TSUGO: PROBING SEARCH EFFICIENCY IN LLM REASONING VIA GO LIFE-AND-DEATH PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The evaluation of LLM reasoning is moving from final-answer accuracy to process-level assessment, yet existing methods still fail to capture how models plan reasoning paths and allocate reasoning resources—that is, how they organize search. Prior process-level methods focus on the coherence and redundancy of chain-of-thought (CoT), and most benchmark tasks have a single objective solvable by static capabilities such as derivation and tool use, leaving search organization unmeasured. We introduce TSUGO, a process-level reasoning benchmark for evaluating Search Efficiency in LLM reasoning through Go life-and-death problems. These problems provide closed and verifiable solution spaces with an inherent adversarial structure, making candidate generation, response checking, branch comparison, and backtracking necessary parts of reasoning rather than incidental trace patterns. By constraining the solution space, TSUGO disentangles domain knowledge from search organization, parses CoT into a structured search tree, and reports Search Efficiency together with Token Efficiency and other diagnostic metrics and visualizations. Experiments show that current LLMs remain far from stable tsumego solving: stronger models succeed by finding the correct candidate earlier and sustaining effort on productive branches, but most models still behave much closer to unguided search algorithms than to neural-guided KataGo. Longer CoT or higher Token Efficiency does not necessarily imply better search. Our results identify search organization and reasoning-resource allocation as missing dimensions in LLM reasoning evaluation.

1 INTRODUCTION

Chain-of-Thought (CoT) and extended thinking improve LLM performance across mathematics, code, and science, motivating evaluation of reasoning effectiveness and efficiency beyond final-answer correctness Wei et al. (2022); OpenAI (2024); DeepSeek-AI (2025).

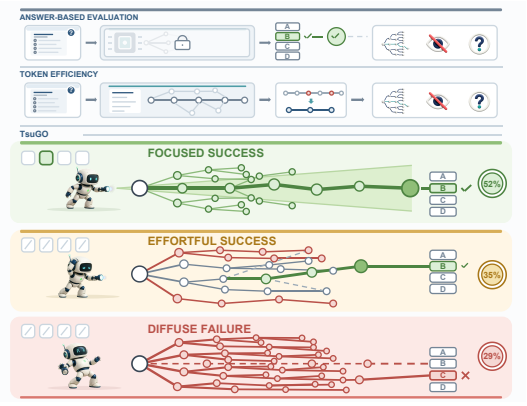


Figure 1: TSUGO reveals search organization beyond answer and token efficiency.

Existing evaluations mainly study the quality and efficiency of reasoning along a single trajectory. Traditional benchmarks such as GSM8K Cobbe et al. (2021), MATH Hendrycks et al. (2021), and ARC Clark et al. (2018) focus on final-answer correctness, while recent process-level methods analyze reasoning traces themselves. CoTJudger measures necessary reasoning and structural redundancy from dependency graphs Li et al. (2026); ReEfBench maps traces into logical structures to analyze reasoning efficiency and behavioral patterns Fu et al. (2026); and process-supervision or PRM-based methods evaluate intermediate steps through step-level feedback Lightman et al. (2024); Song et al. (2025). These studies show whether a reasoning chain is correct, concise, and logically coherent, but mainly assume that effective reasoning

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.

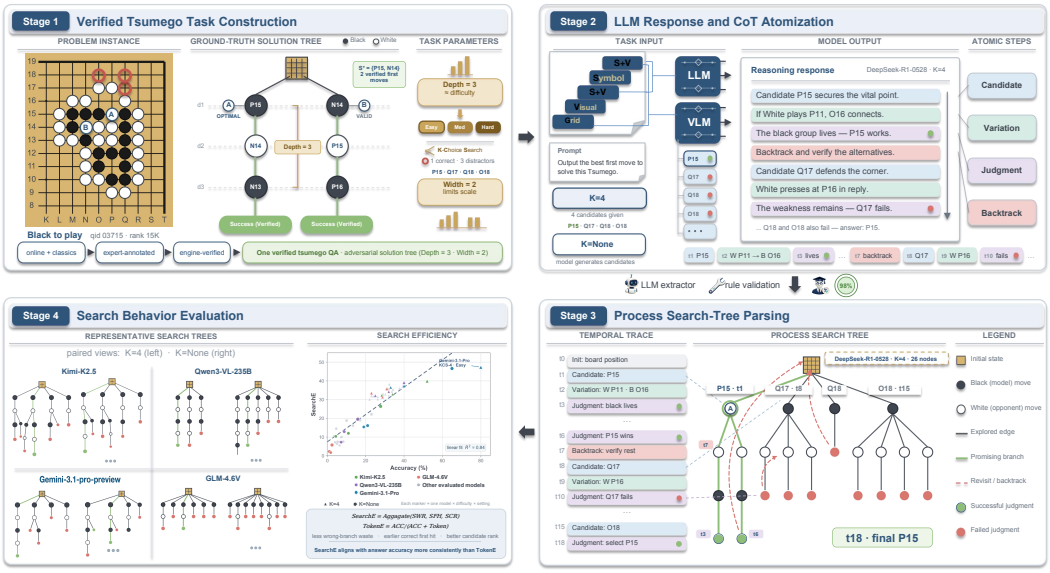


Figure 2: Overview of TSUGO: verified problems are rendered into five modalities, solved through bounded or open K-Search, parsed into process search trees, and evaluated by Search Efficiency with supporting diagnostics.

follows a single derivation path, where the key question is which steps are necessary and which can be removed.

Many challenging problems require more than static problem solving along a single derivation path: models must explore alternatives, identify promising directions, and allocate resources across competing paths. We call this capability **Search Efficiency** (**SearchE**): organizing reasoning search toward effective solutions across multiple possible trajectories. Compared with Token Efficiency, which reflects observable token cost, SearchE is more internal: it tracks how models move through the solution space and redistribute effort across branches, rather than only how much surface text they produce.

Adversarial tasks naturally expose this ability because each decision must survive future responses, forcing comparison, verification, and revision. Branching is therefore not inherently redundant; effective reasoning depends on whether exploration is organized around valuable paths. Classical game AI and recent LLM search studies show the importance of explicit search Silver et al. (2016; 2017); Gandhi et al. (2024); Feng et al. (2024), but LLM benchmarks still lack a controlled way to evaluate search organization itself.

To address this gap, we introduce TSUGO, a benchmark based on Go life-and-death problems for evaluating SearchE in LLM reasoning. The name draws on *tsumego* (*tsume-go*), the Japanese term for Go life-and-death problems, and emphasizes how models choose where to go next. Tsumego offers a controlled adversarial environment with verifiable solution spaces, enabling analysis of both final answers and search over alternatives. We parse free-form CoT into process search trees and report SearchE, which measures whether resources target the correct branch, and TokenE, which normalizes accuracy by observable token cost. Experiments show that stronger models succeed by proposing the correct candidate earlier and sustaining effort on productive branches, establishing search organization as a dimension beyond outcome accuracy and token efficiency.

Our contributions are threefold:

- We introduce TSUGO, a process-level benchmark for evaluating Search Efficiency in controlled adversarial tasks with verifiable solution spaces (§3).
- We propose a search-trace framework that converts free-form CoT into process search trees and diagnoses resource allocation with Search Efficiency and supporting trajectory/scale diagnostics (§4).

- We show that Search Efficiency aligns more closely with search organization than token-level efficiency, and that TSUGO remains far from saturated: even frontier LLMs lag far behind neural-guided KataGo, making search organization a key underdeveloped direction for LLM reasoning (§5).

2 RELATED WORK

Reasoning Evaluation Benchmarks. Benchmarks such as GSM8K, MATH, ARC, and BIG-Bench evaluate outcome accuracy Cobbe et al. (2021); Hendrycks et al. (2021); Clark et al. (2018); Srivastava et al. (2023), while formal and synthetic logic benchmarks test rule-based inference and constraint satisfaction Han et al. (2024); Saparov & He (2023); Lin et al. (2025); Liu et al. (2025). Self-consistency improves outcomes through sampling Wang et al. (2023), but outcome-based evaluation does not reveal the reasoning process.

Recent studies have begun to analyze intermediate reasoning traces. ROSCOE and ReCEval evaluate CoT quality Golovneva et al. (2023); Prasad et al. (2023); CoTJudger measures reasoning redundancy through dependency graphs and shortest effective paths Li et al. (2026); ReEfBench analyzes reasoning structures and efficiency Fu et al. (2026); ProcessBench, PRMBench, and process reward modeling evaluate step-level correctness and supervision signals Zheng et al. (2025); Song et al. (2025); Lightman et al. (2024). Other works study the cost, scaling behavior, inference-time behavior, and faithfulness of long CoT Li et al. (2025); Luo et al. (2025); Sui et al. (2025); Chen et al. (2025); Shen et al. (2025); Parashar et al. (2025); Gandhi et al. (2025). These methods mainly evaluate the quality of a single reasoning trajectory. In contrast, TSUGO studies how models organize search across multiple possible trajectories.

Search-based Reasoning. Meanwhile, recent work improves LLM reasoning by introducing explicit search procedures. Tree-of-Thought and Graph-of-Thoughts represent intermediate thoughts as trees or graphs Yao et al. (2023); Besta et al. (2024); RAP formulates reasoning as planning Hao et al. (2023); LATS, AlphaZero-like search, Stream of Search, and LE-MCTS explore search-based inference and training Zhou et al. (2024); Feng et al. (2024); Gandhi et al. (2024); Park et al. (2025). These methods show that structured search can improve reasoning when supplied as an external procedure, but leave open whether LLMs can organize search within their own reasoning traces. TSUGO evaluates this internal search-organization ability directly.

Adversarial Reasoning Environments. Adversarial environments provide structured spaces for studying planning and search, as demonstrated by AlphaGo and AlphaGo Zero Silver et al. (2016; 2017). LLM studies of chess, Othello, and Go mainly evaluate state tracking, move prediction, or task performance Toshniwal et al. (2023); Li et al. (2023); Ma et al. (2025); TSUGO instead uses life-and-death problems to analyze how models explore, verify, and organize reasoning search.

3 TSUGO

3.1 WHY TSUMEGO?

As introduced above, TSUGO uses Go life-and-death problems to study how models organize search rather than whether they know a final answer. This setting is especially suitable because adversarial solution spaces are large and dynamic: each candidate move changes the space of possible replies, and a line remains valid only if it survives the opponent's strongest response. Tsumego provides this structure in a compact and verifiable form: relevant moves are local, variations can be checked against reference solutions and engine analysis, and positions can be rendered as coordinates, matrices, or images. We therefore use tsumego not to test Go strength, but as a closed adversarial environment where the difficulty of Go-style search comes from dynamically changing branches and the need to separate domain knowledge from search organization.

Table 1 shows that TSUGO, unlike prior benchmarks centered on accuracy, step errors, or CoT redundancy, jointly supports process traces, reasoning topology, verifiable states, controlled candidates, multimodal inputs, branch search, and search-organization analysis.

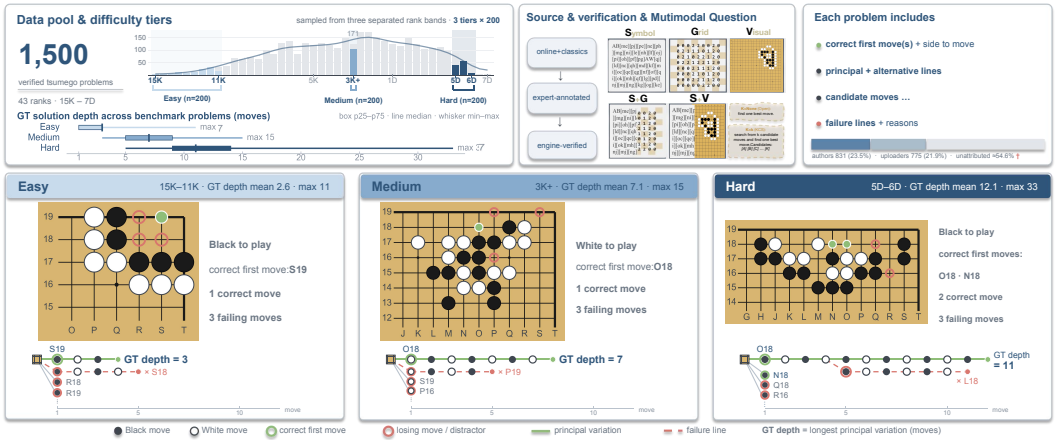


Figure 3: Overview of the TSUGO dataset construction and evaluation settings.

Table 1: Comparison with representative reasoning benchmarks and process-evaluation frameworks. ✓ denotes a primary target, ○ partial support, and × no explicit support. T = trace, P = topology, V = verification, C = controlled candidate space, M = multimodal input, B = branch search, O = search organization.

Benchmark group and examples	**T**	**P**	**V**	**C**	**M**	**B**	**O**
Outcome (GSM8K, MATH, ARC, BIG-Bench)	○	×	○	○	×	×	×
Logic (FOLIO, PrOntoQA, ZebraLogic)	○	○	✓	✓	×	○	×
Process (ROSCOE, ReCEval, ProcessBench, PRMB)	✓	○	✓	○	×	×	×
CoT efficiency (CoTJudger, ReEfBench)	✓	✓	○	○	×	○	○
Behavior (Sys2Bench, CogBehaviors)	✓	○	○	○	×	○	×
Board games (Chess, Othello, Go)	✓	✓	✓	○	×	○	×
TSUGO	✓	✓	✓	✓	✓	✓	✓

3.2 DATA CURATION

Problems are curated from public tsumego materials, classical collections, and traceable source records, including texts such as *Xuanxuan Qijing* and *Go Life-and-Death Dictionary*. To avoid reproducing platform-specific material, TSUGO keeps only normalized board states, candidate points, reference solution trees, and provenance records; each problem is stored in SGF and converted to JSON with a 19×19 matrix, coordinates, side to play, and solution tree.

We retain problems satisfying three criteria: **Locality**, where relevant stones occupy a bounded region; **First-move determinacy**, where correct first moves are unique or finite and not ko-dependent; and **Verifiability**, where ground-truth solutions are expert-checked and cross-validated by a Go engine. Each solution tree stores the correct first move, principal variation, valid alternatives, refutations, and failure annotations for both answer scoring and search-trace evaluation (§4); additional curation and traceability details appear in the appendix.

Each position is rendered in Symbolic, Grid, Symbolic+Grid, Visual, and Symbolic+Visual forms; bounded-setting distractors are locally plausible but tactically refutable moves.

3.3 DATASET OVERVIEW

TSUGO is organized by difficulty, modality, and candidate-space setting. The curated pool contains 1,500 problems in five input forms; the main evaluation uses 600 problems, sampled as 200 problems from each of three evaluated difficulty tiers, while harder remaining subsets are reserved for future models as capabilities improve. Difficulty tiers combine human-rank bands, main-line depth, and

Table 2: Difficulty tiers in TSUGO. Human-rank bands are approximate descriptors; depth is measured on reference solution trees.

Tier	Human rank band	Avg. depth	Max depth
Easy	15K–11K	2.8	7
Medium	3K+	6.6	15
Hard	5D–6D	11.9	37

plausible wrong candidates; Table 2 shows a progression from shallow kyu-level to dan-level amateur problems, with rank bands used only as coarse descriptors.

We evaluate each problem under two candidate-space conditions: $K{=}4$, where the model selects from four first moves including the correct one, and $K{=}\mathrm{None}$, where it must generate, compare, and justify a move from the board. This separates candidate discrimination from independent search organization across symbolic, grid, and visual modalities. We also test rotation, reflection, color inversion, coordinate relabeling, and candidate-order permutation as robustness controls against surface memorization, coordinate shortcuts, and answer-order bias; details appear in the appendix.

4 Search-Trace Analysis Framework

4.1 Trace Source

TSUGO uses each model's observable reasoning output to reconstruct search organization, using full CoT or reasoning content when available. For closed-source models that expose only compressed summaries, we analyze those summaries as observable artifacts rather than complete internal computation. They still reveal candidate proposal, variation reading, position judgment, and branch switching, although summary-length scale metrics are not comparable to full-CoT models. Prompts provide only the board, side to play, and task instruction, without imposing a reasoning format or tree-search strategy.

4.2 Process Search Tree

To compare free-form traces, TSUGO parses each output into a process search tree $T = (V, E, \tau)$, where V is the node set, E the directed edges, and τ a strictly increasing timestamp. The root r is the initial board state; action nodes a are proposed or simulated moves with $\mathrm{side}(a)$ and $\mathrm{pos}(a)$; evaluation nodes j are terminal judgments with $\mathrm{polarity}(j) \in \{\textit{win}, \textit{lose}, \textit{undetermined}\}$.

The root's action children form the first-level candidates $C = \{c_1, \ldots, c_m\}$. For candidate c, $|\mathrm{subtree}(c)|$ measures allocated resources and maximum root-to-leaf distance measures reading depth. Cross-candidate timestamp transitions are search jumps; jumps to previously visited candidates are backtracking. Duplicate actions under the same parent are merged, while revisits under different branches remain separate to preserve search history.

4.3 Extraction Pipeline

We build a domain prior dictionary from open-ended answers with four step types: candidate exploration, variation reading, position evaluation, and backtracking. An LLM-based extractor then identifies action and evaluation nodes, classifies edges from temporal and semantic context, and closes branches with win/lose/undetermined judgments. Structural validation enforces well-formed trees: roots connect only to actions, evaluations are leaves, unclosed branches receive undetermined leaves, repeated sibling actions are merged, and timestamps remain monotonic. Final trees are selected from parallel extractors by rule filtering and cross-validation. We further validate extraction on 300 sampled problems with two human experts assisted by KataGo; Gemini-2.5-Pro reaches 93–98% agreement, with the range reflecting semantic ambiguity in free-form traces, and details appear in the appendix.

Algorithm 1 Process Search Tree Extraction

Require: Free-form reasoning text $\mathcal{T}$, prior dictionary $\mathcal{D}$
Ensure: Process search tree $T = (V, E, \tau)$
Initialize $V \leftarrow \{r\}$, $E \leftarrow \emptyset$, $\tau(r) \leftarrow 0$
Scan $\mathcal{T}$ using $\mathcal{D}$ to identify atomic steps: EXPLORE, READ, EVALUATE, BACKTRACK
for each identified step in temporal order **do**
  if step is EXPLORE or READ **then**
    Create action node a; infer parent by edge classification; assign $\tau(a)$
  else if step is EVALUATE **then**
    Create evaluation node j; assign polarity and $\tau(j)$
  else if step is BACKTRACK **then**
    Record branch switch and update current context
  end if
end for
Apply structural validation and supplement undetermined leaves
return T

4.4 METRICS

TSUGO reports answer accuracy as first-move hit rate: a response is correct when its selected key move, i.e., the tsumego vital point, matches the reference first move. Because accuracy cannot show how the model reaches, misses, or abandons that move, we define process metrics, including SearchE, on extracted trees to measure resource allocation across candidates.

To contrast token-centered process efficiency with search-organization efficiency, we highlight two signals: ***SearchE*** aggregates wrong-branch waste (SWR), first-hit behavior (SFH), and correct-candidate search rank (SCR): $SearchE = 100 \cdot (0.5(1-\mathrm{SWR}) + 0.3\,\mathrm{SFH} + 0.2(1-\mathrm{SCR}))$. ***TokenE*** measures accuracy relative to observable thinking-token cost: $TokenE = 100 \cdot A/(A + \mathrm{ITT}/1000)$, with $A = 100 \cdot Acc$. Full formulas and weight sensitivity appear in the appendix; we assign 0.3 to SFH and 0.2 to SCR because both measure early correct-branch exploration, while SFH more directly reflects early correct-candidate intuition.

Due to space, the main text uses SearchE as the overall search-efficiency summary and reports the remaining metrics as diagnostics: (i) search metrics, including Search Waste Ratio (SWR), Search First Hit (SFH), and Search Correct Rank (SCR); (ii) trajectory metrics, including Trajectory Max Depth (TMD), Trajectory Max Fan-out (TMF), and Trajectory Node Count (TNC); and (iii) scale metrics, including Inference Total Tokens (ITT) and Inference Per Node (IPN). For proprietary summary-only models, scale metrics are summary-derived references rather than internal-compute measures. Non-LLM baselines provide only comparable search-side metrics because they are not token-driven, and neural-guided KataGo does not expose internal decisions in a CoT-like form. Full definitions, LLM parameters, and non-LLM reference settings appear in the appendix.

5 EXPERIMENTS

5.1 SETUP

We evaluate open reasoning, open non-reasoning, and proprietary models on 600 sampled problems, with 200 per difficulty tier, under $K{=}4$ candidate selection and $K{=}\mathrm{None}$ generation. The evaluated LLMs cover Moonshot AI Kimi-K2.5 Kimi Team (2026), Alibaba Qwen3-VL Qwen Team (2025), MiniMax-M2.5 MiniMax (2026), DeepSeek-R1/V3.2 DeepSeek-AI (2025; 2024), Zhipu AI GLM-4.6V Zhipu AI (2026), and Google Gemini models Google DeepMind (2025); Google (2026). Accuracy is averaged over available modalities, while process metrics use Symbolic input; for proprietary models, scale metrics are summary-derived references.

We use the Smargo implementation of MCTS/UCT Kocsis & Szepesvári (2006); Sun (2022) and KataGo Wu (2019) as non-LLM references with 200 playouts or visits per problem; details appear in the appendix. MCTS represents unguided search and KataGo neural-guided search; they locate LLM search organization but are not token-equivalent.

Table 3: Compact main results. Cells report $K{=}4|K{=}\text{None}$; Acc averages available modalities, and trace metrics use Symbolic input. SearchE aggregates SWR/SFH/SCR; TokenE is accuracy per token cost. ITT is in thousand tokens; for search baselines, TNC is reported in thousands and the final scale column reports playout/visit budget. Best within each model-type group and each setting is marked per difficulty: **bold underline** = Easy, underline = Med, **bold** = Hard. † denotes playout/visit budget; gray proprietary values are summary-derived and excluded from ranking. * marks the single best value in each column over the whole table, regardless of model type (shown for Hard).

Model	Diff.	Acc		SearchE		TokenE		Trajectory						Scale			
								TMD		TMF		TNC		IPN ↓		ITT(k) ↓	
		$K{=}4$	$K{=}\text{None}$	$K{=}4$	$K{=}\text{None}$	$K{=}4$	$K{=}\text{None}$	$K{=}4$	$K{=}\text{None}$	$K{=}4$	$K{=}\text{None}$	$K{=}4$	$K{=}\text{None}$	$K{=}4$	$K{=}\text{None}$	$K{=}4$	$K{=}\text{None}$
Reasoning Models																	
Kimi-K2.5	Easy	**52.0**	**27.8**	**39.9**	**26.4**	68.2	47.2	4.6	4.8	2.3	2.4	20.5	24.7	1269	1176	24.3	31.1
	Med	28.4	11.0	32.3	12.0	56.0	27.7	4.7	5.3	2.1	2.5	21.2	26.2	1258	1014	22.3	28.7
	Hard	**34.0**	4.4	32.9	10.4	61.5	12.2	5.0	4.9	2.3	2.2	19.6	26.8	1213	1197	21.3	31.8
Qwen3-VL-235B-Thinking	Easy	40.0	15.8	39.1	19.5	68.6	46.9	5.1	5.1	2.2	2.3	24.4	27.3	854	745	18.3	17.9
	Med	25.0	8.8	32.1	12.9	58.4	32.2	5.4	4.9	2.0	2.1	22.9	23.7	815	858	17.8	18.5
	Hard	32.0	**7.2**	34.2	7.3	62.5	27.6	5.4	5.0	2.2	2.1	23.2	25.0	901	794	19.2	18.9
Qwen3-VL-30B-Thinking	Easy	34.2	10.0	35.4	18.8	65.6	34.1	3.9	3.7	1.5	1.4	15.6	17.5	1216	1264	17.9	19.3
	Med	23.0	4.6	32.0	14.5	56.1	19.3	3.7	3.6	1.4	1.2	14.7	17.3	1285	1304	18.0	19.2
	Hard	30.2	3.6	32.8	**10.5**	62.7	15.8	3.8	3.4	1.5	1.2	15.2	17.7	1259	1256	18.0	19.2
MiniMax-M2.5 (Non-VL)	Easy	37.7	13.3	36.0	16.1	**76.8**	**54.5**	3.5	3.5	1.7	1.6	15.0	18.3	827	687	**11.4**	**11.1**
	Med	22.0	7.3	30.1	10.0	80.9	38.6	3.1	3.5	1.2	1.3	12.6	17.3	540	740	5.2	11.6
	Hard	26.3	6.3	31.5	7.4	**73.5**	**33.5**	3.3	3.3	1.5	1.4	14.5	12.9	734	722	**9.5**	**12.5**
DeepSeek-R1-0528(Non-VL)	Easy	32.3	17.0	38.3	19.1	71.1	53.8	4.6	4.5	2.0	2.1	19.5	23.3	**743**	**676**	13.1	14.6
	Med	26.3	8.3	30.2	7.6	70.3	37.4	4.6	4.3	2.1	2.1	21.2	22.6	547	680	11.1	13.9
	Hard	33.7	5.3	**34.3**	7.6	72.2	27.5	4.6	4.4	2.2	2.0	21.6	21.1	**629**	**691**	13.0	14.0
Non-reasoning Models																	
GLM-4.6V	Easy	31.0	2.4	35.8	5.8	88.3	36.9	2.7	2.5	0.7	0.6	11.5	9.9	447	715	4.1	4.1
	Med	23.0	1.0	33.5	2.3	89.5	14.3	2.7	2.5	0.7	0.4	11.4	8.6	274	901	2.7	6.0
	Hard	28.8	1.8	31.3	1.7	91.7	34.0	2.7	2.5	0.6	0.5	10.9	9.7	289	582	2.6	3.5
DeepSeek-V3.2	Easy	**39.3**	**16.0**	**37.0**	**20.0**	**94.2**	**85.1**	4.7	5.0	1.8	1.8	17.0	16.4	**162**	**220**	**2.4**	**2.8**
	Med	24.0	7.0	31.8	7.4	91.6	72.2	4.6	4.9	1.6	1.9	16.3	16.0	158	225	2.2	2.7
	Hard	**32.0**	**6.3**	**32.2**	**5.5**	**94.1**	**70.8**	4.6	4.9	1.7	1.9	17.2	16.2	**138***	**206***	**2.0***	**2.6***
Proprietary Models																	
Gemini-2.5-Flash	Easy	32.0	23.0	35.7	22.2	95.5	93.1	3.4	4.1	1.2	1.8	13.8	16.0	109	136	1.5	1.7
	Med	29.0	4.0	33.1	7.2	94.2	70.2	4.6	4.3	2.1	1.8	19.7	15.6	102	146	1.8	1.7
	Hard	29.0	5.0	32.1	6.4	94.8	74.6	4.6	4.1	2.1	2.1	19.2	16.1	98	133	1.6	1.7
Gemini-3.1-Pro-Preview	Easy	**80.0**	**65.0**	**47.3**	**46.6**	**99.0**	**98.9**	4.1	3.9	1.5	1.5	13.7	9.6	66	101	0.8	0.7
	Med	40.0	21.0	37.3	16.1	98.5	95.9	4.2	4.3	1.7	1.6	15.3	11.5	52	104	0.6	0.9
	Hard	33.0	**19.0**	**36.5**	**15.4**	**97.9***	**96.4***	4.0	4.3	1.5	1.7	14.3	11.1	59	86	0.7	0.7
Search Baselines												*TNC(k)*				*Budget*	
MCTS/UCT	Easy	33.0	5.0	35.8	13.1	–	–	5.3	4.8	45.9	47.1	7.2	7.8	–	–	200†	200†
	Med	32.0	1.0	35.1	14.1	–	–	5.2	4.8	52.0	54.6	8.7	9.1	–	–	200†	200†
	Hard	22.0	1.0	30.8	13.4	–	–	5.3	4.9	63.1	65.3	11.0	11.4	–	–	200†	200†
KataGo-b18	Easy	**97.0**	**49.0**	**96.9**	**53.3**	–	–	–	–	–	–	–	–	–	–	200†	200†
	Med	75.0	46.0	73.9	47.8	–	–	–	–	–	–	–	–	–	–	200†	200†
	Hard	**58.0*†**	**43.0*†**	**57.0*†**	**41.5***	–	–	–	–	–	–	–	–	–	–	200†	200†

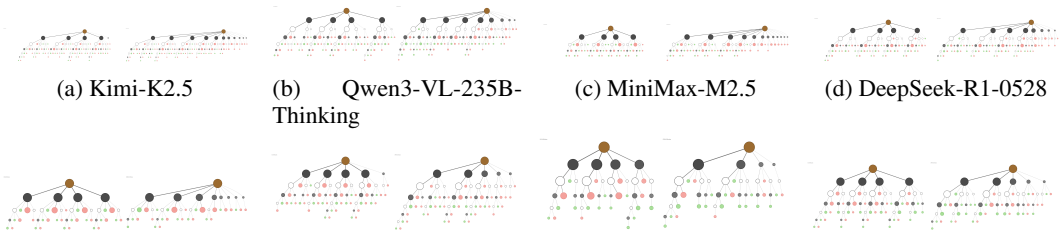
5.2 Main Results

Current LLMs are still far from stable TsuGO solving. Table 3 shows accuracy dropping from constrained easy to open hard problems across systems. Even under $K{=}4$ Easy, the strongest open model, Kimi-K2.5, reaches only 52.0 accuracy; Gemini-3.1-Pro reaches 80.0 but falls to 19.0 on $K{=}\text{None}$ Hard. Model classes show a clear but non-absolute ordering: Gemini models lead on easy settings but lose advantage with difficulty, while open reasoning models generally outperform weaker non-reasoning models in open search. DeepSeek-V3.2, though not a long-reasoning model, approaches or exceeds some reasoning models in accuracy and SearchE; GLM-4.6V retains limited $K{=}4$ discrimination but nearly collapses under $K{=}\text{None}$. Size alone is insufficient: Qwen3-VL-235B usually beats its 30B variant, yet both degrade sharply on open hard problems, pointing to search organization rather than parameter count or token volume.

The drop from $K{=}4$ to $K{=}\text{None}$ exposes the open-search bottleneck. With $K{=}4$, the model mainly discriminates among provided options; with $K{=}\text{None}$, it must generate, rank, and verify candidates itself. The gap shows that many models use partial board-shape knowledge with candidates but struggle to preserve the correct direction in open space. For example, Kimi-K2.5 drops from 52.0

accuracy and 39.9 SearchE on $K{=}4$ Easy to 27.8 and 26.4 on $K{=}\text{None}$ Easy; GLM-4.6V nearly collapses across the open tiers, with SearchE approaching blind-search levels.

Difficulty trends show that candidate constraints can mask open-search failures. Under $K{=}4$, candidate lists compress the search space, so some models do not degrade from Medium to Hard. Once candidates are removed, difficulty produces a more consistent accuracy and SearchE decline, so the open setting better exposes independent search organization.



(a) Kimi-K2.5 (b) Qwen3-VL-235B-Thinking (c) MiniMax-M2.5 (d) DeepSeek-R1-0528

(e) GLM-4.6V (f) DeepSeek-V3.2† (g) Gemini-3.1-Pro (h) Gemini-2.5-Flash

Figure 4: Aggregated process search trees. Each panel shows $K{=}4$ (left) and $K{=}\text{None}$ (right); node size is average subtree weight and color denotes polarity (green = win, red = lose).

Treemaps reveal focused and diffuse search. Under $K{=}4$, candidate constraints make tree shapes similar. Under $K{=}\text{None}$, Kimi-K2.5 and Gemini-3.1-Pro concentrate on fewer candidates and push them toward verification, while GLM-4.6V and MCTS spread resources across shallow wrong branches. Trajectory metrics in Table 3 quantify these structures: TMD measures depth, TMF single-node fan-out, and TNC node count. This matches SearchE: tree width or depth matters less than whether resources point toward the correct direction.

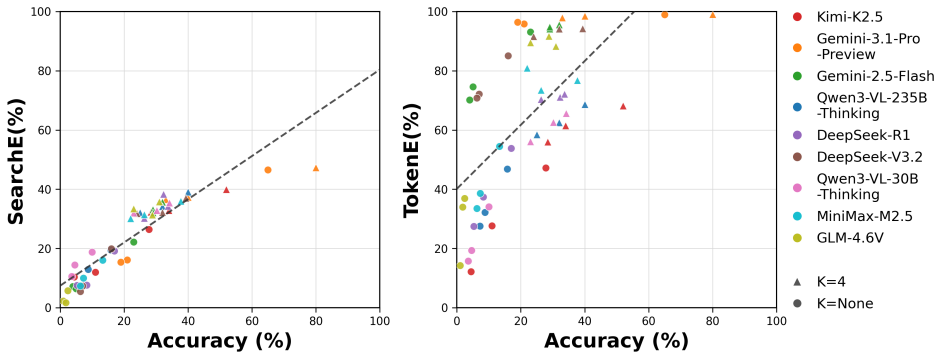


Figure 5: SearchE aligns with accuracy more consistently than TokenE.

5.3 Insight

Failure is a failure of search-resource allocation. The $K{=}4$ results show that many models retain board-shape knowledge when candidates are provided. In the open setting, however, the correct move is often absent or displaced by shallow checks of other branches. Failure is therefore not only missing the right move; often, the right candidate lacks early and sustained resources. Tsumego requires adversarial verification around key candidates, not frequent switching among superficially plausible branches.

SearchE is a stronger process signal than TokenE. Figure 5 shows tighter clustering around accuracy for SearchE than TokenE, indicating stronger alignment with task success. SearchE tracks accuracy more closely because stronger settings allocate more resources to the correct candidate. TokenE is more dispersed, since low token cost does not imply effective search; SearchE better captures search intelligence, while TokenE mainly reflects cost.

LLMs lie between blind and guided search. MCTS reaches mid-tier LLM accuracy under $K{=}4$ but drops to single digits without candidates, showing that unguided expansion cannot maintain direction

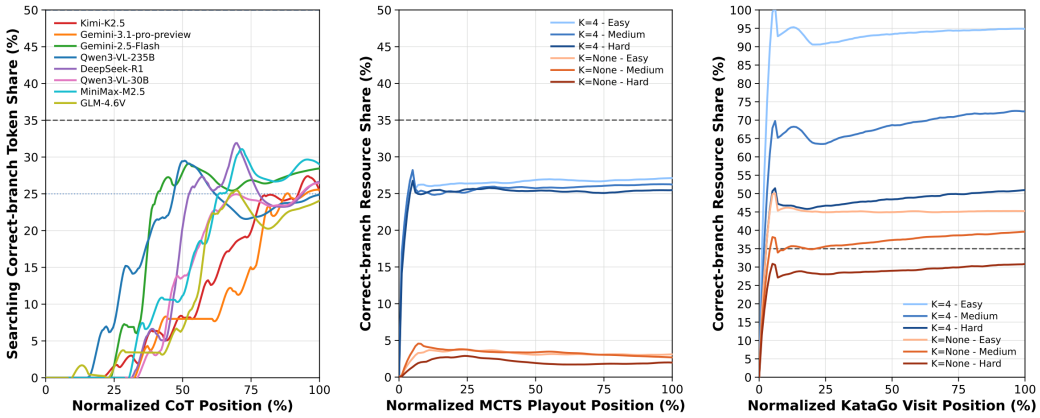


Figure 6: Resource share assigned to the correct branch during search.

in open space. KataGo remains much stronger under the same visit budget, confirming that the task is solvable by well-guided search. Strong LLMs outperform blind search, suggesting useful priors, but their gap to KataGo shows these priors are not yet stable, problem-adaptive search control.

5.4 Further Analysis

Dynamics of resource allocation. Figure 6 compares how LLM CoT, MCTS, and KataGo allocate resources to the correct branch over normalized search progress; the dashed line marks 35%. MCTS and KataGo both form early branch preferences and plateau, but KataGo converges faster and more strongly, especially on Easy and Medium problems. MCTS stays near random four-choice allocation under $K{=}4$ and lower in open search, while LLM CoT is more scattered and volatile, indicating weaker concentration than explicit search.

Domain-specialized model. Logos, a Go LLM trained on KataGo trajectories, often continues whole-game play instead of resolving the local problem, making it unsuitable for trace analysis and motivating TsuGO's focus on goal-directed local search; details appear in the appendix.

Bad-case analysis and distribution. Failed open-setting traces exhibit candidate-generation, coordinate-grounding, perception, state-maintenance, verification, instruction-following, looping, and branch-management errors. These failures indicate joint bottlenecks in knowledge, state tracking, verification, and branch control; distributions and examples appear in the appendix.

6 Conclusion

We presented TsuGO, a process-level benchmark for evaluating Search Efficiency in LLM reasoning. Instead of testing Go-playing strength, it uses life-and-death problems as closed, verifiable, adversarial search spaces; K-Search controls the solution space, and process trees expose resource allocation. Experiments show that current LLMs remain far from stable tsumego solving: stronger models find the correct candidate earlier and sustain productive effort, while failures omit, delay, or abandon the key move, often compounded by state-perception and tactical errors. Search Efficiency is a more effective process signal than token cost alone because it measures how models allocate resources during search. With Token Efficiency as a cost reference, it identifies candidate generation, branch comparison, verification, and backtracking as core search-control bottlenecks. Overall, TsuGO turns free-form traces into diagnostic feedback and remains far from saturated, suggesting future work on Search-Efficiency-sensitive training data and planning frameworks.

AI USE STATEMENT

In this work, we used generative AI tools to aid or polish writing and to assist with manuscript reformatting. We reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The benchmark construction, evaluation settings, metric definitions, and additional implementation details needed to reproduce the reported results are documented in the appendix and supplementary materials.

REFERENCES

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024. doi: 10.1609/aaai.v38i16.29720.

Xingyu Chen et al. Do not think that much for 2+3! overthinking with chain-of-thought. *arXiv preprint*, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. Deepseek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

DeepSeek-AI. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen M McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *ICML*, 2024.

Zhizhang Fu, Yuancheng Gu, Chenkai Hu, Hanmeng Liu, and Yue Zhang. Reefbench: Quantifying the reasoning efficiency of LLMs. *arXiv preprint arXiv:2601.03550*, 2026.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Alane Suhr, and Noah D Goodman. Stream of search (SoS): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STaRs. *arXiv preprint arXiv:2503.01307*, 2025.

Olga Golovneva, Moya Peng Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. ROSCOE: A suite of metrics for scoring step-by-step reasoning. In *Proceedings of the 2023 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 6556–6576, 2023.

Google. Gemini 3.1 pro: A smarter model for your most complex tasks. `https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/`, 2026. Official blog.

Google DeepMind. Gemini: A family of highly capable multimodal models. *arXiv preprint*, 2025.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, et al. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22017–22031, 2024.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *NeurIPS*, 2021.

Kimi Team. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pp. 282–293. Springer, 2006.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. *ICLR*, 2023.

Siyi Li, Jiajun Shi, Shiwen Ni, Ge Zhang, Shuaimin Li, Shijian Wang, Zhoufutu Wen, Yizhi Li, Hamid Alinejad-Rokny, Jiaheng Liu, Min Yang, and Wenhao Huang. Cotjudger: A graph-driven framework for automatic evaluation of chain-of-thought efficiency and redundancy in LRMs. *arXiv preprint arXiv:2603.07078*, 2026.

Zhiyuan Li, Yi Chang, and Yuan Wu. THINK-Bench: Evaluating thinking efficiency and chain-of-thought quality of large reasoning models. *arXiv preprint arXiv:2505.22113*, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *ICLR*, 2024.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, et al. ZebraLogic: On the scaling limits of LLMs for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

Yiwei Liu, Yucheng Li, Xiao Li, and Gong Cheng. LogiNumSynth: Synthesizing joint logical-numerical reasoning problems for language models. *arXiv preprint arXiv:2510.11031*, 2025.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. O1-Pruner: Length-harmonizing fine-tuning for O1-like reasoning pruning. *arXiv preprint arXiv:2501.12570*, 2025.

Yichuan Ma, Linyang Li, Yongkang Chen, Peiji Li, Jiasheng Ye, Qipeng Guo, Dahua Lin, and Kai Chen. Mixing expert knowledge: Bring human thoughts back to the game of Go. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.48550/arXiv.2601.16447.

MiniMax. The MiniMax-M2 series: Mini activations unleashing max real-world intelligence. *arXiv preprint arXiv:2605.26494*, 2026.

OpenAI. Learning to reason with LLMs. *OpenAI Blog*, 2024.

Shubham Parashar, Blake Olson, Sambhav Khurana, Eric Li, Hongyi Ling, James Caverlee, and Shuiwang Ji. Inference-time computations for LLM reasoning and planning: A benchmark and insights. *arXiv preprint arXiv:2502.12521*, 2025.

Jongwon Park et al. Ensembling large language models with process reward-guided tree search for better complex reasoning. *arXiv preprint arXiv:2412.15797*, 2025.

Archiki Prasad, Swarnadeep Saha, Xiang Zhou, and Mohit Bansal. ReCEval: Evaluating reasoning chains via correctness and informativeness. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10066–10087, 2023. doi: 10.18653/v1/2023.emnlp-main.622.

Qwen Team. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *International Conference on Learning Representations*, 2023.

Chenhui Shen et al. FaithCoT-Bench: Benchmarking instance-level faithfulness of chain-of-thought reasoning in large language models. *arXiv preprint arXiv:2510.04040*, 2025.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. PRMBench: A fine-grained and challenging benchmark for process-level reward models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25299–25346, 2025. doi: 10.18653/v1/2025.acl-long.1230.

Aarohi Srivastava et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *TMLR*, 2023.

Yuan Sui et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.

Yize Sun. Smargo: An efficient and highly accurate solver for tsumego. `https://github.com/Sun-Yize/smargo`, 2022. GitHub repository.

Shubham Toshniwal, Sam Wiseman, Karen Livescu, and Kevin Gimpel. Chess as a testbed for language model state tracking. *arXiv preprint arXiv:2302.13071*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.

David J. Wu. Accelerating self-play learning in Go. *arXiv preprint arXiv:1902.10565*, 2019.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025.

Zhipu AI. GLM-4.6V: Open source multimodal models with native tool use. `https://www.zhipuai.cn/en/glm46v`, 2026. Official blog.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2024.

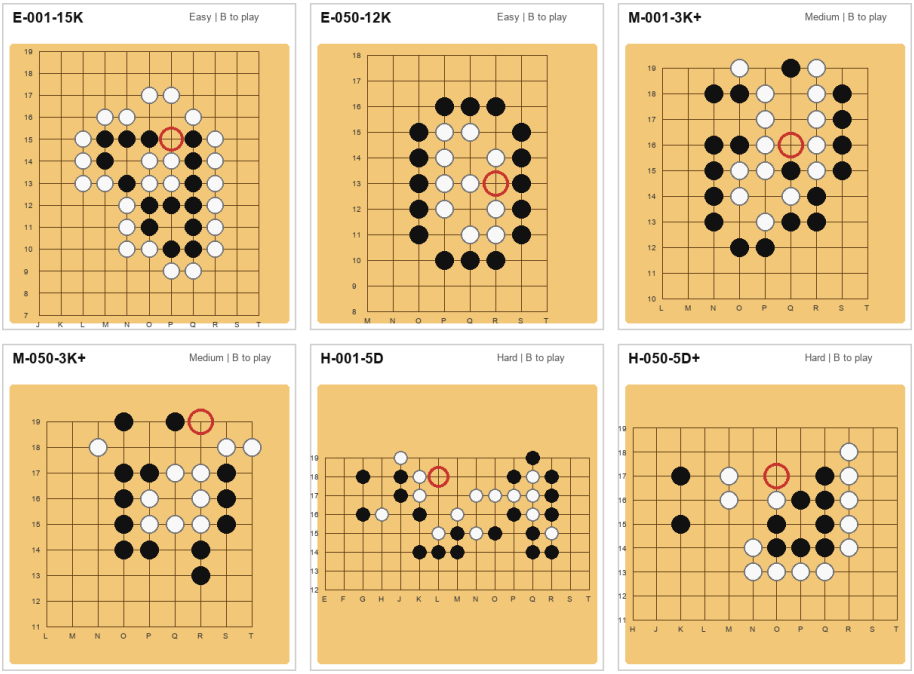


Figure 7: Example normalized tsumego boards sampled from Easy, Medium, and Hard splits. Labels use benchmark display identifiers with rank tags, and the red circle marks the reference first move.

A Data Curation and Traceability

TsuGO is built from traceable tsumego records rather than from free-form puzzle text alone. Each item is normalized into a board state, side to play, candidate first moves for bounded evaluation, and reference solution lines. The released benchmark stores the normalized state and solution metadata, while avoiding redistribution of webpage layout, platform-specific presentation, or explanatory prose from the original sources.

Construction process. We use SGF as the common intermediate representation. Source records are first converted into SGF fields for board size, black stones, white stones, side to play, and verified solution moves; the SGF is then converted into the structured problem representation used by all model evaluations. This keeps the symbolic coordinate state, numeric grid, visual rendering, and answer key tied to the same canonical position.

Each JSON item contains three top-level fields: `Prompt`, `Question`, and `Solution`. `Question` stores the symbolic coordinate state, the 19-by-19 grid state, optional visual rendering metadata, and the side to play. `Solution` stores the reference first move, the bounded candidate set, all equivalent first moves when present, standard winning lines, variations, and losing candidate lines. This structure lets every evaluated answer be checked by coordinate normalization rather than by string matching alone.

SGF-to-symbolic conversion. SGF points are converted to the benchmark coordinate system by mapping the first SGF character to board columns `A,B,...,H,J,...,T` and the second SGF character to rows from top to bottom. Thus an SGF point `[oe]` maps to `P15`: `o` is the fifteenth zero-indexed SGF column, which becomes `P` after skipping `I`, and `e` maps to row $19 - 4 = 15$. The same conversion is applied to setup stones, reference solution lines, candidate options, and model predictions.

K=4 candidate construction. The bounded setting is designed to evaluate first-move discrimination rather than open move generation. For each problem, the correct first move and all verified equivalent first moves are first identified from the solution record. Distractor candidates are then drawn from a

Table 4: Example normalized evaluation interface derived from one SGF problem.

Field	Example value
Benchmark record	`E-001-15K`; Easy split; rank tag `15K`
SGF state	`AB[me][pj]...AW[qj][ld]...PL[B]`
Side to play	Black
Symbolic state (Sym)	Black: `M14,M15,N13,...,Q15`; White: `L13,L14,...,R15`
Grid state (Grd)	19×19 board with 0/1/2 for empty/black/white.
Visual state (Vis)	Rendered board image generated from the same normalized state.
Accepted answers	`correct_answer=P15`; `all_correct_answers={P15,N14}`
Candidate options	A: `P15`; B: `Q17`; C: `Q18`; D: `O18`
Reference lines	Standard: `[P15]`, `[N14]`; variations: `[P15,N14,N13]`, `[N14,P15,P16]`
Input variants	Sym, Grd, Vis, S+G, and S+V share the same side-to-play, options, and answer key.

Algorithm 2 Bounded K=4 Candidate Construction

Require: Normalized board B, side to play s, canonical answer c^*, verified equivalent answers E, search-assisted proposal records $S_{\mathrm{KG}}, S_{\mathrm{MCTS}}$
Ensure: Four-option set O
1: $A \leftarrow \{c^*\} \cup E$ {accepted first-move answer set}
2: $P \leftarrow \{m : m$ is a non-solution local proposal in $S_{\mathrm{KG}} \cup S_{\mathrm{MCTS}}\}$
3: $P \leftarrow P \setminus A$
4: remove occupied, illegal, non-local, and globally context-dependent moves from P
5: rank each $m \in P$ by Euclidean distance to c^*
6: $D \leftarrow$ the nearest three remaining moves
7: $O \leftarrow \mathrm{Shuffle}(\{c^*\} \cup D)$
8: **return** O

search-assisted curation pool rather than from random legal moves. This pool combines non-solution local alternatives recorded during candidate construction, including proposals from non-LLM search references such as KataGo and MCTS/UCT, candidate metadata from the normalized problem record, and geometric locality around the canonical answer. Candidates that duplicate a verified equivalent answer, land on occupied points, or require global context are removed. When more than three distractors remain, we rank them by Euclidean board distance $d(m, c^*) = \sqrt{(x_m - x_{c^*})^2 + (y_m - y_{c^*})^2}$ to the canonical correct point and choose close alternatives first, because near misses are more diagnostic of local tactical discrimination than arbitrary distant legal moves.

The construction has six steps. (i) Normalize the SGF board and solution record into the benchmark coordinate system. (ii) Define the accepted answer set from the canonical solution and all verified equivalent first moves. (iii) Build a proposal pool from non-solution local alternatives in the construction records, including KataGo and MCTS/UCT proposals. MCTS/UCT contributes broad candidate coverage, while KataGo contributes more selective candidate and equivalent-answer signals. (iv) Remove candidates that are occupied, illegal for the side to play, outside the local problem region, or in the accepted answer set. (v) Use Euclidean distance to prioritize the remaining wrong candidates by locality to the canonical correct point. (vi) Select the nearest plausible distractors and combine them with the correct answer to form the K=4 set, then randomly shuffle the option labels before evaluation. This shuffle prevents the verified answer from occupying a fixed A/B/C/D position while preserving the same coordinate-level answer key. The resulting sampled problems are then checked by human review and independent search traces. Search references are therefore used to propose plausible wrong moves, and distance is used to choose local near misses; neither defines correctness, which always comes from the verified solution record.

A post-hoc audit of independent shallow-search traces supports this curation rule. Table 5 summarizes the 300-problem bounded-candidate trace-audit subset, sampled as 100 problems from each evaluated difficulty tier. This subset is half of the 600-problem main evaluation and is the portion for which complete open-root and restricted-root MCTS/UCT and KataGo root-distribution traces are available. The audit is deliberately separate from the construction step above: it does not define the options, but applies the same coverage test to independent root-distribution traces. Coverage is therefore not expected to be 100%. The audit has two purposes. First, it checks that selected non-correct options are usually search-visible rather than arbitrary legal points. Second, it checks that additional equivalent correct moves are recognized as accepted answers rather than counted as distractors. For

Table 5: Audit of K=4 candidate construction using independent shallow-search traces for the 300-problem trace-audit subset. The 881 non-correct option entries equal 300×4 final option entries minus 319 verified correct or equivalent-correct entries. A candidate is counted as covered when it receives a nonzero root visit within budget 0–30. Open-root runs search all legal root moves; restricted-root runs search only the final K=4 option set.

Audit quantity	Count	Rate
Problems in curation subset	300	–
Final option entries	1200	–
Accepted-answer entries	319	–
Non-correct option entries	881	–
Covered by open-root MCTS/UCT	629/881	71.4%
Covered by open-root KataGo	210/881	23.8%
Covered by either open-root search	681/881	77.3%
Covered by restricted-root MCTS/UCT	851/881	96.6%
Covered by restricted-root KataGo	485/881	55.1%
Extra equivalent correct moves	38	–
Equiv. correct visited by open-root MCTS/UCT	26/38	68.4%
Equiv. correct visited by open-root KataGo	23/38	60.5%

Table 6: Example K=4 evidence for benchmark problem `E-001-15K`. Rows A–D are the final answer options, while `N14` is shown as a rejected proposal. Distance is Euclidean distance to the canonical answer `P15`; first-seen columns report the earliest independent audit budget, at most 30, where the move receives a nonzero root visit. `N14` is search-visible but removed because it is a verified equivalent correct answer.

Option	Coord.	Label	Dist.	MCTS-O	KG-O	MCTS-4	KG-4
A	P15	correct	0.00	21	2	1	2
B	Q17	distractor	2.24	11	–	2	30
C	Q18	distractor	3.16	7	–	3	–
D	O18	distractor	3.16	5	–	4	–
–	N14	equiv. correct	2.24	27	4	–	–

each selected option, we scan budgets up to 30 and count it as covered if it receives a nonzero root visit. The 881 non-correct option entries are obtained by counting all final K=4 options in the 300 sampled problems and excluding verified correct or equivalent-correct entries: $300 \times 4 = 1200$ total option entries, 319 accepted-answer entries, and $1200 - 319 = 881$ non-correct entries.

Open-root coverage is a plausibility check, not a correctness signal. In an open-root run, KataGo or MCTS/UCT searches from the original board without being told the four benchmark options, so a distractor covered by open-root search is a move that the search process naturally considered among legal root moves. Restricted-root coverage answers a different question: after the benchmark fixes the final K=4 interface, does the search process allocate visits to those listed options? We therefore use open-root coverage to test whether distractors are naturally plausible, and restricted-root coverage to test whether the final option set remains meaningful under the same answer interface used by LLMs.

The table shows that 77.3% of the final non-correct options are independently encountered by at least one open-root search reference within the shallow budget. MCTS/UCT contributes broader early root coverage, while KataGo is more selective; this difference is useful because it prevents the option set from reflecting a single search policy. These visit counts are used as priority and plausibility signals, not as correctness thresholds. The restricted-root columns provide a second check: once the final four candidates are fixed, many non-correct options receive visits under the same answer interface used for model evaluation. The equivalent-correct rows also show why answer-set filtering must precede distractor selection: many verified equivalent answers are attractive to search references, so using search proposals alone would incorrectly label some valid first moves as distractors.

Here "O" denotes open-root search over legal root moves, and "4" denotes restricted-root search over the final K=4 option set. The example illustrates the full curation-and-audit chain. The final distractors are local near misses around the canonical answer and appear in independent MCTS audit traces, so they are not random legal points. At the same time, `N14` is also search-visible and geometrically close, but it is filtered out before final option selection because the verified solution record marks it as an equivalent correct first move.

Table 7: Composition audit for the 300-problem bounded-candidate curation subset used for structural checks. This audit subset is separate from the 600-problem main evaluation set.

Split	N	Options	B/W	Std. len.	Var. len.
Easy	100	4	99/1	2.77	3.84
Medium	100	4	68/32	6.56	6.86
Hard	100	4	85/15	11.92	10.14

Evaluation prompt example. For the same problem, the symbolic prompt presents the state as normalized coordinates rather than as a natural-language puzzle description: "Black stones: `M14, M15, N13, ..., Q15`; White stones: `L13, L14, ..., R15`; side to play: Black; choose from A: `P15`, B: `Q17`, C: `Q18`, D: `O18`." The final answer is scored by mapping the option letter back to a normalized coordinate and checking whether it belongs to the verified correct-answer set.

Filtering criteria. We retain a problem only when the local life-and-death objective is well defined, the first move is unique or belongs to a finite equivalence class, the candidate set is legal under coordinate normalization, and the reference line is not dependent on unresolved global ko or whole-board context. Ambiguous problems, records with inconsistent side-to-play metadata, positions whose candidate labels cannot be mapped back to legal board coordinates, and K=4 sets containing another unverified equivalent answer are removed before evaluation.

Normalization checks. Every retained item is checked at three levels. First, all coordinates must use the same 19-by-19 convention with the column `I` omitted. Second, the symbolic coordinate state and grid state must agree after color, row, and column normalization. Third, each candidate first move and each reference first move must map to an empty legal coordinate for the stated side to play. These checks make answer evaluation independent of the surface text used to present a problem.

Traceability. The benchmark keeps source-level identifiers, rank tags, candidate labels, and reference-line metadata for auditability, but evaluation uses the normalized board state. This separation is important for reproducibility. A model can be evaluated from the released structured problem alone, while the construction record still allows us to inspect why a puzzle was included, how its candidates were produced, and whether an apparent model error is actually a coordinate or metadata issue.

Why the bounded task is evaluable. The K=4 setting is intentionally conservative. Five human reviewers, assisted by KataGo line checking, review the first-move answer set and the distractor set for 300 sampled problems, corresponding to half of the 600-problem main evaluation. The reviewers self-rate as two stronger amateur Go players, two intermediate amateur players, and one beginner amateur player; each reviewer spends approximately 12 hours on the review, for about 60 reviewer-hours before joint adjudication. The review asks whether the listed correct moves are all acceptable first-move tesuji, whether any distractor is actually an equivalent solution, and whether every option is a legal local move under the normalized state. Accuracy is then defined only at the first move: a model is correct if its selected coordinate is in the verified correct-answer set. This does not claim to certify every possible continuation of an open game; it certifies the local four-choice eye point used by the benchmark.

This bounded evaluation is already non-saturated. Even on the easiest K=4 tier, most evaluated LLMs remain well below perfect accuracy in the main table. Therefore, although K=4 is simpler than open search, it still provides a meaningful first layer of measurement: it tests whether the model can identify the vital point among locally plausible alternatives before the benchmark asks it to generate candidates from the full board.

The split statistics in Table 7 show the intended difficulty progression: harder tiers have longer standard solution lines and longer failure/variation lines. This pattern is important because it verifies that the split is not merely relabeled by rank; the reference solution structure itself becomes longer and more branching as difficulty increases.

Table 8: Average equivalence-transform robustness over evaluated models. For each model, 10 benchmark problems are sampled with difficulty stratification. Acc. is accuracy under the corresponding surface form; Consist. is inverse-mapped agreement with the same model's original prediction.

Transform	Problems/model	Acc.	Consist.
Original	10	26.7	100.0
Rotate 180	10	40.0	60.0
Mirror horizontal	10	31.1	64.4
Color inversion	10	28.9	77.8

B Equivalence-Transform Robustness

The benchmark is designed so that tactical content is invariant to board symmetries and color relabeling. We therefore define a transform suite for robustness evaluation: 90/180/270 degree rotation, horizontal reflection, vertical reflection, color inversion with side-to-play flip, and coordinate relabeling. This test is not a proof that no benchmark position appears in pretraining data. Instead, it is a surface-memorization stress test: the local tsumego answer is preserved while coordinates, colors, or geometry are changed. A model that relies on memorized board layouts or coordinate shortcuts should become less consistent under these equivalent views.

Transform construction audit. Before using the transforms for model evaluation, we audit whether each transformed problem preserves a valid coordinate system and maps the reference answer into the transformed candidate set. The construction audit covers 300 benchmark items and passes for all tested transforms: rotation, horizontal/vertical reflection, and color inversion. This check ensures that a model failure is not caused by an invalid transformed instance.

Robustness protocol. We evaluate transformation invariance with a controlled answer-only protocol at temperature 0. The prompt asks for a single option letter, so this probe is used to study presentation sensitivity rather than to replace the main benchmark accuracy estimates. For each evaluated model, we sample 10 problems from the benchmark with difficulty stratification across Easy, Medium, and Hard items. Each sampled item is evaluated under the original presentation and three answer-preserving transformations. Table 8 reports the average result across models.

Accuracy is computed against the answer after transformation. Consistency is computed by mapping each transformed prediction back to the original coordinate system and comparing it with the same model's prediction on the original presentation of the same problem. The latter is the primary robustness quantity: a model can be inaccurate in this terse setting yet still be invariant, or accurate on some transformed prompts while changing its underlying prediction.

Interpretation. The aggregate results show that answer-preserving surface changes can substantially alter model behavior. Geometric transforms reduce consistency, with 60.0% under 180-degree rotation and 64.4% under horizontal reflection, suggesting that spatial normalization remains imperfect even when the problem is presented symbolically. Color inversion is less disruptive but still below perfect consistency at 77.8%, indicating sensitivity to color and side-to-play relabeling. These results do not prove absence of training-set exposure; they support the more limited claim used in the main paper, namely that surface-equivalent perturbations are a useful robustness control for memorization-like shortcuts and coordinate sensitivity.

C Process-Tree Extraction and Validation

The main paper evaluates visible reasoning by converting free-form responses into process search trees. We use a compact node schema with three node types: `root`, `move`, and `judgment`. Candidate proposal, adversarial reply, variation continuation, and branch switching are metric-level roles derived from node depth, parent links, move side, and temporal order. The root is the initial board state; first-level move children are candidate first moves whenever they can be recovered.

Extractor protocol. We use Gemini-2.5-Pro as the primary visible-trace extractor because it gives stable structured outputs on long free-form reasoning while preserving enough local Go terminology to avoid excessive rule-based preprocessing. The prompt below is a condensed English version of the extraction instructions used in evaluation. The extractor is not given the reference answer as a target to imitate. It receives the model's visible reasoning text and a schema. The prompt asks the extractor to preserve the temporal order of the original response, identify explicitly analyzed moves, attach follow-up moves to the active branch, and mark terminal judgments as `win`, `lose`, or `other`. A structural validation step then enforces exactly one root, valid parent pointers, move labels with side and coordinate when recoverable, no orphaned nodes, and monotone node order. Repeated sibling moves are merged when they clearly refer to the same move under the same parent; incomplete branches are closed with an `other` judgment instead of being deleted.

Extractor prompt. The extraction prompt first defines the rules, then includes a compact structured example showing the required node format, and finally appends the model reasoning trace to be parsed. The version below condenses the instruction portion while preserving the metric-relevant constraints:

> **Task.** Convert the visible reasoning trace into a structured timeline search tree for a Go life-and-death problem. Read the model's thinking/reasoning text and output a JSON tree describing which move branches were explicitly considered, how each branch was expanded, and what conclusion was reached.
>
> **Node types.** Use only three node types. `root` appears once with `id=0` and `parent=null`. `move` represents an explicitly analyzed move hypothesis and must include `side` in {`B`,`W`} and `move_pos`. `judgment` terminates a branch and must be a leaf with `polarity` in {`win`,`lose`,`other`}.
>
> **Extraction scope.** Extract necessary structure only: candidate moves considered by the model, explicitly expanded follow-up moves, and final branch judgments. Do not convert background board description, repeated restatement, global summaries, or comparison prose into nodes. If a move is merely mentioned but not analyzed as a branch, do not create a node for it.
>
> **Timeline and tree rules.** Node ids must strictly increase in the order the response introduces them. The root may have only `move` children. Every non-leaf node except the root must be a `move`. Every leaf must be a `judgment`. If the same move appears under different parents or different search branches, create a new node; if the same move is repeated under the same parent, reuse the existing node. Do not invent intermediate moves to complete a line.
>
> **Judgment rules.** Add `judgment` only when a branch terminates. Do not turn intermediate statements such as "has one liberty", "connects", "preliminary comparison", or "think further" into judgments if the response continues the line. Use `win` for a branch judged feasible, alive, killing, optimal, or successful; `lose` for a branch judged failed, captured, refuted, or inferior; and `other` for pruned, unclear, interrupted, or unresolved branches.
>
> **Output.** Return only a JSON object with a `nodes` array. Each node includes `id`, `label`, `type`, `parent`, and, for moves, `side` and `move_pos`; for judgments, include `polarity`. Do not output explanations or Markdown.

The structured example has the following form:

```
{
  "nodes": [
    {
      "id": 0,
      "label": "Initial board state",
      "type": "root",
      "parent": null
    },
    {
      "id": 1,
      "label": "B:T14",
      "type": "move",
      "parent": 0,
      "side": "B",
      "move_pos": "T14"
    },
    {
      "id": 2,
      "label": "W:S13",
```

Table 9: Node types used by the visible process-tree extractor.

Node type	Extraction rule
Root	Initial board state and side to play.
Move	Explicitly analyzed move or move hypothesis; first-level moves are candidate first moves.
Judgment	Branch-ending win, loss, refutation, success, uncertainty, or pruning claim.
Derived roles	Candidate, reply, continuation, and branch switch are inferred from depth, side, parent, and time order.

Table 10: Human cross-validation of Gemini-2.5-Pro process-tree extraction on 300 sampled problem responses. Rates report whether the automatic extraction passes the human check for each metric-critical property before joint adjudication; the 93–98% range mainly reflects semantic boundary ambiguity in free-form traces.

Validation property	Pass rate	Main risk when failed
First-level candidate set	98%	brief mention vs. analyzed branch
Candidate order / SFH rank	97%	late restatement of an earlier branch
Parent-child branch edges	95%	pronoun or "this line" ambiguity
Terminal polarity	93%	hedged or summary-level life/death judgment
Metric-consistent tree	96%	merged or split repeated branch


```
      "type": "move",
      "parent": 1,
      "side": "W",
      "move_pos": "S13"
    },
    {
      "id": 3,
      "label": "Black lives",
      "type": "judgment",
      "parent": 2,
      "polarity": "win"
    }
  ]
}
```


Validation protocol. Human validation samples across model, difficulty, modality, candidate-space condition, and correctness strata. The validation set contains 300 sampled problem responses, matching the sample size referenced in the main paper. Two human experts independently inspect the Gemini-2.5-Pro extraction against the original visible response. The experts check five metric-critical properties: first-level candidate recovery, candidate appearance order, parent-child branch assignment, terminal polarity, and whether the resulting tree preserves the values needed for SWR, SFH, SCR, TMD, TMF, TNC, IPN, and ITT. The reported 93–98% range comes from this independent cross-validation: most disagreements are semantic boundary cases rather than structurally invalid trees, such as a move being briefly mentioned but not developed into a branch, a branch being revisited without naming the first move again, a pronoun such as "this line" having multiple possible antecedents, or a summary-only trace compressing several branches into one sentence. The experts are assisted by the normalized board state and KataGo line checking when a move label or terminal Go judgment is ambiguous, but the adjudication target remains the visible text, not the engine's preferred solution. Disagreements are resolved by joint adjudication. Ambiguous proprietary summaries are not expanded into hidden internal trees; they are marked as summary-derived traces.

Validation criteria. We judge extraction quality at the level required by the metrics rather than by exact natural-language paraphrase. A tree is considered metric-consistent when it preserves (i) whether the correct candidate appears, (ii) the order in which first-level candidates appear, (iii) the allocation of explored nodes to correct and wrong first-move branches, (iv) the terminal polarity of each explicitly evaluated branch, and (v) the observable token span used to compute scale metrics.

This criterion is stricter than answer extraction but more stable than requiring annotators to agree on every intermediate phrase in a long free-form trace.

Observed extraction risks. The main failure modes of extraction are implicit candidate references, coordinate aliases, summary-only proprietary reasoning, and long responses that revisit a branch without explicitly naming the first move. We handle these cases conservatively: ambiguous branches remain marked as unclear, summary-only traces are not expanded into hidden internal search, and answer metrics are computed separately from process metrics.

Why tree validation is metric-specific. The goal is not to reconstruct private cognition, but to measure the organization of visible evidence. Therefore the extractor is allowed to ignore rhetorical filler, self-corrections that do not introduce a new board state, and repeated restatements of an already represented branch. It is not allowed to add a candidate that is only implied by the reference answer, to move an opponent reply under a more convenient parent, or to convert an uncertain branch into a terminal win/loss. These constraints keep SWR, SFH, CCER, and related metrics tied to the text actually shown by the model.

D Metric Definitions and Weight Sensitivity

Answer Accuracy. For a benchmark set $\mathcal{D} = \{x_i\}_{i=1}^n$, let c_i^* be the reference first move and $\hat{c}_i$ be the first move selected by the model after coordinate normalization. We compute answer accuracy as

$$\mathrm{Acc} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}[\hat{c}_i = c_i^*]. \tag{1}$$

SearchE. The main table reports a 0–100 structural-efficiency score:

$$\mathrm{SearchE} = 100 \cdot \big(0.5(1-\mathrm{SWR}) + 0.3\,\mathrm{SFH} + 0.2(1-\mathrm{SCR})\big). \tag{2}$$

It aggregates wrong-branch waste, early first exploration of the correct candidate, and the search rank of that candidate.

TokenE. Let $A = 100 \cdot \mathrm{Acc}$ be accuracy in percentage points and let $\mathrm{ITT}_{\mathrm{raw}}$ be the raw number of observable thinking tokens. We report token efficiency as

$$\mathrm{TokenE} = 100 \cdot \frac{A}{A + \mathrm{ITT}_{\mathrm{raw}}/1000}. \tag{3}$$

TokenE is a cost reference for LLM traces and is undefined for search baselines, which spend playouts or visits rather than language tokens.

Search-tree notation. For one response, let $T = (V, E, r, \tau)$ be the extracted process tree, where r is the root and τ orders nodes by their appearance in the reasoning trace. Let $C = \{c_1, \ldots, c_m\}$ be first-level candidate moves, and let c^* denote the correct first move. For any node v, $\mathrm{owner}(v)$ denotes the first-level candidate whose subtree contains v. Let $u_1, \ldots, u_L$ be the non-root nodes sorted by τ.

Search metrics.

$$\begin{aligned} &\mathrm{CCER} = \min\{j : \mathrm{owner}(u_j) = c^*\}, \\ &\mathrm{CCER} = \infty \quad \text{if } c^* \text{ is never expanded.} \end{aligned} \tag{4}$$

$$\mathrm{SWR} = \frac{\sum_{c \in C,\, c \neq c^*} |\mathrm{subtree}(c)|}{\sum_{c \in C} |\mathrm{subtree}(c)|}. \tag{5}$$

$$\mathrm{SFH} = \mathbb{1}[\mathrm{CCER} = 1]. \tag{6}$$

$$\mathrm{SCR} = \begin{cases} (\mathrm{CCER} - 1)/\mathrm{SBC}, & \mathrm{CCER} < \infty, \\ 1, & \mathrm{CCER} = \infty. \end{cases} \tag{7}$$

Table 11: Complete metric inventory used in the main and appendix result tables. Search-organization metrics explain where visible reasoning effort is allocated; trajectory metrics describe extracted tree topology; scale metrics describe observable text budget.

Metric	Category	Direction	Definition / interpretation
Acc.	Answer	↑	First-move hit rate after coordinate normalization.
SearchE	Composite	↑	Weighted score from SWR, SFH, and SCR.
TokenE	Composite	↑	Accuracy normalized by observable token cost.
SWR	Search organization	↓	Fraction of explored tree nodes under wrong first-move branches.
SFH	Search organization	↑	Indicator that the correct first-move branch is explored first.
CCER	Search organization	↓	Rank at which the correct first-move branch first appears; ∞ if absent.
SCR	Search organization	↓	Normalized correct-candidate rank derived from CCER and SBC.
SSC	Search organization	context	Number of switches between first-level candidate branches.
SBC	Search organization	context	Number of first-level candidate branches considered.
TMD	Trajectory	context	Maximum root-to-leaf depth of the extracted process tree.
TMF	Trajectory	context	Maximum number of children of any single tree node.
TNC	Trajectory	context	Number of nodes in the extracted process tree.
ITT	Scale	↓	Observable reasoning-token count; reported in thousands in compact tables.
IPN	Scale	↓	Observable reasoning tokens per extracted tree node.

$$\begin{aligned} \mathrm{SSC} &= \sum_{\ell=2}^{L} \mathbb{1}[\mathrm{owner}(u_\ell) \neq \mathrm{owner}(u_{\ell-1})], \\ \mathrm{SBC} &= |C|. \end{aligned} \tag{8}$$

$$\begin{aligned} \mathrm{ITT} &= \text{number of observable reasoning tokens}, \\ \mathrm{IPN} &= \mathrm{ITT}/|V|. \end{aligned} \tag{9}$$

$$\begin{aligned} \mathrm{TMD} &= \max_{v \in V} \mathrm{dist}(r, v), \\ \mathrm{TNC} &= |V|, \\ \mathrm{TMF} &= \max_{v \in V} |\mathrm{children}(v)|. \end{aligned} \tag{10}$$

The formulas above use fractional SWR, SFH, and SCR values in $[0, 1]$. All result tables report these three metrics on a 0–100 scale for consistency with the main table captions and SearchE.

Complete result tables. Table 12 provides the full aggregate diagnostic layout: it reports accuracy, search-organization metrics SWR, SFH, SCR, SSC, and SBC, trajectory metrics TMD, TMF, and TNC, and scale metrics IPN and ITT(k) for LLMs, proprietary systems, and search baselines. It omits SearchE and TokenE because those composite summaries already appear in the main table. Table 13 gives the complementary per-modality LLM accuracy breakdown, while trace-level diagnostics remain in Table 12.

D.1 Metric Discriminative Power

Table 14 reports two complementary measures of metric discriminative power. Cohen's d is computed on correct vs. incorrect samples, while Spearman ρ is computed across model-level metric means vs. accuracy.

Weight sensitivity. We recomputed SearchE under five interpretable weight schemes over the non-baseline LLM rows used for the main correlation analysis. Table 15 reports the correlation between each SearchE variant and accuracy; Figure 8 shows the corresponding scatter plots. The main $(0.5, 0.3, 0.2)$ setting is not uniquely tuned to the data: nearby balanced, waste-heavy, first-hit-heavy,

Table 12: Complete diagnostic results with the original non-composite metric columns. Each metric cell uses the format $K{=}4 \mid K{=}\mathrm{None}$. Accuracy averages available input modalities; all trace metrics are computed under Symbolic input. SWR, SFH, and SCR are reported on a 0–100 scale without percent signs. Scale follows the main table: for LLMs, ITT is reported in thousand tokens; for search baselines, TNC is reported in thousands and the final column reports playout/visit budget rather than tokens. Gray proprietary values are summary-derived and are not directly comparable to full visible traces.

Model	Diff.	Acc	Search Organization					Trajectory			Scale	
			$SWR\downarrow$	$SFH\uparrow$	$SCR\downarrow$	SSC	SBC	TMD	TMF	TNC	$IPN\downarrow$	$ITT(k)\downarrow$
Reasoning Models												
Kimi-K2.5	Easy	52.0 | 27.8	66.8 | 82.3	31.6 | 18.4	30.7 | 39.8	4.2 | 6.5	4.1 | 6.4	4.6 | 4.8	2.3 | 2.4	20.5 | 24.7	1269 | 1176	24.3 | 31.1
	Med	28.4 | 11.0	75.2 | 92.6	23.2 | 8.2	35.5 | 70.9	4.3 | 6.8	4.1 | 6.8	4.7 | 5.3	2.1 | 2.5	21.2 | 26.2	1258 | 1014	22.3 | 28.7
	Hard	34.0 | 4.4	71.8 | 92.4	20.2 | 5.4	36.3 | 74.9	4.5 | 7.8	4.0 | 7.6	5.0 | 4.9	2.3 | 2.2	19.6 | 26.8	1213 | 1197	21.3 | 31.8
Qwen3-VL-235B-Thinking	Easy	40.0 | 15.8	69.9 | 84.2	33.7 | 7.1	30.5 | 52.5	3.6 | 4.5	4.1 | 5.3	5.1 | 5.1	2.2 | 2.3	24.4 | 27.3	854 | 745	18.3 | 17.9
	Med	25.0 | 8.8	75.1 | 92.4	23.5 | 12.6	37.0 | 73.4	3.3 | 4.9	4.0 | 5.6	5.4 | 4.9	2.0 | 2.1	22.9 | 23.7	815 | 858	17.8 | 18.5
	Hard	32.0 | 7.2	73.8 | 94.2	26.5 | 3.2	34.4 | 82.6	3.4 | 5.1	4.0 | 5.9	5.4 | 5.0	2.2 | 2.1	23.2 | 25.0	901 | 794	19.2 | 18.9
Qwen3-VL-30B-Thinking	Easy	34.2 | 10.0	73.0 | 90.0	30.0 | 7.0	35.5 | 41.7	3.7 | 4.7	4.0 | 5.2	3.9 | 3.7	1.5 | 1.4	15.6 | 17.5	1216 | 1264	17.9 | 19.3
	Med	23.0 | 4.6	76.0 | 96.0	24.0 | 5.0	36.1 | 45.1	3.8 | 4.9	4.1 | 5.5	3.7 | 3.6	1.4 | 1.2	14.7 | 17.3	1285 | 1304	18.0 | 19.2
	Hard	30.2 | 3.6	74.0 | 97.0	24.0 | 2.0	36.8 | 57.8	3.9 | 5.5	4.0 | 5.8	3.8 | 3.4	1.5 | 1.2	15.2 | 17.7	1259 | 1256	18.0 | 19.2
MiniMax-M2.5 (Non-VL)	Easy	37.7 | 13.3	70.7 | 87.0	28.0 | 7.1	35.3 | 62.9	4.0 | 5.7	4.0 | 5.6	3.5 | 3.5	1.7 | 1.6	15.0 | 18.3	827 | 687	11.4 | 11.1
	Med	22.0 | 7.3	75.3 | 93.1	18.0 | 9.1	38.2 | 80.8	3.7 | 5.3	4.0 | 5.7	3.1 | 3.5	1.2 | 1.3	12.6 | 17.3	540 | 740	5.2 | 11.6
	Hard	26.3 | 6.3	74.2 | 94.2	21.0 | 4.5	38.7 | 84.4	4.3 | 6.3	4.0 | 6.6	3.3 | 3.3	1.5 | 1.4	14.5 | 12.9	734 | 722	9.5 | 12.5
DeepSeek-R1-0528(Non-VL)	Easy	32.3 | 17.0	69.2 | 83.5	30.3 | 5.3	30.7 | 53.5	3.3 | 4.8	4.0 | 5.6	4.6 | 4.5	2.0 | 2.1	19.5 | 23.3	743 | 676	13.1 | 14.6
	Med	26.3 | 8.3	75.5 | 94.1	20.0 | 3.0	40.2 | 81.2	3.5 | 5.1	4.0 | 5.7	4.6 | 4.3	2.1 | 2.1	21.2 | 22.6	547 | 680	11.1 | 13.9
	Hard	33.7 | 5.3	73.6 | 94.6	27.0 | 4.4	35.2 | 82.2	3.6 | 4.7	4.0 | 5.6	4.6 | 4.4	2.2 | 2.0	21.6 | 21.1	629 | 691	13.0 | 14.0
Non-reasoning Models												
GLM-4.6V	Easy	31.0 | 2.4	73.5 | 95.0	32.0 | 4.1	35.1 | 89.8	3.9 | 3.9	4.0 | 3.8	2.7 | 2.5	0.7 | 0.6	11.5 | 9.9	447 | 715	4.1 | 4.1
	Med	23.0 | 1.0	74.8 | 97.6	26.0 | 1.1	34.7 | 96.1	3.6 | 3.4	4.0 | 3.5	2.7 | 2.5	0.7 | 0.4	11.4 | 8.6	274 | 901	2.7 | 6.0
	Hard	28.8 | 1.8	74.7 | 98.7	21.2 | 1.0	38.4 | 96.1	3.7 | 3.8	4.0 | 3.7	2.7 | 2.5	0.6 | 0.5	10.9 | 9.7	289 | 582	2.6 | 3.5
DeepSeek-V3.2	Easy	39.3 | 16.0	69.8 | 83.3	29.0 | 16.0	33.8 | 65.9	4.0 | 2.8	4.1 | 3.3	4.7 | 5.0	1.8 | 1.8	17.0 | 16.4	162 | 220	2.4 | 2.8
	Med	24.0 | 7.0	75.7 | 93.9	23.0 | 7.0	36.1 | 88.8	4.1 | 2.8	4.1 | 3.2	4.6 | 4.9	1.6 | 1.9	16.3 | 16.0	158 | 225	2.2 | 2.7
	Hard	32.0 | 6.3	72.9 | 95.9	21.0 | 5.0	38.1 | 90.4	4.0 | 2.6	4.1 | 3.2	4.6 | 4.9	1.7 | 1.9	17.2 | 16.2	138 | 206	2.0 | 2.6
Proprietary Models												
Gemini-2.5-Flash	Easy	32.0 | 23.0	68.0 | 80.5	25.5 | 16.1	39.9 | 61.7	3.3 | 2.4	3.8 | 3.3	3.4 | 4.1	1.2 | 1.8	13.8 | 16.0	109 | 136	1.5 | 1.7
	Med	29.0 | 4.0	73.4 | 95.1	24.2 | 7.3	37.1 | 87.2	3.1 | 2.3	4.0 | 3.2	4.6 | 4.3	2.1 | 1.8	19.7 | 15.6	102 | 146	1.8 | 1.7
	Hard	29.0 | 5.0	72.5 | 94.4	20.8 | 4.2	39.3 | 88.3	3.0 | 2.2	4.0 | 3.1	4.6 | 4.1	2.1 | 2.1	19.2 | 16.1	98 | 133	1.6 | 1.7
Gemini-3.1-Pro-Preview	Easy	80.0 | 65.0	57.8 | 52.1	41.0 | 38.5	30.4 | 44.5	2.8 | 1.2	3.6 | 2.1	4.1 | 3.9	1.5 | 1.5	13.7 | 9.6	66 | 101	0.8 | 0.7
	Med	40.0 | 21.0	70.6 | 84.4	32.7 | 14.3	36.1 | 79.9	2.9 | 1.2	3.7 | 2.2	4.2 | 4.3	1.7 | 1.6	15.3 | 11.5	52 | 104	0.6 | 0.9
	Hard	33.0 | 19.0	71.2 | 84.7	31.9 | 13.3	37.1 | 81.2	3.1 | 1.1	3.7 | 2.0	4.0 | 4.3	1.5 | 1.7	14.3 | 11.1	59 | 86	0.7 | 0.7
Search Baselines										*TNC(k)*		*Budget*
MCTS/UCT (Smargo)	Easy	33.0 | 5.0	72.7 | 96.7	30.0 | 0.0	34.4 | 42.7	4.9 | 3.1	3.7 | 35.2	5.3 | 4.8	45.9 | 47.1	7.2 | 7.8	– | –	$200^{\dagger}$ | $200^{\dagger}$
	Med	32.0 | 1.0	73.5 | 97.5	26.0 | 0.0	29.8 | 35.6	5.7 | 2.5	3.8 | 38.0	5.2 | 4.8	52.0 | 54.6	8.7 | 9.1	– | –	$200^{\dagger}$ | $200^{\dagger}$
	Hard	22.0 | 1.0	74.7 | 98.0	21.0 | 0.0	40.6 | 37.8	5.1 | 2.8	3.8 | 49.6	5.3 | 4.9	63.1 | 65.3	11.0 | 11.4	– | –	$200^{\dagger}$ | $200^{\dagger}$
KataGo-b18	Easy	97.0 | 49.0	4.2 | 54.0	97.0 | 53.0	0.7 | 27.9	1.5 | 6.8	2.1 | 8.4	– | –	– | –	– | –	– | –	$200^{\dagger}$ | $200^{\dagger}$
	Med	75.0 | 46.0	25.5 | 59.6	62.0 | 44.0	9.5 | 28.2	5.0 | 8.7	2.6 | 9.9	– | –	– | –	– | –	– | –	$200^{\dagger}$ | $200^{\dagger}$
	Hard	58.0 | 43.0	48.1 | 67.0	48.0 | 36.0	16.8 | 29.0	7.5 | 12.0	2.9 | 10.2	– | –	– | –	– | –	– | –	$200^{\dagger}$ | $200^{\dagger}$

Table 13: Per-modality LLM accuracy results. Accuracy columns report first-move hit rate by input modality: Sym=Symbolic, Grd=Grid, S+G=Symbolic+Grid, Vis=Visual, and S+V=Symbolic+Visual. Trace-level diagnostics are reported separately in Table 12.

			Accuracy (%)				
Model	Cond.	Diff.	Sym	Grd	S+G	Vis	S+V
Kimi-K2.5	$K = 4$	Easy	57	47	49	51	56
		Med	25	30	30	28	29
		Hard	33	25	38	37	37
	$K = \text{None}$	Easy	29	20	35	22	33
		Med	10	11	12	9	13
		Hard	9	1	4	3	5
Qwen3-VL-235B	$K = 4$	Easy	46	47	41	29	37
		Med	23	27	27	26	22
		Hard	31	26	31	39	33
	$K = \text{None}$	Easy	17	12	20	6	24
		Med	11	9	7	6	11
		Hard	6	5	11	6	8
GLM-4.6V	$K = 4$	Easy	33	23	29	36	34
		Med	17	23	28	22	25
		Hard	24	28	26	34	32
	$K = \text{None}$	Easy	4	3	1	0	4
		Med	0	3	1	0	1
		Hard	3	2	2	0	2
Qwen3-VL-30B	$K = 4$	Easy	34	32	35	31	39
		Med	24	24	19	22	26
		Hard	36	30	31	31	23
	$K = \text{None}$	Easy	9	4	15	4	18
		Med	7	2	5	4	5
		Hard	6	2	2	1	7
MiniMax-M2.5	$K = 4$	Easy	41	36	36	—	—
		Med	22	21	23	—	—
		Hard	25	27	27	—	—
	$K = \text{None}$	Easy	10	12	18	—	—
		Med	9	4	9	—	—
		Hard	3	8	8	—	—
DeepSeek-R1-0528	$K = 4$	Easy	32	27	38	—	—
		Med	31	26	22	—	—
		Hard	28	41	32	—	—
	$K = \text{None}$	Easy	13	18	20	—	—
		Med	8	11	6	—	—
		Hard	4	4	8	—	—
Qwen3-30B	$K = 4$	Easy	20	37	23	—	—
		Med	21	24	19	—	—
		Hard	24	39	30	—	—
	$K = \text{None}$	Easy	15	9	17	—	—
		Med	8	6	7	—	—
		Hard	3	3	5	—	—
DeepSeek-V3.2	$K = 4$	Easy	33	41	44	—	—
		Med	21	29	22	—	—
		Hard	37	31	28	—	—
	$K = \text{None}$	Easy	19	10	19	—	—
		Med	6	8	7	—	—
		Hard	5	6	8	—	—

Table 14: Discriminative power of search-trace metrics.

Metric	d	Direction	ρ	Assessment
SWR	2.07	corr. $<$ wrong	-0.82	Strong
SFH	1.02	corr. $>$ wrong	$+0.71$	Strong
SCR	0.57	corr. $<$ wrong	-0.21	Moderate
TMD	0.11	$\approx$	$+0.82$	Cross-model
TMF	0.46	corr. $>$ wrong	$+0.75$	Cross-model
ITT	0.28	corr. $>$ wrong	$+0.46$	Moderate
TNC	0.05	$\approx$	$+0.54$	Moderate

Table 15: Weight sensitivity of SearchE variants against accuracy on non-baseline LLM rows. Weights are ordered as wrong-branch waste, first-hit behavior, and correct-rank cost.

Scheme	Weights	Pearson r	Spearman ρ
Main	.50/.30/.20	.914	.960
Balanced	.33/.33/.33	.884	.945
Waste-heavy	.60/.20/.20	.915	.962
First-hit-heavy	.40/.40/.20	.913	.952
Rank-heavy	.40/.20/.40	.869	.944

and rank-heavy variants preserve high Pearson and rank correlation. This supports using SearchE as a compact summary of search organization rather than a fragile fitted score.

E Evaluation Settings

All LLM evaluations use the same task interface: bounded-candidate runs provide $K = 4$ options, while open-search runs use $K = \mathrm{None}$ and require the model to generate the first move. Final answers are scored after coordinate normalization. Decoding settings vary by model family and provider; most long-form runs use temperature 0.7 with a large output budget, while some self-hosted or deterministic fill-in runs use temperature 0–0.5.

For reproducibility, each evaluation record links the normalized problem state, difficulty, modality, candidate set, side to play, model family, decoding setting, output budget, and final normalized answer. This is sufficient to recompute answer metrics from final moves, while process metrics are interpreted only for the visible reasoning exposed by the model.

For proprietary systems that expose only reasoning summaries, tree size and token-scale measurements are treated as summary-derived observations. They are useful for comparing visible output organization, but they are not claims about hidden internal computation.

F MCTS/KataGo Settings

We compare LLM traces with two non-LLM search references: Smargo MCTS/UCT and KataGo. These are not token-equivalent baselines. They provide search-reference runs that show how non-language search procedures allocate fixed playout or visit budgets on the same normalized tsumego board states.

Root modes. We use two root settings. *Open-root* starts from the original board and lets the search procedure select among legal root moves. For Smargo this uses its local legal-move generator; for KataGo this means no root candidate restriction in the analysis query. *Restricted-root* uses the benchmark $K = 4$ interface: only the four listed answer options are permitted at root after filtering occupied or invalid coordinates. Open-root runs test whether final distractors are naturally search-visible, while restricted-root runs align the search references with the LLM four-choice interface.

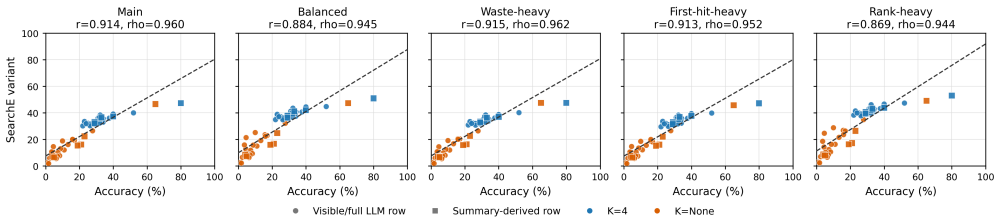


Figure 8: Accuracy–SearchE scatter plots under alternative SearchE weight schemes. The relationship remains monotonic across nearby weight choices, supporting the use of SearchE as a compact search-organization summary rather than a fragile fitted score.

Table 16: Reproducibility summary for model evaluations. Provider names follow the main experimental setup; temperature and output budgets summarize the evaluated run families.

Model family	Provider	Modalities	Temp.	Max output
Kimi-K2.5	Moonshot AI	S/G/S+G	0.7	32k
Qwen3-VL-235B-Thinking	Alibaba/Qwen	S/G/V/S+V/S+G	0.7	32k
Qwen3-VL-30B-Thinking	self-hosted vLLM	S/G/V/S+V/S+G	0.5–0.7	30k–32k
MiniMax-M2.5	MiniMax	S/G/S+G	0.7	32k
DeepSeek-R1-0528	DeepSeek	S/G/S+G	0.7	32k
DeepSeek-V3.2	DeepSeek	S/G/S+G	0.7	32k
GLM-4.6V	Zhipu AI	S/G/V/S+V/S+G	0.7	32k
Gemini-2.5 family	Google	S/G/V/S+V/S+G	0–default	–
Gemini-3 family	Google	S/G/V/S+V/S+G	0–default	–

Smargo MCTS/UCT. Smargo is used as an unguided MCTS/UCT reference, using the public implementation by Sun-Yize et al. Each problem receives 200 MCTS playouts with fixed seeds. In the restricted-root setting, the root action set is limited to the four benchmark candidates; in the open-root setting, the search may choose among legal local root moves. The candidate-construction audit uses the early part of the same playout trajectory to check whether selected distractors are encountered under shallow search.

Smargo selection uses

$$Q(s,a) + c_{\text{puct}} P(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)}, \tag{11}$$

with $c_{\text{puct}} = 5.0$ in our runs. Root priors are uniform over the permitted root moves. At a newly expanded leaf, Smargo expands the available local moves with uniform priors, then performs a rollout until a local tsumego terminal condition is reached, no valid continuation remains, or the board-area depth limit is reached. The terminal condition is local to the life-and-death solver rather than full-game territory scoring, so the baseline is best interpreted as a local search reference rather than a general Go-playing agent.

KataGo. KataGo is used as a neural-guided search reference. We use KataGo v1.16.4 with the b18 network and a 200-visit budget per problem under Chinese rules, komi 7.5, and a 19×19 board. The analysis configuration uses deterministic root behavior with resignation disabled, no root noise, and principal-variation visit reporting enabled. In restricted-root runs, the root is limited to the benchmark's four candidates; in open-root runs, no such candidate restriction is imposed. The same visit trajectory is used for shallow-budget audit at visits 1–30.

Input and output mapping. Both references consume the normalized board state, not natural-language puzzle text. The board matrix is converted to the shared Go coordinate system `A--T` with `I` skipped. Smargo internally indexes board points and maps them back to benchmark coordinates; KataGo receives the same normalized stones and side-to-play information through its analysis interface. Top-1 accuracy is computed from the final root move with the largest visit count after coordinate normalization. Illegal, occupied, pass, missing, or off-format outputs are treated as non-correct; in restricted-root runs occupied options are filtered before evaluation.

Table 17: Reproducible non-LLM search-reference settings.

Method	Search type	Root mode	Budget	N	Output used
Smargo MCTS/UCT	unguided MCTS/UCT	$K = 4$	200 playouts	300	root distribution + full tree
Smargo MCTS/UCT	unguided MCTS/UCT	open	200 playouts	300	root distribution + full tree
KataGo b18	neural-guided search	$K = 4$	200 visits	300	root distribution + principal variations
KataGo b18	neural-guided search	open	200 visits	300	root distribution + principal variations

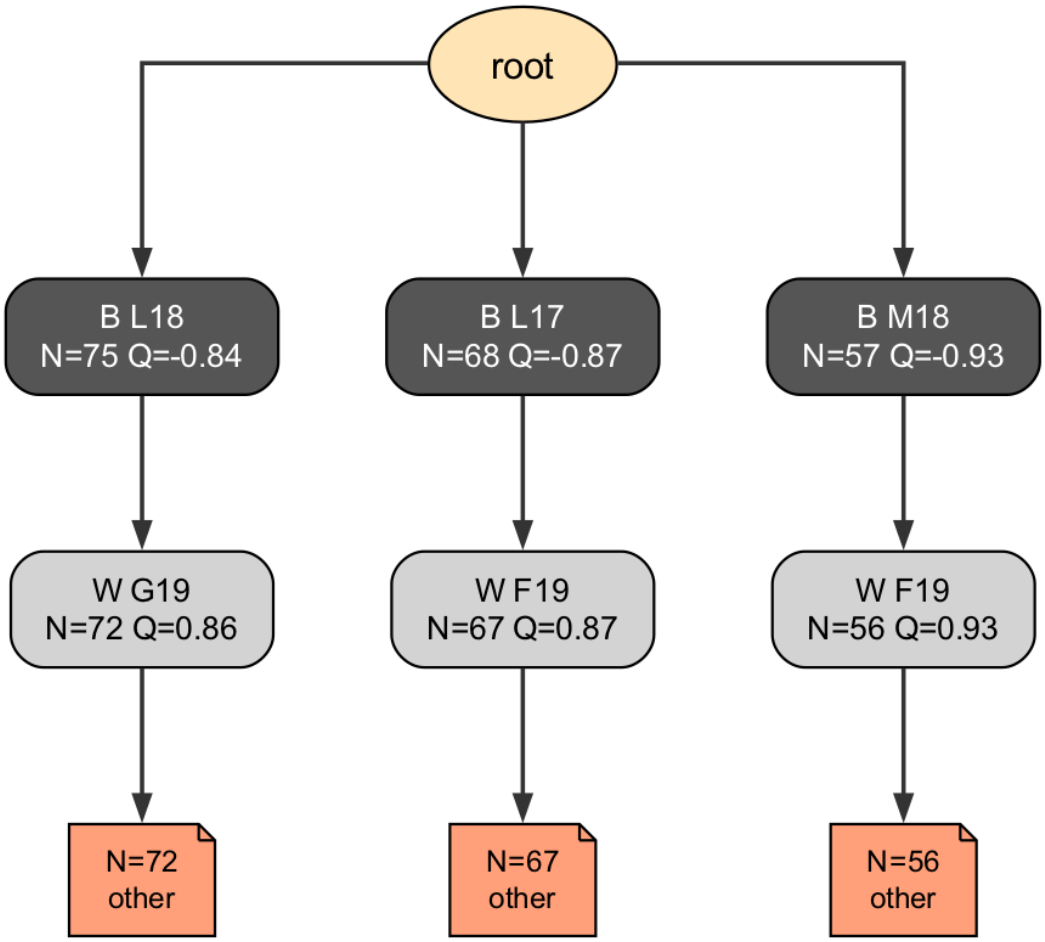


Figure 9: Representative Smargo MCTS/UCT tree for a restricted-root $K = 4$ problem after 200 playouts. The same trajectory format is used to compute MCTS tree-size diagnostics such as $\mathrm{TNC}(200)$, maximum depth, and maximum branching factor.

For metric mapping, Smargo exposes an explicit final MCTS tree, so $\mathrm{TNC}(k)$ is the number of reachable tree nodes after k playouts, with final TNC reported at $k = 200$. KataGo exposes root statistics and principal variations rather than a complete internal search tree, so KataGo tree-size columns are not interpreted as internal TNC. For both references, root-level SearchE components are computed from the root visit distributions: $\mathrm{SWR} = 1-$correct root visit share, SFH indicates whether the earliest top root move is correct, SCR is derived from the final correct-candidate rank and root branch count, and SSC counts top-root switches over the 1–200 trajectory. Budget is reported as playouts for Smargo and visits for KataGo, never as language tokens.

Figure 9 shows a representative final Smargo tree for a restricted-root $K = 4$ problem after 200 playouts. The example is included to make the reported TNC, depth, and branching-factor quantities concrete: unlike KataGo, Smargo exposes the explicit explored tree, so each node-count statistic is computed from the saved MCTS tree rather than inferred from principal variations.

Search-reference budgets are kept in their native units: playouts for Smargo, visits for KataGo, and tokens for LLMs. The main non-LLM setting uses 200 playouts or visits per problem; smaller intermediate budgets are retained only for trajectory analysis and are not used as separate evaluation metrics.

G PROMPT TEMPLATES

We include faithful templates of the Chinese task prompts used for evaluated models. The extraction prompt is already described in Appendix C; here we report only the two task prompts used for bounded candidate discrimination and open first-move generation.

Bounded candidate template.

> You are a top-level Go master. Please carefully analyze this Go life-and-death problem, reason deeply, and choose the optimal first move for Black from the given options. The input then provides one or more board presentations: coordinate lists, a numeric board matrix, a rendered board image, or their combinations. The prompt lists the candidate moves as `A. <coord>`, `B. <coord>`, ... Please think step by step about the continuations after each option, and finally state your answer clearly. Answer format: at the end, explicitly write "answer is X", where X is one of the option labels.

Open-search template.

> You are a top-level Go master. Please carefully analyze this Go life-and-death problem, reason deeply, and find the optimal first move for the side to play. The input then provides one or more board presentations: coordinate lists, a numeric board matrix, a rendered board image, or their combinations. No candidate moves are provided. Please think step by step about possible moves and their continuations, and finally state your answer clearly. Answer format: at the end, explicitly write "answer is X", where X is a board coordinate such as `T18`.

Example bounded response. The following illustrates the expected response style: free-form reasoning followed by an explicit final option label.

> Option A plays directly at the vital point and leaves White without enough eye space after the local reply. Option B lets White connect and keep liberties, so Black loses the forcing sequence. Options C and D are farther from the eye-shape weakness and do not solve the local life-and-death problem. Therefore the strongest first move is option A. **Answer is A.**

H EXAMPLE RESPONSES AND EXTRACTED TREES

We report eight representative audited process-tree examples from the bounded $K = 4$ candidate interface. The cases are labeled as `K-4-E-<id>--<model>` and are chosen to place one model-family example in each panel of a single-page montage. Each panel shows the final extracted tree for one model response, together with the selected option and verified answer.

The resulting cases expose several recurring patterns. Correct traces may be short when the key first move is introduced immediately, or longer when the model explicitly verifies several local replies before choosing. Wrong traces are often still structured: they mention the correct region but either attach the terminal judgment to the wrong branch, drift in coordinate reconstruction, or accept a local eye-shape claim that does not match the verified answer. This is why the process tree is useful beyond final accuracy: it records where the visible search path loses alignment with the answer key.

J DOMAIN-SPECIALIZED MODEL: LOGOS

We do not include Logos in the quantitative tables because the available setup does not provide outputs under the same prompt and response format. The qualitative distinction is still important: a

Table 18: Displayed $K = 4$ extracted-tree cases. The case labels follow `K-4-E-<id>--<model>`.

Case	Tier	Model	Outcome	Correct
K-4-E-001	Easy	Qwen3-30B	wrong	T15
K-4-E-002	Easy	Kimi-K2.5	correct	P14
K-4-E-003	Easy	MiniMax-M2.5	wrong	T13
K-4-E-004	Easy	DeepSeek-V3.2	correct	P15
K-4-E-005	Easy	DeepSeek-R1	correct	T15
K-4-E-006	Easy	GLM-4.6V	correct	S13
K-4-E-007	Easy	Qwen3-VL	correct	S17
K-4-E-008	Easy	Gemini-2.5-Pro	correct	T13

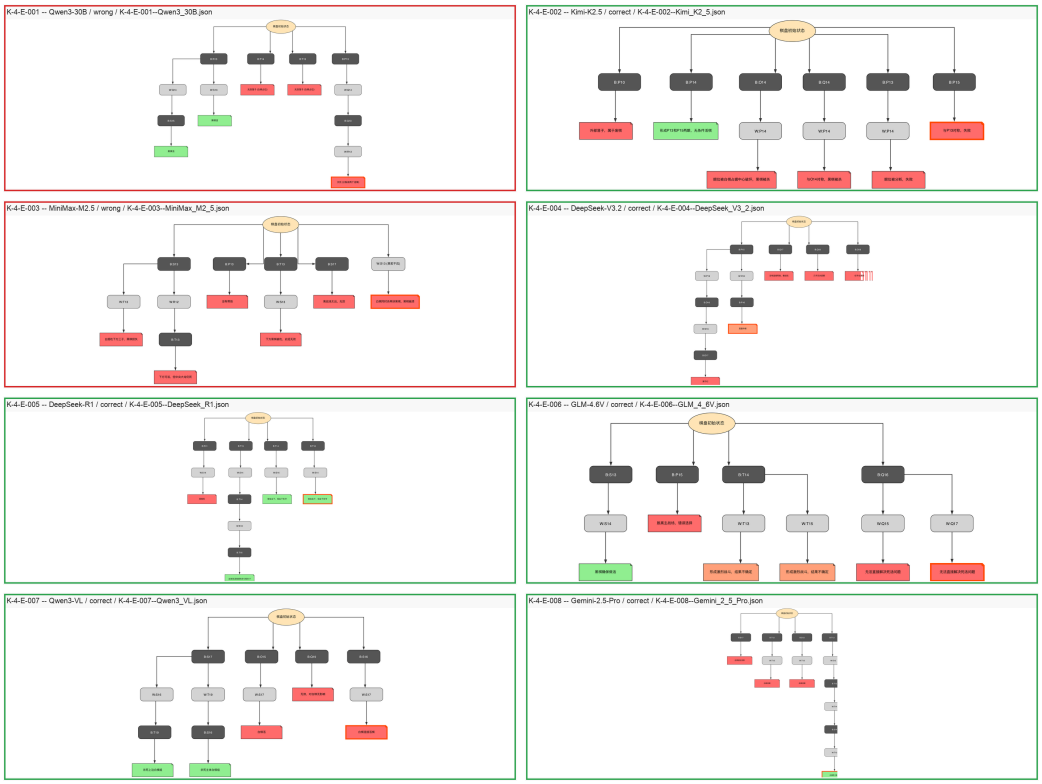


Figure 10: $K = 4$ extracted process trees for eight displayed Easy examples. Green panels denote correct final answers and red panels denote wrong final answers.

Go-specialized policy model may output plausible whole-game moves while failing at local life-and-death search because the objective is not merely to continue the game, but to organize adversarial forcing lines around life, death, capture, and eye-shape constraints.

Figure 11 shows a manual probe using the first benchmark problem in the online Logos interface. For this problem, the verified answer set is `P15` or the equivalent first move `N14`, as shown in Appendix 4. Logos instead returns `R4`, and its explanation is framed as ordinary game continuation and positional commentary rather than as a local life-and-death solution. It discusses global strength and follow-up direction, but does not identify the local live-or-kill objective, enumerate candidate first moves, or verify forcing replies. This case is not used as a quantitative result; it illustrates why a domain-specialized Go interface is not automatically comparable to TsuGO unless its prompt, answer format, and trace extraction are aligned with local tsumego solving.



Figure 11: Qualitative Logos probe on the first benchmark problem. The verified first move is `P15` or equivalent `N14`, whereas Logos returns `R4`. The response behaves like a general Go continuation analysis rather than a local life-and-death solution trace, so it is treated as diagnostic evidence for response-format mismatch rather than as a scored benchmark run.

K Failure Analysis

We audit wrong answers with a balanced slice across model families, difficulty tiers, and both $K = 4$ and open-search interfaces. Figure 12 summarizes the dominant error sources. The largest categories are search-organization failures rather than surface presentation errors: the solver either never enters the correct local branch, accepts a weak local line as decisive, or drifts from the intended coordinate and board state.

The distribution suggests a layered failure structure. At the perceptual layer, visual grounding errors arise when the position is encoded with an incorrect stone, color, or occupancy relation; these errors are comparatively infrequent, but they poison all later tactical computation because the solver is reasoning over the wrong local state. At the state-computation layer, coordinate/state drift reflects failures to maintain liberties, adjacency, captures, or coordinate identities across a multi-step line. These cases are not simply notation mistakes: they indicate that the internal board update is no longer synchronized with the candidate sequence being evaluated.

At the search layer, correct-branch absence and open-search diffusion expose two complementary weaknesses. In the bounded interface, the correct candidate may be available but not treated as the main tactical branch. In the open interface, the model often expands many plausible-looking moves without converging to the vital point, which produces breadth without a decisive local objective. Finally, weak-line acceptance and branch abandonment occur at the adversarial verification layer: the solver reaches a relevant region, but evaluates the opponent's reply too optimistically, prematurely terminates the line, or switches away from a still-viable forcing branch. Thus the common failure is not lack of Go vocabulary, but failure to keep perception, board-state computation, candidate search, and adversarial verification aligned.

L Discussion and Limitations

What TsuGO measures. TsuGO measures local adversarial search organization rather than general Go-playing strength. Its main target is how a model plans a reasoning path and allocates observable reasoning resources across competing candidates: whether it enters the vital branch, sustains verification under opponent replies, avoids spending effort on shallow wrong lines, and reallocates effort when a line is refuted. The process-tree view therefore separates the amount

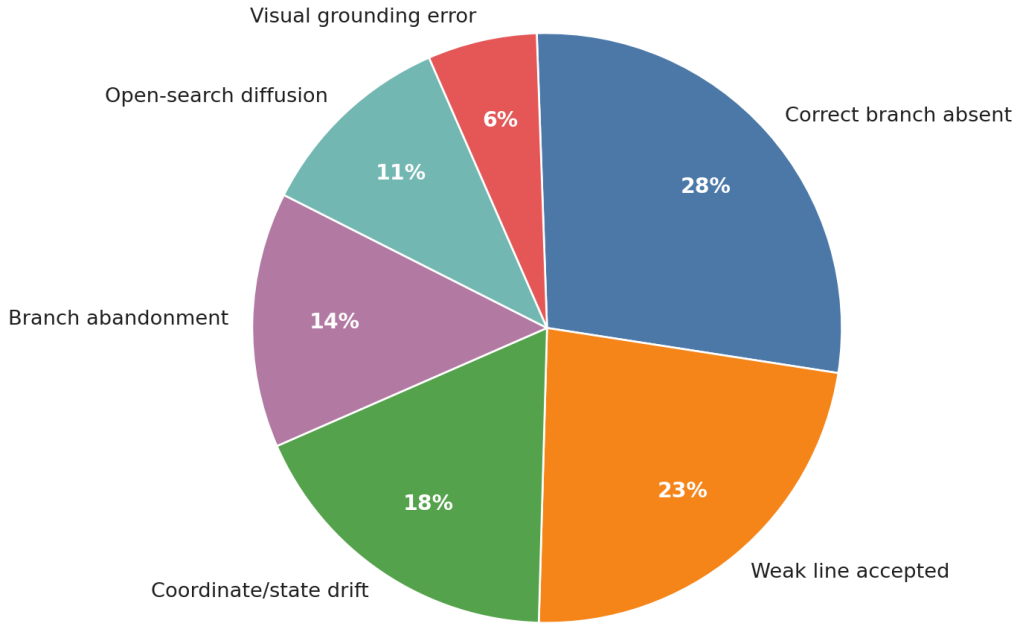


Figure 12: Failure-mode distribution from the balanced qualitative audit. The dominant errors concern search organization and local verification rather than output formatting.

of reasoning from where that reasoning is placed, making branch selection, continuation depth, refutation checking, and backtracking visible as distinct planning behaviors.

Human–AI interaction context. This benchmark also exposes a mismatch between many current optimization settings and adversarial planning tasks. Modern instruction tuning and deployment feedback often emphasize cooperative, non-adversarial, multi-turn interaction: the user can clarify intent, correct mistakes, narrow the search space, or accept partial progress. That setting is valuable, but it gives the model external scaffolding for planning. In TSUGO, the opponent does not cooperate, hidden correction is unavailable, and the model must maintain the local state while anticipating refutations. The resulting failures therefore should not be read only as Go-specific errors; they indicate that models optimized for helpful interactive dialogue may still lack robust autonomous control over where reasoning effort is spent when the environment actively pushes back.

Trace observability. Search trees are extracted from observable CoT traces, so our process metrics describe the visible reasoning path rather than the model's latent computation. This limitation is also an opportunity: future work could analyze internal activations, attention/state trajectories, or planner–verifier signals to recover latent search paths and compare them with the externalized trace.

Future work. A natural next step is to expand from local tsumego to broader agent-system settings where candidate generation, state tracking, tool use, memory, and adversarial feedback interact over longer horizons. Another direction is to compare autonomous solving with human-guided multi-turn repair, measuring when interaction scaffolding compensates for weak internal search control. Finally, TSUGO points to targeted training frameworks that supervise branch prioritization, opponent-response verification, state maintenance, backtracking, and search-budget allocation rather than only final answers or fluent rationales.