MATCHA: BRIDGING THE GAP BETWEEN SAMPLE AND WALL-CLOCK EFFICIENCY IN MODEL-BASED MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-based Multi-Agent Reinforcement Learning (MBMARL) has demonstrated superior sample efficiency compared to model-free baselines. However, its practical impact is often limited by high *wall-clock* training time. To bridge this gap, we introduce **MATCHA**, a system-algorithm co-design that makes search-based MBMARL efficient on modern accelerators. At system level, MATCHA implements an end-to-end, device-centric framework that keeps data collection, planning, and learning on the accelerator, massively reducing costly CPU–GPU communication. At the algorithmic level, we propose *Path-Guided Frontier Search (PGFS)*, which selects nodes directly from the frontier of the tree using a global score based on cumulative path statistics. This reduces selection and expansion to a single vectorized tensor operation, avoiding depth-dependent sequential tree traversal on the GPU. Across SMAX, Overcooked, and MPE, MATCHA matches or exceeds the sample efficiency of strong MBMARL baselines while substantially reducing training time, achieving up to $\mathbf{16.3}\times$ end-to-end wall-clock speedup with minimal GPU resources.

1 INTRODUCTION

Multi-Agent Reinforcement Learning (MARL) has enabled impressive progress on cooperative decision-making problems, but most successes still rely on model-free training that demands millions or tens of millions of environment interactions on standard benchmarks (Samvelyan et al., 2019; Nguyen et al., 2020; Hernandez-Leal et al., 2020). When experience is expensive, *model-based* MARL (MBMARL) (Egorov & Shpilman, 2022; Wu et al., 2023; Liu et al., 2024; Zhang et al., 2025a;b) is an appealing alternative: a learned world model (Ha & Schmidhuber, 2018; Hafner et al., 2020a;b; 2025) can be used to plan or to train from imagined trajectories, often reducing the number of real environment steps required to reach strong performance. Unfortunately, the practical impact of this sample efficiency is frequently negated by *wall-clock* inefficiency. On standard benchmarks (e.g., SMAC (Samvelyan et al., 2019)), recent MBMARL methods can be over multiple orders of magnitude slower than model-free baselines, and training commonly stretches to multi-day runs even on modern accelerators (Egorov & Shpilman, 2022; Liu et al., 2024; Zhang et al., 2025a).

The sources of this latency differ across MBMARL paradigms. Imagine-based approaches are often dominated by the cost of optimizing large models and generating long-horizon rollouts. In this work we focus on *search-based* MBMARL (Liu et al., 2024), where the bottleneck is more structural: **CPU-centric planning**. Concretely, the dominant pipeline runs Monte Carlo Tree Search (MCTS) on the CPU while calling the GPU repeatedly for model inference during node expansion. This tight iterative loop incurs large CPU–GPU communication overhead and caps throughput by available CPU cores. As a result, adding GPUs yields limited speedup in practice, even though the underlying model inference and rollout evaluation are well-suited to accelerator hardware.

A common response is to *scale out* with large distributed systems (Espeholt et al., 2018) that decompose data collection, planning, and learning into many concurrent workers. While effective in single-agent settings, these systems are resource-intensive and difficult to reproduce in typical

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.

laboratory environments.[1] In contrast, we pursue the complementary goal of *scaling up* efficiency on minimal hardware (e.g., one or two GPUs) by addressing the root cause of latency in search-based MBMARL: the mismatch between traditional tree search and modern accelerator architectures.

Naively porting MCTS to a GPU is insufficient. Standard MCTS executes N simulations sequentially, and each simulation performs depth-d pointer-chasing tree traversal plus backpropagation updates. This creates depth-dependent sequential control flow with branch divergence that prevents SIMD (Single Instruction, Multiple Data) hardware from reaching high utilization. Multi-agent settings further exacerbate this mismatch: the joint action space typically induces larger and deeper search trees, making conventional MCTS even more expensive in practice.

We propose **MATCHA** (**M**odel-based MARL **A**rchi**T**ecture with **C**o-designed Pat**H**-Guided Frontier Search **A**lgorithm), a system–algorithm co-design that bridges the gap between sample efficiency and wall-clock efficiency in search-based MBMARL. MATCHA is implemented as a device-centric vectorized framework that keeps *data collection, planning, and learning* resident on the accelerator, massively reducing the frequent CPU–GPU communication inherent in prior designs. At the algorithmic level, MATCHA introduces a **Path-Guided Frontier Search (PGFS)** that replaces step-by-step tree traversal with one-shot tensor operations over the frontier. The search maintains path-based scores for frontier nodes, and performs global batched sampling and updates across the frontier. This removes depth-dependent sequential traversal from the search inner loop and enables massively parallel evaluation of candidate nodes on the GPU.

Our contributions are summarized as follows:

- **Path-Guided Frontier Search (PGFS):** We propose a path-guided frontier search algorithm whose core operations are batched tensor primitives (batched sampling and updates), aligning with SIMD-style execution on modern accelerators. We also provide a theoretical analysis of the proposed search procedure and derive a non-stationary policy improvement guarantee.
- **Device-Centric Vectorized MBMARL framework:** Built on PGFS, we propose an end-to-end vectorized framework **MATCHA** that keeps simulation, planning, and learning on-device, massively reducing costly CPU–GPU communication in search-based MBMARL.
- **Practical speedups without sacrificing sample efficiency:** Extensive experiments on SMAX, Overcooked, and MPE show that MATCHA matches or exceeds the sample efficiency of strong MBMARL baselines while substantially reducing end-to-end training time, achieving up to $16.3\times$ speedup on minimal computational resources.

2 Related Work

Multi-Agent RL. The dominant paradigm for cooperative MARL is Centralized Training with Decentralized Execution (CTDE), which uses global state information during training while restricting execution to local observations. Model-free methods adhering to this paradigm are broadly categorized into value-based approaches (Sunehag et al., 2017; Rashid et al., 2018; Son et al., 2019; Wang et al., 2021) and policy-based approaches (Lowe et al., 2017; Foerster et al., 2018; Iqbal & Sha, 2019; Ryu et al., 2020; Liu et al., 2020; Kuba et al., 2021; Peng et al., 2021; Yu et al., 2022; Zhang et al., 2024; Ye et al., 2025). Despite their success, model-free algorithms suffer from high sample complexity, motivating growing interest in model-based MARL (MBMARL). One line of work, including MAMBA (Egorov & Shpilman, 2022), MARIE (Zhang et al., 2025a), and DIMA (Zhang et al., 2025b), adapts Dreamer (Hafner et al., 2020a) to multi-agent settings, leveraging learned world models for policy optimization via latent imagination. Another line focuses on search-based planning; notably, MAZero (Liu et al., 2024) extends MuZero (Schrittwieser et al., 2020) to enable multi-agent policy improvement via Monte-Carlo Tree Search (MCTS). While these model-based methods improve sample efficiency over model-free baselines, they incur substantial computational cost, especially search-based methods that rely on sequential planning and CPU-centric implementations.

Scalable and Distributed RL. Acceleration strategies generally fall into two categories: *scale-out* and *scale-up*. Scale-out approaches rely on massive distributed infrastructure. Model-free systems

[1] For example, reported performance for SpeedyZero (Mei et al., 2023) uses a whole cluster of dozens of GPUs and hundreds of CPUs.

(Nair et al., 2015; Horgan et al., 2018; Espeholt et al., 2018) and model-based systems (Ye et al., 2021; Wang et al., 2024; Mei et al., 2023) have achieved high speeds via asynchronous execution on large clusters, but their resource demands limit accessibility and often increase sensitivity to hyperparameter tuning. Scale-up approaches focus on algorithmic efficiency. Techniques like Sampled MuZero (Hubert et al., 2021), Gumbel MuZero (Danihelka et al., 2022), and V-MCTS (Ye et al., 2022) reduce search costs by optimizing simulation counts or action spaces. However, they retain the sequential execution and tree-traversal structure of MCTS, and thus cannot fully exploit the massive parallelism of modern GPUs.

Vectorized and Hardware-Accelerated RL. The rise of JAX has enabled end-to-end vectorized environments (Lange, 2022; Koyamada et al., 2023; Radji et al., 2026). Libraries like Brax (Freeman et al., 2021), Jumanji (Bonnet et al., 2024) and JaxMARL (Rutherford et al., 2024) provide comprehensive benchmark suites, including MARL environments SMAX, Overcooked, and MPE. By leveraging massive parallelism and on-device execution, these environments make large-scale parallel simulation on GPUs possible. Building on this foundation, efficient end-to-end vectorized implementations have been established for model-free algorithms (Kostrikov, 2021; Rutherford et al., 2024; Hessel et al., 2021; de Kock et al., 2023; Mahjoub et al., 2025). However, efficient vectorized implementations for search-based MBMARL remain absent. Existing related efforts (Toledo, 2024) are limited to single-agent settings and do not provide a complete and efficient realization of search-based MBRL in a device-centric framework. To the best of our knowledge, our work is the first to close this gap for search-based MBMARL.

3 Preliminaries

3.1 Search-Based MBRL and Reanalysis

Search-based reinforcement learning methods such as MuZero and its multi-agent extensions typically follow a three-stage training pipeline: (1) **Data Collection**, where agents interact with the environment using a search-based policy and store trajectories in a replay buffer; (2) **Reanalysis**, where historical states are sampled from the buffer and re-evaluated using a fresh search to produce improved learning targets; and (3) **Optimization**, where model parameters are updated to fit these targets.

Among these stages, reanalysis is essential for sample efficiency and training stability but is computationally expensive, as it requires executing planning algorithms over large batches of states. To reduce the latency of the reanalysis process, previous MBRL algorithms usually launch multiple reanalysis workers on host-side to reanalyze the data in a concurrent and asynchronous manner.

3.2 Monte Carlo Tree Search Variants

Monte Carlo Tree Search (MCTS) and its variants (Hubert et al., 2021; Danihelka et al., 2022; Liu et al., 2024; Wang et al., 2024) build a search tree rooted at an initial state s_0 by iteratively performing three phases: **Selection**, **Expansion**, and **Backpropagation**. Each node in the tree corresponds to a state, and each edge corresponds to an action. During selection, the algorithm traverses the tree from the root to a leaf according to stored statistics such as action priors and value estimates. Upon reaching a leaf, the learned model is used to expand new child nodes and to estimate their values. Backpropagation then updates statistics along the traversed path, which influences subsequent selections.

4 Method: MATCHA Framework

In this section, we introduce **MATCHA** (**M**odel-based MARL **A**rchi**T**ecture with **C**o-designed Pat**H**-Guided Frontier Search **A**lgorithm), a high-throughput MBMARL framework designed to bridge the gap between sample efficiency and time efficiency. MATCHA achieves this through a system-algorithm co-design: (1) A device-centric end-to-end vectorized architecture for MBMARL that confines the entire training lifecycle within accelerator memory; (2) The Path-Guided Frontier Search (PGFS) algorithm, a fundamentally different search paradigm that replaces the sequential tree traversal of traditional MCTS variants with one-shot tensor operations. We will first detail the device-centric architecture and then introduce the PGFS algorithm.

Table 1: Per-component profiling results of different algorithms on two SMAX environments with two RTX 5090. Hyper-parameters are aligned with the MAZero baseline for fair comparison.

Algorithm	**3s_vs_5z** (sec/train step) ↓ Data	Reanalysis	Optim.	**2s3z** (sec/train step) ↓ Data	Reanalysis	Optim.
MATCHA	**0.0010**	**0.0045**	**0.0092**	**0.0010**	**0.0067**	**0.0091**
MATCHA w/ GPU MCTS	0.0200	0.0585	**0.0123**	0.0220	0.0548	**0.0122**
MATCHA w/ GPU Gumbel MCTS	0.0157	0.0261	**0.0113**	0.0159	0.0260	**0.0112**
MAZero	0.1712	0.1472	0.0838	0.1720	0.1236	0.0889

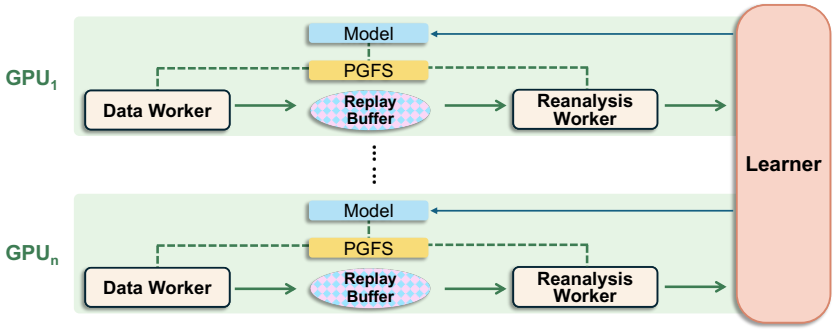


Figure 1: Overview of MATCHA's device-centric, end-to-end vectorized architecture. Each accelerator hosts a self-contained processing pipeline, while a centralized learner coordinates learning across devices. The patterned replay buffer denotes flexible buffer placement: GPU-resident by default, with CPU-resident buffers supported for high-dimensional observation spaces such as image inputs.

4.1 Device-Centric End-to-End Vectorized Architecture

In the standard CPU-centric training lifecycle of search-based MBMARL, illustrated by MAZero in Figure 16, the two dominant computational bottlenecks are **Data Collection** and **Reanalysis** (profiled in Table 1), as they require executing expensive planning algorithms at every step. Reanalysis, in particular, is computationally prohibitive because it requires running MCTS on large batches of data simultaneously for sample efficiency and training stability. Directly parallelizing this batched search on standard hardware is non-trivial due to two system-level constraints: (1) the degree of parallelism is hard-capped by the number of available CPU cores, leaving the GPU unsaturated; (2) the iterative nature of MCTS necessitates frequent bidirectional data transfer between the CPU (search logic) and GPU (inference), creating a severe host-device communication bandwidth bottleneck that cannot be resolved simply by adding more threads. MATCHA addresses these constraints with a **Device-Centric End-to-End Vectorized Architecture** that performs storage and computation directly on the accelerator whenever possible (see Figure 1).

On-device reanalysis chains. On each of the n available accelerators, MATCHA instantiates a self-contained on-device chain for batched data collection, planning, reanalysis, and model inference. Each chain runs independently and in parallel, producing a reanalyzed batch while keeping the search logic and model computation on the same device. The replay buffer in this chain is GPU-resident by default to avoid host-device transfers, but MATCHA also supports CPU-resident buffers for high-dimensional observations such as images. Since buffer access is sparse relative to planning and model inference, this fallback does not dominate the runtime in our setting.

Centralized learner. We use a centralized learner to aggregate the cross-device reanalyzed batches produced by each chain, perform model updates for them sequentially, and then broadcast the updated parameters back to the on-device chains for the next iteration.[2] This design parallelizes the expensive reanalysis stage across accelerators while keeping the learning process centralized and synchronized.

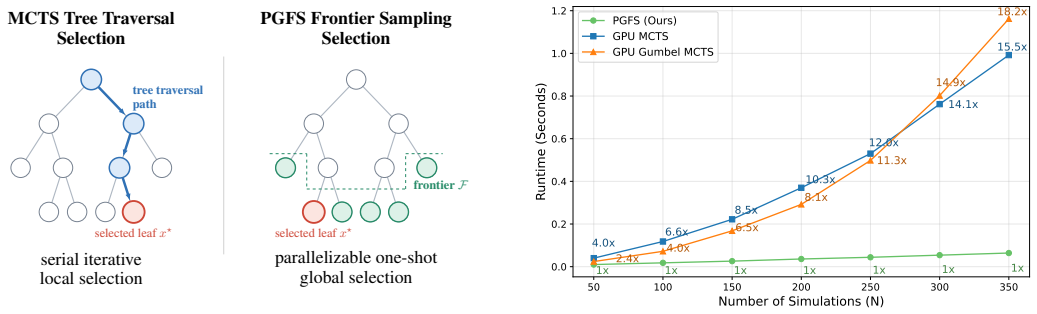
[2]In practice, MATCHA replaces explicit parameter broadcasting with implicit parameter synchronization. We place a fully synchronized learner on each device and ensure that all learners use exactly the same data and perform identical computations, thereby keeping model parameters synchronized across devices.

Unlike Distributed Data Parallel (DDP)-style frameworks such as Anakin (Hessel et al., 2021), which improve environment-step throughput by forming a larger effective batch across devices and synchronizing local updates, MATCHA consumes the n reanalyzed batches as n sequential optimization updates. This enables MATCHA to reduce the average latency per optimization step without reducing the Update-to-Data ratio (UTD), which is important for preserving sample efficiency (D'Oro et al., 2023). In contrast, DDP-style scaling can linearly reduce the effective UTD ratio as the number of devices increases, potentially degrading sample efficiency. We provide a detailed ablation study for UTD in Appendix A.4, and a more detailed comparison with DDP-style scaling in Appendix A.5.

Together, the on-device reanalysis chains and centralized learner substantially reduce CPU–GPU communication overhead while effectively leveraging the parallelism of modern accelerators.

4.2 Path-Guided Frontier Search

For traditional search algorithms such as MCTS and its variants, merely porting them to the device is insufficient to unleash the full potential of modern accelerators (Figure 2). The core reason is that the serial tree traversal procedure in selection and backpropagation phases is not suitable for SIMD devices like GPU. This procedure involves sequential pointer chasing and branch-heavy control flow whose cost scales with the search depth d, preventing the hardware from utilizing its massive parallel throughput. This motivates a new search procedure that eliminates the sequential tree traversal from selection and backpropagation.



(a) MCTS vs. PGFS selection.

(b) Scalability of search algorithms on GPU.

Figure 2: **Search algorithm comparison.** (a) MCTS selects a leaf through a serial, iterative sequence of local comparisons along a single root-to-leaf traversal path (highlighted in blue), whereas PGFS scores every leaf in the frontier $\mathcal{F}$ in one parallel, global operation and samples directly from it. (b) Wall-clock runtime for one complete search under different simulation budgets N; annotations show runtime multipliers relative to PGFS.

4.2.1 Search as Frontier Sampling

Traditional MCTS requires traversal because selection decisions are made *locally*, node by node, using *subtree* statistics such as value estimates and visit counts. After that, backpropagation is also required to update the subtree information.

The key idea behind PGFS is to conduct a *global selection* with a one-shot frontier sampling, by maintaining the statistics of the *root-to-node path* for each node in the frontier $\mathcal{F}$ (the set of unexpanded leaf nodes); see Figure 2a for an illustration of this contrast. This also removes the need for recursive backpropagation, since the cumulative path statistics of newly expanded children can be computed directly from their parent.

Formally, for every node x in the frontier $\mathcal{F}$, we maintain three cumulative statistics of the root-to-x path: 1) $\boldsymbol{l(x)}$: the log prior probability of the path; 2) $\boldsymbol{g(x)}$: the cumulative discounted reward of the path; and 3) $\boldsymbol{d(x)}$: the product of discount factors along the path.

For any node $x \in \mathcal{F}$, we define its path-based value score $V(x)$ as:

$$V(x) = g(x) + d(x) \cdot v_\theta(x) \tag{1}$$

which is the sum of the cumulative reward and the discounted state-value (reward-to-go). Both rewards and state-value are estimated by the learned model. This estimates the expected return of selecting leaf x.

With these statistics, the Selection and Expansion are formulated with vectorized operations:

1. **Vectorized Selection**: Instead of walking the tree, we directly sample a node x^* from $\mathcal{F}$ based on a Boltzmann distribution derived from our cumulative statistics:
$$\eta(x) \propto \exp\left(l(x) + \alpha \cdot V(x)\right) \tag{2}$$
where α is a temperature hyperparameter balancing priors and value scores. This reduces selection to sampling over a flattened frontier tensor.
2. **Vectorized Expansion**: For the selected node x^*, we invoke the model to expand it, by computing the prior probability, immediate reward, and the reward-to-go for all its children y. The new children are then added to $\mathcal{F}$, and their cumulative statistics $l(\cdot), g(\cdot), d(\cdot)$ are updated via simple vector addition and multiplication. After that, x^* is removed from $\mathcal{F}$.

The complete procedure is summarized in Algorithm 1.

4.2.2 Theoretical Properties

In this section, we characterize the theoretical properties of PGFS (Algorithm 1) from three perspectives: node selection as trust-region value maximization, policy improvement, and time complexity.

Node Selection as Trust-Region Value Maximization The PGFS node-selection rule in Equation (2) can be viewed as a trust-region value maximization. Specifically, it selects a node in the frontier that maximizes expected path-based value while remaining in-distribution with respect to the prior. We characterize this property with the following proposition, which is proved in Appendix B.2.

Proposition 4.1. *The node selection distribution η defined in Equation* (2) *is the optimal solution of the following constrained optimization problem:*

$$\begin{aligned} \arg\max_{\mu \in \Delta(\mathcal{F})} \quad & \sum_{x \in \mathcal{F}} \mu(x) V(x) \\ \textit{s.t.} \quad & \mathrm{KL}(\mu \| P_{\mathcal{F}}) \leq \epsilon \end{aligned}$$

where $P_{\mathcal{F}}(x) = \exp(l(x))$ is the prior probability of the root-to-x path, defined for all nodes in the frontier, ϵ is defined by Equation (3)*.*

Policy Improvement MATCHA can be viewed as an instance of generalized policy iteration, in which PGFS acts as a policy improvement operator. The following proposition establishes a policy improvement guarantee for the non-stationary policy induced by PGFS. Its formal statement and proof are deferred in Appendix B.2.

Proposition 4.2 (Policy Improvement, informal)**.** *If v_θ is the value function of the prior policy π, then the non-stationary policy $\widetilde{\pi}$ induced by PGFS achieves an expected return no lower than that of the prior policy π.*

Time Complexity By removing the dependency on tree depth d, PGFS reduces the worst-case search complexity from $O(N^2)$ to $O(N \log N)$, where N is the number of simulations (see Figure 2b). We conduct a detailed analysis for the time complexity of the search algorithms in Appendix B.3.

5 Experiments

In this section, we conduct experiments in three aspects: **(A1) Wall-Clock Efficiency:** How much does MATCHA accelerate training relative to standard CPU-centric search-based MBMARL baselines? **(A2) Sample Efficiency:** Does MATCHA retain the sample-efficiency advantage of strong model-based MARL baselines despite replacing standard tree search? **(A3) Ablation Study:** How do PGFS and the device-centric architecture contribute to the wall-clock efficiency separately? Is MATCHA sensitive to its hyper-parameter selections?

5.1 Experimental Setup

Environments. We evaluate MATCHA on three cooperative MARL benchmarks implemented in fully vectorized JAX environments: (1) **SMAX** (Rutherford et al., 2024), a hardware-accelerated SMAC-like benchmark covering a diverse set of combat scenarios; we evaluate on 10 scenarios spanning both SMACv1-like and SMACv2-like settings. (2) **Overcooked** (Carroll et al., 2019; Rutherford et al., 2024), a coordination-intensive benchmark with challenging exploration under tight time constraints. (3) **MPE** (Lowe et al., 2017), a suite of partially observable cooperative tasks in a 2D physical world with communication.

Baselines. We compare MATCHA against two categories of strong baselines. (1) **Model-based baselines:** MAZero (Liu et al., 2024) as a representative search-based method, and MAMBA (Egorov & Shpilman, 2022) as a representative imagine-based method. (2) **Model-free baselines:** the policy-based methods MAPPO and IPPO (Yu et al., 2022), and the value-based methods QMIX (Rashid et al., 2018), VDN (Sunehag et al., 2017), and IQL. For plotting, all methods are evaluated with 5 random seeds.

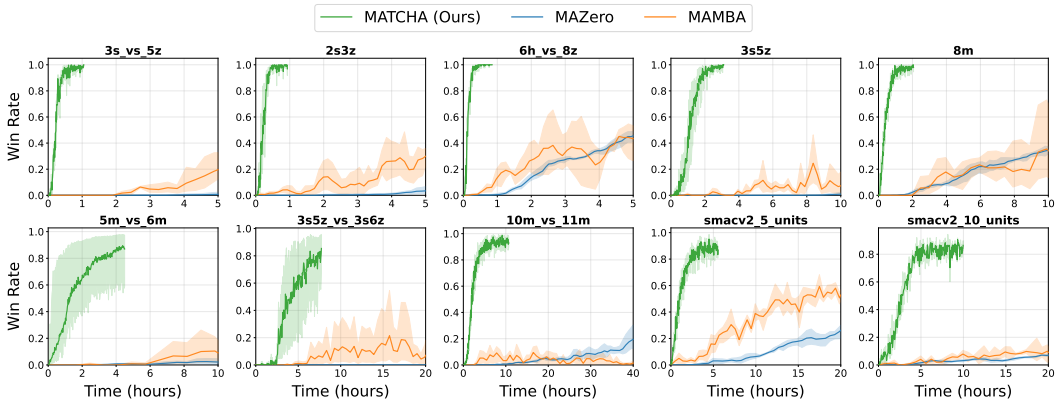


Figure 3: **Wall-clock efficiency (win rate vs. time) on SMAX.** The results are evaluated using 1 RTX 5090 GPU. See Appendix A.2 for the full results and details of the resource settings.

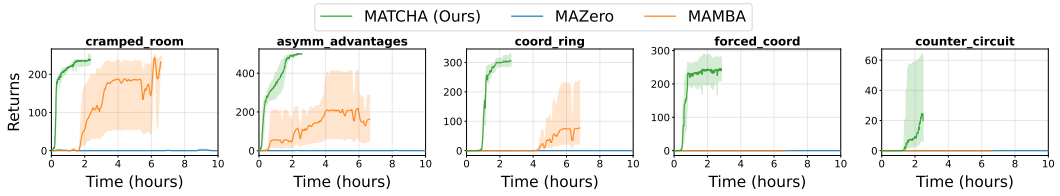


Figure 4: **Wall-clock efficiency (returns vs. time) on Overcooked.** The results are evaluated using 1 RTX 5090 GPU. See Appendix A.2 for the full results and details of the resource settings.

Table 2: **Wall-Clock Speed Benchmarking.** Comparison of training throughput averaged across SMAX and Overcooked tasks. MATCHA achieves an order-of-magnitude speedup over the baseline.

	Training Throughput (FPS) ↑			
Method	SMAX	Overcooked	Average	Speedup
MAZero (Baseline)	3.54	7.11	4.73	1.0×
MATCHA (Ours)	**63.84**	**103.42**	**77.03**	**16.3×**

5.2 Wall-Clock Efficiency (A1)

Figures 3 and 4 show the performance as a function of wall-clock time on SMAX and Overcooked, demonstrating MATCHA's ability to approach asymptotic performance much faster than the baselines.

To quantify this acceleration, Table 2 provides a direct comparison of training throughput (FPS) averaged across SMAX and Overcooked tasks. The quantitative results are consistent with the learning curves: MATCHA achieves a **16.3**× overall speedup compared to the standard CPU-centric baseline MAZero. These findings support our central claim that the dominant bottlenecks in MBMARL stem from CPU–GPU communication and sequential search—both of which are effectively eliminated by our design.

In addition, we conduct further experiments on MPE to broaden the diversity of our benchmark suite (Appendix A.1), as well as an 8-GPU scalability study to verify that MATCHA continues to scale as more GPUs become available (Appendix A.6). Although the experiments in this paper focus on GPU environments to present a clean and convenient comparison, MATCHA is not limited to GPU environments and is equally effective on CPU-centric ones, as shown in Appendix A.3.

5.3 Sample Efficiency (A2)

As shown in Figures 5, 6 and 9, MATCHA consistently achieves the highest sample efficiency among all compared baselines across SMAX, Overcooked, and MPE. These significant gains over model-free methods show that MATCHA preserves the core sample-efficiency advantage of model-based multi-agent RL. Notably, MATCHA also outperforms strong model-based baselines, with especially large margins on Overcooked, where both MAZero and MAMBA perform poorly. We hypothesize that this advantage comes from stronger exploration of our PGFS. The *global selection* mechanism in PGFS may improve exploration over MCTS-style *iterative local selection* by comparing all frontier branches directly, rather than filtering them iteratively through a sequence of local subtree comparisons. As a result, a promising branch is less likely to be overlooked merely because one of its ancestors has uncompetitive subtree statistics caused by other weak sibling branches. This interpretation is consistent with the poor performance of MAZero and MAMBA on the sparse-reward, long-horizon Overcooked tasks, and is further supported by the ablation in Figure 7, where MATCHA with PGFS matches or outperforms MATCHA with GPU Gumbel MCTS, which in turn outperforms MATCHA with standard GPU MCTS.

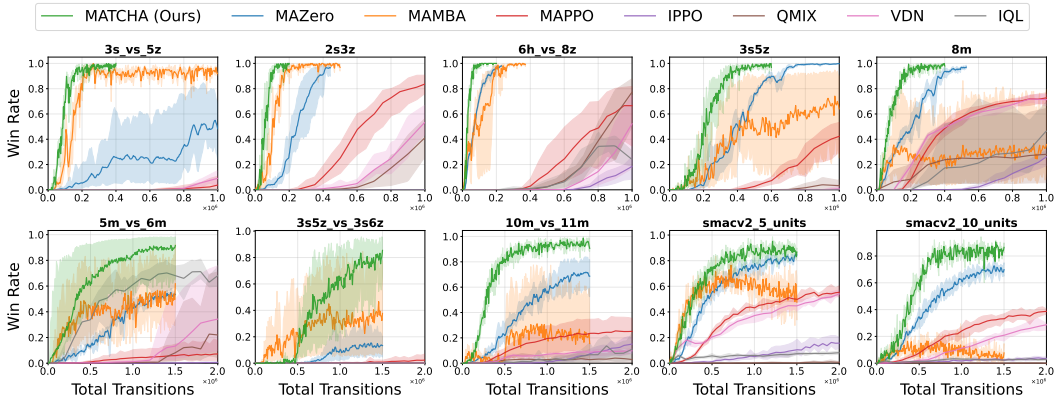


Figure 5: **Sample efficiency (win rate vs. total transitions) on SMAX.**

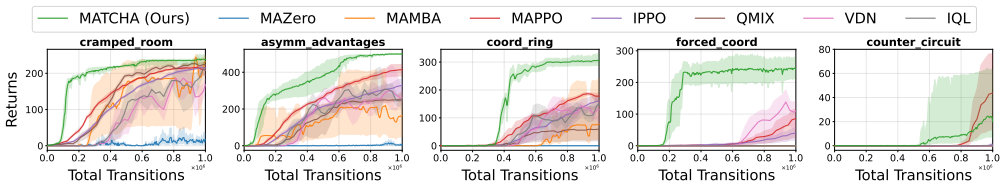


Figure 6: **Sample efficiency (returns vs. total transitions) on Overcooked.**

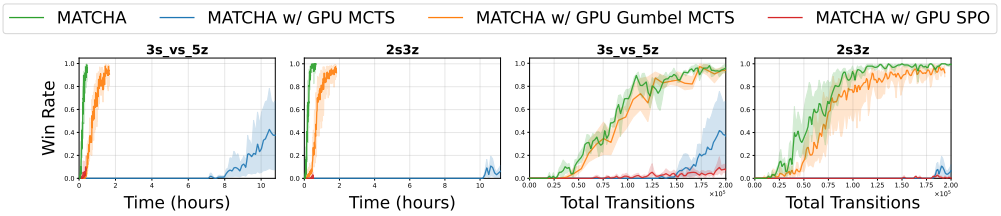


Figure 7: **Search Algorithm Comparison.** We compare MATCHA with PGFS (ours) against MATCHA with GPU-resident implementations of MCTS and Gumbel MCTS in both sample and wall-time efficiency.

5.4 Ablation Study and Per-component Profiling (A3)

We also perform an ablation study to disentangle the contributions of the system architecture and the algorithmic design.

Search Algorithm Comparison. To verify that PGFS improves wall-clock efficiency over MCTS without sacrificing sample efficiency, we compare MATCHA against three variants in the same device-centric framework, each using a GPU-resident (not the standard CPU-based) implementation of the respective search algorithm: **MATCHA w/ GPU MCTS**, **w/ GPU Gumbel MCTS**, and **w/ GPU SPO** (Macfarlane et al., 2024). As Figure 7 shows, MATCHA substantially improves wall-clock efficiency over all baselines while achieving the highest sample efficiency.

To isolate the search procedure's efficiency and verify our time-complexity analysis in Section 4.2.2, we compare the runtime scalability of the search algorithms in Figure 2b. We fix a trained SMAX *3s_vs_5z* model with win rate near 1 and measure the time each algorithm needs for one full search under different simulation budgets. Consistent with our analysis, the MCTS variants scale quadratically with the number of simulations, whereas PGFS is quasi-linear. At $N = 350$, PGFS is 15–20$\times$ faster than the MCTS baselines.

Per-Component Profiling. To understand the specific roles of our two contributions, the device-centric architecture and PGFS, we profile MATCHA, MATCHA w/ GPU Gumbel MCTS, and the CPU-centric baseline MAZero per component (see Table 1). For MAZero, the most time-consuming parts are the search-related components: data collection and reanalysis. For MATCHA w/ GPU Gumbel MCTS, the device-centric architecture accelerates each component. Reanalysis, which involves large-batch search, becomes the dominant bottleneck. For full MATCHA, PGFS further reduces the cost of reanalysis and data collection, making them cheaper than optimization. The dominant cost thus shifts from reanalysis to optimization, indicating that search is no longer the primary bottleneck.

Hyper-parameter Ablation. We also conducted hyperparameter ablations, including one on α, which balances value score and prior constraint in PGFS (Equation (2)). Results in Appendix A.8 show that PGFS is insensitive to α: varying it from 50 to 400 yields nearly identical sample efficiency.

6 Conclusions

To bridge the gap between sample and wall-clock efficiency in MBMARL, we propose MATCHA, combining a system-level **device-centric end-to-end vectorized architecture** with an algorithm-level **Path-Guided Frontier Search**. Across SMAX, Overcooked, and MPE, it matches or exceeds strong MBMARL baselines in sample efficiency, with up to 16.3$\times$ speedup on minimal GPU resources.

Limitations MATCHA targets the search and reanalysis bottlenecks in MBMARL. Once these costs are reduced, model optimization becomes the dominant bottleneck, limiting further speedup. Accelerating this less parallelizable stage remains an open challenge for future MBMARL systems.

REPRODUCIBILITY STATEMENT

We provide the full pseudocode of PGFS in Algorithm 1 (Appendix B), together with complete formal statements and proofs of Propositions 4.1 and 4.2 in Appendix B.2, and a detailed derivation of the time-complexity analysis in Appendix B.3. The model architecture, training objective, and the complete hyperparameter configuration used across all environments are listed in Appendix C. All environments used in our experiments (SMAX, Overcooked, MPE, and the original SMAC) and all baselines (MAZero, MAMBA, MAPPO, IPPO, QMIX, VDN, IQL) are official, publicly available open-source implementations cited in Section 5. All learning curves are averaged over 5 random seeds, and the exact hardware configurations used for all wall-clock measurements are reported in Appendices A.2, A.3 and A.6.

AI USE STATEMENT

In this work, we used generative AI tools, including GPT and Claude, as research and writing assistants. Specifically, we used these tools to draft, revise, polish, and translate portions of the manuscript; assist with code writing and debugging; check proofs and identify potential issues in mathematical derivations; create or refine figures; and help locate relevant literature.

The authors independently reviewed and revised all AI-assisted text, translations, and citations. All AI-assisted code was manually inspected, tested, and executed by the authors. Mathematical proofs and derivations were independently verified by the authors; AI tools were used only as an auxiliary aid for checking potential issues, not as a substitute for author verification. The authors also verified the original sources and bibliographic information for all cited literature, and checked the data sources, accuracy, and final presentation of all figures.

We did not use generative AI tools to generate synthetic datasets, develop the core theoretical or conceptual framework, formulate the central mathematical claims, generate the final proofs, propose or refine the main hypotheses, design the research methodology or experiments, clean or reformat datasets, conduct qualitative or thematic data analysis, or interpret the main experimental results. We take responsibility for the final content of this work, including all text, claims, code, figures, and other artifacts produced with the aid of generative AI.

REFERENCES

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2024. URL https://arxiv.org/abs/2306.09884.

Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=bERaNdoegnO.

Ruan de Kock, Omayma Mahjoub, Sasha Abramowitz, Wiem Khlifi, Callum Rhys Tilbury, Claude Formanek, Andries Smit, and Arnu Pretorius. Mava: a research library for distributed multi-agent reinforcement learning in jax, 2023. URL https://arxiv.org/abs/2107.01460.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider,

Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=OpC-9aBBVJe.

Vladimir Egorov and Aleksei Shpilman. Scalable multi-agent model-based reinforcement learning, 2022. URL https://arxiv.org/abs/2205.15023.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018. URL https://arxiv.org/abs/1802.01561.

Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24, 1994. URL https://api.semanticscholar.org/CorpusID:7519761.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation. http://github.com/google/brax, 2021. Version 0.14.0.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020a. URL https://arxiv.org/abs/1912.01603.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020b.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pp. 1–7, 2025.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A very condensed survey and critique of multiagent deep reinforcement learning. In *Proceedings of the 19th international conference on autonomous agents and multiagent systems*, pp. 2146–2148, 2020.

Matteo Hessel, Manuel Kroiss, Aidan Clark, Iurii Kemaev, John Quan, Thomas Keck, Fabio Viola, and Hado van Hasselt. Podracer architectures for scalable reinforcement learning, 2021. URL https://arxiv.org/abs/2104.06272.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay, 2018. URL https://arxiv.org/abs/1803.00933.

Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces, 2021. URL https://arxiv.org/abs/2104.06303.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2019. URL http://proceedings.mlr.press/v97/iqbal19a.html.

Ilya Kostrikov. JAXRL: Implementations of Reinforcement Learning algorithms in JAX, 10 2021. URL https://github.com/ikostrikov/jaxrl.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 45716–45743, 2023.

Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2021.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Qihan Liu, Jianing Ye, Xiaoteng Ma, Jun Yang, Bin Liang, and Chongjie Zhang. Efficient multi-agent reinforcement learning by planning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=CpnKq3UJwp.

Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. Multi-agent game abstraction via graph attention neural network. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020. doi: 10.1609/AAAI.V34I05.6211. URL https://doi.org/10.1609/aaai.v34i05.6211.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Matthew V Macfarlane, Edan Toledo, Donal Byrne, Paul Duckworth, and Alexandre Laterre. Spo: Sequential monte carlo policy optimisation, 2024. URL https://arxiv.org/abs/2402.07963.

Omayma Mahjoub, Sasha Abramowitz, Ruan de Kock, Wiem Khlifi, Simon du Toit, Jemma Daniel, Louay Ben Nessir, Louise Beyers, Claude Formanek, Liam Clark, and Arnu Pretorius. Sable: a performant, efficient and scalable sequence model for marl, 2025. URL https://arxiv.org/abs/2410.01706.

Yixuan Mei, Jiaxuan Gao, Weirui Ye, Shaohuai Liu, Yang Gao, and Yi Wu. Speedyzero: Mastering atari with limited data and time. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Mg5CLXZgvLJ.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning, 2015. URL https://arxiv.org/abs/1507.04296.

Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. FACMAC: Factored multi-agent centralised policy gradients. In *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=WxH774N0mEu.

Waris Radji, Thomas Michel, and Hector Piteau. Octax: Accelerated CHIP-8 arcade environments for reinforcement learning in JAX. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=BdeUsYlyIf.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning, 2018. URL https://arxiv.org/abs/1803.11485.

Pierre H. Richemond and Brendan Maginnis. A short variational proof of equivalence between policy gradients and soft Q learning. *CoRR*, abs/1712.08650, 2017. URL http://arxiv.org/abs/1712.08650.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, Chris Lu, and Jakob Nicolaus Foerster. Jaxmarl: Multi-agent rl environments and algorithms in jax. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Heechang Ryu, Hayong Shin, and Jinkyoo Park. Multi-agent actor-critic with hierarchical graph attention network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 (7839):604–609, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-03051-4. URL http://dx.doi.org/10.1038/s41586-020-03051-4.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2019. URL https://proceedings.mlr.press/v97/son19a.html.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017. URL https://arxiv.org/abs/1706.05296.

Edan Toledo. Stoix: Distributed single-agent reinforcement learning end-to-end in jax, April 2024. URL https://github.com/EdanToledo/Stoix.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning, 2021. URL https://arxiv.org/abs/2008.01062.

Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. Efficientzero v2: Mastering discrete and continuous control with limited data, 2024. URL https://arxiv.org/abs/2403.00564.

Zifan Wu, Chao Yu, Chen Chen, Jianye Hao, and Hankz Hankui Zhuo. Models as agents: Optimizing multi-step predictions of interactive local models in model-based multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 10435–10443, 2023.

Jianing Ye, Chenghao Li, Yongqiang Dou, Jianhao Wang, Guangwen Yang, and Chongjie Zhang. Towards global optimality in cooperative marl with the transformation and distillation framework, 2025. URL https://arxiv.org/abs/2207.11143.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data, 2021. URL https://arxiv.org/abs/2111.00210.

Weirui Ye, Pieter Abbeel, and Yang Gao. Spending thinking time wisely: Accelerating mcts with virtual expansions, 2022. URL https://arxiv.org/abs/2210.12628.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2022. URL https://arxiv.org/abs/2103.01955.

Yang Zhang, Shixin Yang, Chenjia Bai, Fei Wu, Xiu Li, Xuelong Li, and Zhen Wang. Towards efficient llm grounding for embodied multi-agent collaboration. *arXiv preprint arXiv:2405.14314*, 2024.

Yang Zhang, Chenjia Bai, Bin Zhao, Junchi Yan, Xiu Li, and Xuelong Li. Decentralized transformers with centralized aggregation are sample-efficient multi-agent world models. *Transactions on Machine Learning Research*, 2025a. ISSN 2835-8856. URL https://openreview.net/forum?id=xT8BEgXmVc.

Yang Zhang, Xinran Li, Jianing Ye, Shuang Qiu, Delin Qu, Xiu Li, Chongjie Zhang, and Chenjia Bai. Revisiting multi-agent world modeling from a diffusion-inspired perspective, 2025b. URL https://arxiv.org/abs/2505.20922.

A Additional Experimental Results

A.1 Performance on Multi-Particle Environment (MPE)

We further evaluate MATCHA on the Multi-Particle Environment (MPE) to assess its capabilities in continuous observation spaces and communication-oriented tasks.

In terms of wall-clock efficiency, Figure 8 shows that MATCHA continues to dominate both model-based baselines, MAZero and MAMBA, on MPE.

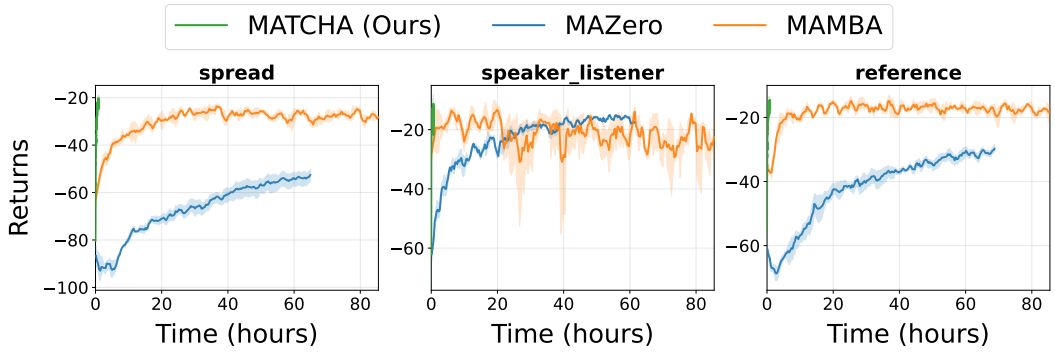


Figure 8: **Wall-clock Efficiency on MPE.** Comparison of MATCHA against model-based baselines on MPE scenarios.

As shown in Figure 9, MATCHA achieves substantially higher sample efficiency than all model-free baselines (MAPPO, IPPO, QMIX, VDN, IQL) as well as the model-based baseline MAZero, across all three tested scenarios. MAMBA improves faster during early training, but consistently converges to a worse (or, at best, comparable) asymptotic performance than MATCHA.

- **Spread:** Requires agents to cover landmarks while avoiding collisions. MATCHA converges faster and achieves a higher collective return, indicating effective spatial coordination.
- **Speaker-Listener:** A communication-heavy task where one agent must signal a target landmark to another. MATCHA's rapid convergence suggests that the search mechanism effectively facilitates the emergence of communication protocols.
- **Reference:** Agents must move to a target referenced by another agent. MATCHA shows significantly lower variance and higher asymptotic performance compared to QMIX and VDN.

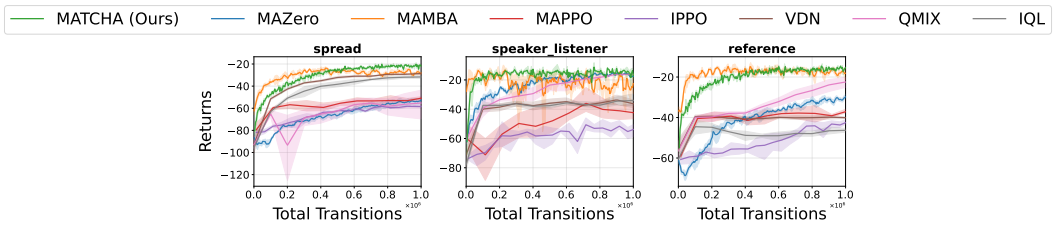


Figure 9: **Sample Efficiency on MPE.** Comparison of MATCHA against model-based and model-free baselines on MPE scenarios.

A.2 Full Wall-Clock Efficiency Performance Results on SMAX and Overcooked

Resource settings. Unless otherwise noted (i.e., the GPU-scalability study in Appendix A.6 and the CPU-centric SMAC results in Appendix A.3, which use the separate hardware described in their respective sections), all experiments reported in the main paper use the same hardware configuration:

2× NVIDIA RTX 5090 GPUs, a 48-core AMD Ryzen Threadripper PRO 5965WX CPU, and 126 GB of RAM. Each experiment uses a single GPU by default, including the wall-clock efficiency results on SMAX and Overcooked (Figures 3, 4, 10 and 11); the sole exception is the per-component profiling result (Table 1), which uses both GPUs. We profile on two GPUs rather than one because cross-device aggregation, where a centralized learner aggregates reanalyzed batches produced by each device's on-device chain (Section 4.1), is a defining feature of MATCHA's architecture; profiling on a single GPU would collapse to the degenerate case where the centralized learner has nothing to aggregate across, and would not reflect the per-component costs of this cross-device execution. All compared methods within a given experiment are evaluated on the same hardware configuration to ensure a fair comparison.

Figures 3 and 4 present time-truncated results to facilitate visual comparison across methods, rather than letting slower approaches such as MAMBA and MAZero until reach the same number of environment steps as MATCHA. This section reports the full wall-clock results without any time truncation, as shown in Figures 10 and 11.

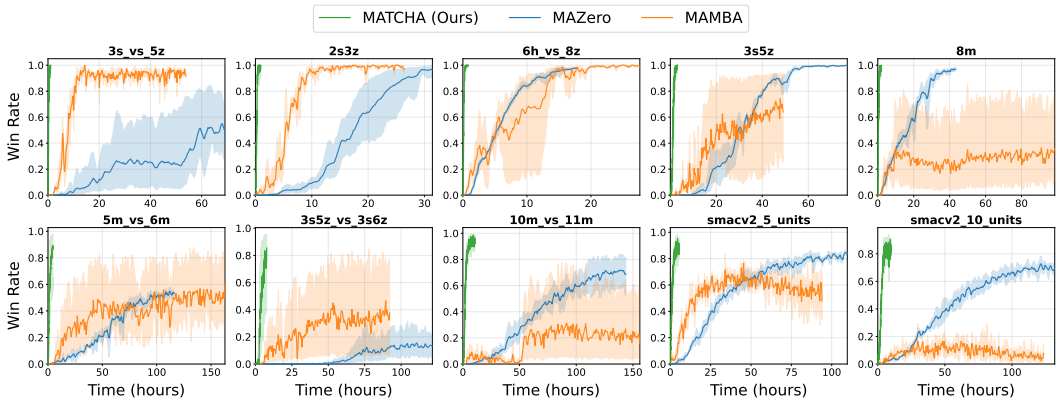


Figure 10: **Full Wall-clock efficiency performance (win rate vs. time) on SMAX.**

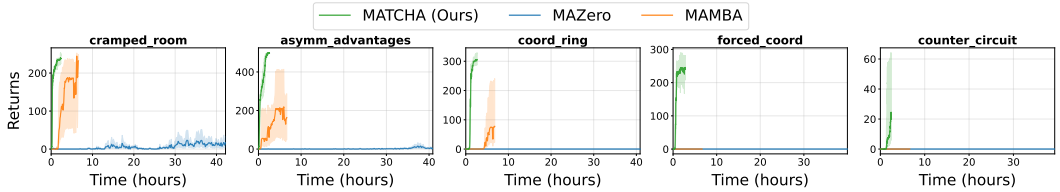


Figure 11: **Full Wall-clock efficiency performance (returns vs. time) on Overcooked.**

A.3 Compatibility with CPU-Centric Environments: Results on the Original SMAC

The experiments in the main paper are conducted on fully vectorized, GPU-native environments (SMAX, Overcooked, MPE). To verify that MATCHA's wall-clock efficiency advantage is not specific to such GPU-vectorized environments, we further evaluate it on the original Python/C++-backed SMAC (Samvelyan et al., 2019) environment, which is CPU-centric and does not support GPU-vectorized simulation. We test on the *3s_vs_5z* scenario and compare MATCHA against the model-based baselines MAZero and MAMBA.

As shown in Figure 12, within 250 minutes MATCHA already reaches a win rate close to 80%, whereas both MAZero and MAMBA remain at a 0% win rate throughout the entire run; these baselines typically require a week or longer to reach a comparable win rate on this task. This confirms that MATCHA's device-centric design yields substantial wall-clock speedups even when interfacing with a legacy, CPU-centric environment, rather than relying solely on the environment itself being GPU-vectorized.

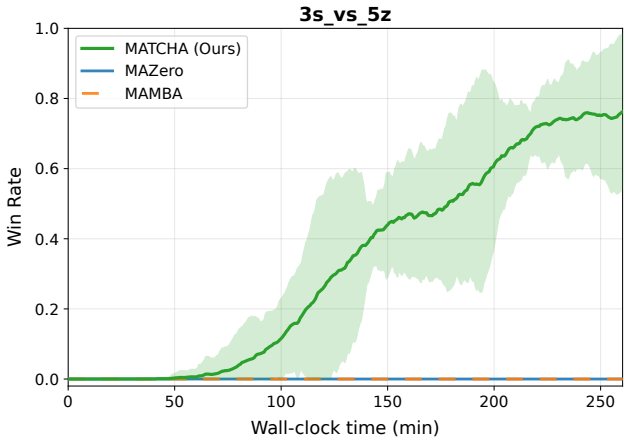


Figure 12: **Wall-Clock Efficiency on the Original SMAC *3s_vs_5z*.** Win rate vs. wall-clock time on the CPU-centric, non-vectorized SMAC environment.

A.4 Ablation Study for Update-to-Data Ratio (UTD)

In this section, we study the effect of the update-to-data ratio (UTD) on both the sample efficiency and wall-clock efficiency of the algorithm.

UTD is an important factor that affects both the sample efficiency and the running time of an algorithm (D'Oro et al., 2023). It measures the average number of gradient updates performed per environment interaction. Its impact on algorithmic performance can be understood from two aspects.

First, within a certain range, increasing the UTD generally improves sample efficiency, since the algorithm performs more optimization steps on the same amount of collected data. Second, increasing the UTD also increases the wall-clock training time. Environment interactions are usually easy to parallelize, whereas model updates are inherently more sequential. When the UTD becomes sufficiently large, model learning, including reanalysis and optimization in our algorithm, becomes the main bottleneck. Consequently, the time required to process the same number of environment interactions grows approximately linearly with the UTD.

We conduct ablation experiments with different UTD values, as shown in Figure 13. The results confirm that, within a reasonable range, a higher UTD improves sample efficiency but also increases the total training time. However, a longer total training time does not necessarily imply worse wall-clock efficiency. As shown in the figure, different UTD values achieve comparable wall-clock efficiency, and lower UTD values are even slightly more efficient. This is because, for the same number of model updates, a lower UTD allows the algorithm to incorporate fresher data from the environment. Since the running time is approximately proportional to the number of model updates, lower UTD values can lead to slightly better wall-clock efficiency.

A.5 Comparison with DDP-style Scaling (e.g., Anakin)

As discussed in Section 4.1, a key difference between our framework and a DDP-style framework lies in how computation is scaled across devices. A DDP-style framework scales out the entire learning process: multiple workers collect data and perform gradient updates in parallel, after which their gradients or model parameters are averaged. Although this design increases the amount of data collected per synchronization step, it still produces only one effective optimizer update after aggregation. Therefore, if the number of effective optimizer updates is kept fixed, increasing the number of workers reduces the effective UTD approximately in inverse proportion to the number of workers. This can substantially degrade sample efficiency.

In contrast, our framework only scales out the pipeline from data collection to reanalysis. The model optimizer sequentially consumes the reanalyzed batches produced by different workers. This

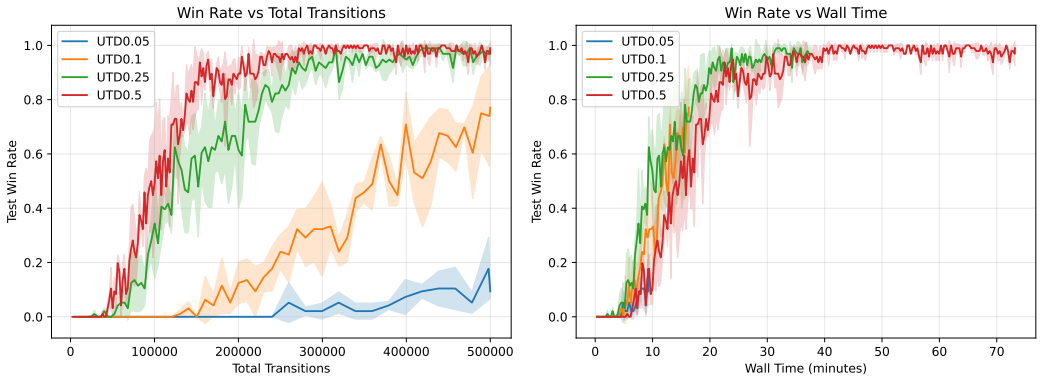


Figure 13: **Ablation Study for Different UTD on SMAX 3s_vs_5z.**

design allows us to increase the throughput of data generation and reanalysis while keeping the UTD unchanged, thereby preserving sample efficiency.

In summary, a DDP-style framework improves the throughput of environment interactions by scaling out the entire learning process, but it reduces the effective UTD and therefore may fail to maintain sample efficiency. By contrast, our framework scales out only the pipeline from data collection to reanalysis. Together with a faster search algorithm, this effectively increases the amount of useful training data available within each optimization cycle, improves training throughput, and preserves sample efficiency.

A.6 GPU Scalability of MBMARL Algorithms

Another key property of an efficient learning system is that it can leverage additional compute to run faster *without* sacrificing performance. To evaluate GPU scalability, we scale MATCHA and model-based baselines from 1 to 8 GPUs on a single server equipped with 8× NVIDIA RTX 5090 GPUs and 256 GB of total VRAM (32 GB per card), paired with 4× AMD EPYC 9654 96-core processors (384 cores total) and 1032 GB of RAM. As shown in Figure 14, MATCHA achieves higher throughput while preserving sample efficiency as GPUs are added, demonstrating strong scalability. In contrast, MAMBA does not meaningfully benefit from GPU scaling: its highly sequential control flow leaves additional GPUs largely idle. MAZero can take advantage of extra RAM/VRAM by increasing the number of reanalysis workers, leading to modest scaling from 1 to 4 GPUs. However, using more reanalysis workers causes CPU thread resources to become saturated, preventing MAZero from scaling effectively in an 8-GPU environment.

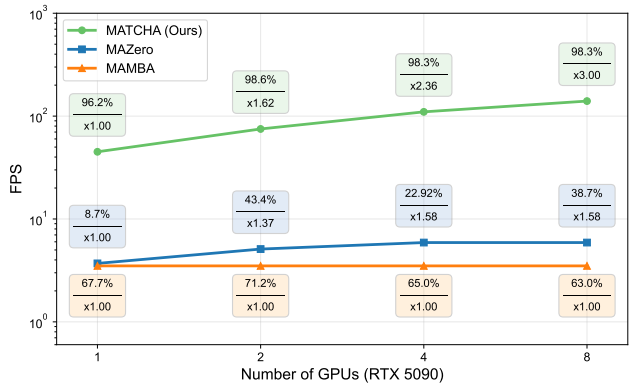


Figure 14: **Scalability of MBMARL algorithms.** FPS and truncated win rate on SMAX *6h_vs_8z* across model-based MARL methods under different numbers of RTX 5090 GPUs; annotations show win rate at 80K environment steps above the line and speedup over the single-GPU setting below it.

A.7 Comparison between CPU buffer and GPU buffer

Our framework MATCHA supports the use of both CPU and GPU buffers. The primary reason we choose a GPU buffer as default is that it completely avoids CPU–GPU communication, allowing the entire algorithm to be compiled into a single kernel and executed fully on the GPU, thereby best reflecting our device-centric design philosophy. As a result, aside from the initial few minutes required for compilation with CPU involvement, the subsequent execution speed depends only on the GPU. This enables full reproducibility across machines with the same GPU model, independent of differences in CPU or memory.

On the other hand, for tasks that require large buffer capacity, such as image-based tasks, we also support the use of a CPU buffer. In fact, since the buffer implementation is based on a sum tree, a data structure that is more CPU-friendly. Therefore the buffer execution on CPU can even be faster than on GPU (Table 3). However, this comes at the cost of additional CPU–GPU communication overhead, which may vary across different machines. However, even for CPU buffer, the buffer-related time cost is not the bottleneck (see Table 1), since the CPU–GPU communication induced by the replay buffer is sparse.

Table 3: Per-step time cost comparison between CPU and GPU buffers.

Buffer type	Time per training step (sec)		
	Buffer operations	CPU–GPU communication	Total
CPU buffer	0.000018025	0.00223	0.002248
GPU buffer	0.0000515	0	0.0000515

A.8 Sensitivity Test for α

We further evaluate the sensitivity of PGFS to the hyperparameter α, which controls the trade-off between the value score and the prior constraint. Specifically, we conduct experiments on two SMAX maps, `3s_vs_5z` and `2s3z`, with $\alpha \in \{50, 100, 200, 300, 400\}$. As shown in Figure 15, the learning curves under different choices of α exhibit similar sample efficiency and converge to comparable performance. This suggests that PGFS is not sensitive to the precise value of α within a broad range, demonstrating the robustness of the proposed method to this hyperparameter.

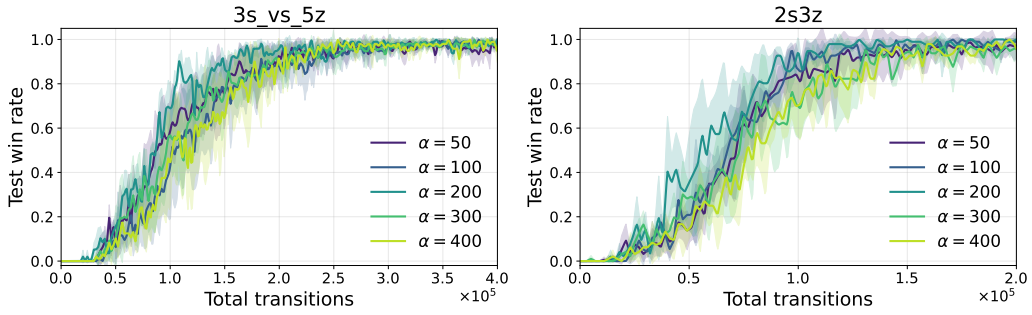


Figure 15: **Ablation Study for Different αs on two SMAX Maps.**

B PATH-GUIDED FRONTIER SEARCH

B.1 ALGORITHM DESCRIPTION

The Path-Guided Frontier Search (PGFS) algorithm is described in Algorithm 1.

Algorithm 1 Path-Guided Frontier Search

Require: Root node x_0; learned model $\mathcal{M}_\theta = (v_\theta, T_\theta)$; temperature α; discount factor γ; number of simulations N.

Ensure: Root policy π' and non-stationary policy $\widetilde{\pi}$

Initialize frontier $\mathcal{F} \leftarrow \{x_0\}$

Initialize statistics: $l(x_0) \leftarrow 0, g(x_0) \leftarrow 0, d(x_0) \leftarrow 1, V(x_0) \leftarrow v_\theta(x_0)$

for $h = 1$ to N **do**

  Sample node $x^\star \in \mathcal{F}$ according to

$$P(x) \propto \exp(l(x) + \alpha V(x))$$

  Remove $x^\star$ from the frontier

$$\mathcal{F} \leftarrow \mathcal{F} - \{x^\star\}$$

  for in parallel $a \in \mathcal{A}$ **do**

    Get child node y, reward r, and prior probability p from the model

$$y, r, p \leftarrow T_\theta(x^\star, a)$$

    Update the statistics for y

$$g(y) \leftarrow g(x^\star) + d(x^\star) \cdot r$$
$$l(y) \leftarrow l(x^\star) + \log p$$
$$d(y) \leftarrow d(x^\star) \cdot \gamma$$
$$V(y) \leftarrow g(y) + d(y) \cdot v_\theta(y)$$

    Add the child to the frontier $\mathcal{F} \leftarrow \mathcal{F} \cup \{y\}$

  end for

end for

Compute the improved policy

$$\pi'(a) \propto \sum_{x \in \mathcal{F}_a} \exp(l(x) + \alpha V(x)), \quad a \in \mathcal{A}$$

where $\mathcal{F}_a$ is the part of the frontier in the subtree of the root action a.

Define $\widetilde{\pi}$: execute $a \sim \pi'$, sample $x \in \mathcal{F}_a$ with probability proportional to $\exp(l(x) + \alpha V(x))$, follow the remaining path to x, then follow the prior policy π.

return $\pi', \widetilde{\pi}$

B.2 PROOF OF THE THEOREMS IN SECTION 4.2.2

We first introduce the following lemma.

Lemma B.1 (Donsker–Varadhan)**.** *Suppose Ω to be a finite support of all the functions related here. Let $Z(\alpha) = \sum_{x \in \Omega} P(x) e^{\alpha A(x)}$, and $\bar{\eta}_\alpha(x) = \frac{P(x) e^{\alpha A(x)}}{Z(\alpha)}$.*

Whenever ϵ and α satisfies

$$\epsilon = \mathrm{KL}(\bar{\eta}_\alpha \| P) \tag{3}$$

The penalized optimization problem

$$\arg\max_{\mu\in\Delta(\Omega)}\left\{\sum_{x\in\Omega}\mu(x)A(x)-\frac{1}{\alpha}KL(\mu\|P)\right\}$$

and the constrained optimization problem

$$\arg\max_{\mu\in\Delta(\Omega)}\sum_{x\in\Omega}\mu(x)A(x) s.t.\ KL(\mu\|P)\le\epsilon$$

have the same solution $\bar{\eta}_\alpha$.

Proof. This is a known result called Donsker–Varadhan variational formula. See proof of Proposition 1 in Richemond & Maginnis (2017) □

We the prove Proposition 4.1.

Proposition B.2. *The node selection distribution* η *defined in Equation* (2) *is the optimal solution of the following constrained optimization problem:*

$$\begin{aligned}\arg\max_{\mu\in\Delta(\mathcal{F})}\quad&\sum_{x\in\mathcal{F}}\mu(x)V(x)\\ s.t.\quad&\mathrm{KL}(\mu\|P_{\mathcal{F}})\le\epsilon\end{aligned}$$

where $P_{\mathcal{F}}(x)=\exp(l(x))$ *is the prior probability of the root-to-*x *path, defined for all nodes in the frontier,* ϵ *is defined by Equation* (3)*.*

Proof. By Lemma B.1, we only need to show that $P_{\mathcal{F}}$ is a distribution, i.e.

$$\sum_{x\in\mathcal{F}}P_{\mathcal{F}}(x)=1$$

This can be proved by induction.

Initial step: At the very beginning of the search (line 1-3 in Algorithm 1), we have $\mathcal{F}=\{x_0\}$, and $l(x_0)=0$. Then it is easy to obtain $\sum_{x\in\mathcal{F}}P_{\mathcal{F}}(x)=\exp(l(x_0))=1$.

Induction step: Suppose we have $\sum_{x\in\mathcal{F}}P_{\mathcal{F}}(x)=1$ currently. Consider expanding any node $x^\star\in\mathcal{F}$. Let $y_1,\cdots,y_{|\mathcal{A}|}$ be its children, and $p_1,\cdots,p_{|\mathcal{A}|}$ be the prior probabilities defined in line 10 of Algorithm 1. Then we have

$$\begin{aligned}&\sum_{u\in\mathcal{F}'}P_{\mathcal{F}'}(u)\\ =&\sum_{u\in(\mathcal{F}-\{x^\star\})\cup\{y_i\}_{i=1}^{|\mathcal{A}|}}\exp(l(u))\\ =&\sum_{u\in\mathcal{F}}\exp(l(u))-\exp(l(x^\star))+\sum_{i=1}^{|\mathcal{A}|}\exp(l(y_i))\\ =&\sum_{u\in\mathcal{F}}P_{\mathcal{F}}(u)-\exp(l(x^\star))+\sum_{i=1}^{|\mathcal{A}|}\exp(l(x^\star)+\log p_i)\\ =&1-\exp(l(x^\star))+\exp(l(x^\star))\sum_{i=1}^{|\mathcal{A}|}p_i\\ =&1\end{aligned}$$

□

Before we prove Proposition 4.2, we introduce the following lemma:

Lemma B.3. *Let x_0 be the root, $p(a) := P(a|x_0)$ is the prior of the root action a, $\mathcal{F}_a$ be the intersection of $\mathcal{F}$ and the subtree of root action a, and $\bar{P}_a(x) := P(x)/p(a)$ for $x \in \mathcal{F}_a$.*

Denote $\rho(a) \propto \sum_{x\in\mathcal{F}_a} \exp(l(x) + \alpha V(x))$ for all root actions as the probability we selection root action a in the selection phase. Also denote $M_a = \max_{x\in\mathcal{F}_a} V(x)$ as the maximal path-based value in $\mathcal{F}_a$.

Then we have

1. $\rho = \mathrm{softmax}(G)$, where

$$G(a) = \log p(a) + \alpha f(a), \quad f(a) = \frac{1}{\alpha} \log \sum_{x\in\mathcal{F}_a} \bar{P}_a(x) \exp(\alpha V(x))$$

2. $M_a + \frac{1}{\alpha}\log p^* \le f(a) \le M_a$, where $p^* = \sum_{x\in\mathcal{F}_a, V(x)=M_a} \bar{P}_a(x) > 0$.

This implies $f(a) \to M_a$ when $\alpha \to \infty$ and $\rho \to \arg\max_a M_a$ when $\alpha \to \infty$.

Proof. The first formula is straightforward.

$$\begin{aligned}
\rho(a) &\propto \sum_{x\in\mathcal{F}_a} \exp(l(x) + \alpha V(x)) \\
&\propto \exp\left(\log \sum_{x\in\mathcal{F}_a} P(x) \exp(\alpha V(x))\right) \\
&= \exp\left(\log p(a) + \log \sum_{x\in\mathcal{F}_a} \bar{P}_a(x) \exp(\alpha V(x))\right) \\
&= \exp\left(\log p(a) + \alpha f(a)\right) \\
&= \exp(G(a))
\end{aligned}$$

For the second formula, we have

$$f(a) = \frac{1}{\alpha} \log \sum_{x\in\mathcal{F}_a} \bar{P}_a(x) \exp(\alpha V(x)) \le \frac{1}{\alpha} \log \exp(\alpha M_a) = M_a$$

and

$$\begin{aligned}
f(a) &= \frac{1}{\alpha} \log \sum_{x\in\mathcal{F}_a} \bar{P}_a(x) \exp(\alpha V(x)) \\
&\ge \frac{1}{\alpha} \log \sum_{x\in\mathcal{F}_a, V(x)=M_a} \bar{P}_a(x) \exp(\alpha M_a) \\
&= \frac{1}{\alpha} \log \left(p^* \exp(\alpha M_a)\right) \\
&= M_a + \frac{1}{\alpha} \log p^*
\end{aligned}$$

□

We then restate and prove the formal version of Proposition 4.2. Let $q = \eta$ be the final frontier selection distribution. The returned policy $\widetilde{\pi}$ induces this same path distribution, since

$$\pi'(a)q(x \mid a) = \left(\sum_{y \in \mathcal{F}_a} q(y)\right) \frac{q(x)}{\sum_{y \in \mathcal{F}_a} q(y)} = q(x), \qquad x \in \mathcal{F}_a.$$

Proposition B.4. *Assume an accurate model of transitions, rewards, and discount factors, and the state-value prediction $v_\theta(x)$ accurately predicts $V^\pi(x)$, where π is the prior policy, then for the non-stationary policy $\widetilde{\pi}$ returned by Algorithm 1, we have*

$$J(\widetilde{\pi}) = \sum_{x \in \mathcal{F}} q(x)V(x) \geq V^\pi(s)$$

where s is the root and $J(\widetilde{\pi})$ denotes the expected return of executing $\widetilde{\pi}$ from s.

This gives a non-stationary policy that improves the current policy π.

Proof. If $v_\theta(x) = V^\pi(x)$, we can guarantee by induction that for anytime in the search procedure, the following property holds:

$$V^\pi(s) = \sum_{x \in \mathcal{F}} P(x)V(x)Q^\pi(s,a) = \sum_{x \in \mathcal{F}_a} \bar{P}_a(x)V(x)$$

since the frontier is a cut of the tree, and the bootstrapping value remains identical anytime after expansion:

$$\forall x : P(x)V(x) = P(x) \sum_{y \in \text{sons}(x)} \pi(y|x)V(y) = \sum_{y \in \text{sons}(x)} P(y)V(y)$$

Thus we have

$$\begin{aligned} J(\widetilde{\pi}) &= \sum_{x \in \mathcal{F}} q(x)V(x) \\ &\geq \sum_{x \in \mathcal{F}} q(x)V(x) - \frac{1}{\alpha}\text{KL}(q\|P) \\ &\geq \sum_{x \in \mathcal{F}} P(x)V(x) - \frac{1}{\alpha}\text{KL}(P\|P) && \text{Proposition 4.1} \\ &= V^\pi(s) \end{aligned}$$

□

B.3 Time Complexity

We briefly analyze the theoretical time complexity of MCTS and PGFS. Here we denote N as the number of simulations, $K = |\mathcal{A}|$ as the branching factor. We also consider the time complexity of one model forward pass as C.

Time Complexity of MCTS The vanilla MCTS conducts N simulations, each with three stages: Selection, Expansion, and Back Propagation. The expansion stage requires a constant number of model forward passes, and therefore has time complexity $O(C)$. Both selection and backpropagation traverse the tree from the root to a leaf. In the worst case, the tree depth can be $\Theta(i)$ at the i-th simulation, so each of these two stages takes $O(i)$ time. Therefore, the total time complexity over all simulations is

$$O\left(\sum_{i=1}^{N}(C+i)\right) = O(CN + N^2).$$

When executed on GPUs, model forward passes are highly parallelized and typically much faster than tree traversal. Thus, when C is treated as a constant with respect to N, the overall complexity is dominated by tree traversal and becomes $O(N^2)$.

Time Complexity of Gumbel MCTS The Gumbel MCTS uses a sequential halving procedure for root actions, which divides the whole N simulations into $c = \lfloor \log_2 K \rfloor$ stages. In each stage, Gumbel MCTS performs simulations evenly for the remaining subtrees, and drops half of them. In this way, there are $\left\lfloor \frac{K}{2^{i-1}} \right\rfloor$ subtrees remaining in stage i, each with the same size

$$\sum_{j=1}^{i} \left\lfloor \frac{\left\lfloor \frac{N}{c} \right\rfloor}{\left\lfloor \frac{K}{2^{j-1}} \right\rfloor} \right\rfloor = \left\lfloor \frac{N}{c} \right\rfloor O\left(\sum_{j=1}^{i} \frac{2^{j-1}}{K} \right) = O\left(\frac{N 2^i}{cK} \right)$$

And the total complexity for selection and backpropagation in the worst case is

$$\sum_{i=1}^{c} \left\lfloor \frac{K}{2^{i-1}} \right\rfloor O\left(\left(\frac{N 2^i}{cK} \right)^2 \right) = \sum_{i=1}^{c} O\left(\frac{N^2 2^i}{c^2 K} \right) = O\left(\frac{N^2}{c^2} \right) = O\left(\frac{N^2}{\log^2 K} \right)$$

The complexity for expansion is still $O(CN)$, making the total complexity $O\left(CN + \frac{N^2}{\log^2 K}\right)$. When C and K are both treated as constants, the overall complexity is also $O(N^2)$.

Time Complexity of PGFS on a single CPU Consider one simulation of our PGFS (Algorithm 1). The selection phase in i-th iteration is a sampling over the frontier of the size iK, which can be done in $O(\log i + \log K)$ with a Fenwick tree (Fenwick, 1994). The expansion phase involves a network forward pass, one point update (removal of $x^\star$), and append-only insertions for the K newly generated children. The point update costs $O(\log i + \log K)$. For the append-only insertions, we use the lazy construction of the Fenwick tree: when appending the m-th entry, we only construct the newly activated Fenwick node and do not update future nodes. This costs $O(1 + \nu_2(m))$, where $\nu_2(m)$ is the exponent of the largest power of two dividing m. Since $\sum_{m=1}^{M} \nu_2(m) = O(M)$, each append-only insertion has amortized $O(1)$ cost. Thus, inserting the K newly generated children costs amortized $O(K)$ per simulation. Therefore, the overall time complexity of PGFS on a single CPU core is

$$O\left(\sum_{i=1}^{N} (C + \log i + \log K + K) \right) = O(CN + N \log N + NK)$$

Time Complexity of PGFS on GPU PGFS is more suitable for GPUs, where abundant parallelism is available. We assume enough GPU parallelism, so that parallelizable operation can be fully parallelized. Then for PGFS (Algorithm 1), a sampling over the frontier can be easily implemented as a tensor operation with parallel-time complexity also in $O(\log i + \log K)$, which has a smaller constant in complexity if we implement segment tree on GPU. Apart from the network forward pass, the expansion phase can be parallelized with $O(1)$ parallel time. Therefore, the overall parallel-time complexity of PGFS is

$$O\left(\sum_{i=1}^{N} (C + \log i + \log K) \right) = O(CN + N \log N + N \log K)$$

Is the worst case realized in practice? The bounds above are worst-case complexities; a natural concern is whether this worst case is actually reached in practice, or whether it is a loose upper bound that rarely binds. Figure 2b provides direct empirical evidence that it is reached: as the simulation budget N grows, the measured runtime of MCTS and Gumbel MCTS scales quadratically, closely tracking the $O(N^2)$ bound, while PGFS scales quasi-linearly, consistent with $O(N \log N)$. This confirms that the asymptotic gap between PGFS and MCTS-style search is not merely a theoretical artifact of worst-case analysis, but is realized in practice.

C HYPER-PARAMETERS

Our model structure follows MAZero (Liu et al., 2024), except using an additional head G to predict the discount factor.

$$\mathbb{P}(\gamma_t = 1|s_t) = G_\theta(\mathbf{s}, \mathbf{a})$$

This head is trained with BCE Loss.

Besides, since the improved policy produced by PGFS already incorporates value information. We do not use AWPO here for policy training, instead, we use a simple Cross Entropy Loss for policy training.

To address the exponentially large joint action space in MARL, we follow MAZero (Liu et al., 2024) and Sampled MuZero (Hubert et al., 2021) and apply standard action down-sampling: at each node, we sample K joint actions and restrict search to this subset.

The GPU MCTS and GPU Gumbel MCTS baselines used in our ablation studies are implemented based on DeepMind's open-source JAX library `mctx` (DeepMind et al., 2020).

The base configuration of our method is shown in Table 4.

We use the base configuration for all SMAX maps.

For MPE and OverCooked tasks, we enable `Model predict discount` as `True` to handle the reset of the environment after fixed steps.

Table 4: Hyperparameter settings.

Parameter	**Setting**
Batch size	256
Buffer size	10^6
Learning rate	5×10^{-4}
Gradient clipping	10
ϵ-greedy	$1.0 \rightarrow 0.05$
ϵ annealing time	10000
Action down-sampling K	40
PGFS simulations N	100
α in frontier selection	200
Number of Envs	8
Model update frequency (per Env Step)	0.5
Reward loss coefficient	1
Value loss coefficient	0.25
Policy loss coefficient	1
Consistency loss coefficient	2
Discount loss coefficient	1
Prior entropy regularization loss	0.25
Action selection type	Greedy
fc_discount_layers	[32]
fc_dynamic_layers	[128, 128]
fc_policy_layers	[32]
fc_representation_layers	[128, 128]
fc_reward_layers	32
fc_value_layers	32
Hidden state size	128
Number of rollout steps	5
TD steps	5
Observation n_stack	1
pred_hid	64
pred_out	128
proj_hid	128
proj_out	128
prioritized_replay_eps	10^{-6}
priority_prob_alpha	0.6
priority_prob_beta	0.4
Reward support	$[-2, 2, 5]$
root_dirichlet_alpha	0.3
root_exploration_fraction	0.25

D BASELINES

We include the CPU-centric architecture of MAZero here for comparison. As shown in Figure 16, MAZero primarily relies on CPU-side orchestration for coordinating simulation, inference, and training, inducing frequent CPU-GPU communication which serves as a contrast to our GPU-centric architecture (Figure 1).

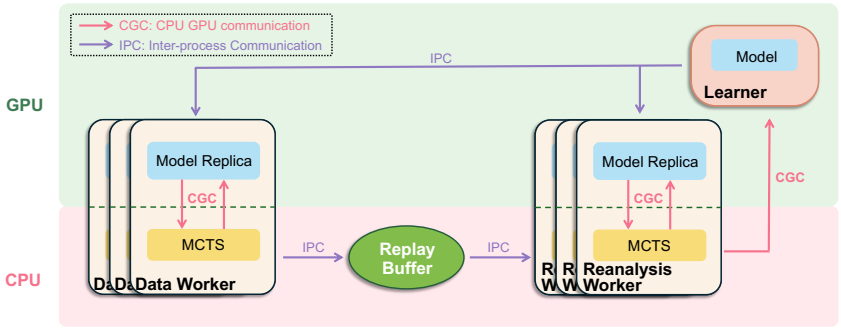


Figure 16: Overview of MAZero's CPU Centric Architecture.