated prompt	Protoss	0.767	279	0.79	0.11
	Terran	0.491	279	0.51	0.00
	Zerg	0.312	279	0.32	0.11
playbook-fed prompt	Protoss	0.821	279	0.84	0.33
	Terran	0.459	279	0.47	0.00
	Zerg	0.470	279	0.47	0.33
coding agent	Protoss	0.803	279	0.83	0.11
	Terran	0.681	279	0.70	0.11
	Zerg	0.828	279	0.85	0.22
playbook-fed coding agent	Protoss	0.900	279	0.93	0.11
	Terran	0.613	279	0.63	0.22
	Zerg	0.602	279	0.61	0.22
SC2Forge	Protoss	0.943	279	0.96	0.44
	Terran	0.853	279	0.87	0.33
	Zerg	0.857	279	0.86	0.67

Table 4: The comparison rows measured on sealed suite A, which is two maps that no decision used. Each row played three seeds (180 games), and the rows with a six-seed column six (360); the playbook-fed coding agent's row is the package of Table 2. The columns give the outcome and the sign statistic of the framework's package against the row, with z positive when the framework's package won more of the disagreeing games, under the rule of §3. For the rows that played the six-seed extensions the last column gives their six-seed outcome on suite A and on suite B (§5.4).

Row	Three seeds	z, framework vs row	Six seeds, suite A / B
SC2Forge (Protoss, the run)	0.944	—	0.919 / 0.908
SC2Forge, replicate 1	0.956	−0.51	—
SC2Forge, replicate 2	0.911	1.33	—
SC2Forge, no playbook (run 1)	0.889	2.10	—
SC2Forge, GLM-5.2	0.933	0.56	—
VibeCraft Protoss	0.911	1.42	0.914 / 0.939
playbook-fed coding agent	0.922	1.02	0.906 / —
coding agent	0.772	4.78	0.778 / 0.775
seed package	0.167	—	—

(Spearman 0.58); Paul, which attacks early, does not keep the order (Spearman 0.36; framework 0, VibeCraft Protoss package 6, coding agent 0, seed 0): no package here was written against an early attack (EC1 and EC3, Appendix I).

Six-seed scores on the released suite. On the released suite's six seeds the framework's package scores 0.921, the coding agent's without the playbook 0.785, VibeCraft's Protoss 0.923, and the playbook-fed coding agent's 0.853; Table 5 gives the paired statistics.

Sealed suite A, its construction and amendments. The frozen packages of the three rows first compared, the framework's, VibeCraft's Protoss and the coding agent's without the playbook, plus

Table 5: Summary of the headline comparisons: cells, seeds and games per row, the sign statistic z (framework's package first, $\rho = 0.939$ or the comparison's $\hat{q}$ upper bound), the cell test's p, the per-cell difference (framework's package minus the row), the decision under §3's rule, and whether the 90% cell-bootstrap interval lies within ± 0.05 per cell, the equivalence reading. "Both" means the two tests agree. The sealed six-seed rows take $\hat{q}$ from their pre-flight block; the three-runs-each rows pair each condition's three runs with the framework's three by cell, seed and run index, p Holm-corrected over that block's four conditions, the three part-removals and the timing inversion. Sealed suite A is AutomatonLE and CyberForestLE, suite B KairosJunctionLE and KingsCoveLE.

Framework's package vs	Cells × seeds	Games/row	z	Cell test	Per-cell diff.	Decision	Within 0.05
iterated prompt, released	93×3	279	5.89	$p < 0.001$	$+0.176$	separated (both)	no
playbook-fed prompt, released	93×3	279	4.58	$p < 0.001$	$+0.122$	separated (both)	no
coding agent, released	93×3	279	4.93	$p < 0.001$	$+0.140$	separated (both)	no
coding agent, released, six seeds	93×6	558	7.16	$p < 0.001$	$+0.136$	separated (both)	no
coding agent, sealed A	60×3	180	4.78	$p < 0.001$	$+0.172$	separated (both)	no
coding agent, sealed A, six seeds	60×6	360	5.25	$p < 0.001$	$+0.142$	separated (both)	no
coding agent, sealed B, six seeds	60×6	360	5.34	$p < 0.001$	$+0.133$	separated (both)	no
playbook-fed coding agent, released	93×3	279	1.81	$p = 0.12$	$+0.043$	not separated	no
playbook-fed coding agent, released, six seeds	93×6	558	3.80	$p = 0.005$	$+0.068$	separated (both)	no
playbook-fed coding agent, sealed A	60×3	180	1.02	$p = 0.31$	$+0.022$	not separated	no
playbook-fed coding agent, sealed A, six seeds	60×6	360	0.89	$p = 0.41$	$+0.014$	not separated	yes
VibeCraft Protoss, released	93×3	279	0.52	$p = 0.732$	$+0.011$	not separated	yes
VibeCraft Protoss, released, six seeds	93×6	558	-0.12	$p = 1.00$	-0.002	not separated	yes
VibeCraft Protoss, sealed A	60×3	180	1.42	$p = 0.112$	$+0.033$	not separated	no
VibeCraft Protoss, sealed A, six seeds	60×6	360	0.26	$p = 0.83$	$+0.006$	not separated	yes
VibeCraft Protoss, sealed B, six seeds	60×6	360	-1.34	$p = 0.07$	-0.031	not separated	no
VibeCraft Terran, released (cross-race; the framework's Protoss)	93×3	279	6.24	$p < 0.001$	$+0.197$	separated (both)	no
VibeCraft Zerg, released (cross-race; the framework's Protoss)	93×3	279	8.43	$p < 0.001$	$+0.323$	separated (both)	no
VibeCraft Terran, released (same race; the framework's Terran)	93×3	279	3.97	$p < 0.001$	$+0.107$	separated (both)	no
VibeCraft Zerg, released (same race; the framework's Zerg)	93×3	279	6.83	$p < 0.001$	$+0.237$	separated (both)	no
iterated prompt (Terran), released	93×3	279	9.25	$p < 0.001$	$+0.362$	separated (both)	no
iterated prompt (Zerg), released	93×3	279	11.66	$p < 0.001$	$+0.545$	separated (both)	no
playbook-fed prompt (Terran), released	93×3	279	9.79	$p < 0.001$	$+0.394$	separated (both)	no
playbook-fed prompt (Zerg), released	93×3	279	9.69	$p < 0.001$	$+0.387$	separated (both)	no
coding agent (Terran), released	93×3	279	5.91	$p < 0.001$	$+0.172$	separated (both)	no
coding agent (Zerg), released	93×3	279	1.12	$p = 0.32$	$+0.029$	not separated	no
Terran seed package, released	93×3	279	13.56	$p < 0.001$	$+0.706$	separated (both)	no
Zerg seed package, released	93×3	279	12.89	$p < 0.001$	$+0.645$	separated (both)	no
playbook-fed coding agent (Terran), released	93×3	279	7.13	$p < 0.001$	$+0.240$	separated (both)	no
playbook-fed coding agent (Zerg), released	93×3	279	7.47	$p < 0.001$	$+0.255$	separated (both)	no
greedy acceptance, three runs each, released	$93 \times 3 \times 3$	837	3.91	$p < 0.001$	$+0.050$	separated (both)	no
playbook without build orders, three runs each, released	$93 \times 3 \times 3$	837	5.21	$p < 0.001$	$+0.072$	separated (both)	no
no playbook, three runs each, released	$93 \times 3 \times 3$	837	5.39	$p < 0.001$	$+0.076$	separated (both)	no
playbook with inverted timings, three runs each, released	$93 \times 3 \times 3$	837	0.82	$p = 0.49$	$+0.010$	not separated	yes

the seed package added by amendment, were played on 60 further cells; the replicate, ablation, model-family and playbook-fed packages of Table 4 were played there afterwards under registered protocols (Appendix I). The cells were built the same way on two ladder maps the project has never played, AutomatonLE and CyberForestLE. Two amendments were recorded before the first game: the seed package's addition, and one setup game per package per map on a seed used once, to check that a package written against three other maps can start a game on these. No package's same-seed

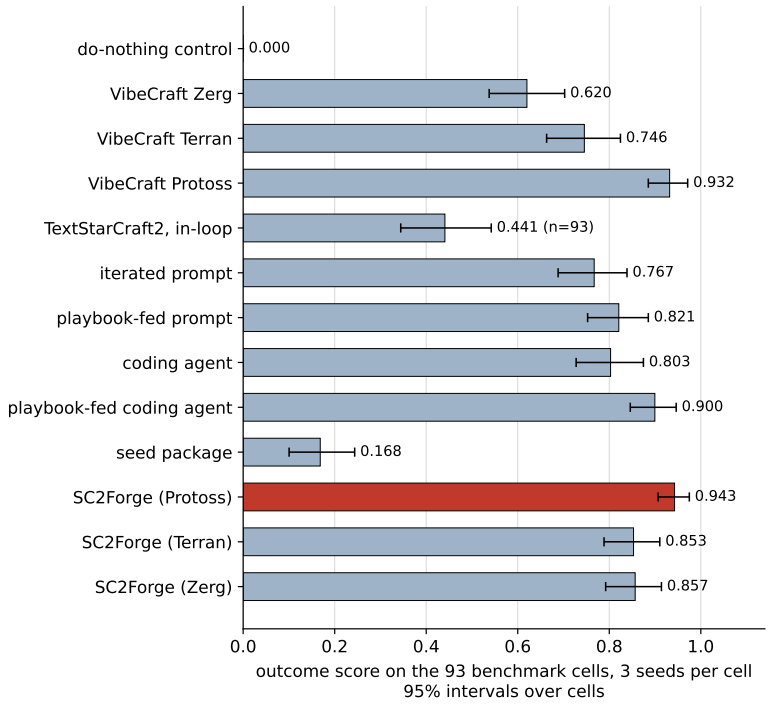

Figure 4: Outcome score on the 93 cells with 95% intervals over cells, Wilson where a row plays one game per cell and a cell bootstrap where it plays several, as §3 defines them: three seeds per cell for every compiled package and one for the in-loop agent, whose row is labeled with its game count. TextStarCraft2 plays with a model in the control loop; the iterated-prompt row is the best package from 150 rounds of prompting without a verifier; the coding-agent row is a package delivered by DeepSeek Harness; the three VibeCraft rows are that project's expert-system race bots (catmaniii, 2026); the framework's Protoss package is in red.

Table 6: Cell tests for seven rows of Table 2, and the two post-freeze seeds. Each row is compared with the framework's package by a paired randomization test that resamples whole cells, a cell's seeds together, 20,000 permutations, Holm-corrected over the seven, with a cell bootstrap of the same statistic as the interval. The test-seed column is the score on the two seeds played after every package was frozen: the framework's package scores 0.943 on all seeds and 0.946 on those alone. VibeCraft Protoss is the one of the seven not separated from it; the playbook-fed rows are compared in Table 5 and the race rows in §5.2.

Row	All seeds	Test seeds	Mean diff.	95% CI	Holm p
do-nothing control	0.000	0.000	+0.943	$[+0.907, +0.975]$	< 0.001
VibeCraft Zerg	0.620	0.624	+0.323	$[+0.244, +0.405]$	< 0.001
VibeCraft Terran	0.746	0.753	+0.197	$[+0.129, +0.269]$	< 0.001
VibeCraft Protoss	0.932	0.930	+0.011	$[-0.029, +0.050]$	0.732
iterated prompt	0.767	0.763	+0.176	$[+0.111, +0.240]$	< 0.001
coding agent	0.803	0.796	+0.140	$[+0.075, +0.208]$	< 0.001
seed package	0.168	0.177	+0.774	$[+0.695, +0.846]$	< 0.001

agreement was measured on these cells, so the three-seed comparison of Table 4 starts from the registration-time ρ, as every released-suite comparison does; the six-seed extension of §5.4 added a separate pre-flight block.

Table 7: Per-game cost of the compiled package and of the in-loop agent, measured on the benchmark's own games.

	compiled package (ours)	in-loop agent (TextStarCraft2)
model calls per game	0	581 on average
wall-clock s per game s	0.27–1.50, median 0.66 (940 games)	5.5 (vendor API) / 18 (relay)
tokens	7.77M input plus output, 3.6M billable	spent in every game

Per-game cost of the compiled package and the in-loop agent. Table 7 gives both, measured on the benchmark's own games. Both were measured on the same host, an 80-thread Intel Xeon Gold 6133, against the same pinned game build, so the ratio between them is a property of the two designs rather than of the machines they ran on. The games behind these rows take a median of 5.2 minutes of wall clock, from 3.6 at the first quartile to 10.6 at the ninth decile, over the per-game records.

A further cost is setup: getting the published in-loop agent to complete a scored game took two disclosed source patches (a dependency pin and the decision-heading pattern) and a rebuilt action index the repository does not include (Appendix E).

The scores in the two protocol paragraphs below are on the benchmark's released default over the 90 built-in cells, one game per cell, the basis on which their predictions were registered, unless a basis is named, and their verdicts follow the sign test they were registered under.

The iterated-prompt control. The same model was prompted for 150 rounds from the seed package, each round carrying the package contract, the current source and the previous round's per-seed results on one fixed cell, Simple64 against lv7 Terran, with plain file writes and no verifier (`run_bare_iterate.py` in the release). Its best-of-150 package scores 0.800 on the released default, level-equivalent 9.4: 1.000 through lv5 and 0.222 at lv10; economy 1.01, tech and production not measured (its probe game ended before the sample, Table 8), combat 0.74, defense 1.00. Against the framework's package $S = +15$ over 17 disagreeing cells ($z = 2.98$, separated); against the coding agent's $S = -4$ over 8 ($z = -1.04$, not separated).

The coding-agent protocol. The playbook-fed coding agent ran from the same clean workspace as the agent without the playbook, a workspace holding the seed package, the task statement and the runtime's documentation, plus the playbook file, under a guard that stops the run at the framework's billable spend (3.6M tokens), 420 games or 13 hours, whichever comes first, and freezes the workspace as it stands. The guard stopped the run at the token cap: 3.63M tokens, 4.60 hours and 206 games. The package scores 0.900 on three seeds, and it is not separated from the framework's on the released suite ($z = 1.81$, §5.2) or on sealed suite A ($z = 1.02$ on three seeds, 0.89 on six). It is separated below the framework's on the released suite's six seeds ($z = 3.80$, $+0.068$ per cell). DeepSeek Harness (Shi et al., 2026) was run in its headless profile from a clean workspace holding the seed package, a task statement and the runtime's documentation, with shell access to the same game server and DEEPSEEK-V4-FLASH as its model. The task statement quoted the framework's registered spend (3.6M billable tokens, counted as fresh input plus output plus a tenth of cached input) and asked for a finished package; the protocol registered that figure as recorded rather than enforced, so the agent was allowed to finish. It delivered on its own after 11.35 hours, 318 steps, 176 ledgered games and a 90-game sweep of its own design on other seeds, at 8.08M billable tokens, 2.25× the framework's spend in that same accounting. The registered prediction LA1, that the framework's package exceeds the agent's by 0.05 or is separated, holds for the delivered package (0.967 against 0.844 on the released default, separated).

We first ran the coding agent and the iterated prompt without the playbook. The playbook-fed rows of §5.2 were run afterwards, under protocols registered before their games (`knowledge_matched_baselines_v1` for the prompt, `knowledge_matched_capped_v1` for the coding agent), with the same model, seed package, task statement, runtime documentation and game server, plus the playbook file the framework read. The iterated prompt's playbook-fed package is the first-round package of one of ten prompting chains (independent 150-round sequences): the other nine produced only packages that failed to start (a wrong argument type in a

Table 8: Capability scores for the rows of Table 2 below; the eight race-mirrored baselines and the two race seed packages are not scored. Economy and tech are ratios to the calibration ladders' maximum, Protoss constants, so the scores are indicative only across races; a score above 1 exceeds that maximum; combat is the arena's value-differential share, defense the fraction of workers kept, a diagnostic (§3) marked †. Two scores have a floor above zero: the do-nothing control scores 0.41 on economy, 0.71 on combat, and above that floor the combat probe does not order the packages here. – = not measured: the in-loop row's combat probe, since a text-action agent cannot control the army the arena spawns for it, the loop's Terran and Zerg packages, and the iterated prompt's tech probe, whose game ended at 5:32, before its 8:00 sample.

Agent	Econ.	Tech	Comb.	Def.†
do-nothing control	0.41	0.00	0.71	0.00
VibeCraft Zerg	1.26	1.08	0.69	1.00
VibeCraft Terran	1.05	1.15	0.63	1.00
VibeCraft Protoss	1.08	0.80	0.69	1.00
TextStarCraft2, in-loop (one seed)	0.86	0.06	–	0.00
iterated prompt	1.01	–	0.74	1.00
playbook-fed prompt	1.00	0.64	0.77	0.00
coding agent	1.00	0.73	0.68	1.00
playbook-fed coding agent	1.04	0.92	0.68	1.00
seed package	0.46	0.20	0.71	0.00
SC2Forge (Protoss)	1.02	0.72	0.74	1.00
SC2Forge (Terran)	–	–	–	–
SC2Forge (Zerg)	–	–	–	–

call the playbook names), which the prompt's per-seed feedback cannot report, so the row measures one implementation of the playbook rather than 150 rounds of iteration.

Single-opening packages. Nine hand-written packages of this project, each implementing one opening in the seed package's skeleton, score 0.367 to 0.978 on the released default over the 90 built-in cells (one game per cell), and the framework's package is not separated from the four strongest on that basis. The VibeCraft Protoss package of Table 2 is a different, multi-module package drawn from the open-source project VibeCraft (catmaniii, 2026), registered as the comparison's reference row before the framework's run; its released-default score on the 93 cells is 0.935.

B THE FRAMEWORK'S RUN IN DETAIL

This appendix follows the run generation by generation: what each of the six arms proposed, what the ladder rejected and on which check, the verdict every judged candidate received, the archive it entered, the three nominees, the files the run changed, and what the run spent.

Candidates. Generation one produced twelve proposals, two from each of the six arms. The four patch-type arms (three anchored editors and the plan-module writer) produced eight of them, and they split three ways: one failed at the proposal stage, six were rejected by the reference check for an identifier or import that does not exist in the vendored runtime, and the remaining anchored edit passed. All four rewrite candidates, two module rewrites and two agentic rewrites, also passed and played the panel, and the agentic arm's revision is the package the run adopted. Generations two to five produced 48 logged attempts and accepted none as a gain: the best moved its target dimension by 0.18 against the rule's 0.20; the archive kept every judged candidate as a lineage record (42 over these generations) and the elites added over these generations occupied two capability cells. The game-second check was triggered before the revision candidate of generation five, with the token cap already exceeded by then, because the driver checks the caps at generation and judging-batch boundaries rather than after every game. Table 9 splits the adopted package's outcome on the 90 built-in cells three ways, by the opponent's level, by its race and by map: it wins every game at lv1–lv4, lv6 and lv9, and its lowest slice is 0.852, at the cheating levels lv8 and lv10.

Table 9: The framework's adopted package on the 90 built-in cells over the released suite's three seeds, split three ways, by opponent level, by opponent race and by map. Every slice is three seeds in each of its cells: nine cells and 27 games for a level, thirty cells and 90 games for an opponent race and for a map. The three panels partition the same 270 games.

Slice	Outcome	Games
opponent level		
lv1	1.000	27
lv2	1.000	27
lv3	1.000	27
lv4	1.000	27
lv5	0.926	27
lv6	1.000	27
lv7	0.963	27
lv8 (cheat)	0.852	27
lv9 (cheat)	1.000	27
lv10 (cheat)	0.852	27
opponent race		
Protoss	0.967	90
Terran	0.933	90
Zerg	0.978	90
map		
Simple64	0.978	90
AbyssalReefLE	0.956	90
AcropolisLE	0.944	90

Table 10: Generation one's judged candidates, with the panel card, five dimensions, the panel outcome over the twelve development cells, the judge's verdict, read against the parent's card and outcome under the rule of §4, a revision against the candidate it revises, and, for the three of this generation that played the benchmark, their score on the released default. Quality is the card mean plus twice the panel outcome; card values come from the loop's own probe games on the development seed, not the benchmark's probes, which is why the seed package's defense scores 1.00 here and 0.00 in Table 8. The last row is the package the run delivered; the run's other two nominees come from later generations and are in the Nominees paragraph below.

Candidate	Touched	Econ.	Tech	Combat	Def.	Scout	Panel	Quality	Verdict	Bench.
seed	—	0.46	0.20	0.74	1.00	0.00	2/12	0.82	—	—
anchored edit	entry	0.46	0.20	0.75	0.00	1.00	1/12	0.65	regressed	—
module rewrite (1)	entry	0.48	0.20	0.75	1.00	1.00	0/12	0.69	accepted	—
module rewrite (2)	opening	0.98	0.52	0.73	1.00	1.00	6/12	1.85	accepted	0.806
agentic rewrite (1)	entry	0.46	0.20	0.74	0.00	1.00	1/12	0.65	regressed	—
agentic rewrite (2)	opening	0.94	0.71	0.70	0.00	1.00	8/12	2.00	regressed	0.892
agentic rewrite (2), revised	opening	1.00	0.72	0.76	1.00	1.00	10/12	2.56	no gain	0.935

Verdicts and the archive. Table 10 lists the six judged candidates of generation one and the seed. The adopted package was judged *no gain*: its target dimension, scouting, was already at 1.000 in its parent, and the rule's outcome clause requires a gain of 0.25 where the revision gained 0.167. It entered the archive as the elite of a new capability cell (each card number binned at 0.5), $(1, 1, 1, 1, 1)$, and became the parent of generation two by quality. Measured again on the panel as generation two's parent, its card read 0.74 on tech and 0.73 on combat, with nine of the twelve panel games won, so generation two targeted combat. The second agentic candidate was judged regressed (defense $1.00 \rightarrow 0.00$) and was likewise retained as the occupant of its cell.

Failure modes, generation by generation. The same logs record where each attempt stopped. Table 11 gives the run generation by generation: attempts, how many passed the ladder's game-free checks, the proposal-stage failures, the verifier's refusals, the judge's verdicts, and the range of the target dimension's gain against the $+0.20$ it asks for. After generation one no acceptance failed for want of a proposal: by generations four and five every attempt passed the ladder and the target dimension kept moving, to $+0.183$ in generation five, but no candidate reached the $+0.20$ rise, or the $+0.25$ outcome rise, that the judge requires. The verdicts are therefore no gain; the

Table 11: Failure modes of the framework's run, generation by generation, from its own logs: attempts and how many passed static verification, where the rest stopped (a stage the arms could not deliver, or a verifier check), the judge's verdicts, the duplicates among the attempts, and the range of the target dimension's measured gain against the $+0.20$ the judge asks for. **A generation's verdicts include its one revision**, which is not one of the twelve attempts, so, counting duplicates as verdicts, a generation that revised a candidate shows one more verdict than it has static passes, while generation five, which revised none, shows twelve; the rule holds wherever the log recorded that generation's judging. The static column counts the verifier's five game-free checks; a candidate that also passes the smoke game is judged.

Gen.	Target	Att.	Static	Prop.	Verif.	Acc.	No gain	Regr.	Dupl.	Target gain
1	scouting	12	5	1	6	2	1	3	0	0.000–0.000
2	combat	12	12	0	0	0	11	1	1	−0.003–0.049
3	tech	12	9	2	1	0	8	2	0	0.000–0.117
4	tech	12	12	0	0	0	11	2	0	−0.108–0.178
5	tech	12	12	0	0	0	5	2	5	0.014–0.183

Table 12: Every entry the archive retained, by the arm that proposed it: how many entries, which generations, the best archive quality and the best panel outcome among them, and what the arm writes, under the framework's own names: three anchored editors that write one located region of the opening plan, one plan-module writer that writes that file from a specification, and two rewrite arms, the second with a bounded tool loop. The archive held 48 entries; the adoption rule ranked them by quality and the three nominees were played on the benchmark.

Arm	Entries	Gens.	Best quality	Best panel	Kind
curriculum_pcc	5	1,2,3,4	2.702	0.917	anchored edit
playbook_rewrite	10	1,2,3,4,5	2.566	0.833	module rewrite
agentic_rewrite	14	1,2,3,4,5	2.563	0.833	agentic rewrite
staged_direct	6	2,3,4,5	2.555	0.833	anchored edit
sc2forge_gen	7	2,3,4,5	2.402	0.750	plan-module writer
pcc_fg	6	2,3,4,5	2.394	0.750	anchored edit

remaining failure modes are proposal-stage ones, a token limit or a provider error or an anchor that no longer matches, one contract-check refusal, duplicates as the opening plan saturated (five of the twelve attempts in generation five), and regressions where a local edit broke another dimension by more than 0.10, one in generation two and two in each generation after it. Table 12 shows what the archive held in the end, by the arm that proposed each entry. Nothing in the archive is a new macro plan: every candidate the run judged touched one module, and of the proposals the reach count in Appendix C covers, only one touched anything other than the opening plan and the entry module, the combat-plan gate removal of generation two.

Nominees. At the end of the run the three entries of highest quality in the archive were frozen and played the benchmark's released default (one game per cell; on the body's three seeds the adopted package scores 0.943): the adopted package (0.935 on 93 cells, 0.967 on the 90 built-in cells; third of the three by quality) and two generation-two archive entries, both at 0.925 and neither separated from it ($S = +1$). Two other generation-one candidates score 0.922 and 0.822 on the 90 built-in cells (0.892 and 0.806 on the 93 cells, Table 10).

Files the run changed. Every candidate of the run touched one file. All but one of the 77 proposals the reach count covers touched the entry module or the opening plan module; the exception is a combat-targeted edit in generation two, which removed an attack gate from the combat plan module and passed the ladder. The adopted package differs from the seed in the opening module alone; its revision differs from the unrevised candidate in that module alone. Fig. 5 shows the adopted package's opening plan beside one of its benchmark games.

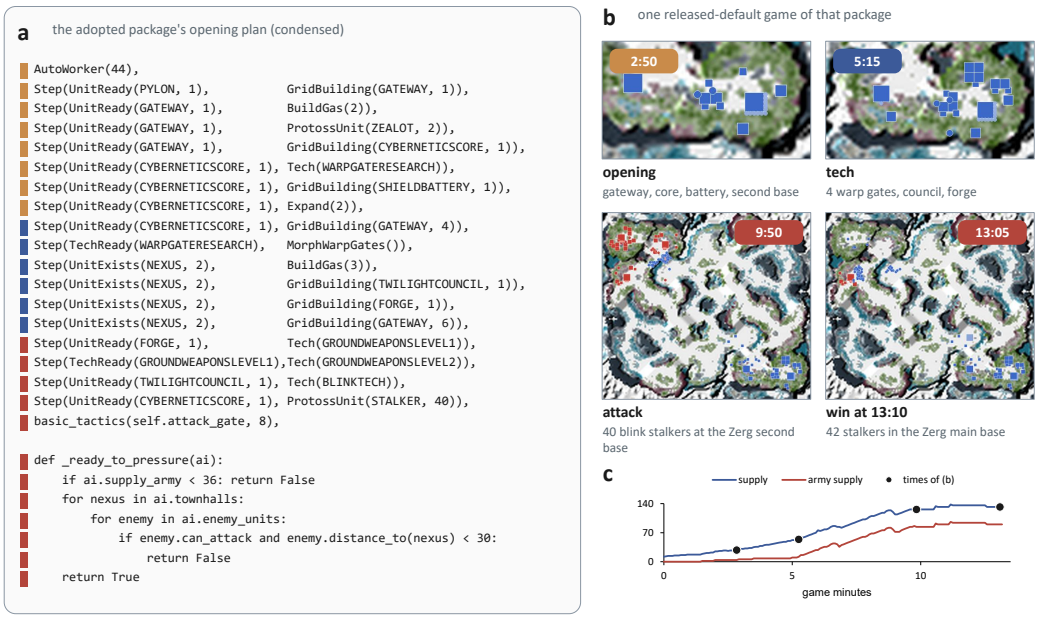

Figure 5: The adopted package and one of its games. (a) Its opening plan: the build-order steps, banded by phase, with the worker target, the tactic line and its pressure predicate (`shell_open.py`, condensed: `UnitTypeId`/`UpgradeId` prefixes, the `PROTOSS` prefix of the two weapon upgrades and priority flags removed). (b) Its released-default game on AbyssalReefLE against the built-in lv10 Zerg, drawn from the replay's tracker events at four times (own bases at 2:50 and 5:15, whole map at 9:50 and 13:05, the win coming at 13:10; squares structures, dots units, blue the package, red the Zerg). (c) Supply and army supply over that game, the four times marked.

What the ladder rejected. A proposal is one arm output, a candidate a proposal that passed the ladder and was judged. The ladder ran on all 77 proposals of the run that produced a patch and rejected 7 of them. One proposal here is one entry in the run's pool files, so a retry inside an arm, and two arms that independently wrote the same wrong patch, count separately; deduplicating by patch content would give 71 faced and 4 rejected, and the tables above count the logs' attempts instead, twelve a generation, which is the smaller basis. 6 are the same mistake: a scouting edit importing a name the vendored runtime does not have, `from sharpy.plans.acts import WorkerScout`, which the reference check reports as `name not found` and the contract check then fails at import, so those 6 failed both checks. 1 failed the contract check alone, a tech edit adding `Tech(UpgradeId.BLINKTECH)` behind a Twilight Council. The check that refuses an API which would crash a game on its first tick rejected nothing here; its case is Appendix H, where three published packages reference an upgrade that is in the build's table without a research ability.

Cost. The run spent 7.77M tokens as input plus output, 497,756 game seconds and 17.1 hours against caps of 7.21M tokens, 469,280 game seconds and 25.9 hours, so the token and game-second caps were crossed by under a tenth each and the wall-clock cap was not reached. The benchmark evaluations are outside the caps.

C THE ABLATION, REPLICATE AND RACE RUNS IN DETAIL

This appendix gives every run behind §5.3 and the questions those runs raise: the three part-removals and the timing inversion with their run-level spreads, the replicate and race runs, the second model family, the four baselines run for each race, what the loop did with the inverted timings, the archive's bin width, what the arms proposed and where their edits landed, and the judge's thresholds.

Table 13: The twelve ablation runs of §5.3: spend at the stop, all runs stopped at the game-second cap, the adopted candidate, its score on the released default over the 90 built-in cells, the basis the run's own verdict used, and on the comparison's three seeds over the 93 cells, the numbers §5.3 pools. Run 1 of each condition is the run described below; an anchored edit is a located patch from one of §4's three anchored editors. **The three inverted-timing runs depart from the registered adoption rule**: they delivered the archive entry their ranking put first rather than the best of three nominees on the released default, which Appendix I records with its direction; their released-default column is that package's own game at the released seed.

Condition	Run	Tokens (M)	Game s	Hours	Adopted candidate	Released default	Three seeds
Greedy acceptance	1	4.19	485,768	13.1	agentic rewrite, gen. 4	0.933	0.864
	2	3.31	482,495	14.5	agentic rewrite, gen. 5	0.833	0.821
	3	3.50	470,783	11.2	agentic rewrite, gen. 4	0.978	0.907
Playbook without build orders	1	1.73	478,105	14.7	revised agentic rewrite, gen. 2	0.911	0.857
	2	1.83	476,451	15.7	agentic rewrite, gen. 4	0.878	0.835
	3	2.43	470,866	12.8	module rewrite, gen. 6	0.822	0.835
No playbook	1	2.70	479,431	15.9	anchored edit, gen. 6	0.911	0.878
	2	2.32	477,766	15.7	anchored edit, gen. 6	0.889	0.835
	3	1.96	470,794	13.0	agentic rewrite, gen. 6	0.800	0.799
Playbook with inverted timings	1	2.15	477,168	6.48	agentic rewrite, gen. 2	0.933	0.903
	2	3.55	488,493	9.09	anchored edit, gen. 6	0.922	0.893
	3	4.34	472,003	10.54	anchored edit, gen. 4	0.956	0.918

Table 14: The three part-removals and the timing inversion of §5.3, three runs each: three-seed outcomes over the 93 cells, against the framework's own three runs (§5.3). The z is pooled over games; the run-level t intervals, which exclude zero for the build-orders removal alone, are in Appendix C.

Change to the loop	Three runs	Mean	Framework above	z (Holm p)
greedy acceptance	0.864, 0.821, 0.907	0.864	+0.050	3.91 (< 0.001)
playbook without build orders	0.857, 0.835, 0.835	0.842	+0.072	5.21 (< 0.001)
playbook	0.878, 0.835, 0.799	0.838	+0.076	5.39 (< 0.001)
playbook with inverted timings	0.903, 0.893, 0.918	0.904	+0.010	0.82 ($= 0.49$)
none (the framework's runs)	0.943, 0.910, 0.889	0.914	—	spread 0.054

Selection on the released suite. For 7 of the runs with a nominee list, among them the framework's run, its two replicates and the first run of each of the three ablations of §5.3, the nominee was chosen on a random half of the 90 built-in cells and measured on the other half, 2,000 splits; the in-sample best on the measured half exceeds the chosen nominee's score there by 0.005 (framework's run), 0.023 and 0.016 (its replicates), 0.036 at most (greedy acceptance). Every run's nominees were scored on one seed.

Sample size from Eq. 1. By Eq. 1 at $\alpha = 0.05$ one-sided and 80% power, resolving a per-cell difference of 0.04 at a typical measured $\hat{q}$ upper bound of 0.15 needs 580 paired games, 6.2 seeds per cell on 93 cells; resolving 0.03 needs 11.1; the three-seed design's 279 pairs resolve 0.058 at that bound. The six-seed extension of the released suite supplied 558 paired games for the rows it covered.

A score in this appendix is on the benchmark's released default, one game per cell over the 93 cells, unless it is labeled three seeds, a cell count or a sealed suite. Verdict tallies in this appendix count distinct candidates (a run's ledger records each revision id twice), and archive counts exclude the seed package the run started from. Every adopted package below was replayed on the comparison's three seeds under a protocol frozen before its first game; §5.3 reports those numbers, and Table 13 lists the twelve ablation runs' spend and adopted packages.

Greedy acceptance. Only accepted candidates are archived. The run made eight generations (4.19M tokens, 485,768 game seconds, 13.1 hours), judged 53 candidates (5 accepted, 33 no gain, 11 regressed, 3 crashed, 1 duplicate) and archived five entries. Its three nominees played the panel at 9/12 each and scored 0.933, 0.922 and 0.933 on the 90 built-in cells. The adopted one, the agentic rewrite at 0.933 (released default over those cells, one seed; the tie with the module rewrite broken by archive rank, as the adoption rule prescribes), clears the sign test against the coding agent's pack-

age ($S = +8$, $z = 1.71$), for which no cell test was run, and is not separated from the framework's ($S = +3$ over 5 disagreeing cells) on that basis; on three seeds it is 0.864, the first of the three greedy runs of §5.3.

No playbook. The loop ran with the documentation and no playbook. It made six generations and reached the game-second cap before generation seven's candidates were judged (2.70M tokens, 479,431 game seconds); 5 candidates were accepted. The three nominees played the panel at $9/12$, $8/12$ and $8/12$, and scored 0.892, 0.849 and 0.796 on the released default over the 93 cells. The adopted one is an anchored edit from generation six: it scores 0.911 on the 90 built-in cells and 0.878 on three seeds, the first of the three no-playbook runs of §5.3. On those 90 built-in cells it equals the adopted package of the first run without build orders, at 0.911. The plan that package plays was edited by the run's own arms: this run had no playbook text at all, the adopted one is an anchored edit of the seed package's opening plan in generation six, and its two other nominees are anchored edits of that same plan. Its card, read from the loop's own probe games, is economy 1.00, tech and production 0.88, combat 0.74, defense 1.00 and scouting 1.00 (§4). The build it plays was therefore written into that plan during the run rather than copied from an expert's text, which is the sense in which the loop is not confined to reproducing an opening it is handed (§5.1).

Playbook without build orders. The playbook is stripped of every supply-timed line by the released stripping script; the notes and the API index are unchanged. The run made seven generations (1.73M tokens, 478,105 game seconds, 14.7 hours; 2 accepted) with a best panel result of $8/12$. Nominees: a revised agentic rewrite ($7/12$ on the panel; 0.911 on the 90 built-in cells), an anchored edit ($8/12$; 0.900) and the unrevised agentic rewrite ($8/12$; 0.867). The adopted package (0.911 on the released default over those cells, one seed; 0.857 on three seeds, the first of the three runs without build orders of §5.3) is not separated from the framework's ($S = +5$ over 9, $z = 1.24$) or from the coding agent's ($S = +6$, $z = 1.35$) on that basis. Two of the three nominees rank differently on the panel and on the benchmark.

The run with the verifier ladder off. The run whose crash count §5.3 cites made seven generations to the game-second cap at 1.97M tokens; of the 74 candidates the judge saw, 18 crashed at their first panel game.

Second model family. The GLM-5.2 run, under the Protoss run's caps, made 13 generations and stopped at the wall-clock cap, checked at generation boundaries, after 26.5 hours, at 2.19M tokens and 81% of the game-second cap; the judge saw 62 candidates, 13 accepted, 7 no gain, 20 regressed, 7 crashed and 15 duplicates. Its adopted package, an anchored edit from generation twelve, scores 0.914 on the released default over the 93 cells, 0.922 on the 90 built-in cells, and 0.885 on three seeds (§5.3).

Framework replicates. Replicate one made seven generations and stopped at the game-second cap after 8.73 hours and 3.72M tokens; its three nominees scored 0.935, 0.946 and 0.957 on the released default over the 93 cells and the adopted one, a revised agentic rewrite ranked third by archive quality, reads 0.957 there and 0.910 on three seeds. Replicate two made six generations and stopped at the game-second cap before generation seven, after 14.7 hours and 2.86M tokens. Its nominees scored 0.925, 0.946 and 0.914, and the adopted one, an anchored edit ranked second by archive quality, scores 0.946 on the released default and 0.889 on three seeds. The three ablations' run-level spreads, 0.086, 0.022 and 0.079, are of the same size as the effects in §5.3, against the framework's own 0.054: a three-against-three t interval on the run means excludes zero only for the playbook's build orders, $[+0.017, +0.127]$.

Terran and Zerg runs. Each run took the Protoss run's protocol and caps with three inputs replaced: a seed package of that race, generated from the Protoss seed package by literal substitution (sixteen workers, one production building and its one unit type, nothing else); a playbook written from that race's VibeCraft package; and the probe games' unit compositions. The card's four normalization constants stayed those of the Protoss calibration ladders, and the loop's card clips each capability score at 1, so on these races' probe quantities the card reads 1.0 and no candidate can show the 0.20 rise the judge requires. Neither run accepted a candidate under that rule (the Terran run made five generations, the Zerg run six, each to its game-second cap), and both filled their

archives with entries kept for occupying a new capability cell, the same mechanism that kept the Protoss run's adopted package. On the released default over the built-in cells the package each run delivered scores 0.822 (Terran) and 0.867 (Zerg). On the comparison's three seeds each is separated above the VibeCraft package of its race (§5.2). The two seed packages the runs started from were measured on the same three seeds and the same 93 cells, 279 games each: they score 0.147 and 0.211 (Table 2).

Baselines for the two races. The iterated prompt and the coding agent of §5.2 were run for Terran and Zerg with the Protoss protocols mirrored: the race's seed package, its entry file and its race in the task statement, and otherwise the same prompts, caps and selection; the playbook-fed variants received the playbook the framework's run of that race read. Without the playbook, the packages the registered selection rules returned for Zerg were poor, far below their Terran counterparts, and the rules the Protoss rows had used did not hold for them: the iterated prompt's Zerg run judged 142 candidates, of which 31 crashed at import and 107 raised an exception in their first step, and the registered rule, most cheap-block wins with ties broken by order, returned one that fails at import, so that row scored 0.000 over 279 games that ended in an import error. The coding agent's Zerg run stopped after 38 minutes with its package left mid-rewrite, and its package resigned at the first step of every benchmark game. We attribute the weakness to two causes we cannot separate with these runs: the Zerg build differs in kind from the other two races (production through larvae and injects, structures by morphing drones, a tech tree the runtime library exposes through different calls), and the model's knowledge of that build without a playbook is thin. We therefore adjusted the selection so that each baseline row is measured on a package that plays. For the iterated prompt the rule is most cheap-block wins, then no crash in the cheap block, then the longest six games as measured by their replay files, then order. For the coding agent it is the last version the agent itself tested on the game server. The adjusted rule is applied to every race baseline row, including the playbook-fed ones, and the packages the registered rules had returned stay in the records with their scores. A per-cell difference is the mean over the cells the two rows share, computed from unrounded scores, so it can sit a unit in the last decimal away from the difference of the two rounded scores printed in Table 2. Against their own versions without the playbook the four race baselines move as follows, 279 pairs each: the Zerg prompt $+0.158$ per cell ($S = +44$, $z = 4.83$, cell $p < 0.001$, separated), the Terran prompt -0.032 ($S = -9$, $z = -1.5$, cell $p = 0.20$, not separated), the Terran coding agent -0.068 ($S = -19$, $z = -1.93$, cell $p = 0.16$, not separated) and the Zerg coding agent -0.226 ($S = -63$, $z = -6.21$, cell $p < 0.001$, separated) — the separation §5.2 reports for that row. The eight rows' three-seed scores are in Table 2 and their comparisons in Table 5. The coding-agent runs delivered after 4.4 hours for Terran, 38 minutes for Zerg, 1.3 hours for Terran with the playbook and 3.0 hours for Zerg with the playbook, each the last version the agent itself tested. The playbook-fed Zerg prompt is the second of two runs, the first having produced no candidate that played past its first step.

The misleading playbook, in detail. The playbook ablation above removes the playbook's build orders; this study keeps every line and gets the timings wrong. We generated the perturbed playbook from the shipped one mechanically, with the generator refusing to write a file unless five invariants held: the line and trigger counts unchanged (1084 lines, 272 triggers), the shift monotone so that no prerequisite moves earlier and no trigger can fire sooner than written, no line newly gated on the structure it is the sole source of, every new prerequisite named in the same opening, and the shift exactly invertible. 222 lines changed and 34 were held back by the self-gate guard. The shifts are monotone because two earlier designs were rejected for turning wrong timings into absent timings: a global ladder cascaded and collapsed eight of one opening's eleven milestones onto a single clause, and a per-opening rotation wrapped a third base's gas onto the first Gateway.

The run stopped at its own game-second cap before generation 6, at 6.48 hours of a 25.92-hour cap, having spent $2{,}153{,}009$ tokens and $477{,}168$ game seconds, with 5 generations and 48 archive entries. Its verdicts were 8 accepted, 30 no gain, 10 regressed, 3 crashed and 1 duplicate. The adopted package is `g02.agentic_rewrite`, from the agentic rewrite arm on the defense target.

Two further runs of the same loop on the same perturbed playbook, registered with the same caps before their first game, stopped at the same game-second cap: 9.09 hours at $3{,}553{,}020$ tokens and $488{,}493$ game seconds over 7 generations, adopting `g06.pcc_fg`; and 10.54 hours at $4{,}337{,}691$ tokens and $472{,}003$ game seconds over 7 generations, adopting `g04.staged_direct.cca15c`.

We then benchmarked the first run's package on the basis of Table 14: 279 pairs over 93 cells and three seeds, Tie 0. It scores 0.903 over all cells, 0.911 over the 90 built-in cells and 0.667 over the 3 external ones, winning 6 of 9 head-to-head matches against the community bot. By level it holds 1.000 through lv5, then 0.889 at lv6, 0.963 at lv7, 0.889 at lv8, 0.815 at lv9 and 0.556 at lv10; by opponent race 0.933 against Protoss, 0.833 against Terran and 0.967 against Zerg. For comparison the same basis gives the playbook-without-build-orders condition 0.842 (its runs 0.857/0.835/0.835), the no-playbook condition 0.838 (0.878/0.835/0.799) and the framework 0.914 (0.943/0.910/0.889). The condition's three runs average 0.904 with a spread of 0.025 (0.903, 0.893, 0.918): 0.010 per cell below the framework's three and not separated from them under the rule of §3 ($z = 0.82$, Holm $p = 0.49$, its 90% interval inside ± 0.05, the equivalence reading), and separated above the playbook-without-build-orders condition by $+0.062$ per cell ($z = 5.03$) and above the no-playbook condition by $+0.067$. The condition's spread is under half the framework's own 0.054.

What the loop did with the wrong timings. Comparing each accepted package with its parent's: 7 of the 16 accepted candidates over the three runs (4 of 8, 2 of 5, 1 of 3) rewrote the opening plan's gates rather than only its contents. Counted from the patch records alone rather than from the packages, the same count is 3 of 8; the number above is the packages' for all three runs. Several wrote explicit time gates of their own, among them `Time(0)`, `Time(240)` and `Time(480)`. That is the loop supplying the timing the playbook no longer supplies. One caveat belongs here: the single clearest re-timing, the same action moved from a Gateway prerequisite to a Pylon one, is in a candidate the judge scored no gain, so the accepted candidates mostly *added* gates rather than re-pointing the milestones they were given.

The archive's bin width. The archive keeps **every judged candidate** as a lineage record, and the run's final selection ranks those records by quality, the card mean plus twice the panel outcome. The bin width is not one of that ranking's terms, so no choice of grid can change which package is adopted. What the grid governs is which entry becomes the elite of a capability cell; the parent, the highest-quality entry, is the same under any grid. We examined the alternatives on the 48 entries the run archived, and the measurement says coarser is right at this sample size: at 0.5 bins the entries occupy 4 cells, the largest holding 43, and 1 entry is alone in its own cell, 2.1% of the archive; at 0.25 bins they spread over 9 cells, at 0.1 over 10, and at 0.05 over 26 with 12 entries alone, 25.0%. A grid fine enough to make a quarter of the archive's cells singleton elites removes the concentration the archive exists to provide, so refining it would have weakened the mechanism rather than sharpened it, and it would not have moved the answer either way.

What the arms' edits look like. In the inverted-timing run, the patch records classify each edit by what it touches. Of the 34 proposals delivered as whole-file rewrites, 26 changed the opening plan's gate structure rather than only what is built inside it; the rest altered contents. The remaining proposals arrived as targeted replacements against a recorded digest. The verdicts over the run were 8 accepted, 30 no gain, 10 regressed and 3 crashed, and the crash class needs a definition: a package that dies while being constructed reports a loss before any strategy runs, which is what the ladder's smoke game exists to catch (Appendix B).

The judge's thresholds. The three numbers the judge's verdict turns on, a rise of 0.20 in the target with no dimension falling more than 0.10, or of 0.20 in two numbers at once, or a twelve-cell outcome rise of 0.25, are not parameters of the search. The archive cannot see them. Its insertion takes a candidate and a card and nothing else: every judged candidate is written in as a lineage record, and the elite of a cell is chosen by comparing quality. Quality is the mean of the scored card dimensions plus twice the outcome, and the verdict is not one of quality's terms. The parent of a generation is the highest-quality elite, so the parent sequence is fixed by cards and outcomes alone, and adoption ranks the archive by the same quality. For a given set of judged candidates, moving the thresholds cannot change which are retained, which are elites, which become parents, or which is delivered; it changes the labels in the ledger and the wording of the feedback the next generation is shown, and through that feedback what later generations propose, which we did not measure. The one verdict that does gate anything is the ladder's crash check, which is not one of these three numbers. The same holds for the two paths that do spend more on a candidate. The revision base is the highest-quality rewrite of the generation, chosen by quality and not filtered by verdict, so a

candidate judged no gain or regressed can be the one revised, and the repair path is entered by a verification failure, which belongs to the ladder rather than to the judge.

What the arms contributed, and how far their edits reached. In the framework's Protoss run, the verdicts above say how often a candidate was accepted; this says where the accepted and rejected proposals were edited, which is the question of whether the loop reaches past the opening. Over the generations the verifier ladder rejected 7 of the 77 proposals that produced a patch and faced it, and the arms differ in how reliably they propose: the two rewrite arms are the most dependable, at 17 **of** 17 and 15 **of** 15, against 9 to 10 of 11 to 12 for the four patch-type arms.

Where the edits landed is uneven. 54 proposals edited the opening plan, 22 the entry module, and exactly one edited the combat plan. That is the sense in which the loop's reach is narrower than its target selection: the generations tasked with combat and with scouting still edited the opening plan in 9 of 14 and 4 of 13 proposals, so the loop picks a weak capability and then mostly rewrites the opening that capability is read from.

The reach is narrow, and it is not zero. The one candidate that edited neither the opening plan nor the entry module was a staged-direct edit in generation two, and it removes an attack gate from the combat plan: `Step(attack_gate, PlanZoneAttack(attack_requirement))` becomes a bare `PlanZoneAttack(attack_requirement)`. It passed the verifier ladder, so a control-layer edit that changes when the bot attacks is a thing this loop can produce and accept. It is not the adopted package, and it is one edit in the 77 proposals the reach count covers, which is why the paragraph above states the reach as measured rather than as a capability.

Whether the panel's composition decided anything. The judge reads a candidate against its parent on the same twelve cells, so the question is whether a different twelve would have produced a different reading. The run's own panel games answer it for the three generation-one candidates that played the benchmark (Table 10), each candidate's twelve games paired by cell with the seed package's own twelve. Dropping any one cell moves a candidate's rise by at most 0.121: against the seed package, whose panel reads 2 of 12, the agentic rewrite rises 0.500 and its eleven-cell panels stay within 0.455–0.545, and the module rewrite (2) rises 0.333 within 0.273–0.455; the agentic rewrite's revision, read against the candidate it revises, rises 0.167 within 0.091–0.182. No eleven-cell panel moves a rise across the judge's 0.25 outcome threshold, the module rewrite (2) at 0.273 being the closest. Halving the panel would not have been safe: across the 924 six-cell halves, the paired rise spans 0.000 to 1.000 for the agentic rewrite and -0.167 to 0.833 for the module rewrite (2). This bounds composition sensitivity for the outcome criterion alone, which is what the panel games record; the judge's five card dimensions are read from the probe games and are not in them.

D BENCHMARK CALIBRATION: THE TWO LADDERS AND EVERY CELL

Table 15: Both calibration ladders in full. SC2FullGame-Bench orders two ladders whose strength order is known before any game. Outcome score on the 90 built-in cells with Wilson 95% intervals over the seed–cell pairs, the two replays of a seed counting as one pair (270 pairs per built-in level, 90 per capped variant); level-equivalent on the built-in scale (anchors lv1–lv7, lv9, lv10); capability scores are the ratio to the maximum over these two ladders for economy and tech-and-production, so those two columns fall in $[0, 1]$ by construction and a submission can exceed 1; combat is the arena's value-differential share and defense the fraction of workers kept, a diagnostic.

Subject	Outcome (90)	95% CI	Lv-eq.	Economy	Tech	Combat	Defense
Group A: built-in AI as subject							
lv1	0.015	[0.01, 0.04]	1.0	0.49	0.15	0.49	0.52
lv2	0.148	[0.11, 0.20]	2.0	0.52	0.21	0.49	1.00
lv3	0.256	[0.21, 0.31]	3.0	0.54	0.58	0.48	1.00
lv4	0.359	[0.30, 0.42]	4.0	0.57	0.59	0.49	1.00
lv5	0.433	[0.38, 0.49]	5.0	0.57	0.63	0.50	0.67
lv6	0.574	[0.51, 0.63]	6.0	1.00	0.82	0.53	1.00
lv7	0.644	[0.59, 0.70]	7.0	1.00	0.82	0.54	1.00
lv8 (cheat)	0.600	[0.54, 0.66]	6.4	0.84	0.67	0.51	1.00
lv9 (cheat)	0.744	[0.69, 0.79]	9.0	0.84	0.84	0.51	1.00
lv10 (cheat)	0.900	[0.86, 0.93]	10.0	0.98	1.00	0.52	1.00
Group B: one scripted package under a worker cap / an army-supply cap							
worker cap 12	0.000	[0.00, 0.04]	0.0	0.67	0.00	0.71	0.00
worker cap 16	0.506	[0.40, 0.61]	5.5	0.71	0.35	0.73	0.33
worker cap 22	0.567	[0.46, 0.66]	5.9	0.75	0.35	0.70	0.24
worker cap 30	0.578	[0.47, 0.67]	6.1	0.75	0.35	0.72	0.24
army cap 10	0.139	[0.08, 0.23]	1.9	0.75	0.23	0.70	0.24
army cap 20	0.322	[0.23, 0.42]	3.6	0.75	0.31	0.70	0.24
army cap 40	0.533	[0.43, 0.63]	5.7	0.75	0.35	0.71	0.24
army cap 60	0.561	[0.46, 0.66]	5.9	0.75	0.35	0.70	0.24

Table 15 and Fig. 6 give both calibration ladders in full: the built-in ladder (Group A of §3) and the capped ladder (Group B), one scripted package degraded by a worker cap or an army-supply cap. The built-in levels and the uncapped ladder package played three seeds per cell, and each capped variant one seed, every seed replayed twice: 7,893 games in all, of which 513 are probe games; Table 15's intervals count one draw per seed–cell pair, the two replays of a seed being one pair. The four probes of §3 are defined next. The ladder scores are on the released rule of §3, a tie counting 0; the calibration's own verdict file predates that rule and scores the same games with a tie counting 0.5; the released rule's tie of 0 therefore scores every level below those numbers. Economy counts workers and bases at 5:00 of game time, the tech-and-production probe counts tech structures and army supply at 8:00, combat is three mirror arenas of three games each scored as the value differential, and defense is the fraction of workers kept against a Zerg rush. The economy, tech-and-production and combat probes order Group A lv1–lv7 with Spearman 0.982, 0.964 and 0.857 and the two passes agree across the 19 validation subjects at Spearman 1.000, 1.000 and 0.836, the defense probe at 1.000; the defense probe orders lv1–lv7 at 0.401, since eight of the ten levels keep every worker against that rush. Under the army cap the outcome and the tech-and-production probe rise with the cap, and the other three probes do not, by construction. The uncapped package averages 0.515 over these cells, below its 22-worker, 30-worker, 40-army and 60-army variants. The registered claim, and the order in Table 1, is monotonicity within each cap family, and the uncapped package is an anchor of neither family.

Table 17 lists three quantities per map and opponent level. The first is the outcome of the Group-B package, that package being the basic build the caps are applied to, averaged over the calibration games of that cell. The second is the Group-A discrimination, the Spearman correlation over the seven non-cheating levels played as subjects, between a subject's level and its outcome on that cell.

Table 16: The four capability probes. The archive keys a candidate by these four and scouting, each binned at 0.5 (§4).

Probe	Measures	Read at	Scale
economy	workers and bases	300 s	ratio to the calibration ladders' maximum
tech and production	tech structures and army supply	480 s	ratio to the calibration ladders' maximum
combat	three mirror arenas, as the value differential	arena end	0.5 is parity
defense	workers kept against a Zerg rush	rush end	lies in $[0, 1]$

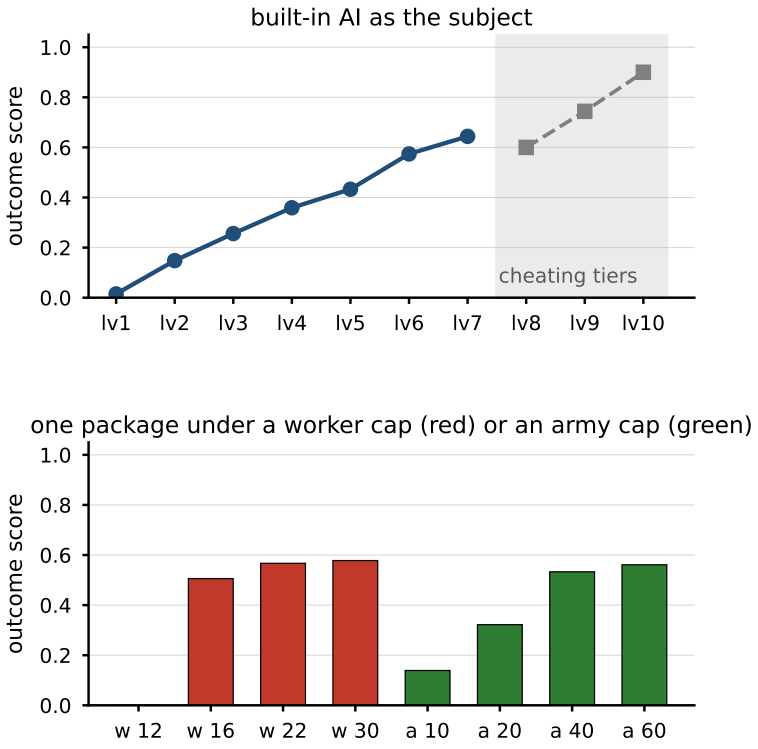

Figure 6: Both calibration ladders plotted: the game's own difficulty levels played as the subject (top) and one scripted package under a single worker or army cap (bottom; w = worker cap, a = army-supply cap); every score is the subject's outcome over the 90 built-in cells, and Table 15 gives them with Wilson 95% intervals over the seed–cell pairs.

The third is the Group-B order statistic, the mean over the two families of the Spearman correlation between cap value and per-cell outcome. Every cell card in the released suite carries these numbers together with the cell's same-seed agreement ρ and the built-in ladder's outcome at each level on that cell.

E TextStarCraft2 on the benchmark: setup and the parse defect

This appendix gives the in-loop agent's setup on the benchmark, its parse defect re-measured on its own games and repaired, and the scores of the row on three seeds.

Setup. Table 18 gives this row's configuration. TextStarCraft2 (Ma et al., 2024a) was run from its public repository behind DEEPSEEK-V4-FLASH, playing Protoss on the benchmark's 93 cells with the same maps and opponents as every other row, on the released default's seed 95001; the game runs through the agent's own environment wrapper, and our driver records the outcome, the wall time and the model endpoint of each game. Protoss is the only player race the system is published for: its repository states that only Protoss is supported, Zerg and Terran being under development there; the

Table 17: Per map and opponent level (three opponent races pooled): Group-B package outcome, Group-A discrimination (Spearman over lv1–lv7 as subjects), Group-B order statistic. – = undefined: every subject's outcome in the cell is the same, so no rank order exists.

Map	Level	B pkg. outcome	A: Spearman	B: order
Simple64	lv1	1.000	0.61	0.71
Simple64	lv2	1.000	0.77	0.71
Simple64	lv3	0.944	0.82	0.55
Simple64	lv4	0.722	0.88	0.74
Simple64	lv5	0.444	0.80	0.52
Simple64	lv6	0.722	0.72	0.53
Simple64	lv7	0.667	0.69	0.22
Simple64	lv8	0.889	0.75	0.45
Simple64	lv9	0.611	0.76	0.15
Simple64	lv10	0.056	–	0.17
AbyssalReefLE	lv1	1.000	0.61	0.71
AbyssalReefLE	lv2	1.000	0.80	0.76
AbyssalReefLE	lv3	1.000	0.90	0.79
AbyssalReefLE	lv4	0.667	0.95	0.59
AbyssalReefLE	lv5	0.389	0.92	0.56
AbyssalReefLE	lv6	0.000	0.77	–
AbyssalReefLE	lv7	0.000	0.80	–
AbyssalReefLE	lv8	0.000	0.67	–
AbyssalReefLE	lv9	0.000	0.61	–
AbyssalReefLE	lv10	0.000	–	–
AcropolisLE	lv1	1.000	0.61	0.77
AcropolisLE	lv2	1.000	0.80	0.79
AcropolisLE	lv3	1.000	0.90	0.80
AcropolisLE	lv4	0.778	0.89	0.75
AcropolisLE	lv5	0.556	0.90	0.87
AcropolisLE	lv6	0.000	0.74	–
AcropolisLE	lv7	0.000	0.78	–
AcropolisLE	lv8	0.000	0.80	–
AcropolisLE	lv9	0.000	0.79	–
AcropolisLE	lv10	0.000	–	–

Table 18: The in-loop row's configuration.

Model	DEEPSEEK-V4-FLASH, the framework's model
Race	Protoss, the only race the system's release supports
Cells	the benchmark's 93, on the same maps and against the same opponents as every other row
Basis	the released default, one seed per cell
Source	its public repository, with two disclosed patches
Cost	581 model calls per game, and 5.5–18 wall-clock seconds per game second
Outcome	0.441 over the benchmark's cells

environment it ships carries a Protoss bot and a Zerg bot on that unfinished path, but no Terran bot. The dashes in its two rows of Table 2 are that limit of the published system rather than a cell left unplayed. Two source patches were needed before a game could be scored in this environment, and both are disclosed in the release: a dependency and protocol-buffer pin so the agent's client loads against the pinned game build, and the decision-heading pattern described next. The action index the agent looks its decisions up in is not shipped by the repository and was rebuilt from the agent's own action descriptions. The agent's wall time is set by model latency rather than by the game, so games were run concurrently: 5.5 wall-clock seconds per game second through the vendor's API (14 games) and 18 through a relay endpoint to the same model (the rest); the endpoint of each game is recorded in its metadata (model `deepseek-chat` through the vendor's API, DEEPSEEK-V4-FLASH through the relay). The two endpoints were compared level by level: at lv1–lv3 both score 1.000 and at lv8–lv10 both score 0.000. Two games hung with no game process alive and were re-queued once after being recorded as wall-timeout errors; their records are kept beside the row.

The parse defect, re-measured on these games. The agent recovers its chosen actions from the model's reply in two steps: a fixed-string lookbehind finds the heading `Decisions:`, and an item pattern `\d+:  <action>` reads the numbered lines below it; a reply that either pattern fails to match yields no action, and the agent then plays that step with an empty decision list while its log records a coherent plan. The heading-pattern patch widens the heading pattern to tolerate markdown emphasis and a missing colon, which is how the benchmark row was played. We then replayed both extractors over every reply recorded during the benchmark games (52,329 decision replies from the 90 built-in-cell games; the replay uses the extractor's regular expressions; the action-index lookup that follows is unchanged). Under the shipped heading pattern 2,520 replies (4.8%) yield at least one action; under the widened heading 17,544 (33.5%) do — and that is the extraction the row was played with, so on about two of every three decision steps the agent issued nothing. The remaining failures are the item format: this model writes its list as `1. **<BUILD PYLON>**, 1. ` TRAIN PROBE` (0) -- ...` or a markdown table, and the shipped item pattern requires `1: <BUILD PYLON>`. Widening the item pattern as well raises the replay to 45,160 replies (86.3%). Whether that defect bounded the outcome was then tested: the agent was run again on the same 93 cells with the item pattern widened (registered before launch as `tsc2_submission_v2_p3`; the three external cells under the external-cell protocol), obtaining an action from 88% of its own 94,514 replies from the 90 built-in-cell games, 1,050 calls per game (the 52,329 replies above are the benchmark row's; the two runs differ in length). It scores 0.376 against the first run's 0.441, not separated ($S = -6$, $z = -1.28$; cell test $p = 0.18$): the registered prediction that the repaired extractor would raise the row (TP1, Appendix I) failed, and Table 2 reports both rows, the row as played first ($S = +46$ against the framework's package on those cells, $z = 6.19$; its Wilson 95% interval over the 93 cells is $[0.34, 0.54]$).

The row on three seeds. The in-loop row of §5.2 plays one seed per cell; to check that its verdict does not rest on that seed, we registered `tsc2_reduced_multiseed_v1` before its first game and played Simple64 at lv5–lv8 against all three races on seeds 95001, 95101 and 95102: 36 games, of which seed 95001's 12 are the 93-cell row's own games copied verbatim with a sidecar naming their source and digest, and 24 are new. The agent is the same system, unchanged: the as-played extractor, the text environment, the chain-of-summarization agent, the prompt set, the action space and the decision cadence are the ones the 93-cell row used. This basis does not replace that row. It adds a second one, and the two are not interchangeable: the 93 cells are one game each, this is 36 games over three seeds, and scores from the two are labeled apart wherever both appear.

Against the framework's package on the same cells and seeds, the row scores 0.139 where the framework scores 0.944, a difference of $+0.806$ per cell. Paired by cell and seed the framework's package is separated above it under both tests of §3: $S = +29$ of 36 with $z = 5.09$, the disagreement rate 0.806 giving a Wilson upper bound of 0.903 and so the $elevated$ branch, and the cell test at $p < 0.001$ with a 95% interval of $[0.694, 0.917]$. By seed the row scores 0.250 ($3/0/9$ wins/ties/losses), 0.000 ($0/1/11$) and 0.167 ($2/0/10$); the per-cell three-seed spread averages 0.417, with 5 of the 12 cells moving between seeds. The row reaches neither 0.50 nor the framework's package on any seed.

TextStarCraft2 is published by its authors for Protoss alone, with Zerg and Terran under development in their repository; we ran the system as released rather than extending it. Its rows therefore carry one race, and the multi-seed basis above is offered beside the one-seed basis, not as a change to the agent.

F Derivations

Null variance. Fix cell i; let $A_i, B_i \in \{0, 1\}$ be outcomes of the two packages and $s_i = \mathrm{sgn}(A_i - B_i)$. Under the identical-behavior null (A_i, B_i) is distributed as two repeats of one package on cell i — precisely what a ρ measurement observes — so $\Pr[A_i{=}B_i] = \rho$, $\Pr[s_i{\neq}0] = 1 - \rho$, and symmetry gives $\Pr[s_i{=}{+}1] = \Pr[s_i{=}{-}1] = (1 - \rho)/2$, hence $\mathbb{E}[s_i] = 0$ and $\mathrm{Var}[s_i] = 1 - \rho$ (only exchangeability of the two repeats and independence across cells are used). A normal approximation at one-sided level α and power $1 - \beta$ then yields Eq. 1 *under a common-variance approximation*: exactly, with $q_1 = \Pr[s_i{\neq}0 \mid H_1]$ the alternative's disagreement rate, $\mathrm{Var}[s_i|H_1] = q_1 - \delta^2$ and the planning equation is $\delta\sqrt{n} \approx z_\alpha\sqrt{1-\rho} + z_\beta\sqrt{q_1 - \delta^2}$; Eq. 1 is its $q_1 - \delta^2 \approx 1 - \rho$ special case, adequate at the small δ budgeted here, and when a comparison's own disagreement is elevated

Table 19: The separation rule under alternative same-seed agreements and equivalence bands. Each comparison keeps its own measured disagreement bound $\hat{q}_U$, so the registered ρ enters only where $1-\rho$ exceeds it; the table gives z and the threshold on S at $\rho = 0.900$, at the registered 0.939, and at the measured 0.946, the cell-level test's p (which ρ does not touch), and whether the 90% cell-bootstrap interval of the per-cell difference falls inside bands of ±0.03, ±0.05 and ±0.08.

						z (threshold on S) at $\rho =$			interval inside		
Comparison	S	n	$\hat{q}_U$	cell p	90% interval	0.900	0.939	0.946	±0.03	±0.05	±0.08
vs iterated prompt	49	279	0.248	< 0.001	$[0.122, 0.233]$	5.89 (13.7)	5.89 (13.7)	5.89 (13.7)	no	no	no
vs coding agent	39	279	0.225	< 0.001	$[0.086, 0.197]$	4.93 (13.0)	4.93 (13.0)	4.93 (13.0)	no	no	no
vs VibeCraft Protoss	3	279	0.121	0.732	$[-0.021, 0.043]$	0.52 (9.5)	0.52 (9.5)	0.52 (9.5)	no	yes	yes

the $\hat{q}$ pre-flight replaces $1-\rho$ entirely. Because a replayed view draws from a fixed pool of per-cell differences, the exact finite-pool model is multivariate hypergeometric; in the connect-four gate study of Appendix G, where the truth ordering is exact, it reproduces the measured rejection rates of the null (false acceptances of a difference, in the terms of Appendix G) where the Gaussian approximation does not, so the Gaussian form is used for budgeting only.

The same-seed agreement and the equivalence band, recomputed. Table 19 recomputes the three headline comparisons at $\rho = 0.900$, at the registered 0.939 and at the measured 0.946. Each comparison keeps its own measured bound $\hat{q}_U$, and every one of them exceeds $1-\rho$ at all three values, so ρ never reaches the arithmetic: z and the threshold on S are identical at all three, and no verdict can move with it. The band moves the equivalence reading and never a verdict: of the three comparisons one interval falls inside ±0.05 and ±0.08, and none falls inside ±0.03.

Comparison-level disagreement and the equivalence reading. Every comparison first measures its own disagreement rate $\hat{q}$ with a Wilson interval, from the games it decides on; the six-seed comparisons on the sealed suites of §5.4 estimate $\hat{q}$ from a pre-flight block of games played before the decision games. Where the interval's upper bound exceeds $1-\rho$, it replaces $1-\rho$ in the variance of S, and the reported z is S over the resulting standard deviation. Table 5 also reports whether each difference's 90% cell-bootstrap interval lies within ±0.05 by two one-sided tests, the equivalence reading of "not separated". The band is reported beside a verdict and never decides one, and it is reported at ±0.05, close to the smallest per-cell difference the three-seed design resolves, 0.058 at the measured disagreement bound (Appendix C).

G VALIDATION OF THE SEPARATION RULE, AND THE TWO OTHER GAMES

The gate study on connect-four. On connect-four, behind the benchmark's package and cell interfaces, the candidate policy has a seed-derived RNG and a blunder rate b (truth ordering exact by construction), and its opponent plays a random move with probability ε drawn from operating-system entropy (three cells, $\varepsilon \in \{0.05, 0.15, 0.35\}$, one pre-data amendment recorded). Fig. 7b and c give the registered results. P1 held: ρ differs across the three cells and the budget follows it, with a ρ spread of 0.22 and an $n_{\min}$ ratio of 1.82. P2 failed as registered, because the sign criterion compared an accept rate above 0.5 against the 80%-power threshold $\delta_{\min}$ where the 50%-power point was meant. All four "disagreements" are boundary points where predicted and measured power agree within 0.06, and the per-point values are in the released records; the model-agreement quantity the prediction intended is an MAE of 0.0189 over 72 points. P3 held on its first clause: at every matched budget up to $n = 120$, discrimination between a better and a worse policy dominates discrimination between a better and an unchanged one. Its second clause, that the better-versus-unchanged comparison stays below a true-positive rate of 0.5 at $n = 15$, holds in the two cells whose false-positive condition is met (rates 0.361 and 0.251) and is not satisfiable as worded in the third, where the identical pair's false-positive rate is 0.104. P4 failed: against a behaviorally different opponent of equal strength the identical-behavior null's variance $n(1-\rho)$ is undersized, and realized false acceptance reached 0.112 at $\alpha = 0.05$. Two post hoc variants of the corrected gate were evaluated. As run, the replay thresholded at $z_\alpha\sqrt{n\,\hat{q}}$ with $\hat{q}$ the point estimate taken from the decision view itself: on a fresh seed block, false acceptance (FA) max 0.073 / mean 0.058, true-positive rate (TPR) 0.947 at n=120. The shipped form takes $\hat{q}_U$ from a Wilson upper bound measured on a

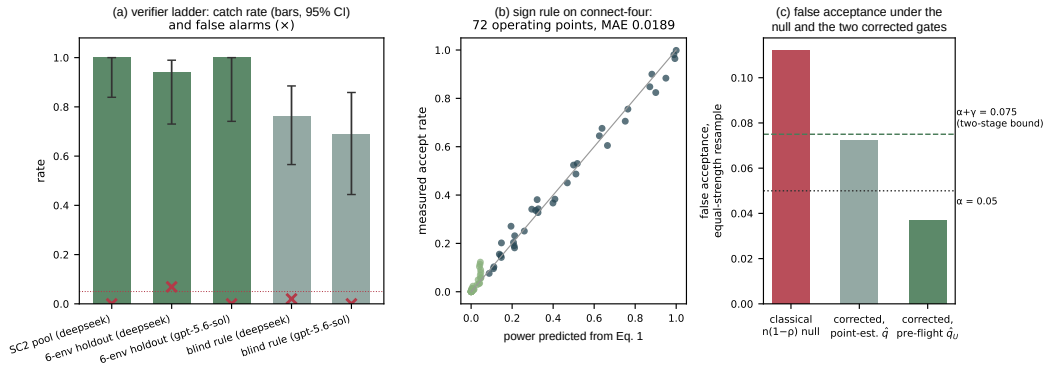

Figure 7: Validation of the separation rule on connect-four, and the verifier ladder in six other environments. **(a)** The verifier ladder across environments, rules and generator families (50–63 live programs per family, held-out): catch rates with 95% CIs, false alarms as crosses (Appendix H). **(b)** The sign test in the connect-four environment: the bound's predicted power against measured accept rates over 72 operating points, mean absolute error (MAE) 0.0189 (the registered sign-test prediction P2 failed as worded). **(c)** False acceptance under the identical-behavior null and under the two corrected gates: both corrected-gate variants, computed post hoc: the point-estimate replay (0.073) and the shipped independent-pre-flight form (0.037).

pre-flight sample independent of the decision view, and we then evaluated it on a further fresh block: false acceptance max 0.037 / mean 0.027, TPR 0.920 at n=120. The independent pre-flight form is the more conservative of the two and is the one the protocol below prescribes; both are approximate and valid conditional on sign-symmetry of the null. Because P4 shows this null can be undersized against a behaviorally different opponent of equal strength, the seven comparisons of Table 6 are also read with a test that does not use it, the cell test of §3; none of their verdicts differs between the two; the sealed suites of §5.4 use the independent pre-flight form.

The connect-four run. We registered the run of §5.4 before its first generation. Its seed policy plays a random legal move, its capability card has four dimensions and is admitted by the same two-pass calibration as the StarCraft II card, and its development cell sits at $\varepsilon = 0.15$. It keeps the framework's acceptance thresholds and structure. There are four single-reply arms (rewrite, patch, tactic, generator) and an agentic rewrite arm with a tool loop of at most 24 replies and two smoke blocks of 20 panel games. Each rewrite-type arm's best candidate gets one revision, a crash gets one repair attempt, and every judged candidate that is the best occupant of its cell enters the archive. The run is capped at 400 model calls. Finalization ran by nomination, on a held-out benchmark ($\varepsilon \in \{0.05, 0.15, 0.35\}$, seeds 230001–230300) and the 400-seed sealed block (220001–220400). One amendment was recorded during generation two, before any benchmark game: the reply parser rejected code blocks with a language tag other than `python` and unfenced replies as syntax errors; it was repaired and generation two replayed. The run stopped at the call cap after eleven generations with 81 judged candidates (9 accepted, 51 no gain, 13 regressed, 4 verification failures, 3 duplicates, 1 proposal failure) and seven archive cells; the agentic arm produced 4 accepted candidates of 11, and its revisions yielded 1 of 9. Nominees: a patch candidate (0.985 on the panel; benchmark 0.977 $[0.965, 0.985]$, cells 0.970/0.972/0.988), a revised rewrite (0.935; 0.941) and a generator candidate (0.653; 0.693); anchors on the same benchmark: the reference 0.611, the seed 0.058. The adopted policy's sealed outcome is 0.995 $[0.982, 0.999]$ (seed 0.058 $[0.039, 0.085]$); the panel and the benchmark order the nominees identically.

The full loop in microRTS. The microRTS run of §5.4 uses gym-microrts (Huang et al., 2021) with the shipped rules and maps. The package has the structure of §4: a controller registry, plan modules the arms may edit, and a protected runtime. External knowledge given to every arm: a domain note (concepts, unit costs, timings) and a playbook of four named plans with step timings, in prose; no code from the reference. The capability card and the development panel were admitted by the same two-pass calibration as the StarCraft II card, and the loop kept the arms, judge, archive, revision, repair and top-three nomination of §4, with caps of 15 generations and 800 model

calls. The run reached the generation cap, 15 generations, at 605 calls with 89 judged candidates (3 accepted, 35 *inert*, 30 regressed, 10 no gain, 11 proposal or verification failures), where *inert* is a candidate whose probe-game actions match its parent's. The adopted package, the agentic arm's revised candidate of generation 13, was judged no gain and entered the archive as the best occupant of a new capability cell, the mechanism that kept the StarCraft II package. Finalization: the three nominees, the seed and the reference each played the 12-cell benchmark (three maps × randomBiased, workerRush, lightRush and coac opponents) at 30 games per cell, seeds fixed before the run. Nominees: 0.528 $[0.476, 0.579]$, 0.489 $[0.438, 0.540]$ and 0.167 $[0.132, 0.209]$, in the panel's order; the reference 0.464 $[0.413, 0.516]$ and the seed 0.000. Paired over cells and game index, the adopted package is above the reference at $S = +23$ over 360 games ($\hat{q} = 0.125$, $\hat{q}_U = 0.163$, $z = 3.00$) and above the seed at $z = 13.2$; for the reference comparison the cell test over the 12 cells gives $p = 0.09$, with a 95% cell-bootstrap interval of $[+0.006, +0.142]$ around the $+0.064$ per-cell difference, so under the rule of §3, which needs both tests, separation is not claimed there; against the seed the sign test separates it ($z = 13.2$), and a cell test computed after the run, not registered, agrees at $p = 0.0037$. Per opponent, averaged over the three maps, it scores 1.000 against randomBiased, 0.400 against workerRush, 0.700 against lightRush and 0.011 against coac; the reference 0.978, 0.411, 0.467 and 0.000. The shipped opponents are ordered on the benchmark, and the panel and the benchmark order the nominees identically; the registered margin of 0.10 over the reference was measured as $+0.064$ (M2, Appendix I).

Pre-flight protocol on the benchmark. A paired comparison of two packages computes $\hat{q}$ as their observed disagreement rate over the paired games, with a Wilson interval. If $\hat{q}_U$ lies at or below $1-\rho$ with the registration-time $\rho = 0.939$ ($1-\rho = 0.061$), the identical-behavior budget and classical gate stand; otherwise the threshold on the sign sum is recomputed from $\hat{q}_U$ (Eq. 1 with $1-\rho$ replaced) and the corrected gate applies. On the released suite the pre-flight and the decision are computed from the same 279 paired games. The false-acceptance bound $\alpha + \gamma$ of the independent-pre-flight form therefore does not apply there as stated, where γ is the probability that the pre-flight Wilson upper bound falls below the true disagreement rate, 0.025 at the 95% interval; those verdicts are read as the more conservative corrected threshold, with that caveat. The sealed suites of §5.4 take $\hat{q}_U$ from a separate pre-flight block. Every three-seed comparison of §5.2 took the elevated branch: $\hat{q}$ ranged from 0.082 (the framework's package against the VibeCraft Protoss package) to 0.351 (against the VibeCraft Zerg package), against $1 - \rho = 0.061$.

The rule validated on StarCraft II. A validation registered before its first game (`gate_calibration_v1`) played the framework's package twice and the VibeCraft Protoss package once on all 93 cells over six seeds no study had used (1,674 games): two passes of one package realize the identical-behavior null exactly, and the framework against the VibeCraft Protoss package is the behaviorally different near-null pair that undersized the null in connect-four. T1 held: the identical pair is not separated under either form of the rule (released form, $\hat{q}$ from its own 558 pairs, $z = -1.06$, cell $p = 0.36$; pre-flight form, $\hat{q}_U$ from a 186-pair block and the decision on 372 pairs, $z = -1.05$, -0.013 per cell, $p = 0.50$). T2 set two bounds over 20,000 random subsets of 279 pairs, the size of the three-seed comparisons. The rule as used, with both tests and either direction, declares the identical pair separated in 0.0055 of subsets against the registered 0.07; the sign test alone declares it separated in 0.1075 against 0.10. T2 **failed** on the second clause by 0.0075 and held the first with margin; one-sided in the framework's favor, both rates are 0.0000. T3 (measurement): the framework against the VibeCraft Protoss package is not separated (pre-flight form $z = -0.74$, -0.013 per cell); its subset false-separation rates are 0.0131 (sign test) and 0.0087 (both), the pre-flight $\hat{q}_U = 0.122$ absorbing the behavioral difference the connect-four replay had left to the null. T4: same-seed agreement across the two passes is 0.944 (Wilson lower bound 0.922), covering the registered 0.939. T5: the framework's two passes average 0.905 and 0.918 on these seeds, and the VibeCraft Protoss package 0.921, all below their released-suite scores as the six-seed extension had also found.

How the arms were adapted for the two other games. The six arms of §4 are a StarCraft II arrangement, so the two transfer runs required a decision about them, and they made different ones. The microRTS run kept the arms unchanged: it carries the same six, the same judge, the same archive, revision, repair and top-three nomination, over a package with the same shape, a controller registry and plan modules the arms may edit. The connect-four run reduced and renamed them: four

Table 20: The verifier ladder's reference check in six other environments: catch rate of reference-dead programs by rule and generator family, with false alarms. Held-out rows are the registered measurement; in-sample rows are descriptive. The family column names the generator family: DEEPSEEK-V4-FLASH and GPT-5.6-SOL.

rule (row status)	family	catch [95% CI]	false alarms
execution-observed (held-out)	deepseek	0.94 [0.73, 0.99]	0–1 in 50–63 programs
execution-observed (held-out)	gpt	1.00 [0.74, 1.00]	as above
verifier-blind (held-out)	both	0.60 / 0.76 per family; pooled 0.69 [0.44, 0.86]	—
execution-observed, lenient (in-sample, descriptive)	deepseek	0.37 [0.22, 0.54]	0/120
verifier-blind, lenient / strict (in-sample)	deepseek	0.43 / 0.63	0/120

single-reply arms, a rewrite, a patch, a tactic and a generator, against §4's three anchored editors and one plan-module writer, plus the same agentic rewrite arm with a bounded tool loop. Its capability card has four dimensions rather than five because the fifth, scouting, is a StarCraft probe with no counterpart in a game of perfect information.

The capability-keyed archive did comparable work in both. In microRTS the adopted package was judged no gain and became the best occupant of a new capability cell, which is exactly the mechanism that kept the StarCraft II package: a candidate the judge would not have promoted became the run's delivery through the archive's cell structure. The connect-four run occupied seven cells over eleven generations, out of the eighty-one candidates it judged, so its archive was sparse in the same way and for the same reason, a five-dimension or four-dimension card binned coarsely against a few dozen entries (Appendix C).

H THE VERIFIER LADDER IN SIX OTHER ENVIRONMENTS

Held-out catch rates by rule and generator family. The reference checks resolve names against a deployed environment rather than against anything StarCraft-specific, so we extracted them into a dependency-free tool and re-ran the identical frozen protocol in six environments on 15 held-out tasks, then again with a second generator family, GPT-5.6-SOL. Held-out rows appear in Fig. 7a and Table 20; the two families' intervals overlap under both rules. Hallucination is a property of *obscure* APIs — three popular libraries produced zero reference deaths in 75 programs per family, so a benchmark built on those libraries alone would have measured none of it — and the rate is family-dependent (0.15 vs. 0.23).

The execution-observed rule counts a candidate reference-dead when the interpreter kills it on a module/attribute that does not exist; the verifier-blind rule counts *any* failed name resolution on any object, including logic bugs the ladder never claimed, which lowers measured catch. In-sample rows are descriptive only: repairs were designed on the first rule's misses. Family-2 candidates: 74/75 program requests succeeded (one API failure, counted); reference-death rate 0.149 vs. deepseek's 0.227 on identical tasks.

The loop in the six library environments. The framework's loop shape — propose, static ladder, execute, typed feedback, re-propose, with a ladder-rejected candidate never executed — was run in all six environments (registered before any candidate): 30 tasks, at most three rounds each. Completion within three rounds exceeds round-one completion by 0.133 (0.767 → 0.900), and the ladder rejected 4 candidates before execution in-loop. Iteration repaired tasks in three environments (gymnasium, minigrid, scikit-learn); two of five PettingZoo tasks stayed incomplete through all rounds and are reported as such. These six environments are deterministic, so they exercise the loop and execution layers; the measurement and decision layers are exercised where outcomes are stochastic (connect-four above, and StarCraft II).

The upgrade-reference extension. The reference table now carries the build's upgrades in two lists — present (296) and researchable (116) — because an `UpgradeId` can be present and still have no research ability, which is fatal the moment `already_pending_upgrade` reads it. Re-run over the three VibeCraft bots of Table 2, the extended check flags three references. One is the Zerg package's baneling morph, a call the check classes as fatal if its path runs; none of that package's 279 benchmark games crashed, so the path did not run there. The other two — the Terran package's

SUPPLYDEPOTDROP and the Protoss package's TEMPESTGROUNDATTACKUPGRADE — lie in code paths that never executed in these games, and both packages played clean. A reference absent from the build is fatal only if its path runs; the check is conservative (0/9 false alarms on candidates), and the VibeCraft bots are measured by it, not gated.

Wildcard imports in the reference check. Both implementations of the reference check, the framework's and the standalone tool's, follow "`from X import *`" with bounded recursion and cycle protection; an unreachable re-export target returns OPAQUE, the verdict for a name the check can neither confirm nor refute. A re-run of the ladder over all 170 recorded candidate packages leaves 22 rejected. The runtime resolver used for the cross-environment transfer numbers imports and reads `dir()` rather than parsing source, so those results are untouched.

I REGISTERED PREDICTIONS AND THEIR OUTCOMES

Every study reported here was frozen as a protocol, with its predictions, before the games it governs were played; *measurement* rows registered what would be measured and no direction. Predictions that failed are marked and none was retried without a fresh registration. A threshold or score in this table is on the benchmark's released default, one game per cell, which is the basis on which the predictions were registered, unless the protocol column names another basis: three seeds, six seeds, a sealed suite, or another game's benchmark. Released-default scores therefore differ from the body's three-seed scores of the same packages, and the framework's package scores 0.935 here against 0.943 in the body. Predictions registered under the sign test alone are decided by it, while the both-tests rule governs the body's comparisons. The three-seed comparison of §5.2 is covered by its own rows (B1–B2); the connect-four gate study's predictions (P1–P4) are reported with their outcomes in Appendix G and not repeated here. Labels missing from the sequence are predictions about studies, or bases the body does not report, or labels withdrawn from the paper's claim set; in every case their outcomes, failures included, are in the released protocols. The sealed ladder's capped packages carry the labels SB1–SB3 in the released protocol, shown as SLB1–SLB3 here so that the labels of sealed suite B (SB1–SB5) stay unique in this table; the in-loop rows' RM1–RM4 are likewise shown as IL1–IL4. The verifier-off ablation was registered inside the four-condition family of `ablation_replicates_v1` and is reported outside it: §5.3 uses that run for the ladder's crash count and draws no score claim from it. Its three runs average 0.883 against the framework's 0.914, and the framework's runs are $+0.031$ per cell above it ($z = 2.62$, Holm $p = 0.02$ over the four registered conditions). The four conditions the body reports are the three part-removals and the timing inversion of the playbook.

Protocol	Prediction	Outcome	Where
Benchmark calibration			
built-in ladder	BL1: lv1–lv7 strictly increasing	held (Spearman 1.0)	§3
capped ladder	HL1: both cap families monotone	held (Spearman 1.0)	§3
probes	CP1: each capability score orders lv1–lv7	held for three scores; **failed** for defense	§3
probes	CP2: capability scores reproduce pass to pass	held	§3
Comparisons			
coding agent	LA1: framework above by 0.05 or separated	held	§5.2
in-loop agent	row, two disclosed patches, parse replay	measurement	App. E
in-loop agent, re-run	TP1: the repaired extractor raises the row (separated)	**failed** (0.376 vs 0.441, $S = -6$)	App. E
in-loop agent, re-run	TP2: the repaired extractor yields an action for $\geq 75\%$ of replies	held (88%)	App. E
VibeCraft bots, iterated prompt	paired on the built-in cells	measurement	§5.2
Ablations, replicates, other games, sealed suites, gate validation and extensions			

Protocol	Prediction	Outcome	Where (cont.)
connect-four, its held-out benchmark	CG2: sealed floor 0.490 at the lower bound	held (sealed 0.995, lower bound 0.982 against the floor)	§5.4
connect-four, its held-out benchmark	CG3: an agentic candidate archived as best of its cell	held	§5.4
connect-four, its held-out benchmark	CG4: panel order vs benchmark order	measurement (same order)	§5.4
connect-four, its held-out benchmark	CG5: seed at most 0.15 on the benchmark	held (0.058)	§5.4
microRTS, its 12-cell benchmark	M1: the shipped opponents ordered on the benchmark	held (Spearman 1.0)	§5.4
microRTS, its 12-cell benchmark	M2: adopted package above the reference by 0.10, separated above the seed	**failed** on the margin ($+0.064$, $z = 3.00$; seed $z = 13.2$)	§5.4
microRTS, its 12-cell benchmark	M3: an agentic candidate is the archive's best at some generation	held (generation 13)	§5.4
microRTS, its 12-cell benchmark	M4: panel order vs benchmark order	measurement (same order)	§5.4
ablation replicates, three runs per condition, three seeds	AR1: each of greedy acceptance, playbook without build orders (registered as *playbook without timings*) and no playbook has a three-run mean below the framework's	held (0.864, 0.842, 0.838 vs 0.914)	§5.3
ablation replicates, three seeds	AR3: pooled over three runs, each removal separated below the framework's runs	held ($z = 3.91$, 5.21, 5.39)	§5.3
ablation replicates, three seeds	AR4: run-level spreads and t intervals	measurement (spreads 0.086, 0.022, 0.079; the t interval excludes zero for the playbook's build orders only)	§5.3
replication of the framework's run, three seeds	RP1: the replicates' scores and the spread over three runs	measurement (0.943, 0.910, 0.889)	§5.3
second model family (GLM-5.2), three seeds	MF1: separated above the coding agent	held ($S = +23$, $z = 2.96$)	§5.3
second model family (GLM-5.2), three seeds	MF2: not separated from the VibeCraft Protoss package	held ($z = -1.94$, cell $p = 0.08$)	§5.3
second model family (GLM-5.2), three seeds	MF3: within 0.10 of the first model's run	held (0.885 vs 0.943)	§5.3
sealed suite B (two further maps)	SB1: ours separated above the coding agent run without the playbook	held ($z = 5.34$)	§5.4
sealed suite B	SB2: ours not separated from the VibeCraft Protoss package	held ($z = -1.34$)	§5.4
sealed suite B	SB4: the four rows keep suite A's order	**failed**	§5.4
sealed suite B	SB5: every row within 0.05 of its suite-A six-seed score	held (largest change 0.025)	§5.4
gate validation, six fresh seeds	T1: the identical-behavior pair (one package, two passes) not separated under both forms of the rule	held ($z = -1.06$, -1.05)	App. G
gate validation, six fresh seeds	T2: false-separation rate over 20,000 subsets of 279 pairs at most 0.07 (both tests) and at most 0.10 (sign test alone)	**failed** on the second clause (0.0055; 0.1075)	App. G
gate validation, six fresh seeds	T3: framework vs VibeCraft Protoss package, both forms and subset rates	measurement (not separated, $z = -0.74$; rates 0.0131, 0.0087)	App. G
gate validation, six fresh seeds	T4: same-seed agreement across the two passes	measurement (0.944, lower bound 0.922)	App. G

Protocol	Prediction	Outcome	Where (cont.)
gate validation, six fresh seeds	T5: each row's six-seed mean	measurement (0.905, 0.918, 0.921)	App. G
sealed suite A, six seeds	SS2: ours separated above the coding agent run without the playbook	held ($z = 5.25$)	§5.4
sealed suite A, six seeds	SS3: ours not separated from the VibeCraft Protoss package	held ($z = 0.26$, cell $p = 0.83$)	§5.4
sealed suite A, six seeds	SS4: pre-flight against decision-block $\hat{q}_U$	measurement (coding agent 0.262 vs 0.194; VibeCraft Protoss 0.167 vs 0.110)	§5.4
sealed suite A, six seeds	SS5: every row within 0.03 of its three-seed sealed score	held	§5.4
playbook-fed coding agent, capped run	KC1: spend and stopping mode	measurement (stopped at the token cap, 3.63M, 4.60 hours, 206 games)	§5.2
playbook-fed coding agent, capped run	KC2: the package separated below the framework's on three seeds	**failed** (framework first, $z = 1.81$; cell test $p = 0.12$; not separated under both tests)	§5.2
playbook-fed coding agent, capped run	KC5: the package on the sealed cells	measurement (0.922 on three seeds; 0.906 on six)	§5.4
community bots (Paul, 2020; PlayerPBrief, 2021; Sir Slimestone, 2020)	EC1: each row's nine-game score against each bot	measurement	App. A
community bots	EC3, per bot: ours wins at least as many of 9 as the coding agent and the seed	held for all three	App. A
six seeds per cell	SX1: ours vs the playbook-fed coding agent on six seeds	measurement (separated: $z = 3.80$, $+0.068$ per cell, $p = 0.005$)	§5.2
six seeds per cell	SX3: ours not separated from the VibeCraft Protoss package	held ($z = -0.12$, cell $p = 1.00$)	§5.2
six seeds per cell	SX4: ours separated above the coding agent run without the playbook	held ($z = 7.16$)	§5.2
External opponent and three seeds			
external cells	X3: ours separated above the coding agent, not separated from the VibeCraft Protoss package, on 93 cells	held	§5.2
external cells	X5: the opponent beats the do-nothing control and the seed package	held	the released protocol
three-seed comparison	B1: ours separated above the coding agent over 279 pairs	held ($S = +39$, $z = 4.93$)	§5.2
three-seed comparison	B2: ours not separated from the VibeCraft Protoss package	held ($S = +3$, $z = 0.52$)	§5.2
sealed maps, three seeds	SE1: ours still separated above the coding agent	held ($S = +31$, $z = 4.78$)	§5.4
sealed maps, three seeds	SE2: ours still not separated from the VibeCraft Protoss package	held ($S = +6$, $z = 1.42$; cell test $p = 0.112$)	§5.4
sealed ladder	SL1: lv1–lv7 strictly increasing on the sealed cells	held (Spearman 1.0)	the released protocol
sealed ladder	SL2: built-in same-seed agreement ≥ 0.95	held (1.000)	the released protocol
sealed ladder, capped packages	SLB1 (SB1 in the protocol): both cap families monotone on the sealed cells	held (Spearman 1.0)	the released protocol
sealed ladder, capped packages	SLB2 (SB2 in the protocol): capped packages' same-seed agreement ≥ 0.90	held (0.976)	the released protocol
sealed ladder, capped packages	SLB3 (SB3 in the protocol): the sealed per-cell agreement recorded	measurement (0.976)	the released protocol
The Terran and Zerg packages			

Protocol	Prediction	Outcome	Where (cont.)
race rows, three seeds	RM1: framework Terran separated above VibeCraft Terran	held ($S = +30$, $z = 3.97$)	§5.2
race rows, three seeds	RM2: framework Zerg separated above VibeCraft Zerg	held ($S = +66$, $z = 6.83$)	§5.2
race rows, three seeds	RM3: each race package's three-seed score	measurement (0.853, 0.857)	§5.2
race rows, three seeds	RM4: neither race package reaches the framework's Protoss package	held ($z = -3.34, -3.11$)	§5.2
The playbook-fed rows			
playbook-fed, three seeds	KM2: the iterated prompt given the playbook is not separated below the framework's package	**failed** ($S = +34$, $z = 4.58$)	§5.2
playbook-fed, three seeds	KM3: both rows score above their versions without the playbook	held (0.900 vs 0.803; 0.821 vs 0.767)	§5.2
Registered 2026-09-19: the in-loop three-seed row and the inverted-timing playbook			
in-loop, three seeds	IL1 (RM1 in the protocol): the row's three-seed outcome is below the framework's on the same games	held	§5.2
in-loop, three seeds	IL2 (RM2 in the protocol): the framework's package is separated above it under both tests	held ($S = +29$, $z = 5.09$, $p < 0.001$)	§5.2
in-loop, three seeds	IL3 (RM3 in the protocol): its per-seed outcomes and per-cell three-seed spread, as measured	measurement	App. E
in-loop, three seeds	IL4 (RM4 in the protocol): the row does not reach 0.50	held (0.139)	§5.2
inverted-timing playbook	PI1: the package scores below the framework's original benchmark outcome	held (0.933 vs 0.967; within the framework's three-run range, IR1–IR5)	§5.3
inverted-timing playbook	PI2: it scores above the playbook-without-build-orders condition's three-run mean	held (0.903 vs 0.842)	§5.3
inverted-timing playbook	PI3: at least one accepted candidate edits a milestone's timing or ordering rather than the opening plan it was set to	held (4 of 8 accepted, by added time gates)	App. C
inverted-timing playbook	PI4: its spend, as measured	measurement (App. C)	§5.3
Registered 2026-09-20: the inverted-timing replicates			
inverted timings, three runs	IR1: the condition's three-run mean is above the playbook-without-build-orders condition's 0.842 and the no-playbook condition's 0.838	held (0.904)	App. C
inverted timings, three runs	IR2: pooled over the three runs against the framework's three, the per-cell difference is smaller than the build-orders condition's +0.072, separation reported as measured	held (+0.010 per cell, not separated)	§5.3
inverted timings, three runs	IR3: each replicate's adopted package scores above 0.842 on its own	held (0.893 and 0.918)	§5.3
inverted timings, three runs	IR4: the condition's run-level spread against the framework's 0.054, and a run-level t interval on the difference of means	measured (0.025; the interval spans zero)	App. C
inverted timings, three runs	IR5: per run, the accepted candidates that rewrite the opening plan's gates	held (4 of 8, 2 of 5, 1 of 3)	App. C
inverted timings, three runs	AM1: the delivered package is the archive entry the ranking put first, not the best of three nominees on the released default, which is the adoption rule the other conditions registered	departure, disclosed: it forgoes a step whose measured effect is +0.005 to +0.036 per cell (Appendix C)	Table 13

Every benchmark that scores a package this table names was registered under the same rule before its first game.

J RELEASE INVENTORY AND ITS STATUS

Ships independently (stdlib-only, no StarCraft): the verifier ladder as a standalone package (MIT, unit-tested, validated on StarCraft II, six other environments and two model families); the statistics module (the sign test, the cell test and the sample-size calculator); the suite index with its 93 cell cards (each built-in card carrying its measured ρ, the built-in ladder's outcome at every level and the capped variants' outcomes on that cell), the four probe definitions and the level-equivalent scale, with JSON schema and validators; the frozen protocols with registered predictions (including every failed one, unmodified); the frozen package of every leaderboard row with its file digests; per-game records for all games with their telemetry, and every analysis script with its fixed seed. **Requires the pinned environment:** full-game evaluation needs StarCraft II Linux 4.10 Base75689, the seven maps, the released suite's three and the sealed suites' four, all but Simple64 from the ladder pool and recorded by digest, the sibling package-generator repository, and (for generation) an LLM endpoint; an install guide states these tiers, a self-test distinguishes the tiers mechanically, and a Dockerfile builds the evaluation image, which passes both tiers of the self-test (the stdlib-only checks and two full games). **Known gaps:** package tree hashes drop platform-specific binaries but are still not portable across OSes; the framework still assumes one top-level bot namespace; the capability scores are normalized against the two Protoss calibration ladders; and the leaderboard's rows are the ones measured here, no package from outside this work having been submitted to it yet. Provenance of the code in this inventory: the harness, the analysis scripts and the figures were written by the generative AI whose use the AI-use statement describes, under the authors' direction; every protocol and every registered prediction was frozen before the games it governs, and no experimental number was produced, imputed or adjusted by a model (the AI use statement).

K SYMBOLS

Symbol	Meaning
ρ	same-seed outcome agreement of one artifact with itself, per cell
$1-\rho$	per-cell noise floor of a paired comparison *under the identical-behavior null*
$\hat{q}$	disagreement rate measured on the comparison being run
$\hat{q}_U$	Wilson upper bound of $\hat{q}$
α, β	sign-test level (0.05) and one minus its power; z_α, z_β their normal quantiles
γ	probability that a pre-flight $\hat{q}_U$ is below the true disagreement rate (0.025)
δ	expected per-cell sign $\mathbb{E}[s_i]$: win-over minus loss-to probability
$\delta_{\min}$	smallest δ detectable at a given n, level, and power
$n_{\min}$	paired games needed for a target δ
S	paired sign sum $\sum_i s_i$ over a comparison's paired games
level-equivalent	built-in level whose outcome matches a submission's (isotonic fit of lv1–lv7, lv9, lv10)

The suite's arithmetic is as follows. The 93 cells are 3 maps $\times$ 10 levels $\times$ 3 races, plus one external cell per map. The released default is one game per cell on seed 95001. Four probes are defined, of which combat is three arenas of three games each. The calibration used 19 validation subjects: the ten built-in levels, the uncapped package `protoss_basic` that both cap families are cut from, and its eight capped variants. The calibration played $7{,}893$ games, two replays of $3{,}690$ seed–cell pairs plus 513 probe games, whose definitions are the four probes of Table 16. Pooled ρ is 0.986. Above the lv10 anchor the scale extends linearly, reaching 11 at an outcome of 1, which is how the framework's package reads 10.6. Every comparison is registered at $\rho = 0.939$, whose $1-\rho$ enters its variance only where it exceeds that comparison's own measured bound; the calibration measured 0.946 for packages. A tie scores 0.