International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 03 | March - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Graphzero Agent: Monte Carlo Graph Search as Reasoning-As-A-Service for General Agentic Systems

Karthik Rajkumar Kannan[1], Bipin Chandra[2]

Abstract

Current agentic AI systems, predominantly based on the Reason-Act (ReAct) paradigm, execute single-trajectory reasoning with limited backtracking, no principled exploration, and no reuse of repeated subproblems. Empirical analyses reveal brittleness to prompt perturbations and catastrophic performance degradation on tasks exceeding 15–120 steps. We propose GraphZero Agent, a reasoning architecture that adapts AlphaZero's policy/value-guided Monte Carlo Tree Search (MCTS) to Monte Carlo Graph Search (MCGS) over general agentic belief-states. GraphZero Agent generalizes the search space from trees to Directed Acyclic Graphs, enabling transposition merging of semantically equivalent reasoning states via dense embedding similarity and Approximate Nearest Neighbor search. Intermediate states are scored by Process Reward Models (PRMs) trained without human annotation through a self-rewarding rollout pipeline that produces Process Preference Models (PPMs) from pairwise trajectory comparisons. Beyond single-session inference, a dual-loop Empirical-MCTS architecture provides continuous non-parametric learning: a local loop refines the LLM's generation policy within a search session via contrastive meta-prompting (PE-EMP), while a global loop distills winning trajectories into reusable heuristics across sessions. The system is deployed as Reasoning-as-a-Service (RaaS) via Model Context Protocol (MCP) or OpenAPI, enabling any agentic orchestration framework to invoke deep deliberative search with independent scaling and LLM hot-swapping.

[1]Accenture, k.rajkumar.kannan@accenture.com

[2]Accenture, bipin.a.chandra@accenture.com

Key Words: Monte Carlo Graph Search, Reasoning-as-a-Service, Agentic AI, Process Reward Models, AlphaZero, Large Language Models.

1. INTRODUCTION

The artificial intelligence ecosystem is transitioning from stochastic text completion to the autonomous execution of complex, multi-step objectives. This evolution has produced **agentic AI** — systems that perceive dynamic environments, execute sequential plans, orchestrate external tools, and adapt to feedback [4, 7, 25]. Large Language Models (LLMs) serve as the cognitive engines, but as demand for reliability in high-stakes domains intensifies, the structural limitations of autoregressive generation have become the primary bottleneck.

The prevailing approach enhances LLM reasoning by optimizing **test-time compute** — shifting computational effort from pre-training to inference [5, 6, 7]. While search strategies such as Tree of Thoughts (ToT) [5] and LATS [7] demonstrate the value of exploring multiple reasoning paths, they remain computationally expensive, structurally rigid, and fail to exploit redundancies in the logic space. ReAct-style agents [4] improve grounding but offer no principled search control, no explicit value estimation, and no state reuse.

This paper proposes **GraphZero Agent**, an architecture that fuses LLM semantic comprehension with the mathematically rigorous search dynamics of AlphaZero [1, 2]. The key advancement is generalizing Monte Carlo Tree Search to **Monte Carlo Graph Search (MCGS)** [9, 10] — modeling reasoning as a Directed Acyclic Graph (DAG) where semantically identical states are detected and merged as transpositions.

Contributions

1. **MCGS for agentic reasoning.** We adapt AlphaZero-style graph search to open-ended LLM reasoning with three-tier transposition detection (exact, semantic, structural) and delta-corrected DAG backpropagation [9]. (See Appendices C–D for mathematical details and algorithm pseudocode.)

2. **Self-rewarding Process Preference Models.** MCGS rollouts generate pairwise preference datasets that train PPMs without human annotation, extending [12, 13] to general agentic tasks.

3. **Dual-loop continuous learning.** Pairwise-Experience-Evolutionary Meta-Prompting (PE-EMP) for within-session adaptation and a Memory Optimization Agent for cross-session heuristic distillation.

4. **Reasoning-as-a-Service.** MCGS exposed via MCP/OpenAPI with independent scaling, LLM hot-swapping, and integration patterns for LangGraph, CrewAI, and AutoGen.

5. **Comprehensive evaluation framework.** Six-benchmark evaluation plan with six ablation experiments isolating each contribution. (See Appendix H for full ablation design.)

2. PROBLEM ANALYSIS: LIMITATIONS OF REACT-STYLE AGENTS

Stateless execution and memory collapse. ReAct architectures rely on stateless prompt-response execution where each decision lacks persistent memory of prior reasoning [4]. The MEM1 framework [37] shows degradation begins at 32k–64k tokens. On the LORE benchmark [38], model accuracy approaches zero beyond 120 steps.

Error cascades and absent lookahead. ReAct operates via greedy, step-wise action selection — arbitrarily suboptimal under delayed rewards [8]. A single hallucinated premise cascades through the entire plan with no capacity to backtrack.

Prompt brittleness. Sensitivity analysis [36] demonstrates that ReAct performance is driven by exemplar-query similarity rather than the reasoning-action interleaving itself.

Table 1: Comparison of reasoning approaches

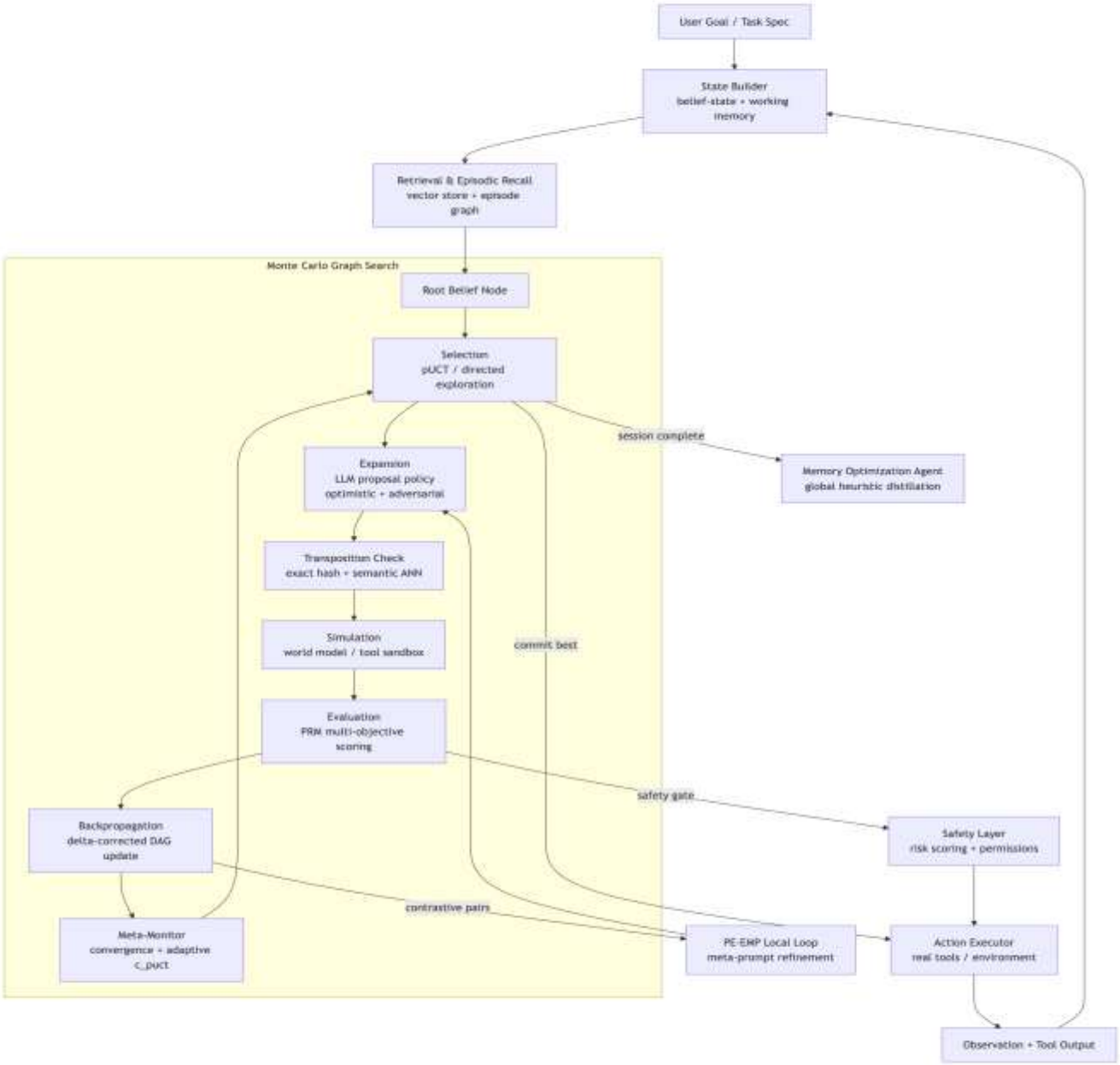
Feature	ReAct [4]	ToT [5]	AlphaZero MCTS [1]	GraphZero (Ours)
Search topology	Linear / single path	Rigid spanning tree	Expanding tree	DAG (transpositions)
Lookahead	None (myopic)	Limited layers	Deep rollouts + value	Deep rollouts + PRM + multi-objective
Evaluation	Post-action observation	LLM self-evaluation	Learned V_θ	PRM/PPM + $(V_{\text{succ}}, V_{\text{cost}}, V_{\text{risk}})$
Exploration	Greedy	BFS or DFS	pUCT	pUCT + Gumbel + adversarial
Backtracking	Full restart	Rigid rules	Fluid via Q-updates	Fluid + delta-corrected DAG
State reuse	None	None	None (tree)	Exact + semantic + structural
Cross-session learning	None	None	Self-play training	Dual-loop (PE-EMP + Memory Agent)

3. GRAPHZERO AGENT ARCHITECTURE

GraphZero Agent replaces single-trace prompting with policy/value-guided MCGS in a typed agent state space:

- **Nodes** — belief-states: observations, tool results, working memory, retrieved context, latent embeddings.
- **Edges** — typed actions: tool calls, memory operations (Retrieve, Reflect, Compress), reasoning operators (propose_subgoal, critique, compress).
- **MCGS** — selection via PUCT; expansion via LLM proposal policy; evaluation via PRM; delta-corrected backpropagation; transposition merging.

3.1 Architecture Diagram



GraphZero Agent Architecture — MCGS search loop with dual-loop learning and RaaS integration.

Fig. 1: GraphZero Agent architecture showing the MCGS search loop with dual-loop learning and RaaS integration.

The MCGS search loop iterates through selection (PUCT), expansion (LLM proposals), transposition checking, evaluation (PRM), and delta-corrected DAG backpropagation. Full algorithm pseudocode is provided in Appendix D; the graph schema is detailed in Appendix E.

4. KEY TECHNICAL INNOVATIONS

4.1 Semantic Transposition Detection

In language environments, distinct surface forms can represent identical logical states (*"x equals 5"* vs. *"the value of x is found to be 5"*). GraphZero Agent implements three-tier detection:

Table 2: Transposition detection tiers

Tier	Method	Speed	False Positive Risk
Exact	Environment snapshot + tool state + memory buffer hash	Sub-microsecond	Zero
Semantic	Dense embedding + ANN search (cosine >= 0.92)	Millisecond	Low
Structural	Relational structure-mapping over memory templates	~100ms	Medium (safeguarded)

Conservative thresholds, downstream Q-value consistency monitoring, and merge rollback protect against incorrect merges. The full detection algorithm is given in Appendix D (Algorithm 3).

4.2 Process Reward Models and Self-Rewarding Training

Outcome Reward Models (ORMs) assign reward only at trajectory termination, creating severe credit assignment failures [11]. **Process Reward Models (PRMs)** provide step-level supervision, scoring each intermediate node and enabling real-time pruning of hallucinatory branches.

To eliminate human annotation, GraphZero Agent synthesizes process rewards autonomously: MCGS rollouts from intermediate nodes produce empirical success probabilities, which are converted to **pairwise preference datasets** that train a **Process Preference Model (PPM)** via ranking loss — extending ReST-MCTS* [12] and rStar-Math [13] to general agentic tasks.

4.3 Dual-Loop Continuous Learning

Standard inference-time search is stateless across sessions. GraphZero Agent transitions to continuous non-parametric learning via two loops:

Local Loop (PE-EMP). Within a search session, contrastive sibling pairs (high-Q vs. low-Q nodes) are analyzed by an LLM to extract task-specific principles that refine the system prompt in real time. (Algorithm 4 in Appendix D.)

Global Loop (Memory Optimization Agent). A dedicated agent manages a cross-session heuristic repository via four atomic operations — **Add**, **Modify**, **Merge**, **Delete** — using the Bradley-Terry model [39] for quantitative integration and Enhanced Borda Count for globally consistent ranking.

4.4 Hierarchical Planning and Extensions

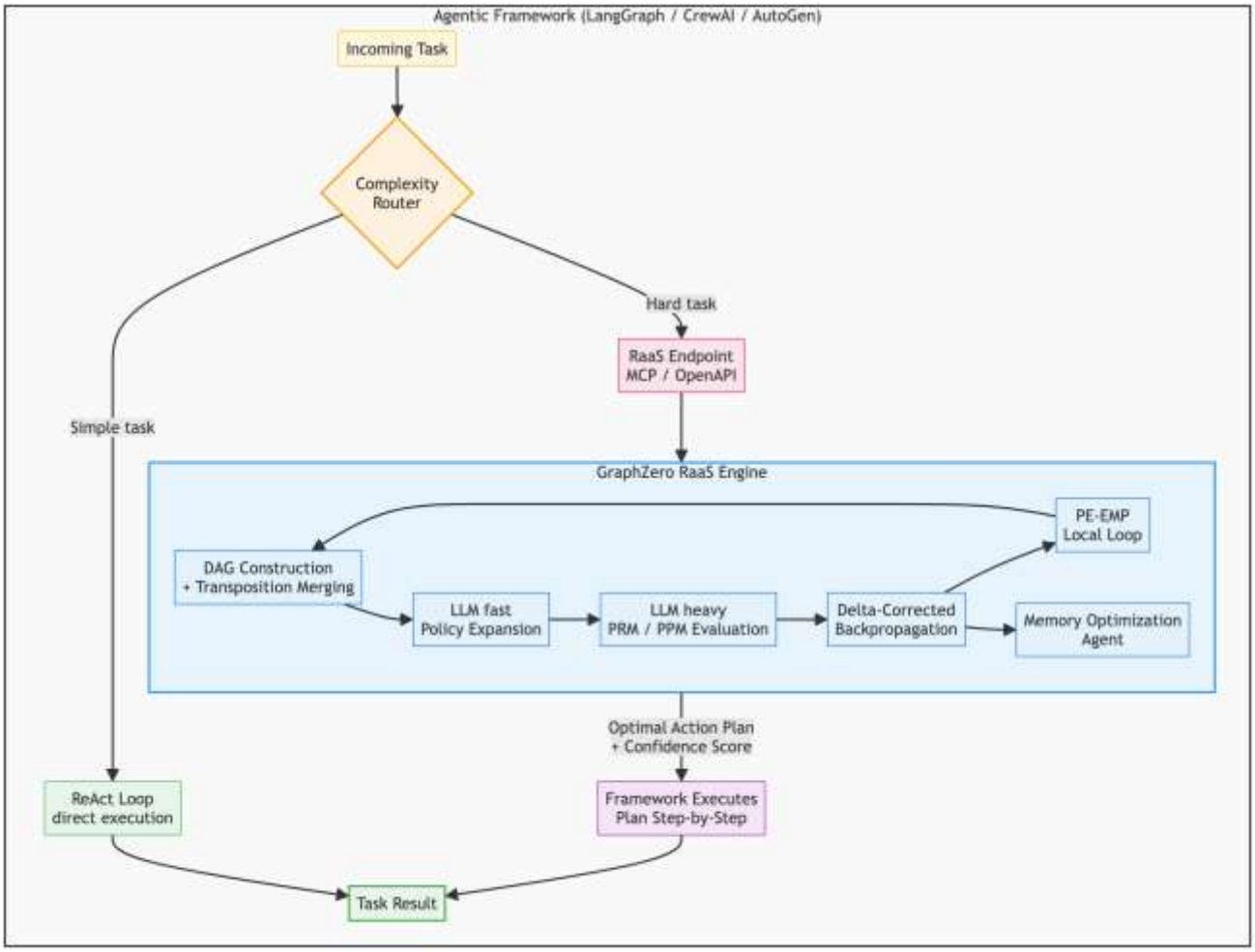
Following the options framework [18], GraphZero Agent supports temporal abstraction via learned action macros, two-level search, and option discovery from successful trajectories. Partial observability is handled through belief nodes (POMCP [19]) and information-set hashing [20].

5. REASONING-AS-A-SERVICE DEPLOYMENT

GraphZero Agent is deployed as a **Reasoning-as-a-Service (RaaS)** module behind standardized protocols:

- **Model Context Protocol (MCP)** — for native integration with MCP-compatible agents.
- **OpenAPI / REST** — for universal HTTP-based consumption.

This achieves: (a) **independent scaling** of reasoning compute vs. orchestration; (b) **framework-agnostic consumption** by LangGraph, CrewAI, AutoGen, or any custom framework; (c) **LLM hot-swapping** — fast models for expansion, heavy models for PRM evaluation.



RaaS deployment architecture — complexity routing, GraphZero engine internals, and framework integration.

Fig. 2: RaaS deployment architecture showing complexity-based routing, the GraphZero engine internals, and framework integration.

6. TRAINING AND EVALUATION OVERVIEW

6.1 Three-Loop Self-Improvement

Following AlphaZero's self-play paradigm [1] and MuZero's extension [3]:

1. **Search-Data Generation** — Run MCGS-guided episodes on task mixtures; store search graphs, trajectories, and outcomes.

2. **Supervised Distillation** — Train policy head to match root visit distributions; train value head to predict outcomes; train PRM/PPM via self-rewarding rollouts.

3. **Preference/Safety Alignment** — Integrate RLHF [33], Constitutional AI [34], or DPO [35] where automated reward is incomplete.

Loss functions and hyperparameter ranges are provided in Appendix G.

6.2 Benchmark Suite

Table 3: Evaluation benchmarks

Benchmark	Domain	Key Properties	Reference
WebArena	Interactive web	Realistic, reproducible, large gap to humans	[26]
WebShop	E-commerce navigation	Grounded actions, human trajectories	[27]
ALFWorld	Embodied text tasks	Long horizons, sparse rewards	[28]
SWE-bench	Repo-level coding	Real GitHub issues, test-suite evaluation	[29]
HumanEval	Unit-test coding	Functional correctness, pass@k	[30]
HotpotQA / FEVER	Multi-hop QA	Retrieval actions, evidence-based answers	[31, 32]

6.3 Metrics

- **Task success** — success rate, exact match, evidence F1.
- **Cost** — tool calls/episode, latency, tokens, API spend.
- **Search efficiency** — nodes expanded, transposition rate, value calibration.
- **Robustness** — performance under prompt perturbations [36].
- **Safety** — prompt injection success rate, unsafe calls blocked [23, 24].
- **Cross-session learning** — episode-over-episode improvement.

Baselines include ReAct [4], ToT [5], RAP [6], LATS [7], tree search for web agents [14], and SWE-agent [25]. Six ablation experiments (Appendix H) isolate each contribution.

7. SAFETY, ALIGNMENT, AND FAILURE MODES

Search amplifies both capability and risk. Safety is integrated at **search time**, not only at output time [23, 24].

Table 4: Failure modes and mitigations

Failure Mode	Mitigation
Prompt injection [23]	Provenance tagging; instruction/data separation
Search-induced overreach	Risk-aware pUCT; capability-based permissions
World-model hallucination	Calibration monitoring; real-env verification
Semantic-transposition errors	Conservative thresholds; consistency checks; rollback
Reward hacking [11]	Multi-objective value; verifier-based validation

Built-in mitigations include typed action schemas with capability-based permissions [25], search-time safety gating via V_{risk}, provenance-aware retrieval [23], alignment training (RLHF [33], DPO [35]), and full audit logging.

8. IMPLEMENTATION ROADMAP

Table 5: Phased implementation plan

Phase	Focus	Success Criterion
1: Foundation	Baselines + state format	Apples-to-apples baseline match
2: Search	MCGS core	Node-efficiency gains on >=1 benchmark
3: Learning	PRM/PPM + distillation	Match high-budget search with fewer sims
4: Continuous + RaaS	Dual-loop + service layer	Cross-session improvement; framework integration
5: Extensions	World model + hierarchy	Long-horizon task completion improvement

9. DISCUSSION AND LIMITATIONS

Strengths. GraphZero Agent combines individually validated components — MCTS/MCGS from game AI [1, 9], PRMs from math reasoning [11], episodic memory from agent research [15, 16] — into a unified architecture. The self-rewarding PPM pipeline removes the annotation bottleneck. The RaaS packaging and dual-loop learning address practical deployment concerns that purely algorithmic proposals ignore.

Semantic transposition reliability. Embedding similarity is a necessary approximation. States with cosine similarity >= 0.92 may differ in critical details. While conservative thresholds and consistency checks mitigate this, the fundamental reliability of semantic merging remains an empirical question.

PRM quality as binding constraint. In verifiable domains (math, code), the self-rewarding pipeline has empirical backing [12, 13]. In open-ended domains, effectiveness is unproven.

Latency. At 200 nodes, ~400 model calls per decision. Adaptive compute and hierarchical planning partially address this, but real-time applications may be infeasible.

Evaluation gap. This paper presents a design and evaluation plan, not empirical results. All expected outcomes are hypotheses requiring experimental confirmation.

Future Directions

- Learned transposition functions (neural hash for merge/no-merge decisions).
- Formal verification integration (terminal solvers + proof assistants).
- Multi-modal belief states (visual observations for web/GUI agents).
- Federated memory (shared heuristic repositories with privacy constraints).

10. CONCLUSIONS

GraphZero Agent addresses the structural limitations of ReAct through four compounding innovations:

1. **Monte Carlo Graph Search** with semantic transposition detection and delta-corrected DAG backpropagation.

2. **Self-rewarding Process Preference Models** providing dense, step-level supervision without human annotation.

3. **Dual-loop Empirical-MCTS** combining within-session meta-prompt refinement with cross-session heuristic distillation.

4. **Reasoning-as-a-Service** packaging via MCP/OpenAPI with independent scaling and heterogeneous model routing.

Grounded in AlphaZero's search-and-learn paradigm [1, 2] and extended with MuZero-style world models [3], hierarchical options [18], and POMDP-capable search [19], this architecture provides a concrete path from single-trajectory prompting to robust general-purpose agentic reasoning.

REFERENCES

[1] Silver, D., Hubert, T., Schrittwieser, J., et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science*, 362(6419), 2018.

[2] Silver, D., Schrittwieser, J., Simonyan, K., et al. "Mastering the game of Go without human knowledge." *Nature*, 550(7676), 2017.

[3] Schrittwieser, J., Antonoglou, I., Hubert, T., et al. "Mastering Atari, Go, chess and shogi by planning with a learned model." *Nature*, 588(7839), 2020.

[4] Yao, S., Zhao, J., Yu, D., et al. "ReAct: Synergizing reasoning and acting in language models." *ICLR*, 2023.

[5] Yao, S., Yu, D., Zhao, J., et al. "Tree of thoughts: Deliberate problem solving with large language models." *NeurIPS*, 2024.

[6] Hao, S., Gu, Y., Ma, H., et al. "Reasoning with language model is planning with world model." *EMNLP*, 2023.

[7] Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., Wang, Y.-X. "Language agent tree search unifies reasoning, acting, and planning in language models." *ICML*, 2024.

[8] Browne, C., Powley, E., Whitehouse, D., et al. "A survey of Monte Carlo tree search methods." *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 2012.

[9] Czech, J., Korus, P., Kersting, K. "Monte Carlo graph search for AlphaZero." *arXiv:2012.11045*, 2021.

[10] Leurent, E., Maillard, O.-A. "Monte Carlo graph search: Efficient and exact solving of combinatorial problems." *arXiv*, 2020.

[11] Lightman, H., Kosaraju, V., Burda, Y., et al. "Let's verify step by step." *ICLR*, 2024.

[12] Zhang, D., Wu, J., Lei, J., et al. "ReST-MCTS*: LLM self-training via process reward guided tree search." *arXiv:2406.03816*, 2024.

[13] Qi, Z., Ma, M., Xu, J., et al. "Mutual reasoning makes smaller LLMs stronger problem-solvers." *arXiv:2408.06195*, 2024.

[14] Koh, J. Y., McAleer, S., Fried, D., Salakhutdinov, R. "Tree search for language model agents." *arXiv*, 2024.

[15] Shinn, N., Cassano, F., Gopinath, A., et al. "Reflexion: Language agents with verbal reinforcement learning." *NeurIPS*, 2023.

[16] Park, J. S., O'Brien, J. C., Cai, C. J., et al. "Generative agents: Interactive simulacra of human behavior." *UIST*, 2023.

[17] Lewis, P., Perez, E., Piktus, A., et al. "Retrieval-augmented generation for knowledge-intensive NLP tasks." *NeurIPS*, 2020.

[18] Sutton, R. S., Precup, D., Singh, S. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." *Artificial Intelligence*, 112(1-2), 1999.

[19] Silver, D., Veness, J. "Monte-Carlo planning in large POMDPs." *NeurIPS*, 2010.

[20] Cowling, P. I., Powley, E. J., Whitehouse, D. "Information set Monte Carlo tree search." *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2), 2012.

[21] Danihelka, I., Guez, A., Schrittwieser, J., Silver, D. "Policy improvement by planning with Gumbel." *ICLR*, 2022.

[22] Wu, D. J. "Accelerating self-play learning in Go." *arXiv:1902.10565*, 2019.

[23] Greshake, K., Abdelnabi, S., Mishra, S., et al. "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection." *AISec*, 2023.

[24] OWASP Foundation. "OWASP Top 10 for Large Language Model Applications." 2025.

[25] Yang, J., Jimenez, C. E., Wettig, A., et al. "SWE-agent: Agent-computer interfaces enable automated software engineering." *arXiv*, 2024.

[26] Zhou, S., Xu, F. F., Zhu, H., et al. "WebArena: A realistic web environment for building autonomous agents." *ICLR*, 2024.

[27] Yao, S., Chen, H., Yang, J., Narasimhan, K. "WebShop: Towards scalable real-world web interaction with grounded language agents." *NeurIPS*, 2022.

[28] Shridhar, M., Yuan, X., Côté, M.-A., et al. "ALFWorld: Aligning text and embodied environments for interactive learning." *ICLR*, 2021.

[29] Jimenez, C. E., Yang, J., Wettig, A., et al. "SWE-bench: Can language models resolve real-world GitHub issues?" *ICLR*, 2024.

[30] Chen, M., Tworek, J., Jun, H., et al. "Evaluating large language models trained on code." *arXiv:2107.03374*, 2021.

[31] Yang, Z., Qi, P., Zhang, S., et al. "HotpotQA: A dataset for diverse, explainable multi-hop question answering." *EMNLP*, 2018.

[32] Thorne, J., Vlachos, A., Christodoulopoulos, C., Mittal, A. "FEVER: A large-scale dataset for fact extraction and VERification." *NAACL*, 2018.

[33] Ouyang, L., Wu, J., Jiang, X., et al. "Training language models to follow instructions with human feedback." *NeurIPS*, 2022.

[34] Bai, Y., Kadavath, S., Kundu, S., et al. "Constitutional AI: Harmlessness from AI feedback." *arXiv:2212.08073*, 2022.

[35] Rafailov, R., Sharma, A., Mitchell, E., et al. "Direct preference optimization: Your language model is secretly a reward model." *NeurIPS*, 2023.

[36] Wang, L., Ma, C., Feng, X., et al. "A survey on large language model based autonomous agents." *Frontiers of Computer Science*, 2024.

[37] Khandelwal, U., et al. "MEM1: Memory-augmented language models for long-horizon reasoning." *arXiv*, 2024.

[38] Chen, Y., et al. "LORE: Long-horizon reasoning evaluation for LLM agents." *arXiv*, 2024.

[39] Bradley, R. A., Terry, M. E. "Rank analysis of incomplete block designs: I. The method of paired comparisons." *Biometrika*, 39(3/4), 1952.

[40] Mineiro, P., et al. "Model2Vec: Distill a small fast model from any sentence transformer." *arXiv*, 2024.

APPENDICES

Appendix A: Notation and Preliminaries

Table A-1: Symbol definitions

Symbol	Definition
s, s_t	Belief-state (node) at step t
a, a_t	Action (edge) — reasoning step, tool call, or memory operation
$Q(s,a)$	Action-value estimate for taking action a in state s
$V_\theta(s)$	Parameterized value network estimate of state s
$P(s,a)$	Policy prior — probability of action a from the LLM
$N(s)$	Visit count of state s
$N(s,a)$	Visit count of action edge (s,a)
c_{puct}	Exploration constant for PUCT selection
π	Search-improved policy (root visit distribution)
z	Terminal outcome reward (success/failure or scalar)
r_t	Intermediate reward at step t

Symbol	Definition
$V_{\text{succ}}, V_{\text{cost}}, V_{\text{risk}}$	Multi-objective value components
$U(s)$	Scalarized utility combining multi-objective values
λ_c, λ_r	Cost and risk penalty weights
Q_δ	Delta evaluation correction for DAG backpropagation
S_c, S_p	Self-principled scores for Bradley-Terry preference model
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Reasoning DAG with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$
PRM	Process Reward Model — step-level value estimator
PPM	Process Preference Model — pairwise comparative evaluator
ORM	Outcome Reward Model — terminal-only value estimator
PE-EMP	Pairwise-Experience-Evolutionary Meta-Prompting

Appendix B: Related Work

B.1 AlphaZero and MuZero

AlphaZero [1, 2] couples a learned policy/value network with MCTS to achieve superhuman performance in chess, shogi, and Go without domain-specific heuristics. MuZero [3] extends this by learning a latent dynamics model, enabling planning without known environment rules. KataGo [22] demonstrates that algorithmic improvements can reduce AlphaZero-scale compute by ~50x, and Gumbel MuZero [21] improves policy improvement guarantees under small simulation budgets.

B.2 Monte Carlo Graph Search

Standard MCTS treats the search structure as a tree even when the underlying MDP contains transpositions. Czech et al. [9] introduce MCGS for AlphaZero, generalizing to a DAG and reporting +310 Elo improvement in crazyhouse chess. Leurent and Maillard [10] provide theoretical analysis of graph-based planning exploiting state similarity.

B.3 Search-Enhanced LLM Reasoning

Tree of Thoughts (ToT) [5] expands LLM inference to deliberate tree search. RAP [6] frames reasoning as planning with an LM world model. LATS [7] integrates MCTS with LM-based value functions and reflection. Best-first tree search methods for web agents [14] demonstrate performance scaling with test-time search compute. These works use tree structures without transposition handling.

B.4 ReAct and Its Brittleness

ReAct [4] interleaves reasoning traces with tool actions. Sensitivity analysis [36] reveals performance driven by exemplar-query similarity. The LORE benchmark [38] shows accuracy approaches zero beyond 120 steps, and MEM1 [37] demonstrates context degradation beginning at 32k–64k tokens.

B.5 Reward Modeling for LLMs

Lightman et al. [11] demonstrate that PRMs significantly outperform ORMs for multi-step math reasoning. ReST-MCTS* [12] synthesizes process rewards autonomously. rStar-Math [13] demonstrates that small LLMs can match larger models when guided by self-generated process rewards.

B.6 Agent Memory and Episodic Learning

RAG [17] formalizes hybrid parametric/non-parametric memory. Generative Agents [16] demonstrate that storing and retrieving experiences changes agent behavior. Reflexion [15] uses episodic self-reflective feedback. GraphZero Agent promotes memory to an active search operator.

B.7 Partial Observability and Hierarchical Planning

POMCP [19] extends MCTS to large POMDPs. ISMCTS [20] addresses imperfect information via information-set search. The options framework [18] formalizes temporal abstraction.

B.8 Agent Safety

LLM-integrated applications are vulnerable to prompt injection [23]. OWASP's LLM Top 10 [24] highlights key risks. SWE-agent [25] demonstrates that constrained agent-computer interfaces reduce failure modes.

Appendix C: Mathematical Foundations

C.1 Policy-Guided MCTS

MCTS constructs a search tree through four iterated phases [8]:

Selection. Traverse from root to leaf by selecting actions maximizing the PUCT criterion [1, 3]:

$$a_t = \operatorname*{argmax}_a \left(Q(s_t, a) + c_{\text{puct}} \cdot P(s_t, a) \cdot \frac{\sqrt{N(s_t)}}{1 + N(s_t, a)} \right)$$

MuZero [3] extends this with tunable (c_1, c_2) constants. GraphZero Agent adds Dirichlet root noise for training-time exploration [1] and Gumbel-style planning [21] for small budgets.

Expansion. Add child nodes representing candidate next actions. GraphZero Agent generates both optimistic and adversarial candidates for risk-aware search.

Evaluation. AlphaZero replaces random rollouts with a learned value network $V_\theta(s)$ [1]. GraphZero Agent extends this with multi-objective PRMs.

Backpropagation. Propagate the evaluation signal upward, updating $Q(s, a)$ as a running mean and incrementing visit counts.

C.2 Generalization to MCGS

In complex state spaces, distinct action sequences can yield identical states — **transpositions** [8]. MCGS [9, 10] generalizes to a DAG: when a new state matches an existing node, a directed edge is added to the existing node instead of creating a duplicate.

C.3 Cycle Prevention

A discrete step counter is embedded in the transposition hash key, ensuring acyclicity [9].

C.4 Delta-Corrected DAG Backpropagation

Transposition nodes with multiple parents must channel value updates without information leaks:

$$Q_\phi(s_t, a) = N(s_t, a) \cdot Q_\delta(s_t, a) + V^*(s_{t+1})$$

where $Q_\delta = Q(s_t, a) - q_{\text{Target}}$. This prevents Q-value plateau under both high and low simulation counts [9].

C.5 Terminal Solvers

When outcomes are provably deterministic, exact terminal values bypass neural network queries [9]. In crazyhouse chess, MCGS with terminal solvers yielded **+310 Elo** over standard MCTS.

C.6 ORM vs. PRM Comparison

Table C-1: ORM vs. PRM properties

Property	ORM	PRM [11]
Granularity	Full trajectory only	Step-by-step
Signal density	Sparse, delayed	Dense, immediate
Credit assignment	Opaque	Explicit error localization
MCGS suitability	Incompatible for mid-search	Essential for real-time pruning

C.7 Bradley-Terry Integration

Stored heuristics are integrated quantitatively using the Bradley-Terry model [39]:

$$Q_{\text{local}}(s_c) = \frac{\exp(S_c)}{\exp(S_c) + \exp(S_p)}$$

An Enhanced Borda Count produces globally consistent ranking, fusing local probabilities with global ranks.

Appendix D: Algorithm Pseudocode

Algorithm 1: GraphZero MCGS Search

ALGORITHM 1: GraphZero MCGS Search

```
------------------------------------------------------------
Input:  root state s0, policy LLM pi_theta, value model V_theta,
        state hasher H, config (K, c_puct, width, depth_max)
Output: optimal trajectory tau*

1:  G <- InitGraph(s0)
2:  for iteration = 1 to K do
3:      -- SELECTION --
4:      (leaf, path) <- SelectLeaf(G, c_puct)      // PUCT traversal
5:
6:      -- EXPANSION --
7:      candidates <- pi_theta.generate(leaf, n=width)   // LLM proposes actions
8:      for each candidate c in candidates do
9:          s_new <- BuildState(leaf, c)
10:         s_new.depth <- leaf.depth + 1
11:
12:         -- TRANSPOSITION CHECK --
13:         s_existing <- H.find_match(s_new, G.nodes_at_depth(s_new.depth))
14:         if s_existing != null then
15:             G.add_edge(leaf -> s_existing, action=c)
16:         else
17:             G.add_node(s_new)
18:             G.add_edge(leaf -> s_new, action=c)
19:
20:             -- EVALUATION --
21:             v <- V_theta.score(s_new)              // PRM multi-objective
```

```
            s_new.value <- v
            s_new.visits <- 1

            -- BACKPROPAGATION --
            DeltaBackprop(G, s_new, v)          // DAG-aware, Eq. (2)
        end if
    end for

    -- META-MONITORING --
    if SatisficingCheck(G) then break           // early stop
    AdaptCpuct(G, iteration)                    // adaptive exploration
end for

tau* <- ExtractBestTrajectory(G)
return tau*
```

Algorithm 2: Delta-Corrected DAG Backpropagation

```
ALGORITHM 2: DeltaBackprop
------------------------------------------------------------
Input:  graph G, node s, value v
Output: updated Q-values and visit counts in G

visited <- {}
stack <- [(s, v)]
while stack != {} do
    (current, val) <- stack.pop()
    if current in visited then continue
    visited <- visited + {current}
    current.visits <- current.visits + 1
    current.value_sum <- current.value_sum + val
    for each incoming edge e = (parent -> current) do
        old_q <- e.q_value
        e.visits <- e.visits + 1
        e.q_value <- old_q + (val - old_q) / e.visits  // incremental mean
        stack.push((parent, e.q_value))
    end for
end while
```

Algorithm 3: Semantic Transposition Detection

```
ALGORITHM 3: SemanticTranspositionCheck
------------------------------------------------------------
Input:  new node s_new, existing nodes S_d at depth d,
       embedding model E, threshold tau
Output: matching node or null

v_new <- E.encode(s_new.state_text)

// Tier 1: exact hash
h_new <- SHA256(s_new.env_snapshot || s_new.tool_state || s_new.depth)
for each s in S_d do
    if hash(s) = h_new then return s
```

```
7:  end for
8:
9:  // Tier 2: semantic embedding similarity
10: candidates <- ANN_search(v_new, {E.encode(s) : s in S_d}, top_k=5)
11: for each (s, similarity) in candidates do
12:     if similarity >= tau then
13:         if ConsistencyCheck(s_new, s) then
14:             return s
15:         end if
16:     end if
17: end for
18:
19: return null
```

Algorithm 4: PE-EMP Local Loop

```
ALGORITHM 4: PE-EMP Meta-Prompt Adaptation
------------------------------------------------------------
Input:  graph G, base system prompt P0, update interval I,
        LLM for principle extraction
Output: adapted system prompt P_current

1:  principles <- []
2:  for each search iteration i do
3:      if i mod I != 0 or i = 0 then
4:          P_current <- P0 + FormatPrinciples(principles[-10:])
5:          continue
6:      end if
7:
8:      // Select contrastive sibling pairs
9:      (high_nodes, low_nodes) <- SelectContrastivePairs(G)
10:
11:     // Extract principles via LLM analysis
12:     for each (h, l) in zip(high_nodes, low_nodes) do
13:         principle <- LLM.analyze(
14:             "Why did this reasoning score higher?",
15:             high=h.state, low=l.state,
16:             high_score=h.q_value, low_score=l.q_value
17:         )
18:         principles.append(principle)
19:     end for
20:
21:     P_current <- P0 + FormatPrinciples(principles[-10:])
22: end for
23: return P_current
```

Appendix E: Graph Schema

BELIEF_STATE (Node)

Table E-1: Belief-state node schema

Field	Type	Description
state_id	string	Unique identifier
observation_digest	string	Hash of the current observation
working_memory	string	Current working memory contents
latent_key	vector	Embedding for semantic transposition matching
visit_count	int	$N(s)$: times this node has been visited
depth	int	Step counter (prevents cycles in hash key)

ACTION_EDGE (Edge)

Table E-2: Action-edge schema

Field	Type	Description
edge_id	string	Unique identifier
parent_id	string	Source belief-state
child_id	string	Target belief-state
prior	float	$P(s,a)$ from LLM policy
q_value	float	$Q(s,a)$
n_visits	int	$N(s,a)$
cost_est	float	Estimated execution cost
risk_est	float	Estimated risk score

VALUE_ESTIMATE (Multi-Objective)

Table E-3: Multi-objective value estimate schema

Field	Type	Description
v_success	float	V_{succ}: task success probability
v_cost	float	V_{cost}: cumulative cost

Field	Type	Description
v_risk	float	V_{risk}: cumulative risk
uncertainty	float	Epistemic uncertainty

Entity Relationships

BELIEF_STATE --expands via--> ACTION_EDGE --produces--> BELIEF_STATE
ACTION_EDGE --evaluated by-> VALUE_ESTIMATE
BELIEF_STATE --retrieves----> MEMORY_ITEM (with provenance)
BELIEF_STATE --grounded in--> TRANSITION (tool execution record)
MEMORY_ITEM --linked to----> EPISODE_LINK (cross-session reference)

Appendix F: Design Alternatives and Trade-offs

Table F-1: Architectural design alternatives

Design Axis	Option A	Option B	Option C	Trade-offs
Search	Tree MCTS	DAG MCGS (exact)	DAG + semantic merging	Tree wastes compute; DAG reuses; semantic risks incorrect merges [9]
State repr.	Raw text	Structured belief + slots	Latent state (MuZero-like) [3]	Raw is flexible but low transposition rate; latent enables learning
World model	Real env only	LM simulation	Learned latent dynamics [3]	Real is accurate but expensive; LM hallucinates; latent needs data
Value fn.	Heuristics	Learned scalar V	Multi-objective V	Multi-objective supports safety but complicates training
Reward model	ORM	PRM [11]	PPM (self-rewarding) [12, 13]	ORM has credit assignment failure; PPM is self-supervised
Memory	None	RAG [17]	Dual-loop + provenance	Dual-loop enables cross-session learning; increases attack surface

Design Axis	Option A	Option B	Option C	Trade-offs
Hierarchy	Flat actions	Fixed macros	Learned options [18]	Options reduce depth; require discovery
Deployment	Monolithic	Library	RaaS (MCP/OpenAPI)	RaaS enables scaling + hot-swapping

Appendix G: Loss Functions and Hyperparameters

Loss Functions

Policy distillation (AlphaZero-style [1]):

$$\mathcal{L}_\pi = -\sum_a \pi\,(a \mid s)\log p_\theta(a \mid s)$$

Value regression:

$$\mathcal{L}_V = (V_\theta(s) - z)^2$$

Multi-objective utility:

$$U(s) = V_{\text{succ}}(s) - \lambda_c \cdot V_{\text{cost}}(s) - \lambda_r \cdot V_{\text{risk}}(s)$$

MuZero-style dynamics [3] (optional): predict r_{t+1} and next latent state h_{t+1}; predict policy/value from h_t.

Efficiency Strategies

- **Transposition reuse** — reduces duplicated computation [9].
- **Batched inference** — GPU utilization via batch MCTS evaluation.
- **Gumbel planning** [21] — improved policy under small budgets.
- **KataGo-style** [22] algorithmic improvements — ~50x compute reduction.
- **Adaptive compute** — dynamic budgets proportional to uncertainty [14].

Hyperparameters (Starting Ranges)

Table G-1: Hyperparameter ranges and trade-offs

Component	Parameter	Range	Trade-off
Search budget	Simulations / decision	16–512	Latency vs. coverage
pUCT	c_1, c_2	Near MuZero defaults	High -> over-explore; low -> greedy
Root exploration	Dirichlet α	Small; moderate mixing	Noisy targets vs. premature collapse
Proposal width	Top-k actions	4–64	Missing branches vs. cost blow-up

Component	Parameter	Range	Trade-off
Depth limit	Max depth	6–40	Cannot solve long tasks vs. explosion
Transposition thresh.	Cosine similarity	>= 0.92 (conservative)	Incorrect merges vs. wasted compute
Cost/risk weights	λ_c, λ_r	Small -> moderate	Over-penalize -> under-explore
PE-EMP interval	Update every N iterations	20–50	Instability vs. slow adaptation
Memory prune cutoff	Min Q-score for retention	0.15–0.25	Lose useful heuristics vs. stale knowledge

Appendix H: Ablation Experiments and Visualization Plan

Ablation Experiments

Table H-1: Planned ablation experiments

ID	Ablation	Conditions	Hypothesis
A1	**Tree vs. Graph**	MCTS tree vs. MCGS (exact) vs. MCGS + semantic	MCGS improves efficiency at fixed compute [9]
A2	**Value function**	Heuristic vs. scalar V vs. PRM vs. PPM vs. multi-objective	PRM improves credit assignment; PPM removes annotation need [11, 12]
A3	**World model**	Real env vs. LM simulation vs. learned latent [3]	Learned model increases efficiency
A4	**Hierarchy**	Flat vs. fixed macros vs. learned options [18]	Options reduce depth and improve long-horizon tasks
A5	**Memory**	None vs. RAG vs. episodic vs. full dual-loop	Dual-loop shows compounding cross-session gains [15, 16]
A6	**Deployment**	Embedded vs. RaaS (heterogeneous models)	RaaS achieves better cost/accuracy Pareto frontier

Visualization Plan

- Success rate vs. search budget curves (tree vs. graph vs. graph+semantic).
- Cost–performance Pareto plots (multi-objective value benefits).

- Transposition heatmaps (most frequently merged subgoal states).
- Value calibration curves stratified by task difficulty.
- Cross-session learning curves (dual-loop validation).

Appendix I: Resource Estimates and Team

Resource Scenarios

Table I-1: Hardware resource scenarios

Scenario	Hardware	Focus
Small prototype	1–4 GPUs	Inference-time MCGS; minimal distillation; batched MCTS
Medium lab	8–64 GPUs	PRM/PPM training; episodic memory; RaaS deployment
Large scale	TPU/GPU cluster	MuZero-style world model; multi-agent; large-scale self-play

Team Profile (Minimum Viable)

Table I-2: Minimum viable team

Role	Responsibilities
Search + RL Engineer	MCGS, bandits, parallelization, delta backprop, replay buffers
LLM Systems Engineer	Tool APIs, batching/caching, evaluation harness, RaaS packaging
IR / Memory Engineer	Embeddings, ANN indices, provenance, dual-loop memory system
Safety Engineer	Prompt injection defense, sandboxing, red-teaming, risk gating

Release Plan

Table I-3: Phased release plan

Version	Contents
v0	MCGS engine + logging + baselines
v1	PRM/PPM training + distillation + episodic memory
v2	Dual-loop learning + RaaS (FastAPI + MCP)
v3	Hierarchical options + partial observability

Version	Contents
v4	World-model learning (MuZero-style)

Safety artifacts (prompt injection test suites, safety-gating rules) released transparently, aligned with OWASP guidance [24].