Generative Gamer: Learning Equilibrium Strategy by LLM-driven Dynamic Deduction

Yadong Zhang, Xinshu Shen, Yupei Ren, Shangqing Zhao, Man Lan*
School of Computer Science and Technology, East China Normal University
{yadongzhang, xinshushen, ypren, sqzhao}@stu.ecnu.edu.cn
mlan@cs.ecnu.edu.cn

Abstract

Large Language Models (LLMs) have demonstrated remarkable general capabilities, yet they falter in domains requiring deep strategic reasoning. A primary obstacle is the need to navigate a game tree that grows exponentially with search depth, a task for which their generative nature is ill-suited. To address this, we introduce Generative Gamer (GenGamer), a framework that trains LLMs to reason like an expert player. Instead of attempting an exhaustive search, GenGamer learns to generate a compact, pruned reasoning trajectory termed as a Dynamic Deduction. This is achieved by integrating three key strategies: action pruning based on policy confidence, state pruning via value estimation, and branch pruning inspired by alpha-beta principles. Furthermore, to train the model effectively, we propose the Deduction Tree Reward (DTR), a process-oriented mechanism that provides step-by-step feedback on the quality of the reasoning process, rather than relying solely on the final game outcome. Experiments on complex games such as Tic-Tac-Toe and Leduc Poker demonstrate that GenGamer significantly enhances the strategic capabilities of LLMs, enabling them to achieve performance that surpasses current state-of-the-art language models.

1 Introduction

Large language models (LLMs), such as GPT(Achiam et al., 2023) and Llama(Dubey et al., 2024), have recently achieved impressive performance in diverse tasks, including natural language processing, code generation(Jimenez et al., 2023), and complex reasoning(Phan et al., 2025). Their strong contextual understanding and text generation capabilities allow for multi-step reasoning, planning(Xie et al., 2024), and "Chain-of-Thought" processes(Wei et al., 2022; Kojima et al., 2022), simulating aspects of human cognition. These advances have prompted interest in applying LLMs to more challenging decision-making tasks, with strategic games serving as a key benchmark(Zhang et al., 2024a). Unlike routine tasks, strategic games demand deep lookahead planning, recursive reasoning(Zhang et al., 2024b), and modeling of opponents(Southey et al., 2012).

Traditionally, strategic game solving has relied on algorithms like Minimax(Shannon, 1950) and Monte Carlo Tree Search (MCTS) (Browne et al., 2012), which use extensive search and simulation to find equilibrium strategies, i.e., optimal play under rational opponents. Such methods, exemplified by AlphaGo(Silver et al., 2016), depend on massive self-play and rapid, low-cost environment interactions. However, LLMs face fundamental limitations when applied to these search-intensive paradigms. Their autoregressive generation process has high inference latency, making the thousands of simulations per second required by MCTS infeasible. These limitations create a critical gap, highlighting the need for a new paradigm that leverages the innate reasoning strengths of LLMs without relying on external, high-frequency simulation.

To address this, we propose a new paradigm: **Dynamic Deduction Generation**. Instead of relying on traditional search, we train the LLM to directly generate structured text that simulates an expert's reasoning process. This structured text, which we term a **dynamic deduction tree**, represents a compact and pruned reasoning trajectory that culminates in the identification of an optimal action. The core challenge is to transform the iterative logic of algorithms like MCTS or Minimax into a single, generative pass. To control the tree size and avoid combinatorial explosion, we integrate three synergistic pruning strategies: **action pruning** based on the model's policy confidence, **state pruning** by

*Corresponding author.
Our code and dataset are publicly available at: https://github.com/cubenlp/gengamer

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12604–12617

dynamically terminating branches with evaluation confidence, and **branch pruning** inspired by the principles of alpha-beta search in Minimax algorithms(Knuth and Moore, 1975). This enables the LLM to generate a concise and logically coherent deductive process, from which the final decision is derived.

To train the LLM for this task, we adopt a two-stage training paradigm. First, during **Expert Deduction Imitation**, the model learns the fundamental syntax and structure of dynamic deduction trees, which are generated from an expert search process. Subsequently, the Deduction Process Optimization stage refines this foundational ability, sharpening the model's strategic acumen. This optimization is driven by our **Deduction Tree Reward** (DTR). As a process-oriented reward function(Lightman et al., 2023), DTR provides dense, node-level feedback on the precision of both action selection and value estimation throughout the entire reasoning tree.

To validate the effectiveness of our framework, we conduct comprehensive experiments on a suite of classic strategic games, including the perfect-information game Tic-Tac-Toe and the imperfect-information game Leduc Poker(Southey et al., 2012). We evaluate GenGamer's performance against several strong baselines, chief among them being state-of-the-art LLMs using direct prompting and models fine-tuned with conventional outcome-based supervision. These experiments result show GenGamer's ability to learn high-quality policies and demonstrate the superiority of process-oriented training.

In summary, our main contributions are:

1. Proposing a new paradigm that enables LLMs to independently solve strategic games by generating dynamic deduction trees, removing reliance on external search algorithms.

2. Introducing a two-stage training framework and the Deduction Tree Reward (DTR) to provide robust, process-oriented guidance for complex reasoning tasks.

3. Demonstrating through experiments on games such as Tic-Tac-Toe and Leduc Poker that our method significantly outperforms baselines, including state-of-the-art language models.

2 Related Work

Traditional Game AI: From Search to Self-Play. Early AI, inspired by Shannon's 1950 Chess paper(Shannon, 1950), relied on the Minimax algorithm: agents searched all moves assuming opponents played optimally. Alpha-Beta Pruning(Knuth and Moore, 1975) soon optimized this by ignoring moves guaranteed to be worse, allowing deeper searches. However, high-complexity games like Go made exhaustive search impractical; performance depended on expert-designed evaluation functions. Monte Carlo Tree Search (MCTS)(Browne et al., 2012) addressed this by using randomized simulations to approximate values, achieving greater depth efficiency. A breakthrough came with Deep Reinforcement Learning (DRL) (Sze et al., 2017) and self-play: AlphaGo (Silver et al., 2016) used neural networks guided by MCTS and data from human games. AlphaGo Zero(Silver et al., 2017) advanced further by learning solely via self-play, exceeding human capabilities and generalizing to Chess and Shogi. Recent systems like ReBeL(Brown et al., 2020) for Poker show that DRL combined with MCTS or tree search can solve even imperfect-information games, marking the pinnacle of traditional game AI.

Large Language Models for Strategic Reasoning. LLMs are being explored for strategic reasoning and planning, leveraging their linguistic generation abilities and pre-trained knowledge. Current methods for LLM-based planning include external module augmentation, fine-tuning, and search-based approaches(Zhang et al., 2024a). However, LLMs struggle with long-term planning due to their lack of internal world models, making them less effective in traditional game scenarios requiring systematic and tactical exploration. Therefore, LLMs face limitations in achieving optimal strategic performance, as linguistic reasoning doesn't always align with algorithmic optimality(Jia et al., 2025). Comparisons with traditional solvers reveal that LLMs perform best in games involving human-like uncertainty. Hybrid approaches, like DipLLM(Xu et al., 2025) for Diplomacy(, FAIR), show promise in achieving state-of-the-art strategic decision-making by fine-tuning LLMs with specialized policies, demonstrating the potential for combining generative models with mathematical search capabilities.

3 Method: GenGamer

3.1 Problem Formulation

Game Environment. We model strategic games as a Partially Observable Markov Decision Pro-

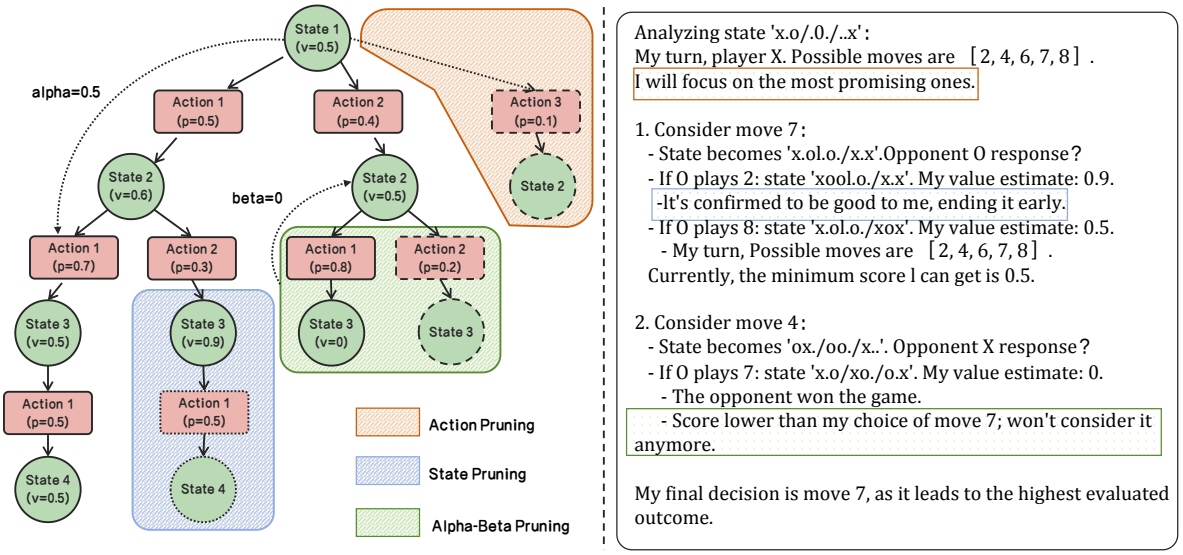

Figure 1: A visual representation of how the Dynamic Deduction Tree is constructed. The left side depicts the theoretical game tree with pruning boundaries, while the right side shows the actual text generated by GenGamer.

cess **(POMDP)**(Huh and Mohapatra, 2023), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{T}$ the transition function, $\mathcal{R}$ the reward function, and $\mathcal{O}$ the observation set. For perfect information games (e.g., Tic-Tac-Toe), this simplifies to a Markov Decision Process **(MDP)**, where the state $s \in \mathcal{S}$ is fully observable. The goal of agent is to learn an optimal policy π^* that selects actions $a \in \mathcal{A}$ based on the state (or observation) s to maximize the expected cumulative return:

$$\pi^* = \underset{\pi}{\text{argmax}}\ \mathbb{E}\left[\sum_t \gamma^t r_t\right] \tag{1}$$

where γ is the discount factor.

Policy as Deduction Tree Generation. Unlike conventional policies $\pi(a|s)$ that output action distributions, our policy $\pi(\tau|s;\theta)$, parameterized by an LLM with parameters θ, generates a **Dynamic Deduction Tree (DDT)** as reasoning content τ.

Definition 1: Dynamic Deduction Tree (DDT). A DDT τ is a serialized text representation of a tree-structured reasoning process $\mathcal{T}$. Each node $n \in \mathcal{T}$ consists of:

- s_n: The game state at node n (or observation $o_n \in \mathcal{O}$ in POMDPs).
- V_n: The LLM-estimated value $V(s_n;\theta)$. For terminal nodes, this represents the final game return G. For chance nodes (e.g., card dealing in poker), it represents the expected value over all probabilistic outcomes.
- A'_n: The pruned action subset, $A'_n \subseteq \mathcal{A}(s_n)$.
- R_n: The returned value from child nodes C_n, propagated upwards following Minimax logic.
- C_n: The set of child nodes reached by taking actions $a \in A'_n$, representing the state transitions $\mathcal{T}(s_n, a)$.

Policy Selection. After generating the full DDT τ, the final action a^* is chosen from the root's children to maximize the returned value. This action selection process follows the **minimax algorithm**, where the agent assumes the opponent plays optimally and chooses the move leading to the best possible outcome. A visual demonstration of this backpropagation and selection is provided in Figure 1.

$$a^* = \underset{a \in A'_{n_{\text{root}}}}{\text{argmax}}\ R_{n_{\text{child}}(a)} \tag{2}$$

where $n_{\text{child}}(a)$ is the child node reached after taking action a.

3.2 Dynamic Deduction Tree Construction

To efficiently explore the game space, we design a recursive DDT generation algorithm that transforms the exploration of a **game tree** into a controlled, serialized text generation task. To generate the ground-truth data for this task, we utilize an expert model, such as MCTS or an AlphaGo-style algorithm, to produce optimal reasoning trajectories.

The construction is guided by **three synergistic pruning strategies**:

- **Action Pruning via Cumulative Probability Sampling:** To form a reduced action set A'_n, we perform sampling from the policy distribution of the expert model. We iteratively draw actions and add them to A'_n until their cumulative probability exceeds a predefined threshold `top_p`.
- **State Pruning based on Search Heuristics:** We employ three forms of heuristic state pruning: **depth-based pruning**, which halts expansion once the tree reaches a predefined `max_depth`; **deterministic pruning**, where branches are terminated if the absolute value of their estimated value $|V_n|$ exceeds a `certainty_threshold` (indicating a decisive outcome); and **stability pruning**, which stops exploration if the value difference between a child node and its parent falls below a `stability_threshold`, signifying converging returns.
- **Alpha-Beta Pruning:** We propagate α (the maximizing player's best-found outcome) and β (the minimizing player's best-found outcome) during DDT construction and prune any branch that is provably suboptimal according to minimax principles. This preserves the equilibrium strategy while significantly reducing the search space.

By integrating these complementary strategies, we dramatically reduce the number of nodes required in the DDT, enabling deep, high-quality reasoning within a manageable generation length. The full recursive process is detailed in Algorithm 1. For imperfect-information games, we further extend this to an expectiminimax variant (Algorithm 2 in Appendix C).

3.3 Two-Stage Training Paradigm

3.3.1 Stage 1: Expert Deduction Imitation

The initial stage teaches the LLM to imitate an expert's deductive reasoning process. To achieve this, we use an expert system (e.g., Alpha-Go style algorithm) to generate a dataset $\mathcal{D}_{\text{SFT}}$ of "golden" deduction trees τ^* for various game states s. The LLM is then trained via supervised learning to maximize the conditional log-likelihood of these expert-generated trees, effectively learning the syntax and structure of high-quality reasoning. The loss function is defined as:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(s,\tau^*)\in\mathcal{D}_{\text{SFT}}}\left[\log P(\tau^*|s;\theta)\right] \quad (3)$$

where $\tau^* = (y_1, y_2, \ldots, y_{|\tau^*|})$ is the sequence of tokens in the expert tree.

Algorithm 1 Dynamic Deduction Tree Construction

Require: state s, remaining depth d_{rem}, alpha α, beta β, expert model θ
Ensure: deduction subtree τ_s, returned value R_s

$V_s \leftarrow V(s;\theta)$
if $|V_s| >$ `certainty_thresh` **or** ($V_{\text{parent}} \neq$ None **and** $|V_s - V_{\text{parent}}| <$ `stability_thresh`) **then**
  {State Pruning: Node is certain or stable}
  $\tau_s \leftarrow$ `format_leaf`(s, V_s)
  return τ_s, V_s
end if
$P(A|s) \leftarrow$ `softmax`(`logits`$(s;\theta)$)
$A' \leftarrow$ `NucleusSampling`$(A(s), P(A|s), p)$ {Action Pruning}
$\tau_s \leftarrow$ `format_node_header`(s, V_s)
Initialize `children_results` $= \{\}$
Set $R_s \leftarrow -\infty$ if maximizing player else $+\infty$

for each action a in A' **do**
  $s' \leftarrow$ `transition`(s, a)
  $\tau_{\text{child}}, R_{\text{child}} \leftarrow$ `GenerateDDT`$(s', d_{\text{rem}} - 1, \alpha, \beta, \theta)$
  `children_results`$[a] \leftarrow (\tau_{\text{child}}, R_{\text{child}})$
  if maximizing player **then**
    $R_s, \alpha \leftarrow \max(R_s, R_{\text{child}}), \max(\alpha, R_s)$
  else
    $R_s, \beta \leftarrow \min(R_s, R_{\text{child}}), \min(\beta, R_s)$
  end if
  if $\beta \leq \alpha$ **then**
    break {Alpha-Beta Pruning}
  end if
end for
Append formatted children τ_{child} to τ_s using `children_results`
Append formatted return value R_s to τ_s
return τ_s, R_s

3.3.2 Stage 2: Deduction Process Optimization

Building upon the imitator model, we then optimize the precision of its deductive capabilities. This is achieved via Group Relative Policy Optimization (GRPO, Shao et al.), guided by a composite **Deduction Tree Reward (DTR)**. This reward function combines dense node-level feedback with a final decision assessment:

- **Value Reward (r_n^V):** For *every node* $n \in \tau$, we reward value estimates V_n that are close to the oracle target value V_n^*. The reward is calculated using an exponential kernel: $r_n^V = \exp(-(V_n - V_n^*)^2)$.

- **Policy Reward (r^P):** For the *root node* (representing the final decision), we measure the quality of the selected action a_{root} by normalizing its oracle value with respect to the best and worst possible actions in the initial state:

$$r^P = \frac{V^*(s_{\text{root}}, a_{\text{root}}) - \min_{a'} V^*(s_{\text{root}}, a')}{\max_{a'} V^*(s_{\text{root}}, a') - \min_{a'} V^*(s_{\text{root}}, a')}$$

The core of GRPO is to compute a normalized reward $\tilde{r}_i$ for each tree, which then serves as the advantage estimate for its entire token sequence:

$$\hat{A}(\tau_i) = \tilde{r}_i = \frac{R_i - \mu_R}{\sigma_R + \epsilon}, \quad R_i = r^V + r^P \quad (4)$$

where μ_R and σ_R are the mean and standard deviation of rewards in the group sampled from state s, and ϵ is for stability. The policy is then updated using the PPO-clip objective(Schulman et al., 2017):

$$\mathcal{L}_{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t \left[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1 \pm \epsilon)\hat{A}_t) \right] \quad (5)$$

where $\rho_t(\theta) = \frac{\pi_\theta(y_t|y_{<t},s)}{\pi_{\theta_{old}}(y_t|y_{<t},s)}$ is the importance sampling ratio.

4 Experiments

4.1 Environments

To comprehensively evaluate the effectiveness and generalization capability of our proposed **Generative Gamer (GenGamer)** method, we conduct experiments on four representative benchmarks covering board games, imperfect information games, and mathematical strategy games.

- **Tic-Tac-Toe**: A classic two-player, turn-based, perfect information board game with simple rules and a manageable action space.

- **Connect Four**: A two-player connection game where players take turns dropping colored discs into a grid. The objective is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs, providing a more complex strategic challenge than Tic-Tac-Toe.

- **Leduc Poker**: A two-player, limit poker variant with hidden information and stochastic outcomes. The complexity of Leduc Poker offers a challenging scenario to test the method's performance in imperfect information environments.

- **Nim**: A mathematical strategy game where players take turns removing objects from distinct piles. The player who takes the last object typically loses.

4.2 Metrics

For **Tic-Tac-Toe**, **Connect Four**, and **Nim**, we report the *average win rate* of the agent when playing against a standardized opponent, calculated over 100 evaluation games. Win rate is computed as the proportion of games won over the total number of games played, reflecting the agent's strategic proficiency in deterministic, perfect-information settings. For **Leduc Poker**, which involves imperfect information and stochastic outcomes, we assess performance using the *average number of chips won*, calculated across 100 evaluation hands.

In addition to outcome-based metrics, we further analyze the *optimal coverage* of player decisions. Specifically, for each decision point in a game, we compare the action selected by our agent to that of an expert model.

4.3 Implementation Details

For all experiments, we adopt **Qwen2.5-3B-Instruct** (Qwen et al., 2025) as the base language model, jointly trained across all four games. The entire training pipeline, encompassing both Expert Deduction Imitation (Stage 1) and Deduction Process Optimization (Stage 2), is executed using the **veRL** framework. All experiments are conducted on 4 NVIDIA RTX 3090 GPUs. The data utilized for both SFT and RL stages are listed in Table 1. Notably, the expert oracle is only used during training; at inference time, GenGamer operates without any external solver or environment interaction.

Environment	Expert Model	SFT Data Size	RL Data Size
TicTacToe	MCTS	2037 frames	377 games
Connect Four	Alpha-Zero	2502 frames	446 games
Leduc Poker	NFSP	978 frames	179 games
Nim	MCTS	2322 frames	334 missions

Table 1: The training and evaluation data sizes for each environment.

	TicTacToe		Connect Four		Leduc Poker		Nim	
	Win Rate	Optimal Coverage	Win Rate	Optimal Coverage	Avg Score	Optimal Coverage	Win Rate	Optimal Coverage
Base Language Model (zero-shot or few-shot)								
Qwen 2.5-3B-Instruct	0.19	26.7%	0.34	17.0%	-0.71	43.1%	0.59	32.2%
Deepseek R1	0.58	65.7%	0.82	43.8%	0.39	60.3%	0.66	63.4%
GPT-4o	0.33	46.1%	0.26	18.6%	-0.55	57.5%	0.52	20.0%
Claude 3.5 Sonnet	0.42	40.8%	0.61	22.0%	1.21	60.9%	0.34	32.0%
Gemini 2.5 Pro	0.60	70.2%	0.82	41.1%	0.41	65.7%	0.81	57.3%
Reinforcement Learning / Searching Method/ Rule								
Random	0.24	18.3%	0.28	19.7%	-1.52	37.2%	0.62	19.0%
Expert Model	0.77	72.3%	1.00	51.3%	2.00	92.5%	1.00	95.7%
Supervised Training for Language Model (Based on Qwen 2.5-3B-Instruct)								
Behavior Cloning	0.40	45.4%	0.45	26.5%	-0.08	44.5%	0.42	26.2%
GRPO	0.47	38.4%	0.55	15.5%	0.07	65.2%	0.48	22.2%
GenGamer (Ours)	**0.57**	**56.5%**	**0.75**	**43.7%**	**1.32**	**79.2%**	**0.79**	**65.2%**
	(+0.38)	(+29.8%)	(+0.41)	(+26.7%)	(+2.03)	(+36.1%)	(+0.20)	(+33.0%)

Table 2: Overall performance comparison across four game environments, reporting Win Rate (or Average Score for Leduc Poker) and Optimal Coverage averaged over 100 evaluation games. Underlined results are the best among base language models, and **bold** results are the best among supervised training methods.

The core hyperparameters governing the construction of Deduction Trees are kept consistent across all games to ensure a fair comparison. These include a `certainty_threshold` of 0.9, a `stability_threshold` of 0.1, a `max_depth` of 3, and a `top_p` for pruning of 0.7. The specific configurations and training parameters for expert models, along with the detailed hyperparameters for our GenGamer's two training stages, are provided in the Appendix for reproducibility.

For evaluation, we employ the significantly larger **Qwen2.5-72B-Instruct** as the standardized opponent. For baselines, Behavior Cloning fine-tunes the model to directly output optimal actions. The GRPO baseline uses a sparse binary reward, assigning 1 if the action matches the expert's choice and 0 otherwise.

4.4 Main Result

The overall performance of GenGamer compared to baseline models across the four game environments is presented in Table 2. The empirical results provide strong evidence for the effectiveness of our proposed framework.

First and foremost, GenGamer achieves a dramatic performance improvement over its base model, Qwen 2.5-3B-Instruct. In deterministic settings like Tic-Tac-Toe (0.19 to 0.57) and Connect Four (0.34 to 0.75), the win rates more than double. The improvement is even more pronounced in the imperfect-information game of Leduc Poker, where GenGamer transforms a losing policy (Average Score -0.71) into a highly profitable one (1.32), and in Nim, where the win rate improves from 0.59 to 0.79.

GenGamer consistently outperforms traditional methods. *Behavior Cloning* often fails to capture strategic depth (e.g., -0.08 in Leduc), while *GRPO* struggles with sparse outcome rewards (e.g., 0.48 in Nim). This substantial performance gap underscores the necessity of our tree-structured process reward mechanism for instilling robust strategic reasoning where standard baselines fail.

A key observation is the strong correlation between *Optimal Coverage* and outcome metrics (Win Rate/Score). GenGamer does not merely achieve successful outcomes through superficial heuristics; it learns to mimic expert logic. For instance, in Leduc Poker, GenGamer achieves a remarkable 79.2% Optimal Coverage, directly corresponding to its dominant average score of 1.32. Similarly, in Nim, the model matches the optimal move 65.2

Finally, it is noteworthy that GenGamer, built upon the relatively small 3B-parameter Qwen 2.5, demonstrates performance that is competitive with—and often superior to—state-of-the-art models orders of magnitude larger. In the logic game

Nim, GenGamer (0.79) surpasses GPT-4o (0.52) and Deepseek R1 (0.66). While larger models like Deepseek R1 and Gemini 2.5 Pro show strength in Connect Four, GenGamer remains highly competitive (0.75 vs. 0.82) despite its compact size. This indicates that our specialized reasoning framework can effectively distill deep strategic capabilities into smaller, efficient models, enabling them to punch well above their weight class without relying on massive parameter counts.

4.5 Ablation Studies

4.5.1 Pruning Strategies

To validate the effectiveness and necessity of our proposed pruning strategies, we conducted an ablation study analyzing their cumulative impact on both the size of the Dynamic Deduction Tree (DDT) . The results are summarized in Figure 2.

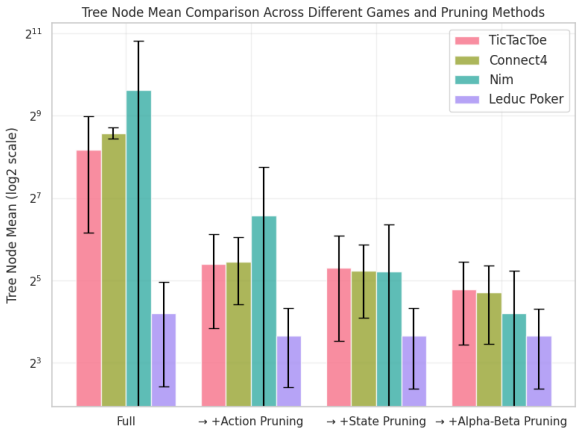

Figure 2: The number of nodes in the DDT for different pruning strategies.

Our first key finding is that the layered pruning mechanism is critical for making LLM-based tree generation feasible. Without any pruning, a game tree of even moderate depth can explode combinatorially, easily exceeding 2^9 (512) nodes. Generating such a large structure, where each node requires a textual description of its state, value estimate, and potential actions, is computationally prohibitive and often exceeds the context window of modern LLMs. As shown in our experiments, applying Action Pruning first dramatically reduces the tree's breadth. The subsequent addition of State Pruning further curtails the exploration of unpromising branches. Finally, integrating Alpha-Beta Pruning efficiently removes provably suboptimal subtrees. Together, these strategies reduce the average number of nodes in the DDT to a manageable range of 2^4 to 2^5 (i.e., 16-32 nodes). This order-of-magnitude reduction is precisely what makes the generative deduction paradigm practical. We provide the detailed token length distributions of the deduction trees generated during game-play in Appendix D.

Crucially, this significant reduction in tree size does not come at the cost of decision-making accuracy. Even after applying all pruning strategies, the optimal action coverage remains remarkably high and experiences only a negligible drop compared to reasoning over a much larger, less pruned tree.

4.5.2 Training Pipeline

Impact of Training Strategies. As shown in Table 3, pure **GRPO** struggles with sparse rewards (e.g., 22.2% in Nim), whereas Deduction Imitation (Stage 1) provides a solid foundation. While adding GRPO improves performance, **GenGamer** (Deduction Imitation + DTR) consistently outperforms the Deduction Imitation + GRPO baseline across all environments (e.g., 79.2% vs. 72.9% in Leduc). This confirms that dense, process-oriented supervision is far more effective than sparse outcome signals for aligning the model with expert reasoning.

Effect of Reasoning Depth. Results reveal a non-linear trend regarding search depth. While *Depth=1* sets a reasonable baseline, extending to *Depth=2* surprisingly degrades performance (e.g., TicTacToe drops to 44.9%), likely due to the unresolved complexity of anticipating counter-moves. However, extending to **Depth=3** (GenGamer) reverses this, achieving the highest coverage. This indicates that a sufficient planning horizon is strictly necessary to resolve intermediate strategic ambiguities and ensure stability.

	Variant	TicTacToe	Connect Four	Leduc Poker	Nim
Training Pipeline	Deduction Imitation	43.3	37.2	61.0	51.9
	GRPO	38.4	15.5	65.2	22.2
	Deduction Imitation + GRPO	56.1	39.4	72.9	60.7
	GenGamer	56.5	43.7	79.2	65.2
Deduction Depth	depth=1	56.0	39.0	72.5	62.1
	depth=2	44.9	35.1	68.5	44.2

Table 3: Ablation studies on the training pipeline and deduction depth.

Training Dynamics and Stability Figure 3 illustrates the comparative training dynamics of GenGamer. First, our method exhibits a vastly superior initialization compared to pure *GRPO*, effectively bypassing the inefficient "cold start" phase

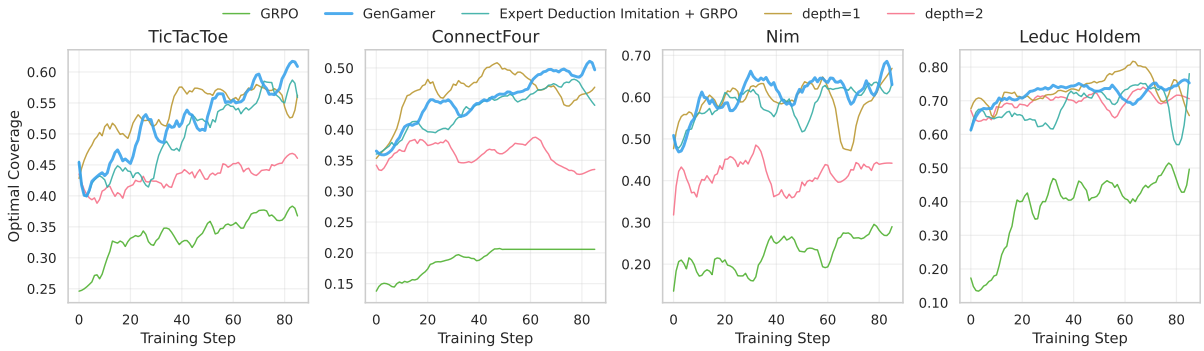

Figure 3: Evolution of decision accuracy across four game environments, comparing the training dynamics of GenGamer against alternative baselines and reasoning depth variants.

of learning from scratch. Second, relative to the *Deduction Imitation + GRPO* baseline, GenGamer demonstrates significantly enhanced stability and higher peak performance. The dense, node-level feedback provided by the Deduction Tree Reward prevents the volatility and premature plateaus often caused by sparse outcome supervision. Finally, regarding reasoning depth, while shallower models (e.g., *depth=1*) converge rapidly but saturate early, GenGamer leverages its deeper reasoning structure to sustain growth momentum throughout training, ultimately converging to a much higher performance ceiling.

4.6 Ability

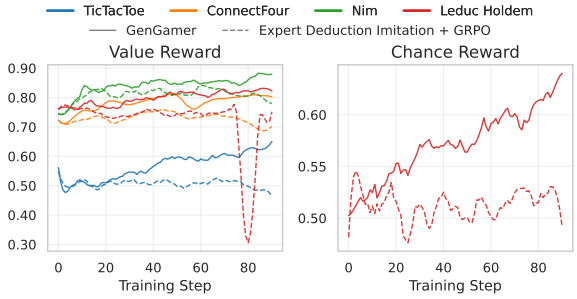

Figure 4: Progression of Value and Chance rewards throughout the training process, contrasting GenGamer with the Expert Deduction Imitation + GRPO baseline.

To demonstrate that GenGamer's performance stems from a genuine enhancement in strategic understanding rather than superficial reward hacking, we analyze the evolution of its internal reasoning metrics in Figure 4.

Value Estimation Stability. The *Value Reward* curves (Left) track the model's ability to accurately evaluate game states. GenGamer (solid lines) exhibits a consistent, monotonic improvement across all environments, indicating that the model progressively refines its understanding of the game's landscape. In stark contrast, the *Deduction Imitation + GRPO* baseline (dashed lines) suffers from significant volatility. Notably, in the complex Leduc Holdem environment, the baseline experiences a catastrophic collapse in value estimation accuracy during the later stages of training. This suggests that without the dense supervision of our Deduction Tree Reward, the baseline struggles to ground its evaluations, leading to unstable and forgetful learning dynamics.

Probabilistic Reasoning under Uncertainty. The *Chance Reward* (Right) specifically highlights the model's capability in the imperfect-information setting of Leduc Poker. This metric reflects the agent's proficiency in handling stochastic nodes—specifically, estimating the probability distribution of the opponent's hidden cards and the dealing of public cards. GenGamer shows a strong, steady upward trend, confirming that it is effectively learning to model uncertainty and infer hidden information. Conversely, the baseline fluctuates noisily without clear improvement, indicating a failure to grasp the underlying probabilistic mechanics.

Together, these results confirm that GenGamer does not merely memorize winning moves but constructs a robust internal model of both state values and environmental probabilities.

5 Conclusion

In this work, we introduced GenGamer, a novel framework for teaching strategic reasoning to Large Language Models. By employing a process-oriented Deduction Tree Reward (DTR) mechanism, we moved beyond sparse, outcome-based

signals to reward the quality of the reasoning process itself. Our empirical results demonstrate that GenGamer not only surpasses state-of-the-art baselines in both perfect and imperfect information games but also achieves remarkable sample efficiency. By effectively translating iterative search algorithms into a single-pass generative paradigm, we bridge the gap between classical game theory and modern language modeling. Ultimately, this framework offers a promising blueprint for enhancing "System 2" reasoning in LLMs, paving the way for autonomous agents capable of complex, multi-step planning.

Limitations

Despite the promising results of GenGamer in strategic reasoning, we acknowledge three primary limitations that outline directions for future research:

Inference Latency and Computational Cost. Unlike standard policy networks that output an action directly in a single forward pass, GenGamer requires generating a complete Dynamic Deduction Tree (DDT) as a textual sequence before making a decision. Although our pruning strategies significantly reduce the token count, the inference process still involves generating hundreds to thousands of tokens per move. This results in higher latency and computational cost compared to "fast-thinking" heuristic agents, potentially limiting the framework's applicability in real-time environments where millisecond-level response times are critical.

Scalability to High-Complexity Games. Our current implementation relies on the LLM's context window to store the serialized reasoning tree. While effective for games like Tic-Tac-Toe, Connect Four, and Leduc Poker, scaling this approach to games with massive branching factors or extreme depths (e.g., Go or Chess) remains a challenge. Even with aggressive pruning, the textual representation of a search tree for such games may exceed the context limits of current LLMs. Future work may need to explore hybrid representations or recurrent state-space models to handle deeper searches without combinatorial explosion in context length.

Dependence on Expert Oracles for Supervision. The success of our two-stage training pipeline—specifically the initial Expert Deduction Imitation phase—relies on the availability of a high-quality oracle (e.g., MCTS or a game-theoretic solver) to generate ground-truth deduction trees. For games where such solvers are computationally intractable or non-existent (e.g., complex open-ended negotiation games or real-world strategic scenarios), acquiring the necessary "golden" reasoning trajectories for SFT is difficult. Reducing this reliance on external experts and enabling the model to self-discover deduction structures from scratch remains an open challenge.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. 2020. Combining deep reinforcement learning and search for imperfect-information games. *Advances in neural information processing systems*, 33:17057–17069.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. 2024. Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations. *arXiv preprint arXiv:2402.12348*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, and 1 others. 2022. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074.

Johannes Heinrich and David Silver. 2016. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*.

Dom Huh and Prasant Mohapatra. 2023. Multi-agent reinforcement learning: A comprehensive survey. *arXiv preprint arXiv:2312.10256*.

Jingru Jia, Zehua Yuan, Junhao Pan, Paul E McNamara, and Deming Chen. 2025. Large language model strategic reasoning evaluation through behavioral game theory. *arXiv preprint arXiv:2502.20432*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Donald E Knuth and Ronald W Moore. 1975. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, and 1 others. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Claude E Shannon. 1950. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and 1 others. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and 1 others. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Finnegan Southey, Michael P Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. 2012. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411*.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Kaixuan Xu, Jiajun Chai, Sicheng Li, Yuqian Fu, Yuanheng Zhu, and Dongbin Zhao. 2025. Dipllm: Fine-tuning llm for strategic decision-making in diplomacy. *arXiv preprint arXiv:2506.09655*.

Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Adrian de Wynter, Yan Xia, Wenshan Wu, Ting Song, Man Lan, and Furu Wei. 2024a. Llm as a mastermind: A survey of strategic reasoning with large language models. *arXiv preprint arXiv:2404.01230*.

Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Yan Xia, Man Lan, and Furu Wei. 2024b. K-level reasoning with large language models. *arXiv preprint arXiv:2402.01521*.

A Implementation Details

Model Architecture and Training Infrastructure We employ Qwen2.5-3B-Instruct (Qwen et al., 2025) as our base language model, trained on 4 NVIDIA GPUs using Fully Sharded Data Parallel (FSDP) strategy with parameter and optimizer offloading enabled. The training pipeline consists of two stages: supervised fine-tuning (SFT) and reinforcement learning (RL) fine-tuning.

Supervised Fine-Tuning Stage In the SFT stage, we train the model on behavioral cloning data from multiple game domains including Tic-Tac-Toe, Connect Four, Leduc Hold'em, and Nim. The SFT training uses:

- Training batch size of 24 with micro-batch size of 1 per GPU
- Maximum sequence length of 4,096 tokens with right truncation
- Total of 1 epoch with evaluation every 32 steps
- AdamW optimizer with gradient checkpointing enabled

Reinforcement Learning Stage For the RL stage, we implement a Group Relative Policy Optimization (GRPO) approach (Shao et al., 2024) with the following configuration:

- **Data Configuration**: Training batch size of 12, generation batch size of 36 with $n = 8$ samples per prompt
- **Prompt and Response**: Maximum prompt length of 1,024 tokens and maximum response length of 3,072 tokens
- **Actor Model**: Learning rate of 1×10^{-6} with 10-step linear warmup, weight decay of 0.1, and gradient clipping at 1.0
- **PPO Parameters**: Mini-batch size of 32, clip ratio of 0.2, and 1 PPO epoch per iteration
- **Generation**: We use vLLM (Kwon et al., 2023) for efficient rollout generation with temperature of 1.0, top-p sampling of 1.0, and 8 responses per prompt during training (reduced to 1 during validation with top-p of 0.7)

Reward Function We adopt the Deduction Tree Reward (DTR) described in Section 3.3.2, which combines a node-level Value Reward r_n^V and a root-level Policy Reward r^P. To enable automated parsing of the generated deduction tree and per-node reward computation, the SFT data incorporates explicit structural markers: special tokens `[VALUE]` and `[ACTION]` distinguish different types of information within each node, and each line is prefixed with a depth indicator to denote the tree level.

Training Stability To ensure training stability, we employ: (1) gradient clipping at 1.0 for actor, (2) dynamic batch sizing with maximum token length of 4,096 per GPU, and (3) parameter offloading to manage memory constraints.

B Expert Model Configurations

We describe the configurations of the expert models used to generate ground-truth deduction trees for the Expert Deduction Imitation stage.

MCTS (Tic-Tac-Toe & Nim) We use a standard Monte Carlo Tree Search with UCT selection[1]. The key parameters are: rollout count of 10, UCT exploration constant $c = 2$, maximum simulations of 1,000 per move.

AlphaFour (Connect Four) For Connect Four, we adopt a pre-trained AlphaZero-style agent[2] with the following architecture and parameters:

- Network: 8 residual blocks with 128 convolutional channels, a policy head (7 actions) and a value head with Tanh activation.
- MCTS iterations: 100, $c_{\text{puct}} = 2.0$.
- Dirichlet noise: $\epsilon = 0.1$, $\alpha = 1.4$.
- Learning rate: 0.001.

NFSP (Leduc Hold'em) For Leduc Hold'em, we train a Neural Fictitious Self-Play[1] (Heinrich and Silver, 2016) agent with the following configuration:

- Hidden layers: [128], optimizer: SGD, loss: MSE.
- RL learning rate: 0.01, SL learning rate: 0.01.

[1] https://github.com/google-deepmind/open_spiel

[2] https://github.com/lucasBertola/Connect-4-Gym-env-Reinforcement-learning

- Anticipatory parameter: 0.1, batch size: 1, learn every: 1 step.
- Target network update every: 19,200 steps, discount factor: 1.0.
- ϵ-greedy: start 0.06, end 0.001, decay duration 2×10^7 steps.
- Evaluation metric: Nash convergence.

C Expectiminimax DDT Construction

For games involving chance events (e.g., card dealing in Leduc Poker), we extend the DDT construction algorithm to handle chance nodes using the expectiminimax principle. The key difference from Algorithm 1 is the addition of a chance node branch, where the returned value is computed as the expectation over all stochastic outcomes rather than a min/max operation.

The chance node branch (lines 9–16) computes the expected value $R_s = \sum_{o \in \mathcal{O}} p_o \cdot R_{\text{child}}(o)$ over all stochastic outcomes, while decision nodes (lines 17–31) retain the standard minimax logic with alpha-beta pruning. Note that alpha-beta pruning is only applied at decision nodes, as chance nodes require evaluating all outcomes to compute the correct expectation.

D Token Length Distribution

Figure 5 presents the token length distributions of the deduction trees generated by the trained model during game-play. Across all four games, the mean token lengths fall within a narrow range of approximately 1,100 to 1,300 tokens, with Nim exhibiting the shortest outputs (mean≈1,149) and Leduc Hold'em the longest (mean≈1,279). Notably, the vast majority of generated sequences remain well below the maximum response length of 3,072 tokens, indicating that the pruning strategies effectively constrain the deduction tree size during inference.

E Game System Prompts

Below we present the system prompts used to describe the game rules and action format to the model during game-play. These prompts are adapted from GTBench (Duan et al., 2024).

Algorithm 2 Expectiminimax DDT Construction

Require: state s, remaining depth d_{rem}, alpha α, beta β, expert model θ
Ensure: deduction subtree τ_s, returned value R_s

1: $V_s \leftarrow V(s;\theta)$
2: **if** $|V_s| >$ `certainty_thresh` **or** ($V_{\text{parent}} \neq$ None **and** $|V_s - V_{\text{parent}}| <$ `stability_thresh`) **then**
3: $\tau_s \leftarrow$ `format_leaf`(s, V_s)
4: **return** τ_s, V_s
5: **end if**
6: $\tau_s \leftarrow$ `format_node_header`(s, V_s)
7: Initialize `children_results` $= \{\}$
8: **if** `is_chance_node`(s) **then**
9: $\mathcal{O} \leftarrow$ `get_outcomes`(s) with probabilities $\{p_o\}_{o \in \mathcal{O}}$
10: $R_s \leftarrow 0$
11: **for** each outcome o in $\mathcal{O}$ **do**
12: $s' \leftarrow$ `apply_outcome`(s, o)
13: $\tau_{\text{child}}, R_{\text{child}} \leftarrow$ `GenerateDDT`$(s', d_{\text{rem}} - 1, \alpha, \beta, \theta)$
14: `children_results`$[o] \leftarrow (\tau_{\text{child}}, R_{\text{child}})$
15: $R_s \leftarrow R_s + p_o \cdot R_{\text{child}}$
16: **end for**
17: **else**
18: $P(A|s) \leftarrow$ `softmax`(`logits`$(s;\theta)$)
19: $A' \leftarrow$ `NucleusSampling`$(A(s), P(A|s), p)$
20: Set $R_s \leftarrow -\infty$ if maximizing player else $+\infty$
21: **for** each action a in A' **do**
22: $s' \leftarrow$ `transition`(s, a)
23: $\tau_{\text{child}}, R_{\text{child}} \leftarrow$ `GenerateDDT`$(s', d_{\text{rem}} - 1, \alpha, \beta, \theta)$
24: `children_results`$[a] \leftarrow (\tau_{\text{child}}, R_{\text{child}})$
25: **if** maximizing player **then**
26: $R_s, \alpha \leftarrow \max(R_s, R_{\text{child}}), \max(\alpha, R_s)$
27: **else**
28: $R_s, \beta \leftarrow \min(R_s, R_{\text{child}}), \min(\beta, R_s)$
29: **end if**
30: **if** $\beta \leq \alpha$ **then**
31: **break** {Alpha-Beta Pruning}
32: **end if**
33: **end for**
34: **end if**
35: Append formatted children τ_{child} to τ_s using `children_results`
36: Append formatted return value R_s to τ_s
37: **return** τ_s, R_s

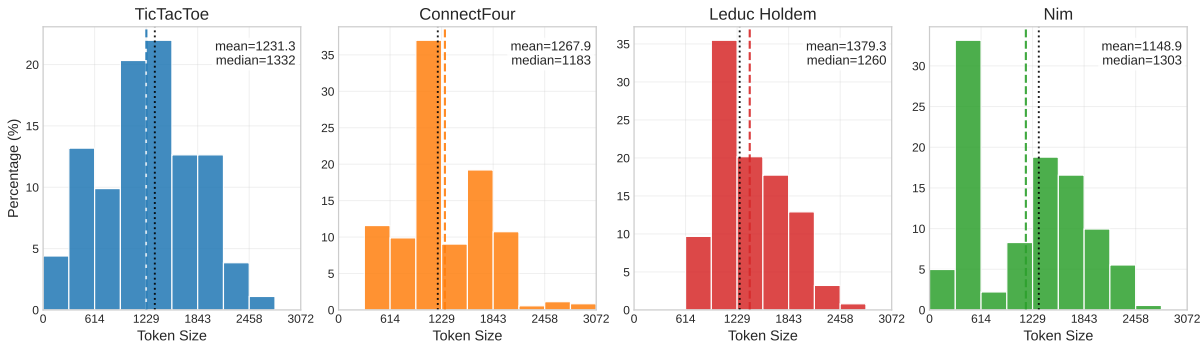

Figure 5: Distribution of token lengths for the deduction trees generated by the model during game-play across different games.

Tic-Tac-Toe System Prompt

Tic Tac Toe is a two-player game played on a grid. Players take turns marking a space with their respective symbols. The goal is to get 3 of one's own symbols in a row, either horizontally, vertically, or diagonally, before the opponent does. If all nine squares are filled and no player has three in a row, the game is a draw. The Tic Tac Toe game is played on a 3 by 3 grid, with the winning length as 3.
Each move is represented by a string consisting of two parts: the column (C) and the row (R), in that order. For instance, <C1R2> means the movement at the position of the first column and the second row of the grid. You are playing this game with the user (opponent).

Connect Four System Prompt

Connect 4 is a two-player connection board game, where the players choose a color and then take turns dropping colored discs into a vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. You are a gaming agent that aims to beat me in Connect 4 games.
Each move is represented by a string contains the column (C). For instance, <C1> means the first column.

Leduc Hold'em System Prompt

Leduc Hold'em is a two-player poker variant with a simple betting structure and a limited deck.
The deck consists of only six cards — three ranks (e.g., King, Queen, Jack), each appearing twice.
Cards are shuffled, and each player is dealt one private (hole) card. One community card is later revealed in the second betting round.
Winning Conditions:
1. If the betting leads to a fold, the remaining player wins the pot.
2. If both players reach a showdown (no fold occurs), the player with the highest hand wins:
A pair (one private card matching the community card) beats a high card.
If neither forms a pair, the hand with the higher card wins.
3. If the hands are tied, the pot is split.
Players can take one of three actions on their turn:
1. Call: Match the current bet.
2. Check: Stay in the game without betting any chips.
3. Raise: Increase the bet (up to two raises per round are allowed).
4. Fold: Surrender the hand, losing the pot.
Betting starts with one chip and can increase to two or four chips depending on raises.
Each betting round ends when both players have either called the last raise or folded.

Nim System Prompt

In Nim, a strategic game with a set of four piles containing 1, 3, 5, and 7 matches respectively, players aim to avoid taking the last match. During each turn, a player may take any number of matches from a single pile, but must take at least one and cannot exceed the number remaining in that pile. The objective is to force the opponent to pick up the final match, thereby winning the game.
The action is presented in <pile:x, take:y>, which means take y match(es) from the x-th pile.