

Answer or Compute? Direct Inference vs. Code Generation for LLM Game-State Understanding in RTS Games

Chang Liu

School of Electrical Engineering and Computer Science
Ohio University, Athens, OH 45701, USA
liuc@ohio.edu

Abstract

A large language model (LLM) can understand a game state in two ways: answer questions about it directly, or write a program that computes the answer. We compare these modalities on three real-time-strategy tasks—factual extraction, strategic assessment, and win prediction—holding states, questions, and models fixed. On mechanical questions, code wins decisively: hit-point (HP) aggregation rises from 30% to 100%, and the direct-mode failures are mostly a refusal to declare ties, not bad arithmetic. On judgment tasks, code seemed to win as well, until a trivial baseline exposed why: on snapshots from bot games the material leader always wins, so $\text{sign}(\Sigma\text{HP})$ alone scores 100%. On contested states built to remove this artifact, no arm—direct, chain-of-thought, or code-based—reliably beats chance (40 tests, Holm-corrected), while a simple unit-count cue reaches 72%. Labels are outcomes of games continued by scripted bots, so we measure prediction under a small, fixed strategy pool, not open-play judgment. Within that scope: code makes LLMs reliable aggregators; it does not make them judges. We contribute the shortcut audit and hard-set construction that judgment benchmarks built on game traces should adopt.

1 Introduction

Large language model (LLM) agents are increasingly deployed in game environments, where every decision begins with *state understanding*: reading the board, aggregating material, judging who is ahead. Prior evaluations of LLMs in real-time-strategy (RTS) settings measure gameplay outcomes (Kim et al., 2026; Tang et al., 2025) or, in diagnostic work, direct question answering over game states (Liu, 2026). That prior diagnostic work identified a stubborn bottleneck: models that predict winners at ceiling fail simple hit-point (HP) aggregation questions at 14–65% accuracy.

Meanwhile, a separate literature shows that LLMs solve *mechanical* problems better by writing code than by answering directly (Gao et al., 2023; Chen et al., 2023; Li et al., 2024), and that executable code is an effective action modality for agents (Wang et al., 2024). Yet this literature tests domains where a correct program provably exists—math word problems, symbolic puzzles, table lookups. Whether an LLM can *codify its own strategic judgment*—write an evaluation function that outperforms its in-context judgment—is untested for game-state evaluation, and recent evidence that LLM-written code is not always better than textual reasoning (Chen et al., 2025) makes the answer genuinely uncertain.

We ask: **is code generation a better modality than direct inference for LLM game-state understanding—and where is the boundary?** We decompose the question along the mechanical–judgmental axis:

- **RQ1 (mechanical):** does synthesized code close the factual/aggregation gaps that direct answering cannot?
- **RQ2 (judgmental):** does a model-synthesized evaluator beat the same model’s direct judgment on strategic assessment and win prediction, measured against rollout-verified outcomes?
- **RQ3 (validity):** what do these comparisons measure? We audit our own evaluation sets with trivial-program baselines.

RQ2 has a two-part answer. On snapshots harvested from bot games—the standard construction, which prior diagnostic work and gameplay benchmarks share—one state-blind synthesized evaluator matched or beat direct judgment and a hand-crafted heuristic, apparently decisively. A

trivial baseline audit then showed the task was broken: bot games snowball, so the eventual winner is *always* the current material leader, and $\text{sign}(\Sigma\text{HP})$ scores 100%. On *contested* states, engineered so that material is chance-level, every modality collapses: direct, chain-of-thought direct, synthesized evaluators, and per-board codegen all land near chance, and none is reliably above it after correction. Yet a trivial composition cue shows 72% is reachable.

Contributions: (1) the first controlled direct-vs-code comparison on *judgmental* game-state tasks, with a state-blind synthesis design that prevents per-instance judgment from leaking into the code arm, and a decoy-board probe quantifying that leakage for per-instance codegen; (2) a **shortcut audit** showing that judgment evaluations built on game traces can be solved by material counting—with a hard-set construction (upset + close-margin states) that removes the artifact; (3) a sharp empirical boundary: code makes LLMs reliable aggregators (30%→100% on extraction; the only Holmsurviving judgment gains are where judgment reduces to aggregation), but *no current modality* demonstrates strategic judgment beyond material; (4) a **bad-theory failure mode** (§4.6): one generation scores 28% while non-degenerate, perfectly antisymmetric, and structurally indistinguishable from its 100% siblings—so structural probes cannot catch it and all k generations must be reported; (5) a revised, honest routing recipe for agent builders.

2 Related Work

Direct answering vs. generated code. PAL (Gao et al., 2023) and Program of Thoughts (PoT) (Chen et al., 2023) established that LLMs solve arithmetic-heavy problems better by emitting programs; Chain of Code (Li et al., 2024) extends to semi-executable pseudocode. CodeSteer (Chen et al., 2025) is closest to our question: across 14 mechanical tasks it studies *when* models should generate code versus reason textually, finding that models mis-choose the modality and that code is not uniformly better. CodeAct (Wang et al., 2024) shows code as a superior *action* format for agents; ViperGPT (Surís et al., 2023) and Binder (Cheng et al., 2023) apply program synthesis to visual and tabular question answering (QA). All of these target tasks with provably correct programs; none test strategic judgment, and none control for judgment

leaking into per-instance code.

LLM-authored evaluators and rewards. Eureka (Ma et al., 2024) and the LLM reward-design line (Sun et al., 2024) show LLMs write strong reward code; SimuScene (Wang et al., 2026) reports generated reward code is more faithful than LLM direct judgment in a physics-sim domain. The heuristic-evolution line (FunSearch (Romera-Paredes et al., 2024), EoH, ReVEL) evolves code with iterative feedback. Our contribution is not the existence of the generation>judgment gap but its *controlled, same-model test on strategic game-state evaluation*: single-shot (no optimization loop), state-blind, against rollout-verified ground truth.

LLMs in strategy games. RTSGameBench (Kim et al., 2026), DSGBench (Tang et al., 2025), and related benchmarks measure gameplay outcomes and explicitly note the absence of mechanistic analysis of *why* models fail. Diagnostic work on game-state assessment (Liu, 2026) supplies our tasks, baselines, and the aggregation bottleneck we attack. Our shortcut result is the game-state instance of a familiar lesson from natural language processing (NLP): benchmarks can be solvable by shallow artifacts (Gururangan et al., 2018), and models (or here, generated programs) can be right for the wrong reasons (McCoy et al., 2019); trivial-program baselines belong in game-agent evaluation for the same reason hypothesis-only baselines—classifiers shown only the hypothesis and not the premise—expose artifacts in natural language inference (NLI) datasets. The abstention behavior we observe under direct prompting (§4.2) connects to work on LLM abstention and calibration (Kirichenko et al., 2025); synthesized evaluators sidestep it entirely by never refusing to score.

3 Setup

3.1 Tasks and data

We reuse the three tasks of Liu (2026) over a 16×16 grid RTS with four unit types (worker, light, heavy, ranged) and two building types (base, barracks). The engine is not microRTS (Ontañón et al., 2018) but is close to it in scope—a compact, fully-observable abstraction of the genre surveyed by Ontañón et al. (2013). An `isBuilding` flag separates buildings (immobile, produce units) from units (move, fight, harvest). The tasks

are: **BOARDREAD** (factual extraction: counts, resources, structures, positions, HP comparison, totals), **STRATEVAL** (score the strategic advantage in $[-10, +10]$), and **WINPREDICT** (predict the eventual winner from a mid-game snapshot). States are mid-game keyframes (turn ≥ 50) harvested from completed bot-vs-bot games, so every snapshot carries its *rollout-verified* outcome—the label is what actually happened, not a judge’s opinion. **BOARDREAD** is scored against ground-truth facts computed from the state; **STRATEVAL** and **WINPREDICT** are scored against the actual winner. For **WINPREDICT** the model predicts the winner directly; for **STRATEVAL** it emits a graded advantage score in $[-10, +10]$ whose *sign* we read as a winner prediction (*direction accuracy*). All sets are pinned (sample identities fixed and replayed), so **every condition scores identical states** and all comparisons are paired.

STRATEVAL and **WINPREDICT** thus take the same input and the same oracle, and differ only in framing. We keep both, following Liu (2026), for two reasons. First, they elicit different behaviors: **WINPREDICT** permits abstention (**TOO_EARLY**), which surfaces the *abstention tax* of §4.2, whereas **STRATEVAL** forces a signed score and additionally exposes magnitude error (correlation, MAE). Second, their agreement is a robustness check on the modality comparison: a code-vs-direct effect that appeared under one framing but not the other would be a prompt artifact, whereas one that holds under both—as every result here does—is a property of the underlying capability, not the question’s wording. The cost: a shared input and oracle make the two tasks’ tests positively correlated, so keeping both doubles the Holm family (§4.4) while adding less than two tasks’ worth of independent evidence, and Holm is conservative accordingly.

We evaluate on two bands of 50 states each per task: the **Material band** follows the standard construction: stratified snapshots from heterogeneous bot games (**WINPREDICT** balanced 25/25 by winner). As §4.3 shows, this band turns out to be solvable by material counting alone—a fact we discovered by auditing. The **Contested band** is our corrective construction. We played 104 new games between *asymmetric* bot strategies (for example, an economy strategy that absorbs an early rush and wins from behind). From these games we harvest, per task, 25 *upset* states (the current material leader eventually loses) and 25 close-margin states, stratified 25/25 by winner. By construction,

$\text{sign}(\Sigma_{\text{HP}})$ scores exactly 50% on these sets. The two hard sets are disjoint; states are not one-per-game (the 50 **WINPREDICT**-hard states come from 21 games, ≤ 4 each), so within-game correlation makes the effective sample somewhat smaller than 50.

3.2 Conditions

A — Direct, answer-first. The prompt of Liu (2026): board text (force lists + ASCII map), answer in-context, *answer before any analysis*. Reasoning models receive a 9,000-token budget so thinking does not truncate answers (§3.4).

A’ — Direct with chain-of-thought (CoT).

Identical except the model is asked to reason step by step first and answer last. This removes a fairness confound in A: the code arms may deliberate at length during generation while A forbids visible deliberation.

B1 — State-blind evaluator (ours). The model receives the game rules, unit-stat table, and state-object schema—*never a board*—and writes one `evaluate(state)` returning $[-10, +10]$. We execute it on all 50 pinned states in a sandbox (2 s timeout, no I/O). $k=3$ independent generations per model; all reported; no best-of- k selection. Failed generations get one error-feedback retry. “Median of $k=3$ ” in the tables means the median-accuracy generation, not a per-state vote: B1 is a single-program arm, matched to the single-sample direct arms rather than to a self-consistency ensemble (Wang et al., 2023). All three generations appear individually in Figure 1.

B2 — Per-board codegen. The model sees the specific board and writes `solve(state)` for that board (the PAL-style deployment). The generated code is saved verbatim for the hardcoding probe.

C — Programmatic references. The engine’s hand-crafted weighted material/economy/infrastructure evaluator—the modern descendant of the hand-tuned combat evaluations studied by Kovarsky and Buro (2005), and the texture arm B1 is implicitly asked to reproduce—and—crucially—**trivial-program baselines:** $\text{sign}(\Sigma_{\text{HP}})$

and `sign(unit count)`. No judgment claim is made without them.

3.3 Validity probes

Decoy probe (hardcoding). Arm B2 writes code for a specific board, so a model could inspect the board, decide the answer, and hardcode it (e.g., `return 7`) instead of computing it. We run each generated program on five *decoy* boards—states sampled evenly across the evaluation set that the program was not written for—and check its outputs alone (a purely behavioral test; we do not inspect the source): a function returning the identical value on all five ignores its input, a direct judgment *laundered* as code. The fraction of programs constant across the decoys is the hardcoding rate. **Mirror antisymmetry** (evaluators): the game is symmetric between players, so *mirroring* a state—swapping the two players and reflecting the board—yields a strategically identical position with the sides reversed. A sound evaluator must then negate its score, $\text{eval}(\text{mirror}(s)) \approx -\text{eval}(s)$; a violation means the evaluator judges by *which* player holds a position rather than the position itself (player-identity bias).

3.4 Models and operational notes

Five open-weight model families served through Ollama’s hosted cloud endpoints (June–July 2026 snapshots; identifiers in Appendix C): qwen3.5, gpt-oss-120b, deepseek-v4-flash, qwen3-coder-480b, glm-5.2. Decoding used temperature 0.3, provider-default top- p , and no seed—the providers expose none, so generation is not bit-reproducible. A sixth family (deepseek-v3.2) was excluded when the provider announced it would be retired two days before this paper’s deadline—a reproducibility hazard of hosted open-weight models that we mitigate by archiving every prompt, pinned state, and generated program alongside the results (prompts in Appendix B, evaluators in Appendix A; the full archive of prompts, pinned states, programs, and result JSONs is released with the paper). Two operational findings that materially affect fairness: (i) reasoning models emit empty content under small output budgets (the budget is consumed by hidden reasoning), which masquerades as abstention—arm A therefore uses a 9,000-token budget rather than disabling reasoning, which would handicap it; (ii) arm B2 disables reasoning, which the same models tolerate for code emission.

Model	STRATEVAL				WINPREDICT			
	A	A′	B1	B2	A	A′	B1	B2
qwen3.5	100	100	98	98	100	100	100	92
glm-5.2	100	100	100	86	98	100	100	68
deepseek-v4-fl.	88	92	94	92	98	98	94	92
gpt-oss-120b	88	92	100 [†]	90	74	78	100	88
qwen3-coder-480b	74	78	100	94	74	62	100	94
mean	90.0	92.4	98.4	92.0	88.8	87.6	98.8	86.8
hand-crafted heur.			76				66	
sign(Σ HP)			100				100	

Table 1: The Material band (original sets, outcomes decided by material): accuracy (%) vs. actual winner over all $n=50$ pinned states per task (abstentions and unparseable answers count as misses). A = answer-first direct; A′ = CoT direct; B1 = state-blind synthesized evaluator, median of $k=3$; B2 = per-board codegen. **Boldface:** the B1 gain survives Holm correction against *both* direct arms (vs. A: $p_{\text{adj}}=.002$; vs. the CoT-fair A′: $p_{\text{adj}}\leq.009$); [†]raw $p=.03$ vs. A, not significant after Holm against either arm. The last row is the audit result of §4.3: a trivial program saturates this band.

4 Results

4.1 RQ1: code generation solves the mechanical faculties

On BOARDREAD, direct answering by qwen3.5 scores 30% on HP comparison while codegen answers **100% of all categories** (500/500 generated programs executed; single-family result—the format ablation below spans four families). Two diagnostic results sharpen the interpretation: (i) the direct-mode failure is largely *parity-blindness*, not arithmetic—on states with a true HP margin, direct scores 8/8, but on exact ties it scores 7/42, refusing to declare EQUAL; (ii) the bottleneck is representation-independent—replacing the ASCII board with clean JSON does not fix HP comparison ($\leq 30\%$ in both formats across four model families), ruling out tokenization. RQ1 thus replicates the PAL/PoT effect in a new domain while correcting its interpretation here: what code offloads is not arithmetic the model lacks, but a *commitment* the model avoids—programs declare ties; models do not.

4.2 RQ2 on the Material band: code wins where judgment is aggregation

Figure 1 (top row) and Table 1 give the Material-band result.

On this band the code arm looks dominant: evaluators match or exceed direct judgment in 8 of 10 cells (worst regression -4pp), and the three

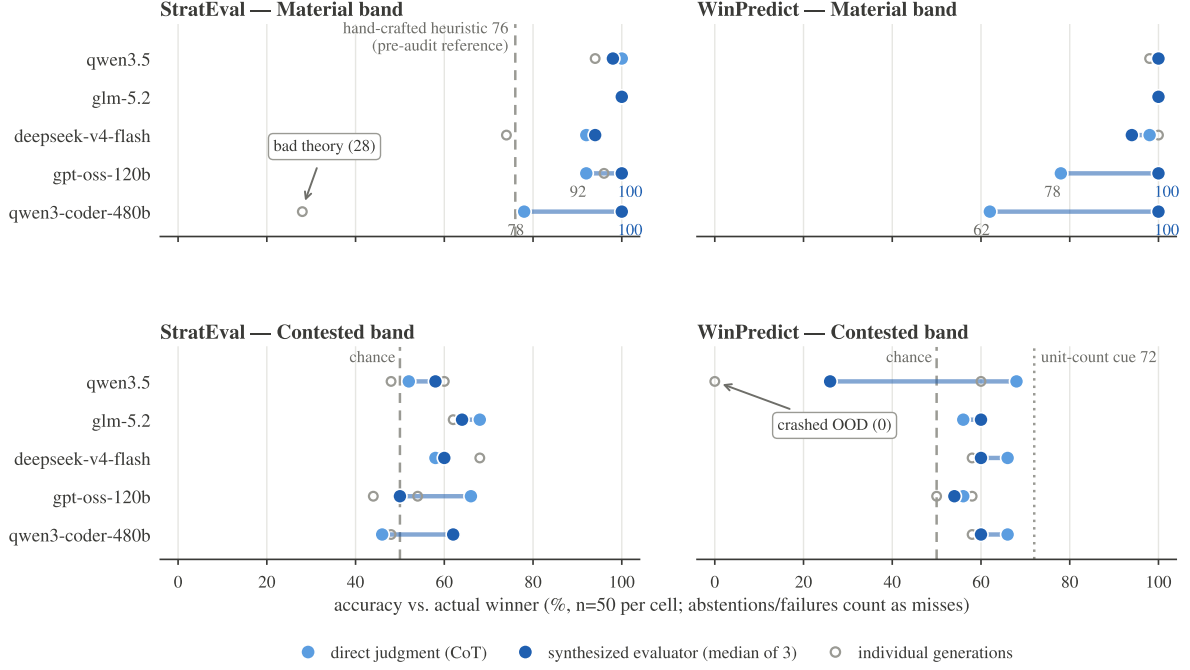


Figure 1: Percentage of 50 game states judged correctly (accuracy against the actual winner) by each of five models, in two ways: *direct judgment*, where the model answers after chain-of-thought reasoning (A', light dots), and *synthesized evaluator*, where we execute an evaluation function the model wrote from the rules alone, never having seen a board (B1, dark dots = median of three generations; open circles = individual generations). Abstentions and code failures count as misses; values for the answer-first direct arm (A) appear in Tables 1 and 2. **Top (Material band, original sets):** evaluators saturate and the large direct-judgment deficits close (three cells survive Holm against both direct arms)—but $\text{sign}(\Sigma\text{HP})$ also scores 100% here, so this band measures reliable aggregation, not judgment. **Bottom (Contested band, hard sets):** with material at chance, every arm collapses to chance-straddling accuracy (LLM arms 36–68%, plus one 26% inversion; Table 2); no LLM arm reliably exceeds chance after correction, one evaluator inverts (26%), one crashes out-of-distribution, and a trivial unit-count cue (72%) beats them all on WINPREDICT.

large deficits close with significance that survives Holm correction (qwen3-coder on both tasks, gpt-oss on WINPREDICT; exact McNemar $p_{\text{adj}}=.002$; gpt-oss's +12pp on STRATEVAL does *not* survive, $p_{\text{adj}}=.22$). Crucially, the gaps are **not an answer-first prompting artifact**: rerunning direct judgment with explicit chain-of-thought (A') lifts the failing models only slightly on STRATEVAL (74→78, 88→92) and *lowers* qwen3-coder on WINPREDICT (74→62, via more abstentions); all three cells remain Holm-significant against A' ($p_{\text{adj}}\leq.009$). Nor do they depend on the median-generation pairing: both WINPREDICT cells are significant against *every* individual generation ($p\leq.001$ each), and coder-STRATEVAL against two of three—the exception being the 28% bad-theory outlier (§4.6), against which direct judgment wins. Part of the direct deficit is an *abstention tax*: gpt-oss declines to predict on 9/50 boards (qwen3-coder with CoT on 7); their

evaluators commit everywhere. Taken alone, this band suggests models write better judges than they are—qwen3-coder judges at 74–78% yet writes 100% evaluators.

4.3 RQ3, the audit: the Material band is solvable by counting

Before believing that story, we ran the baseline that should accompany any judgment claim: $\text{sign}(\Sigma\text{HP})$ scores **100% on both Material-band sets** (50/50 each; so does $\text{sign}(\text{unit count})$). Bot games snowball, so by turn 50 the material leader simply never loses: in 791 frames across our full trace pool there is not a single upset. Three consequences. (i) Material-band judgment is *reducible to aggregation*, the faculty RQ1 already showed code fixes; the evaluators' 94–100% is therefore unremarkable. The honest reading of the Material band is that *direct* judgment fails a task a three-line program solves (74–90%, with

wrong-sign errors and abstentions). (ii) The hand-crafted heuristic (76%) loses to the trivial baseline and cannot anchor claims. (iii) Any evaluation of “strategic judgment” built on snapshots of completed games inherits this hazard; we suspect it extends well beyond our own setup (§5).

4.4 RQ2 on the Contested band: the gains do not survive

The Contested band removes the artifact: material sign is exactly 50% by construction, upsets are half the set, and the states come from matchups where a strategy that is *behind* on material predictably wins. Table 2 and Figure 1 (bottom) show the result: every arm—answer-first direct, CoT direct, the same 15 state-blind evaluators replayed unchanged, and per-board codegen—lands in a chance-straddling 36–68% band (per-board codegen is lowest, its compile failures now costing misses on unfamiliar states). Across 40 above-chance binomial tests (arms $\times$ models $\times$ tasks), **none survives Holm correction**. The only Holm-significant cell is *below* chance: one qwen3.5 evaluator inverts to 26% on WINPREDICT-hard, and a second crashes on 50/50 out-of-distribution states (a latent branch bug never triggered on the Material band). CoT helps direct judgment on average (+3 to +8pp per task) but no per-model improvement is significant. On WINPREDICT-hard, `sign(unit count)` reaches 72%: the eventual winner fields *more, cheaper* units in 36 of the 43 non-tied states. (Worker counts alone are tied in 41/50 states and carry little signal.) Above-chance state-only signal is therefore present in this sample, and no LLM modality reliably finds it. This holds for the sampled states, not the task distribution: the cue rests on 50 states, and composition partly identifies the scripted strategy (§7). On STRATEVAL-hard we found *no* state-only cue above chance (best: total units, $\approx 60\%$, n.s.), so for that set we cannot distinguish “models miss the signal” from “single frames carry too little signal”—the boundary claim rests on WINPREDICT-hard. We also note a construction caveat: the Contested band is selected adversarially against material, and the evaluators are, empirically, material theories—so their collapse is partly by construction. The finding is precisely that: **invited to use any feature of the state, models write only material theories**, and the unit-count cue shows a better theory was available.

Model	STRATEVAL-hard				WINPREDICT-hard			
	A	A'	B1	B2	A	A'	B1	B2
qwen3.5	62	52	58	60	62	68	26	50
glm-5.2	60	68	64	48	60	56	60	36
deepseek-v4-fl.	48	58	60	56	58	66	60	46
gpt-oss-120b	54	66	50	42	48	56	54	42
qwen3-coder-480b	48	46	62	58	44	66	60	68
mean	54.4	58.0	58.8	52.8	54.4	62.4	52.0	48.4
chance	50				50			
sign(Σ HP)	50 (by design)				50 (by design)			
sign(unit count)	46				72			
hand-crafted heur.	46				56			

Table 2: The Contested band (hard sets, material at chance): accuracy (%) vs. actual winner, all-samples metric, $n=50$. A = answer-first direct; A' = CoT direct; B1 = median of the same $k=3$ state-blind evaluators from the Material band, replayed unchanged; B2 = per-board codegen (code failures count as misses). Across 40 above-chance binomial tests, none survives Holm correction; the trivial unit-count cue exceeds every LLM arm on WINPREDICT-hard.

4.5 Reliability, hardcoding, and fragility

On the Material band, all 15 blind evaluators compiled and executed on 100% of boards, with zero degenerate functions and zero antisymmetry violations (max deviation 0.04 on the ± 10 scale). B2 per-board code ran 82–100% of the time depending on model and task (fresh compile/run risk on every board; per-model counts in Table 3). The decoy probe found **0% hardcoding** in B2 on STRATEVAL (0/243 runnable programs constant across decoy boards); on WINPREDICT 2–22% of programs were constant on decoys, but with binary outputs this is weak evidence and we report it only as an upper bound (Appendix C). The Contested band adds a fragility result: replayed on out-of-distribution states, one evaluator that was perfect on the Material band crashed on all 50 boards via a latent branch bug (`hpRatio` is not defined—a code path the Material band never exercised), and another inverted to 26%. Code that executes flawlessly on the distribution it was validated on can still be brittle one distribution away.

4.6 The bad-theory tail

Single-shot evaluator synthesis has a failure mode our structural probes cannot catch: one qwen3-coder generation scored 28% on the Material band while being non-degenerate and perfectly antisymmetric—a *coherent but wrong theory*. It

keeps a running total for each player and returns their difference. Its economy and production terms, however, are *already* Player-1-minus-Player-2 differences, and the code adds each to Player 1’s total *and* subtracts it from Player 2’s; since the final score is the difference of the two totals, every such term is counted *twice* (a true production edge of 5 becomes 10), and together they come to outweigh the material term that actually tracks the winner. The totals are also raw and *unnormalized*—a single base contributes ten points—so the score routinely runs to several tens and slams against the $[-10, +10]$ clamp (*saturation*), erasing the fine material difference that would break a close call. On states with a narrow true margin the sign flips: the evaluator names the wrong winner while looking, structurally, exactly like its 100% siblings (Appendix A annotates the code). Without ground-truth outcomes the two are indistinguishable. The Contested band generalizes the lesson. The same synthesis process that yields 100% evaluators on one distribution yields 26–68% evaluators one distribution away. *All* synthesized evaluators therefore require validation on labeled rollouts from the deployment distribution, and all k generations should be reported—as we do here, since a best-of- k pick would hide exactly this tail.

5 Discussion

A boundary, not a victory. The two bands together answer the title’s question precisely. *Compute* wins wherever the answer is a computable function of enumerable state features. That covers extraction and aggregation (RQ1, 30%→100%), and it covers judgment on any distribution where outcomes reduce to such features: on the Material band, the Holm-surviving gains are exactly the cells where direct judgment failed at what was secretly an aggregation task. Beyond that line, on states where winning is a matter of strategy rather than material, *neither* modality currently works: direct judgment, CoT judgment, and synthesized evaluators are statistically indistinguishable from chance, while a trivial composition cue proves the states are not unpredictable. The honest routing recipe: (1) route extraction/aggregation through generated code; (2) if evaluation is needed on a material-driven distribution, a single state-blind evaluator is the best value—one LLM call ever, microsecond inference, no abstention, matches

or beats direct judgment—but *validate it on labeled rollouts from the deployment distribution first* (§4.4); (3) do not ask for code per state: B2 is dominated by B1 on reliability and cost (50× the calls) with no accuracy advantage; (4) treat any above-chance judgment claim as unproven until trivial-program baselines are reported.

Benchmark shortcuts beyond this paper. Our Material-band audit generalizes. Any evaluation that scores “strategic understanding” against outcomes of completed bot or self-play games inherits the snowball hazard—win prediction from snapshots, advantage annotation, and state-value probes alike. If the material leader never loses, the task is aggregation wearing judgment’s clothes, and code-augmented or even trivial baselines will saturate it. We recommend the two-line audit (report `sign( $\Sigma$ material)`; count upsets) and, where it fails, the Contested-band construction: harvest upset and close-margin states from asymmetric-strategy games until the trivial baseline is at chance.

Relation to modality steering. CodeSteer (Chen et al., 2025) shows models mis-choose *when* to write code on mechanical tasks. Our results add a second axis: the *unit* of code matters (one reusable evaluator beats per-query programs), and there is a task region—judgment beyond computable features—where steering between modalities cannot help because no modality has the capability. Mapping that region’s edge, rather than assuming code or text must win, is the open problem.

Unguided synthesis, and a longitudinal test. One scope decision deserves emphasis: our code prompts are deliberately minimal. The model receives the rules, the unit-stat table, and the state schema, and is asked to use whatever features it finds predictive—no algorithms, no strategy hints. We measure what a model writes *unaided*, not what it can be coaxed into writing. Richer guidance would plausibly help. Prompting for classic game-AI machinery—adversarial lookahead such as alpha-beta pruning, Monte Carlo tree search, or named strategic features like the rock-paper-scissors unit matchup—could elicit stronger judgment code than the static evaluators we obtained. (Search-based methods also require exposing a forward model, which shifts toward the agentic scaffolds we bracket in §7.) But such prompt-

guided iteration is, today, human-in-the-loop development with LLM assistance—valuable, yet a different question from the unaided comparison we pose. The forward-looking version of our question is temporal. Coding models are improving quickly, and one that autonomously writes lookahead-quality evaluators would shift the boundary we draw. Our harness, pinned states, and contested sets are archived, and the synthesized-evaluator arm costs no new model calls to re-score, so re-running this exact comparison on each new coding model is a cheap longitudinal probe of when—perhaps soon—compute begins to cross into judgment.

6 Conclusion

Answer or compute? For LLM game-state understanding the answer is *compute*, within a clear boundary. Wherever the task is a computable function of the state, generated code beats direct answering: extraction and aggregation rise from unreliable to perfect (30%→100%), and a single state-blind evaluator delivers that advantage for one LLM call, with no abstentions, however the direct arm is prompted. The boundary is strategic judgment. On contested states, where material does not decide the game, neither direct answering nor any form of generated code is reliably better than chance—even though a simple composition cue shows that 72% is attainable on these states. The claim is the narrow one: at $n=50$ against a 40-test family the design rules out *strong* judgment ability but cannot exclude a modest above-chance one (§7); the best arm reaches 62%, two predictions short of the cue. Today’s models can be made reliable *computers* of a game state; on this benchmark they cannot yet be shown to be *judges* of one. Closing the gap between their 62% and the cue’s 72% is the concrete next target. Reaching this answer required auditing and rebuilding our own evaluation sets; we release that audit and hard-set construction for others evaluating judgment on game traces.

7 Limitations

Our evidence comes from a single RTS game with a compact, fully documented state schema; richer games (fog of war, continuous state, thousands of entities) may not admit the clean `evaluate(state)` contract that makes state-blind synthesis possible. Second, Contested-band

states come from scripted-bot games with a small strategy pool. Outcomes there are partly a function of *which strategies are playing*; the unit-count cue works precisely because composition identifies the strategy. The Contested band therefore measures recognition of strategy-vs-strategy dynamics, not open-play judgment, and its achievable ceiling is unknown—72% is a lower bound on what a competent judge could reach. Third, $n=50$ per cell means small deltas are noise; we rest no claim on them, report exact McNemar tests with Holm correction, and note that our own headline Material-band significance list shrank under correction (gpt-oss STRATEVAL +12pp did not survive). The Contested-band negative result is correspondingly scoped: with only 50 states per cell and a 40-test family (4 arms $\times$ 5 models $\times$ 2 tasks), a cell would need roughly 70% or more to survive correction. The design therefore rules out *strong* judgment ability, but cannot exclude a *modest, genuinely-above-chance* one ($\approx 60\text{--}65\%$). Fourth, the negative Contested-band result is about *current* models under two prompting styles and single-shot synthesis; agentic scaffolds (evaluators refined against labeled rollouts, tool loops) may cross the boundary and are deliberately out of scope here. Fifth, our five families are hosted open-weight models; one further family was retired by the provider mid-study, and while we archive every prompt, pinned state, and generated program, exact replication of the generation step depends on the provider. Finally, rollout-verified outcomes under bot play are our ground truth; human expert annotation could disagree, though the Material-band audit suggests expert labels would face the same snowball artifact.

AI Assistance Disclosure

Experiment design and implementation, execution and analysis, and the drafting and revision of this manuscript were carried out with assistance from Claude Code (Anthropic; Claude Fable 5 and Claude Opus models) under the author’s direction. The author reviewed and validated all analyses, claims, and text, and takes full responsibility for the content.

References

- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research (TMLR)*. ArXiv:2211.12588.
- Yongchao Chen, Harsh Jhamtani, Srinagesh Sharma, Chuchu Fan, and Chi Wang. 2025. Steering large language models between code execution and textual reasoning. In *International Conference on Learning Representations (ICLR)*. ArXiv:2410.03524.
- Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. In *International Conference on Learning Representations (ICLR)*. ArXiv:2210.02875.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*. ArXiv:2211.10435.
- Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *Proceedings of NAACL-HLT*. ArXiv:1803.02324.
- San Kim, Daechul Ahn, Reokyoung Kim, et al. 2026. RTSGameBench: An RTS benchmark for strategic reasoning by vision-language models. *arXiv preprint arXiv:2606.18950*.
- Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, et al. 2025. AbstentionBench: Reasoning LLMs fail on unanswerable questions. *arXiv preprint arXiv:2506.09038*.
- Alexander Kovarsky and Michael Buro. 2005. *Heuristic search applied to abstract combat games*. In *Advances in Artificial Intelligence (Canadian AI)*, Lecture Notes in Computer Science, pages 66–78.
- Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2024. Chain of code: Reasoning with a language model-augmented code emulator. In *International Conference on Machine Learning (ICML)*. ArXiv:2312.04474.
- Chang Liu. 2026. Evaluating LLM capacity for game state assessment in RTS games. In *Computers and Games (CG)*. Springer.
- Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*. ArXiv:2310.12931.
- R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of ACL*. ArXiv:1902.01007.
- Santiago Ontañón, Nicolas A Barriga, Cleyton R Silva, Rubens O Moraes, and Levi H S Lelis. 2018. The first microRTS artificial intelligence competition. *AI Magazine*, 39(1):75–83.
- Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. 2013. A survey of real-time strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(4):293–311.
- Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, et al. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475.
- Shengjie Sun, Runze Liu, Jiafei Lyu, et al. 2024. A large language model-driven reward design framework via dynamic feedback for reinforcement learning. *arXiv preprint arXiv:2410.14660*.
- Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. ViperGPT: Visual inference via python execution for reasoning. In *International Conference on Computer Vision (ICCV)*. ArXiv:2303.08128.
- Wenjie Tang, Yuan Zhou, Erqiang Xu, et al. 2025. DSGBench: A diverse strategic game benchmark for evaluating LLM-based agents in complex decision-making environments. *arXiv preprint arXiv:2503.06047*.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning (ICML)*. ArXiv:2402.01030.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.
- Yanan Wang, Renxi Wang, Yongxin Wang, et al. 2026. SimuScene: Training and benchmarking code generation to simulate physical scenarios. *arXiv preprint arXiv:2602.10840*.

A Example synthesized evaluators

All code verbatim from archived results (state-blind arm B1; STRATEVAL direction accuracy in parentheses).

A.1 qwen3-coder#e1 (100%): a compact material-ratio theory

HP-weighted material value plus a small resource term, normalized to a ratio. Nine lines of logic outperform the hand-crafted heuristic by 24 points.

```
function evaluate(state) {
  const playerScores = [0, 0];
  const unitValues = {
    worker: 1,
    light: 2,
    heavy: 4,
    ranged: 2,
    base: 10,
    barracks: 5
  };

  for (const entity of state.entities) {
    if (entity.hp <= 0) continue;
    const playerIndex = entity.playerId;
    const value = unitValues[entity.type] || 0;
    playerScores[playerIndex] += value * (entity.hp / entity.maxHp);

    if (entity.type === 'base' && entity.hp > 0) {
      playerScores[playerIndex] += state.resources[
        playerIndex] * 0.1;
    }
  }

  const p1Score = playerScores[0];
  const p2Score = playerScores[1];
  const totalScore = p1Score + p2Score;

  if (totalScore === 0) return 0;

  const normalizedScore = (p1Score / totalScore) *
    20 - 10;
  return Math.max(-10, Math.min(10, normalizedScore));
}
```

A.2 gpt-oss#e2 (100%): material + positional pressure

Derives per-unit values from the stat table (cost, HP, damage, range), scales by current HP, adds proximity-to-enemy-base pressure for combat units and alive-infrastructure bonuses—a texture very close to hand-crafted RTS evaluators, written blind.

```
function evaluate(state){
  const GRID_SIZE = 16;
  const MAX_DIST = GRID_SIZE * 2;
  const RESOURCE_WEIGHT = 0.5;
  const BASE_ALIVE_BONUS = 3;
  const BARRACKS_ALIVE_BONUS = 4;
  const MAX_DIFF = 100; // rough upper bound for scaling

  function baseValue(type){
    switch(type){
      case 'worker': return 1 + 0.2*1 + 0.3*1;
        // 1.5
      case 'light': return 2 + 0.2*5 + 0.3*2;
        // 3.6
      case 'heavy': return 4 + 0.2*7 + 0.3*4;
        // 6.6
      case 'ranged': return 2 + 0.2*2 + 0.3*1 +
        0.1*3; // 3.0 (range bonus)
      case 'barracks': return 5 + 0.2*5;
        // 6
      case 'base': return 10 + 0.2*10;
        // 12
      default: return 0;
    }
  }
}
```

```
// locate each player's base (if any)
const bases = [null, null];
for (const e of state.entities) {
  if (e.type === 'base') bases[e.playerId] = e;
}

const scores = [0, 0];

// material value + positional advantage for
// combat units
for (const e of state.entities) {
  const val = baseValue(e.type) * (e.hp / e.maxHp);
  scores[e.playerId] += val;

  // positional factor for combat units (not
  // workers, not buildings)
  if (e.type !== 'worker' && e.type !== 'base' &&
    e.type !== 'barracks') {
    const enemyId = 1 - e.playerId;
    const enemyBase = bases[enemyId];
    if (enemyBase) {
      const dist = Math.abs(e.x - enemyBase.x) +
        Math.abs(e.y - enemyBase.y);
      const factor = (MAX_DIST - dist) / MAX_DIST;
      // 0 = far, 1 = on base
      scores[e.playerId] += val * 0.5 * factor; //
        0.5 weight for positioning
    }
  }
}

// resources (future production potential)
scores[0] += state.resources[0] * RESOURCE_WEIGHT;
scores[1] += state.resources[1] * RESOURCE_WEIGHT;

// production capacity: alive base -> can make
// workers
if (bases[0]) scores[0] += BASE_ALIVE_BONUS;
if (bases[1]) scores[1] += BASE_ALIVE_BONUS;

// barracks alive -> can make combat units
const barracksAlive = [0, 0];
for (const e of state.entities) {
  if (e.type === 'barracks') barracksAlive[e.
    playerId] = 1;
}
if (barracksAlive[0]) scores[0] +=
  BARRACKS_ALIVE_BONUS;
if (barracksAlive[1]) scores[1] +=
  BARRACKS_ALIVE_BONUS;

const diff = scores[0] - scores[1];
let result = (diff / MAX_DIFF) * 10;

if (result > 10) result = 10;
else if (result < -10) result = -10;

return result;
}
```

A.3 qwen3-coder#e3 (28%): a coherent but wrong theory

Structurally valid (non-degenerate, perfectly anti-symmetric), yet mis-weighted: the resource, production, and position “advantage” terms are each added to one side *and* subtracted from the other (double-counting them in the final difference), and the final score is a raw un-normalized difference that saturates the $[-10, 10]$ clamp—so mid-sized noise terms invert close calls. No structural probe can catch this; only ground-truth outcomes do.

```
function evaluate(state) {
  const playerScores = [0, 0];
  const unitValues = {
    worker: 1,
```

```

    light: 2,
    heavy: 4,
    ranged: 2,
    base: 10,
    barracks: 5
  };

  // Count entities and calculate material value
  const entityCounts = [[], []];
  for (let i = 0; i < 2; i++) {
    entityCounts[i]['worker'] = 0;
    entityCounts[i]['light'] = 0;
    entityCounts[i]['heavy'] = 0;
    entityCounts[i]['ranged'] = 0;
    entityCounts[i]['base'] = 0;
    entityCounts[i]['barracks'] = 0;
  }

  for (const entity of state.entities) {
    const playerId = entity.playerId;
    if (playerId === 0 || playerId === 1) {
      playerScores[playerId] += unitValues[entity.type] * (entity.hp / entity.maxHp);
      entityCounts[playerId][entity.type]++;
    }
  }

  // Economic advantage
  const resourceAdvantage = state.resources[0] - state.resources[1];
  playerScores[0] += resourceAdvantage * 0.1;
  playerScores[1] -= resourceAdvantage * 0.1;

  // Production capabilities
  const productionAdvantage = (entityCounts[0]['base'] * 10 + entityCounts[0]['barracks'] * 5) - (entityCounts[1]['base'] * 10 + entityCounts[1]['barracks'] * 5);
  playerScores[0] += productionAdvantage;
  playerScores[1] -= productionAdvantage;

  // Combat potential
  const combatPotential = [
    entityCounts[0]['light'] * 2 + entityCounts[0]['heavy'] * 4 + entityCounts[0]['ranged'] * 1,
    entityCounts[1]['light'] * 2 + entityCounts[1]['heavy'] * 4 + entityCounts[1]['ranged'] * 1
  ];
  playerScores[0] += combatPotential[0] * 0.5;
  playerScores[1] += combatPotential[1] * 0.5;

  // Worker count (economy growth)
  playerScores[0] += entityCounts[0]['worker'] * 0.5;
  playerScores[1] += entityCounts[1]['worker'] * 0.5;

  // Positioning advantage (proximity to enemy base)
  let positionScore = 0;
  let p1Base = null;
  let p2Base = null;
  for (const entity of state.entities) {
    if (entity.type === 'base') {
      if (entity.playerId === 0) p1Base = entity;
      else p2Base = entity;
    }
  }

  if (p1Base && p2Base) {
    for (const entity of state.entities) {
      if (entity.playerId === 0 && !entity.isBuilding) {
        const dist = Math.abs(entity.x - p2Base.x) + Math.abs(entity.y - p2Base.y);
        positionScore += (15 - dist) * 0.1;
      } else if (entity.playerId === 1 && !entity.isBuilding) {
        const dist = Math.abs(entity.x - p1Base.x) + Math.abs(entity.y - p1Base.y);
        positionScore -= (15 - dist) * 0.1;
      }
    }
  }

  playerScores[0] += positionScore;

```

```

playerScores[1] -= positionScore;

// Final score calculation
const totalScore = playerScores[0] - playerScores[1];
return Math.max(-10, Math.min(10, totalScore));
}

```

B Prompts

B.1 Evaluator-synthesis prompt (arm B1, verbatim)

Identical for STRATEVAL and WINPREDICT; the model never sees a board.

You are an expert RTS game analyst and JavaScript programmer.

THE GAME: a 16x16-grid real-time strategy game between Player_1 (starts top-left) and Player_2 (starts bottom-right). Workers gather resources; a base produces workers; a barracks produces combat units (light, heavy, ranged). A player whose base is destroyed cannot produce workers; a player without a barracks cannot produce combat units. The game ends when one player's base and units are all destroyed (the other player wins), or at the turn limit.

UNIT STATS: worker (hp:1, cost:1, dmg:1), light (hp:5, cost:2, dmg:2, spd:2), heavy (hp:7, cost:4, dmg:4), ranged (hp:2, cost:2, dmg:1, range:3)

Buildings: base (hp:10, cost:10), barracks (hp:5, cost:5)

Write a JavaScript function evaluate(state) that scores who is winning. The 'state' argument has this exact shape:

```

state.turn // number - current turn
state.resources // [p1, p2] - index 0 = Player_1, index 1 = Player_2
state.entities // array; each entity:
                // { type: 'base'|'barracks'|'worker'|'light'|'heavy'|'ranged',
                //   playerId: 0 (Player_1) | 1 (Player_2),
                //   x, y, hp, maxHp,
                //   isBuilding }
state.terrain // 2D array terrain[y][x]; 1 = blocked

```

Return a number from -10 to +10: +10 = Player_1 totally dominant, 0 = perfectly even, -10 = Player_2 totally dominant. The SIGN of your score will be used to predict the eventual winner, so make the score reflect true winning chances as accurately as you can.

Requirements:

- Use whatever features of 'state' you consider predictive (material, hp, economy, production capacity, positioning - your choice).
- Compute the score from 'state'; never return a constant.
- Respond with ONLY the JavaScript code defining function evaluate(state). No prose, no markdown fences.

B.2 Direct prompt (arm A, STRATEVAL; template)

Per-board fields shown as {placeholders}; the WINPREDICT variant asks for PLAYER_1 /

PLAYER_2 / TOO_EARLY instead of the score block.

```
You are an expert RTS game analyst evaluating a 16
x16 strategy game.

=== GAME STATE: TURN {turn}/{maxTurns} ===
RESOURCES: Player_1={r1}, Player_2={r2}

PLAYER_1 FORCES (UPPERCASE on map, starts top-left):
{force list}
PLAYER_2 FORCES (lowercase on map, starts bottom-
right):
{force list}
MAP:
{16x16 ASCII grid}

UNIT STATS: worker(hp:1,cost:1,dmg:1), light(hp:5,
cost:2,dmg:2,spd:2), heavy(hp:7,cost:4,dmg:4),
ranged(hp:2,cost:2,dmg:1,range:3)
Buildings: base(hp:10,cost:10), barracks(hp:5,cost
:5)

Analyze who has the strategic advantage.

IMPORTANT: You MUST start your response with the
SCORE line. Do NOT write any analysis before it
.

Respond in this EXACT format (no markdown, no bold,
no extra text before SCORE):

SCORE: [integer from -10 to +10, where -10 =
Player_2 dominant, 0 = even, +10 = Player_1
dominant]
FACTORS:
1. [most important factor]
2. [second factor]
3. [third factor]
```

B.3 Per-board codegen prompt (arm B2; template)

```
You are writing a JavaScript function solve(state)
for a 16x16 RTS game board.

Here is the board it will run on:
=== GAME STATE: TURN {turn}/{maxTurns} ===
RESOURCES: Player_1={r1}, Player_2={r2}

{force lists + 16x16 ASCII grid, as in the direct
prompt}

UNIT STATS: worker(hp:1,cost:1,dmg:1), light(hp:5,
cost:2,dmg:2,spd:2), heavy(hp:7,cost:4,dmg:4),
ranged(hp:2,cost:2,dmg:1,range:3)
Buildings: base(hp:10,cost:10), barracks(hp:5,cost
:5)

The 'state' argument passed to solve(state) has this
exact shape:
{state schema, as in the evaluator prompt}

Write solve(state) that returns a strategic-
advantage score for this board:
a number from -10 (Player_2 dominant) through 0 (
even) to +10 (Player_1 dominant).
[WinPredict variant: predicts the eventual winner of
this game: return exactly
the string 'PLAYER_1' or 'PLAYER_2'.]

Requirements:
- Compute the answer from 'state'. Do NOT hardcode
values.
- Respond with ONLY the JavaScript code defining
function solve(state). No prose, no markdown
fences.
```

C Reliability, probes, and model snapshots

Model snapshots. Ollama-cloud identifiers, June–July 2026: qwen3.5:cloud, gpt-oss:120b-cloud, deepseek-v4-flash:cloud, qwen3-coder:480b-cloud, glm-5.2:cloud. Runs used the pinned sample files and archived result JSONs released with the paper; the artifact link will be de-anonymized at camera-ready.

Execution reliability. All 15 B1 evaluators (5 models $\times k=3$) compiled and executed on 100% of boards in both tasks. B2 per-board code reliability, out of 50 boards each:

Model	STRATEVAL	WINPREDICT
qwen3.5	50/50	48/50 (run:2)
gpt-oss-120b	49/50 (compile:1)	48/50 (compile:2)
deepseek-v4-fl.	48/50 (run:2)	49/50 (run:1)
qwen3-coder-480b	50/50	49/50 (compile:1)
glm-5.2	46/50 (c:2,n:1,r:1)	41/50 (c:9)

Table 3: Arm B2: generated programs that ran and returned a usable value (c = compile error, r = runtime error, n = non-numeric return).

Contested-band execution health. Replayed on the hard sets, 13/15 Material-band evaluators ran on 100% of states; one crashed on 50/50 (latent branch bug, hpRatio is not defined) and is scored 0%. B2-hard per-board code ran: STRATEVAL 45–50/50 (gpt-oss compile:5, glm-5.2 three failures); WINPREDICT similar, with failures counted as misses in Table 2. **Composition-cue analysis (hard sets):** worker counts are tied in 41–45/50 states (near-uninformative); the 72% WINPREDICT-hard cue is total unit count (winner has more units in 36/43 non-tied states); on STRATEVAL-hard no cue tested exceeded chance significantly.

Probes. B1: 0/15 evaluators degenerate; 0 anti-symmetry violations across all mirror pairs (max deviation 0.04 on the ± 10 scale). B2 decoy probe, STRATEVAL: 0% of runnable programs constant across five decoy boards, for every model (243 programs). WINPREDICT: constant-on-decoys rates were qwen3.5 3/48, gpt-oss 1/48, deepseek-v4-flash 11/49, qwen3-coder 2/49, glm-5.2 8/41—an *upper bound* on hardcoding, since with a binary

output a computed-but-biased predictor can also be constant on five boards.