Breaking the Performance Ceiling in Complex Reinforcement Learning requires Inference Strategies

Felix Chalumeau[*1] **Daniel Rajaonarivonivelomanantsoa**[*1,2] **Ruan de Kock**[*1]
Claude Formanek[1] **Sasha Abramowitz**[1] **Oumayma Mahjoub**[1] **Wiem Khlifi**[1]
Simon Du Toit[1] **Louay Ben Nessir**[1] **Refiloe Shabe**[1] **Arnol Fokam**[1]
Siddarth Singh[1] **Ulrich Mbou Sob**[1] **Arnu Pretorius**[1,2]

[1]InstaDeep
[2]Stellenbosch University

Abstract

Reinforcement learning (RL) systems have countless applications, from energy-grid management to protein design. However, such real-world scenarios are often extremely difficult, combinatorial in nature, and require complex coordination between multiple agents. This level of complexity can cause even state-of-the-art RL systems, trained until convergence, to hit a performance ceiling which they are unable to break out of with zero-shot inference. Meanwhile, many digital or simulation-based applications allow for an inference phase that utilises a specific time and compute budget to explore multiple attempts before outputting a final solution. In this work, we show that such an inference phase employed at execution time, and the choice of a corresponding inference strategy, are key to breaking the performance ceiling observed in complex multi-agent RL problems. Our main result is striking: **we can obtain up to a 126% and, on average, a 45% improvement over the previous state-of-the-art across 17 tasks, using only a couple seconds of extra wall-clock time during execution**. We also demonstrate promising compute scaling properties, supported by over 60k experiments, making it the largest study on inference strategies for complex RL to date. **Our experimental data and code are available at** https://sites.google.com/view/inf-marl[1].

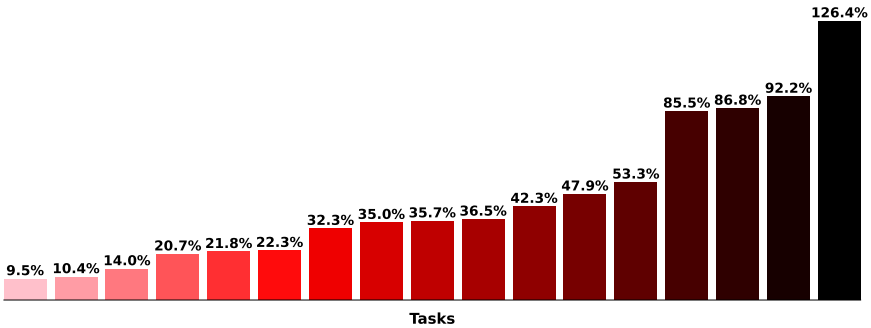

Figure 1: **Improvement from using inference-time search over zero-shot state-of-the-art**. Across 17 complex reinforcement learning tasks, we obtain consistent and significant performance gains using only a 30 second search budget during execution.

[*]Equal contribution. Corresponding author: f.chalumeau@instadeep.com
[1]A GitHub repository will be released shortly.

Preprint.

1 Introduction

Learning to solve sequential decision-making tasks is a central challenge in artificial intelligence (AI), with far-reaching applications ranging from energy-grid optimisation (Ahmad et al., 2021) and autonomous logistics (Laterre et al., 2018) to molecular discovery (Olivecrona et al., 2017) and drug design (Popova et al., 2018). Complex sequential real-world problems that cannot be solved by traditional optimisation techniques are inherently complex and require navigating high-dimensional solution spaces. Reinforcement Learning (RL) presents a promising avenue to improve our capacity to find efficient solutions to these problems, but despite stunning progress over the past decade, such as human-level performance in Atari games (Mnih et al., 2015), defeating the world champion in the game of Go (Silver et al., 2016) or aligning AI systems with human preferences (Stiennon et al., 2020), current approaches are facing challenges that prevent their common deployment in most real-world systems (Dulac-Arnold et al., 2020).

A major source of this difficulty lies in the combinatorial nature of many decision-making tasks. As the problem size increases, the space of possible solutions grows exponentially (Karp, 1975). In multi-agent systems, the challenge compounds: agents must coordinate in environments where only partial information is available, the joint action space is combinatorial, and optimal behaviour depends on precise interaction with other agents (Bernstein et al., 2000; Canese et al., 2021). These properties make it fundamentally difficult to rely on the zero-shot performance of a trained policy, even if that policy was optimised to convergence on a representative training distribution. This causes the gap between zero-shot performance and optimality to grow substantially with increasing complexity (see Fig. 1).

However, numerous practical applications are not restricted to producing a single zero-shot solution. Instead, inference is often permitted to take place over a few seconds, minutes or hours, with a given computational resource. Furthermore, models and simulators are often accessible and very efficient (e.g., energy grid management, train scheduling, package delivery, routing, printed circuit board design) and provide either an exact score or a very accurate approximation. In other applications where the gap to reality may be larger (e.g., protein design, robotics), improving the solution under the simulated score can still arguably provide significant improvement towards the real objective (Hayes et al., 2025; Dona et al., 2024; Hundt et al., 2019; Rao et al., 2020).

This opens up an opportunity: rather than relying on a single attempt of the trained policy, the time budget and compute capacity can be leveraged to actively search for better solutions using multiple attempts, following an *inference-time* strategy. For instance, progressively building a tree of possible solutions, or adapting the policy using outcomes of past attempts. Even straightforward strategies can provide significant performance improvement with low time cost. For instance, generating a large batch of diversified solutions in parallel, using stochastic sampling, rather than a single greedy solution: given a modern GPU, this enables to produce hundreds of solutions, for the same wall-clock time, enabling massive exploration at no time cost. These strategies are rarely emphasized in existing benchmarks (Papoudakis et al., 2021; Mahjoub et al., 2025), and many practitioners invest months of research trying to improve the zero-shot performance of their models on scenarios where only marginal improvement may still be achieved. Whereas they could unlock performance gains from inference-time search, at negligible wall-clock time cost with moderate compute capacity (Fig. 1).

Research in RL for Combinatorial Optimisation (CO) has produced efficient inference strategies (Bello et al., 2016; Hottung et al., 2022; Choo et al., 2022; Chalumeau et al., 2023b), often referred to as active search, or online adaptation methods. However, their empirical study is still limited to a few problems, a narrow range of budget settings (Chalumeau et al., 2024b), and barely no insight on their scaling properties. In the multi-agent case, there is no study on inference strategies for collaborative teams: most work on team adaptation focus on Ad Hoc Teamwork (Mirsky et al., 2022; Wang et al., 2024a; Ruhdorfer et al., 2025), which is adjacent to our objective. Interestingly, most recent studies about the impact of inference strategies come from the Large Language Model (LLM) literature (Snell et al., 2025; Muennighoff et al., 2025; Wu et al., 2025), where the adequate combination of efficient models with inference strategies is currently state-of-the-art (SOTA).

In this work, we formalise and investigate the role of inference strategies in complex decision-making tasks. To capture the full complexity of tasks described above, we formulate our problem setting as a decentralised partially observable Markov decision process (Dec-POMDP) (Kaelbling et al., 1998). This is instead of the typical single-agent MDP used in many RL studies. We make this choice for

several reasons: (1) it more accurately maps onto many complex real-world problems of interest, (2) Dec-POMDPs subsume MDPs by being strictly more complex (Bernstein et al., 2000), and (3) because of this, we expect our findings to translate to all simpler problem formulations. Within this setting, we provide a unifying view of popular inference-strategy paradigms, including policy sampling, tree search (Choo et al., 2022), online fine-tuning (Bello et al., 2016; Hottung et al., 2022), and diversity-based search (Chalumeau et al., 2023b). Strikingly, we show that across a wide range of specifically selected difficult RL problems, inference strategies boost performance on average by 45%, over zero-shot SOTA. Furthermore, in the best of cases, this boost can be as large as 126%. All of this, using only a couple of seconds of additional execution time.

Our results call for a shift in how RL systems are evaluated and deployed: inference strategies are not a minor post-processing step, but a key performance driver in realistic conditions. This work sets the foundation for a more nuanced view of inference in sequential decision-making and provides the tools to build systems that can scale with compute. All our code and experimental data can be accessed at: https://sites.google.com/view/inf-marl.

2 Related work

Inference Strategies from RL for CO Beyond naive stochastic sampling, several paradigms have been explored to generate the best possible solution using a trained policy checkpoint during inference. Online fine-tuning Bello et al. (2016) retrains all policy parameters with RL using past attempts. Hottung et al. (2022) re-trains only a subset of the policy's parameters to reduce memory and compute overheads, enabling more attempts for a given inference budget. It also adds a imitation learning term to the RL term, to force exploration close to the best solution found so far. Choo et al. (2022) uses tree search, with simulation guided node estimates under budget constraints, which outperforms Beam Search and Monte Carlo Tree Search (Coulom, 2006). Diversity-based methods: inspired by previous diversity-seeking approaches, like unsupervised skill discovery (Eysenbach et al., 2019; Sharma et al., 2019; Kumar et al., 2020) and quality-diversity (Chalumeau et al., 2023a; Cully and Demiris, 2017), Grinsztajn et al. (2023) introduces an RL objective that trains a population of diverse and specialized policies, efficient for few-shot performance. Chalumeau et al. (2023b) uses this objective and encodes the diversity in a continuous latent space that can be searched at inference-time, introducing the SOTA method COMPASS; meanwhile Hottung et al. (2024) uses a similar approach but with a discrete encoding space.

These works have introduced most of the inference strategies we consider in this paper, but they fall short on three important aspects that we aim to improve: (i) they evaluate inference strategies on benchmarks where over 95% zero-shot optimality is already achieved, leaving little room for meaningful gains, (ii) these benchmarks rely on domain-specific tricks such as starting points or instance augmentations; and (iii) methods are compared under a unique budget setting, overlooking the fact that relative performance depends on the available compute and time budget. In addition, their ability to scale with compute remains unexplored, despite being a critical property.

Search and adaptation in Multi-Agent RL There is only limited work on inference strategies for MARL. Most work about search and adaptation within MARL focus on the challenge of Ad Hoc Teamwork (Yourdshahi et al., 2018; Hu et al., 2020; Mirsky et al., 2022; Wang et al., 2024a; Ruhdorfer et al., 2025; Hammond et al., 2025), often in the form of zero-shot coordination, where agents must generalize to new partners at execution time. While these lines of work share some methodological similarities, for instance using diversity-seeking training (Long et al., 2024; Lupu et al., 2021) or adapting through tree search (Yourdshahi et al., 2018), they pursue fundamentally different goals and remain orthogonal. In our work, our focus is primarily on solving difficult and complex industrial optimisation tasks.

Inference-time compute for LLMs Recent advances in LLMs are closely intertwined with the use of inference strategies (Snell et al., 2025; Wei et al., 2022; Wang et al., 2024b), and a growing effort has gone into studying their scaling properties (Muennighoff et al., 2025; Wu et al., 2025). However, the typical inference-time setting is usually different from ours. LLMs have very costly forward passes, and cannot access the exact score of their answers, but can approximate them using a reward model (Ouyang et al., 2022). Most popular strategies for LLMs are designed for few shots, namely sampling and ensembling (e.g., majority voting).

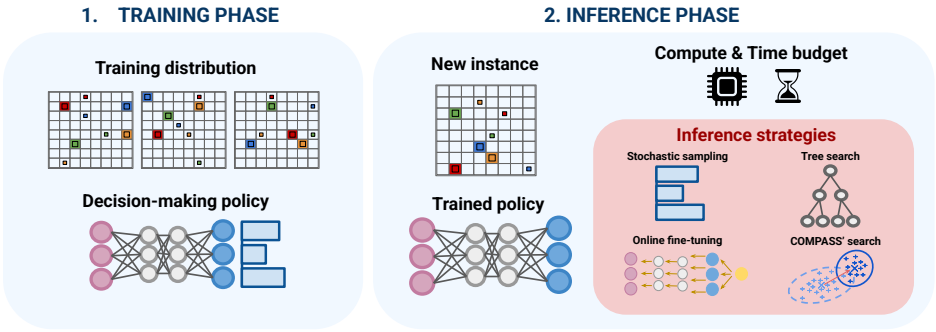

Figure 2: **Numerous applications of RL involve two distinct phases:** (1) a training phase, typically unconstrained in time and compute, during which a policy is optimized over a representative distribution of problem instances; and (2) an inference phase, where a limited time and compute budget are allocated to solving a new instance. The inference phase is often overlooked, despite its crucial role in complex tasks where partial observability and the combinatorial growth of observation and action spaces make good solutions unattainable through zero-shot execution alone.

Overall, numerous inference strategies have been proposed in the literature, yet their efficiency under various evaluation settings remains unexamined. Multi-agent RL, despite its inherent complexity, rarely considers inference-time search beyond Ad Hoc Teamwork. Moreover, the broader field of decision-making has not systematically studied how inference strategies scale with compute. Our work aims at filling these gaps by extending the evaluation of inference strategies in RL, demonstrating major performance gains over a wide range of budget settings with impressive scaling properties.

3 Finding the best solution for a given time and compute budget

3.1 Preliminaries

We focus on RL approaches, and use a neural network (policy) that can construct a solution by taking a sequence of actions. This policy is optimised during a training phase and then used during an inference phase, along with an *inference strategy*, to construct the best possible solution to a new problem instance under a given time and compute budget. These two phases, illustrated in Fig. 2, have different assumptions, objectives and constraints, detailed in the following paragraphs.

Problem instances We assume that each problem instance can be formulated as a Dec-POMDP (Kaelbling et al., 1998), defined by the tuple $\mathcal{M} = \langle N, \mathcal{S}, \boldsymbol{\mathcal{O}}, \boldsymbol{\Omega}, \boldsymbol{\mathcal{A}}, R, P, \gamma, H \rangle$. Here N is the number of agents, $\mathcal{S}$ is the environment state space, $\boldsymbol{\mathcal{O}} = \prod_{i=1}^{N} \mathcal{O}^i$ is the joint agent observation space, $\boldsymbol{\Omega} : \mathcal{S} \mapsto \boldsymbol{\mathcal{O}}$ is the observation function, $\boldsymbol{\mathcal{A}} = \prod_{i=1}^{N} \mathcal{A}^i$ is the joint action space, $R : \mathcal{S} \times \boldsymbol{\mathcal{A}} \mapsto \mathbb{R}$ is the shared reward function, $P : \mathcal{S} \times \boldsymbol{\mathcal{A}} \mapsto \Delta\mathcal{S}$ is the environment transition function, the scalar $\gamma \in [0, 1]$ is the discount factor and H is the finite episode horizon. At each timestep the environment occupies a state s_t which is mapped to a joint partial observation $\boldsymbol{o}_t$ via $\boldsymbol{\Omega}$. The joint action is then sampled following the joint policy, $\boldsymbol{a}_t \sim \boldsymbol{\pi}(\cdot \mid \boldsymbol{o_t})$ and is executed in the environment; after which the environment transitions to the next state s_{t+1}, following $P(\cdot \mid s_t, \boldsymbol{a}_t)$, and the team receives shared the reward $r_t = R(s_t, \boldsymbol{a_t})$.

Training Phase We assume a distribution of problem instances $\mathcal{D}$, that can be sampled from. The joint policy $\boldsymbol{\pi_\theta}$, parameterised by $\boldsymbol{\theta}$, is used to construct a solution sequentially by taking joint actions conditioned on the joint observation at each timestep. We use RL to train this policy to maximise the expected return obtained when building solutions to instances drawn from the distribution $\mathcal{D}$ over a horizon H: $J(\boldsymbol{\pi_\theta}) = \mathbb{E}_{\mathcal{D}}\left[\sum_{t=0}^{H} \gamma^t R(s_t, \boldsymbol{a_t})\right]$. This training objective corresponds to a single attempt (zero-shot). Ideally, this objective should anticipate the multiple attempts allowed at inference, but this is hard to scale. Recent works incorporate such few-shot objectives (Grinsztajn et al., 2023; Chalumeau et al., 2023b, 2024b), but none can yet scale beyond 200 attempts.

The training phase is usually loosely constrained in terms of time and compute capacity, as typically industrial stakeholders are willing to invest days, weeks or even months of training to obtain a high-performing policy that can generate accurate solutions when deployed in production. Hence, in our experiments, we train all policies until convergence.

Inference Phase At inference time, a new problem instance ρ is drawn from a distribution $\mathcal{D}'$ (possibly different than $\mathcal{D}$). Here, there are typically hard constraints to outputting a final solution: a fixed time limit $\mathrm{T}_{\max}$ constrains wall-clock execution, and a compute capacity $\mathrm{B}_{\max}$ constrains the number of operations that can be done in parallel. The trained policy $\boldsymbol{\pi_\theta}$ can be used within these constraints to generate solutions to the problem, and the best solution is ultimately used. The reward function R can still be used to score attempted solutions and inform subsequent attempts. *Inference strategies* can be defined as a function $\mathcal{I} : (\rho, \boldsymbol{\pi_\theta}, \mathrm{B}_{\max}, \mathrm{T}_{\max}) \mapsto (\boldsymbol{a}_1^*, ..., \boldsymbol{a}_H^*)$ that uses the base policy $\boldsymbol{\pi_\theta}$ and any additional inference-time search, adaptation, storage, or optimisation methods under the budget $\mathrm{T}_{\max}$ and $\mathrm{B}_{\max}$ to produce the best possible solution to the problem instance ρ, defined by the sequence of actions $(\boldsymbol{a}_i^*)_{1 \leq i \leq H}$. The objective can hence be written as:

$$I(\mathcal{I}) = \sum_{t=0}^{H} R(s_t, \boldsymbol{a}_t^*) \text{ s.t. } \mathrm{C}(\mathcal{I}) \leq \mathrm{B}_{\max}, \ \mathrm{T}(\mathcal{I}) \leq \mathrm{T}_{\max}$$

where $\mathrm{C}(\mathcal{I})$ and $\mathrm{T}(\mathcal{I})$ represent the compute and time cost of the inference strategy. This formulation highlights that, unlike traditional RL, where zero-shot performance is the primary measure, we focus on strategies which enable further improvement under given constraints.

3.2 One budget, many uses: inference-time search and adaptation

In this section, we detail four types of inference strategies and how we adapt them to work in the multi-agent setting. We implement and release all of these methods in JAX (Bradbury et al., 2018).

Stochastic policy sampling The first natural lever to improve solution quality is to re-sample from a stochastic policy. In other words, beyond the creation of a unique greedy solution (i.e., using $a = \arg\max_{a'} \pi_\theta(a'|o)$ over a trajectory of observations), one can sample stochastically (as in $a \sim \pi_\theta(\cdot|o)$) in order to create diverse solutions. **Multi-agent policy sampling** generalises easily to the multi-agent case by sampling from the joint action distribution, $\boldsymbol{a} \sim \boldsymbol{\pi_\theta}(\cdot|\boldsymbol{o})$.

Tree search These methods store information about partial solutions using past attempts to preferentially search promising regions of the solution space without updating the pre-trained policy. Simulation guided beam search (SGBS) (Choo et al., 2022) provides the best time to performance balance in the literature, outperforming Monte Carlo Tree Search (Coulom, 2006). Like most tree searches, SGBS has three steps: expansion, simulation, pruning. Expansion uses the policy to decide on the most promising next actions (i.e., $a = \text{top-}K(\pi_\theta(\cdot|o))$) from the current node (partial solution). A simulated rollout of an episode is produced greedily using π_θ and the return is collected for each node. Pruning keeps only the best nodes found so far based on the return. Solely the expansion step needs to be adapted for **Multi-agent SGBS**. This is trivial when the explicit joint actions are accessible (de Witt et al., 2020; Yu et al., 2022), since we can still select the top ones (i.e., $\boldsymbol{a} = \text{top-}K(\boldsymbol{\pi_\theta}(\cdot|\boldsymbol{o}))$). For methods using auto-regressive action selection (Mahjoub et al., 2025), having access to the top joint actions is intractable, hence we sample K times stochastically from the same node (i.e., $\boldsymbol{a}[1], ..., \boldsymbol{a}[K] \sim \boldsymbol{\pi_\theta}(\cdot|\boldsymbol{o})$).

Online fine-tuning These methods keep updating policy parameters at inference time. Given a base policy π_θ, online fine-tuning optimises θ using inference-time rollouts and policy gradient updates: $\theta' = \theta + \alpha \nabla_\theta J(\pi_\theta)$, where $J(\pi_\theta)$ represents an adaptation objective. In line with (Bello et al., 2016), we keep maximising expected returns (Bello et al., 2016) over past attempts on the fixed instance (instead of over a training distribution). **Multi-agent online fine-tuning** re-trains $\boldsymbol{\pi_\theta}$ on the new instance using the MARL algorithm that was used during pre-training (Bello et al., 2016; Mahjoub et al., 2025; de Witt et al., 2020; Yu et al., 2022).

Diversity-based approaches These methods pre-train a collection of diverse specialised policies which can be used to search for the most appropriate solution at inference-time. COMPASS (Chalumeau

et al., 2023b) encodes specialised policies in a continuous latent space $\mathcal{L}$ by augmenting a pre-trained policy to condition on both the observation and a latent vector sampled from $\mathcal{L}$ (i.e., $a \sim \pi_\theta(\cdot \mid o,\ z),\ \ z \sim \mathcal{L}$): effectively creating a continuous collection of policies. COMPASS achieves SOTA in single-agent RL for CO. To avoid having the latent space of **Multi-agent COMPASS** growing exponentially with the number of agents, we keep one latent space $\mathcal{L}$ for all agents (i.e., $\boldsymbol{a} \sim \boldsymbol{\pi_\theta}(\cdot \mid \boldsymbol{o},\ z),\ \ z \sim \mathcal{L}$). This allows for tractable training, and for efficient inference search with the covariance matrix adaptation evolution strategy (CMA-ES) (Hansen and Ostermeier, 2001). Aside from being multi-agent, we keep the training and inference phases close to the original method described in Chalumeau et al. (2023b).

4 Experiments

In our experimental study, we combine popular MARL algorithms and inference strategies and benchmark them on a set of complex RL tasks from the literature. Each task was specifically selected for its difficulty. We evaluate all base policies with and without inference-time search across a wide range of budget settings. Our experiments constitute the largest-ever study of inference strategies for decision-making.

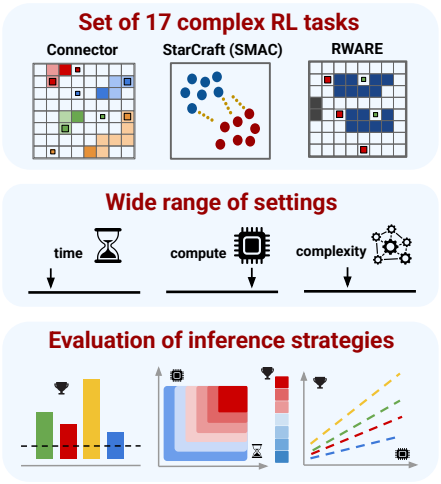

Figure 3: **Overview of our tasks and experimental study.**

Baselines We use three MARL approaches to obtain our base policies: Independent PPO (de Witt et al., 2020) (IPPO) and Multi-Agent PPO (Yu et al., 2022) (MAPPO), which are widely used and well-known MARL methods, and the recent SOTA sequence modelling approach SABLE (Mahjoub et al., 2025). Each of these, referred to as *base policies*, is evaluated with all four inference strategies introduced in Section 3.2, namely stochastic sampling, SGBS, online fine-tuning and COMPASS.

Tasks Mahjoub et al. (2025) established SOTA over the most comprehensive MARL benchmark published in the field to date. Interestingly, their results demonstrate that there remain certain tasks for which no existing method (including SABLE) is able to achieve good performance. Specifically, these are `tiny-2ag-hard`, `tiny-4ag-hard`, `small-4ag`, `small-4ag-hard`, `medium-4ag`, `medium-4ag-hard`, `medium-6ag`, `large-4ag`, `large-4ag-hard`, `large-8ag`, `large-8ag-hard`, `xlarge-4ag` and `xlarge-4ag-hard` from Multi-Robot Warehouse (Papoudakis et al., 2021) (RWARE), `smacv2_10_units` and `smacv2_20_units` from the StarCraft Multi-Agent challenge (Samvelyan et al., 2019; Ellis et al., 2023) (SMAC), and `con-10x10x10a` and `con-15x15x23a` from Connector (Bonnet et al., 2023).

Each environment (illustrated on Fig. 3) introduces distinct challenges that contribute to its complexity. RWARE requires agents to coordinate in order to pick up and deliver packages without collision and has a very sparse reward signal. In SMACv2 tasks, a team cooperates in real-time combat against enemies across diverse scenarios with randomised generation. Connector models the routing of a printed circuit board where agents must connect to designated targets without crossing paths. All three environments feature combinatorial and high-dimensional action spaces, partial observability and the need for tightly coordinated behaviours, making these 17 tasks a compelling test-bed for complex RL with desirable properties modelling aspects of real-world tasks. We use JAX-based implementations of Connector and RWARE from Jumanji (Bonnet et al., 2023) and for SMAC from JaxMARL (Rutherford et al., 2023).

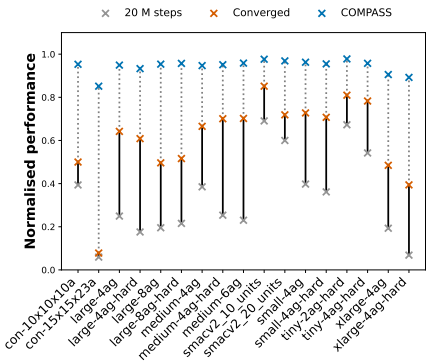

Figure 4: **Training the SOTA algorithm Sable to convergence** is not enough to achieve optimal zero-shot performance.

Training base policies To obtain clear performance ceilings for each algorithm and best isolate the effects of inference strategies, we train all base policies until convergence. For the sake of continuity with previous work with truncated training budgets, typically of 20M steps, we report the zero-shot results for each converged checkpoint compared to its previous corresponding reported performance on Fig. 4. We observe that in most tasks (14 out of 17), the converged policy stays below 70% normalised performance. COMPASS requires an additional training phase, which reincarnates the existing base policies to create the latent space specialisation. For each base policy and task, we also train the COMPASS checkpoint until convergence. This leads to 102 trained policy checkpoints.

Evaluating performance during inference Evaluating inference strategies in a way that is unbiased and aligned with real-world settings is challenging. Most papers report results where the budget is based on a number of attempts, hence not directly incorporating the time cost of the strategy. The time costs are reported, but it is tough to analyse due to the plurality of hardware used to obtain them. Having re-implemented all of the baselines in the same code base and setting the budget in terms of time (in seconds), we can avoid this bias in our study. We use the same fixed hardware for all our experiments, namely a `NVIDIA-A100-SXM4-80GB` GPU. For statistical robustness, we always run 128 independent seeds. In all cases, we control for B_{max} by varying the permitted number of batched parallel attempts instead of altering hardware between experiments. For aggregation across multiple tasks we follow the recommendations made by Agarwal et al. (2021) and use the `rliable` library to compute and report the inter-quartile mean (IQM) and 95% stratified bootstrap confidence intervals.

Hyperparameters To train the base policies, we re-use the hyperparameters reported in Mahjoub et al. (2025), which have been optimised for our tasks. For the inference strategies, we follow recommendations from the literature (Choo et al., 2022; Chalumeau et al., 2023b). All hyperparameters choices are reported in Appendix C.

4.1 A couple of seconds is all you need

In this section, we demonstrate that inference-time search can help reach close to maximum task performance, using base policies for which zero-shot performance stagnates around 60%.

Experiments To demonstrate that inference-time strategies are accessible, we use a small budget: 30 seconds, and a compute capacity enabling to generate 64 solutions in parallel. Each base policy is evaluated greedily for a single attempt, and then evaluated with the search budget, using each inference strategy. We report the performance distribution over the 17 tasks in Fig. 5, and the performance gains offered by the best inference-time search over the best zero-shot on Fig. 1.

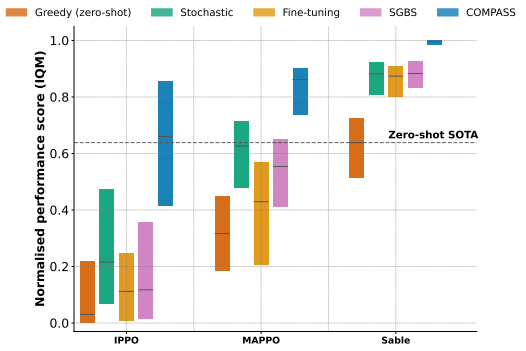

Figure 5: **Performance obtained by inference strategies over the benchmark**. Each base policy is evaluated with each possible inference strategy. We report the inter-quartile mean over tasks with 95% stratified bootstrap confidence intervals.

Discussion We can draw four main conclusions. First, inference-time search does provide a massive performance boost over zero-shot, which stands for every base policy. For the SOTA zero-shot method SABLE, this translates to pushing the best-ever achieved aggregated performance by more than 45% and creating a system (SABLE +COMPASS) that achieves close to 100% win-rate in all tasks where this metric is available. Second, the improvement enabled over zero-shot performance increases significantly (almost exponentially) with respect to the complexity of the task (see Fig. 1). This suggests substantial gains are still ahead as the field moves toward increasingly realistic scenarios. Third, we observe that COMPASS is the leading strategy across tasks and base algorithms, and that SABLE remains the SOTA base policy even when using inference-time search. Interestingly, under a small time budget, stochastic sampling outperforms tree search and online fine-tuning. We nevertheless show in the following section that this order can be reversed when more budget is provided.

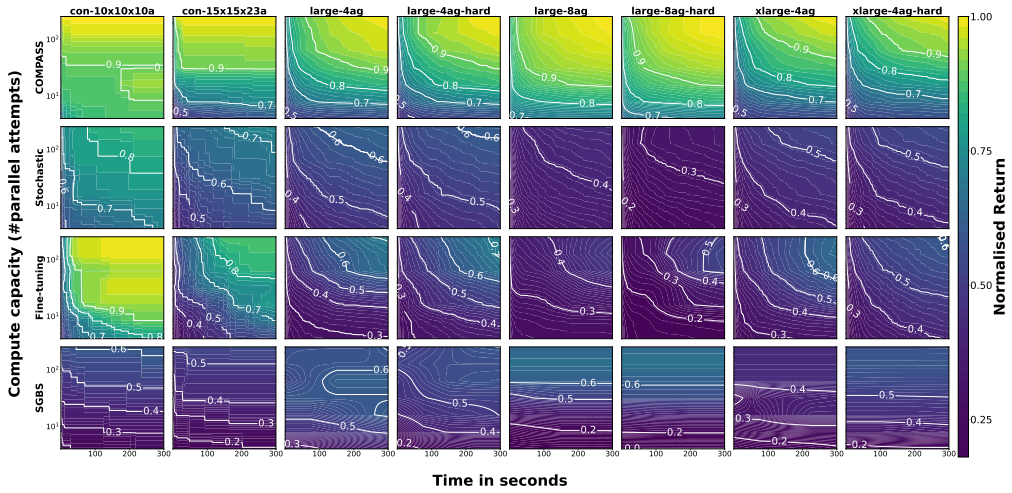

Figure 6: **Performance contour plots of Sable with different inference strategies** on a range of time budget (x-axis) and compute capacities (y-axis), for a set of representative tasks. Colours indicates performance, with brighter colours indicating higher values.

4.2 Mapping performance with compute and time budget

A recurrent limitation in previous work on inference strategies is the use of a fixed budget during evaluation, creating a narrow view on methods, and often creating a bias towards certain types of methods. In this section, we aim at providing a much broader perspective over inference-time search by reporting performance over a grid of time and compute budgets.

Experiments We choose a maximum time of 300 seconds, and evaluate all inference strategies using the leading base policy (SABLE), with a compute budget of $\{4, 8, 16, 32, 64, 128, 256\}$. All in all, we have 4 inference strategies, 7 possible compute budgets, 17 tasks, and 128 seeds per task, leading to 60 928 evaluated episodes. This constitutes the largest study released on inference strategies. We report these results using contour plots, where the x-axis is time, y-axis the number of parallel attempts allowed (our proxy for compute) and colour corresponds to the performance achieved (win-rate when accessible or *min-max* normalised return) going from dark purple (*min*) to yellow (*max*). We keep the 8 hardest tasks, the lower half based on the zero-shot performance of the converged SABLE checkpoints (see Fig. 4), on Fig. 6 and defer remaining tasks to Appendix A.

Discussion As a sanity check, we remark that performance always increases (colours become lighter) when time or compute increases (going towards the upper right corner). We now highlight three main observations. First, COMPASS demonstrates impressive versatility and achieves significant gains over other inference strategies, dominating all maps, except for `con-10x10x10a`, where it gets slightly outperformed by online fine-tuning. Second, we observe high variance for online fine-tuning: getting close to COMPASS for large budgets on `con-10x10x10a`, yet struggling to match stochastic sampling on others (e.g., RWARE's `large-8ag`). This shows that fine-tuning can be detrimental by reducing the number of attempts made within the time budget. Plus, policy gradients can be unstable (small batch size) or converge to local optima. This observation disproves the common belief that inference-time search is as trivial as over-fitting to the problem instance. Finally, we observe that SGBS has good performance for high compute capacity and low time, but the greediness of its simulation step impacts its exploration capacity, preventing to benefit from additional time budget.

4.3 Scaling with increasing budget

In practical applications, time is often more restricted than compute: a couple of seconds or minutes can be allowed, sometimes a few hours (train scheduling for the next day), but rarely more. Being able to improve solutions' quality under a fixed time budget by scaling compute is a valuable property. Consequently, methods that can scale with the compute available are particularly valuable. We look at this property in this section.

Experiments We keep the time budget fixed, 300 seconds, and we plot the final performance for each possible compute budget (still using the number of parallel attempts as a proxy). We use SABLE as the base policy and evaluate all the inference strategies across the 8 hardest tasks. We report results per strategy, and aggregated over the tasks on Fig. 7.

Discussion As expected, stochastic sampling has the lowest scaling coefficient: its search solely relies on chance, no adaptation or additional search is happening. On the other hand, we can see that online fine-tuning benefits from more compute, probably due to a better estimation of the policy gradient. It nevertheless requires a budget of 64 parallel attempts to clearly outperform stochastic sampling. On the other hand, COMPASS consistently provides a significant advantage. Its scaling trend seems linear at first, with a high coefficient, and only seems to decline for higher budgets, as performance limits are reached (over 95% win-rate when accessible, best-ever observed performance elsewhere). We have a two-fold explanation for these particularly impressive scaling properties: (i) the diversity contained in the latent space can be exploited by more parallelism, leading to a massive exploration of the solution space, even when the initial policy is far from optimal, and (ii) the higher the batch size, the better the CMA-ES search, enabling COMPASS to exploit even more information from any additional searching step allowed within the given time budget.

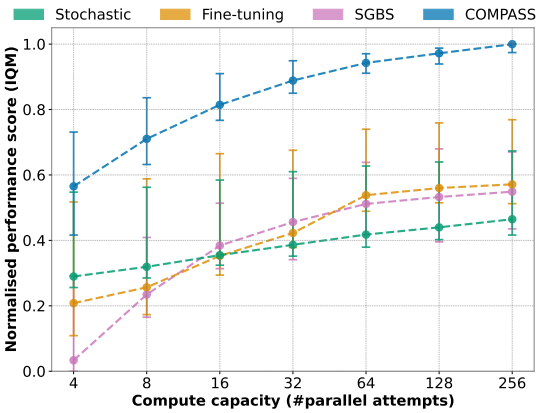

Figure 7: **Scaling of performance with respect to compute capacity, for a fixed time budget.** Compute capacity (x-axis) is set by limiting the number of parallel attempts. SABLE +COMPASS demonstrates impressive scaling properties, reaching the performance bounds of the benchmark.

5 Conclusion

In this work, we demonstrate that inference-time strategies are a critical and underutilized lever for boosting the performance of RL systems in complex tasks, using multi-agent RL as a representative test-bed. While training-time improvements have long dominated the field, our results show that inference-time search may offer significant performance gains using only a few seconds to minutes of additional wall-clock time during execution. We introduce a unified framework for inference strategies, extend it to the multi-agent setting, and empirically validate its effectiveness under varying compute and time budgets. Our large-scale evaluation, the most comprehensive to date on inference-time methods, reveals three **key takeaways**: (i) inference-time search with a relevant strategy yields significant improvements, even under tight time constraints; (ii) the gains depend on the inference budget, and our contour maps provide practitioners with practical guidance based on their constraints; and (iii) SABLE +COMPASS not only dominates the benchmark but also exhibits the most favourable scaling trends, making it particularly effective for increasingly complex decision-making problems.

Altogether, our findings call for a shift in how decision-making models are evaluated and deployed: inference-time strategies should be treated as core components of the solution pipeline, not as optional refinements. We hope our results and open-source tools will encourage broader adoption and inspire further innovation in the design of scalable inference-time algorithms.

Limitations and future work We focus on multi-agent RL to represent complex RL. We are aware that single-agent tasks can be arbitrarily complex. We believe that our main trends should generalise to the single-agent case, but leave it for future work. We intend two main future research directions. First, studying how to best combine existing paradigms together. Second, investigating how inference strategies reacts to evaluation out-of-distribution.

Acknowledgements

We would like to thank Guillaume Toujas-Bernate, Jake Lourie and Thomas Lecat for useful discussions on the use of inference strategies in real-world applications. We thank our MLOps team for developing our model training and experiment orchestration platform AIchor. We thank the Python and JAX communities for developing tools that made this research possible. Finally, we thank Google's TPU Research Cloud (TRC) for supporting our research with Cloud TPUs.

References

R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2108.13264.

T. Ahmad, D. Zhang, C. Huang, H. Zhang, N. Dai, Y. Song, and H. Chen. Artificial intelligence in sustainable energy industry: Status quo, challenges and opportunities. *Journal of Cleaner Production*, 289:125834, 2021. ISSN 0959-6526. doi: https://doi.org/10.1016/j.jclepro.2021.125834. URL https://www.sciencedirect.com/science/article/pii/S0959652621000548.

I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, page 32–37, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607099.

C. Bonnet, D. Luo, D. Byrne, S. Abramowitz, V. Coyette, P. Duckworth, D. Furelos-Blanco, N. Grinsztajn, T. Kalloniatis, V. Le, O. Mahjoub, L. Midgley, S. Surana, C. Waters, and A. Laterre. Jumanji: a suite of diverse and challenging reinforcement learning environments in jax, 2023. URL https://github.com/instadeepai/jumanji.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11), 2021. ISSN 2076-3417. doi: 10.3390/app11114948. URL https://www.mdpi.com/2076-3417/11/11/4948.

F. Chalumeau, R. Boige, B. Lim, V. Macé, M. Allard, A. Flajolet, A. Cully, and T. Pierrot. Neuroevolution is a competitive alternative to reinforcement learning for skill discovery. In *International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=6BHlZgyPOZY.

F. Chalumeau, S. Surana, C. Bonnet, N. Grinsztajn, A. Pretorius, A. Laterre, and T. D. Barrett. Combinatorial optimization with policy adaptation using latent space search. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

F. Chalumeau, B. Lim, R. Boige, M. Allard, L. Grillotti, M. Flageat, V. Macé, G. Richard, A. Flajolet, T. Pierrot, et al. Qdax: A library for quality-diversity and population-based algorithms with hardware acceleration. *Journal of Machine Learning Research*, 25(108):1–16, 2024a.

F. Chalumeau, R. Shabe, N. D. Nicola, A. Pretorius, T. D. Barrett, and N. Grinsztajn. Memory-enhanced neural solvers for efficient adaptation in combinatorial optimization, 2024b. URL https://arxiv.org/abs/2406.16424.

J. Choo, Y.-D. Kwon, J. Kim, J. Jae, A. Hottung, K. Tierney, and Y. Gwon. Simulation-guided beam search for neural combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2207.06190.

R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

A. Cully and Y. Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2017.

R. de Kock, O. Mahjoub, S. Abramowitz, W. Khlifi, C. R. Tilbury, C. Formanek, A. Smit, and A. Pretorius. Mava: a research library for distributed multi-agent reinforcement learning in jax. *arXiv preprint arXiv:2107.01460*, 2021.

C. S. de Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. S. Torr, M. Sun, and S. Whiteson. Is independent learning all you need in the starcraft multi-agent challenge?, 2020. URL https://arxiv.org/abs/2011.09533.

DeepMind, I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, A. Dedieu, C. Fantacci, J. Godwin, C. Jones, R. Hemsley, T. Hennigan, M. Hessel, S. Hou, S. Kapturowski, T. Keck, I. Kemaev, M. King, M. Kunesch, L. Martens, H. Merzic, V. Mikulik, T. Norman, G. Papamakarios, J. Quan, R. Ring, F. Ruiz, A. Sanchez, L. Sartran, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, M. Stanojević, W. Stokowiec, L. Wang, G. Zhou, and F. Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/google-deepmind.

J. Dona, A. Flajolet, A. Marginean, A. Cully, and T. Pierrot. Quality-diversity for one-shot biological sequence design. In *ICML'24 Workshop ML for Life and Material Science: From Theory to Industry Applications*, 2024. URL https://openreview.net/forum?id=ZZPwFG5W7o.

G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. An empirical investigation of the challenges of real-world reinforcement learning. *CoRR*, abs/2003.11881, 2020. URL https://arxiv.org/abs/2003.11881.

B. Ellis, J. Cook, S. Moalla, M. Samvelyan, M. Sun, A. Mahajan, J. Foerster, and S. Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36:37567–37593, 2023.

B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

N. Grinsztajn, D. Furelos-Blanco, S. Surana, C. Bonnet, and T. D. Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

R. Hammond, D. Craggs, M. Guo, J. N. Foerster, and I. Reid. Symmetry-breaking augmentations for ad hoc teamwork. In *ICLR 2025 Workshop on Bidirectional Human-AI Alignment*, 2025. URL https://openreview.net/forum?id=pEQwTKmcks.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

T. Hayes, R. Rao, H. Akin, N. J. Sofroniew, D. Oktay, Z. Lin, R. Verkuil, V. Q. Tran, J. Deaton, M. Wiggert, R. Badkundri, I. Shafkat, J. Gong, A. Derry, R. S. Molina, N. Thomas, Y. A. Khan, C. Mishra, C. Kim, L. J. Bartie, M. Nemeth, P. D. Hsu, T. Sercu, S. Candido, and A. Rives. Simulating 500 million years of evolution with a language model. *Science*, 387(6736):850–858, 2025. doi: 10.1126/science.ads0018. URL https://www.science.org/doi/abs/10.1126/science.ads0018.

J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee. Flax: A neural network library and ecosystem for JAX, 2024. URL http://github.com/google/flax.

A. Hottung, Y.-D. Kwon, and K. Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

A. Hottung, M. Mahajan, and K. Tierney. Polynet: Learning diverse solution strategies for neural combinatorial optimization. *arXiv preprint arXiv:2402.14048*, Feb 2024.

H. Hu, A. Lerer, A. Peysakhovich, and J. Foerster. "other-play " for zero-shot coordination. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

A. Hundt, B. Killeen, H. Kwon, C. Paxton, and G. Hager. "good robot!": Efficient reinforcement learning for multi-step visual tasks via reward shaping. 09 2019. doi: 10.48550/arXiv.1909.11730.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

S. Kumar, A. Kumar, S. Levine, and C. Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. *Advances in Neural Information Processing Systems*, 33:8198–8210, 2020.

A. Laterre, Y. Fu, M. K. Jabri, A.-S. Cohen, D. Kas, K. Hajjar, T. S. Dahl, A. Kerkeni, and K. Beguir. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization, 2018. URL https://arxiv.org/abs/1807.01672.

W. Long, W. Wen, P. Zhai, and L. Zhang. Role play: Learning adaptive role-specific strategies in multi-agent interactions, 2024. URL https://arxiv.org/abs/2411.01166.

A. Lupu, B. Cui, H. Hu, and J. Foerster. Trajectory diversity for zero-shot coordination. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7204–7213. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/lupu21a.html.

O. Mahjoub, S. Abramowitz, R. de Kock, W. Khlifi, S. du Toit, J. Daniel, L. B. Nessir, L. Beyers, C. Formanek, L. Clark, and A. Pretorius. Sable: a performant, efficient and scalable sequence model for marl. In *Proceedings of the 42th International Conference on Machine Learning*, ICML'25, 2025.

R. Mirsky, I. Carlucho, A. Rahman, E. Fosong, W. Macke, M. Sridharan, P. Stone, and S. V. Albrecht. A survey of ad hoc teamwork research. In *Multi-Agent Systems: 19th European Conference, EUMAS 2022, Düsseldorf, Germany, September 14–16, 2022, Proceedings*, page 275–293, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-20613-9. doi: 10.1007/978-3-031-20614-6_16. URL https://doi.org/10.1007/978-3-031-20614-6_16.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen. Molecular de novo design through deep reinforcement learning. *Journal of Cheminformatics*, 9, 09 2017. doi: 10.1186/s13321-017-0235-x.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021. URL http://arxiv.org/abs/2006.07869.

M. Popova, O. Isayev, and A. Tropsha. Deep reinforcement learning for de novo drug design. *Science Advances*, 4(7):eaap7885, 2018. doi: 10.1126/sciadv.aap7885. URL https://www.science.org/doi/abs/10.1126/sciadv.aap7885.

K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real. pages 11154–11163, 06 2020. doi: 10.1109/CVPR42600.2020.01117.

C. Ruhdorfer, M. Bortoletto, A. Penzkofer, and A. Bulling. The overcooked generalisation challenge, 2025. URL https://arxiv.org/abs/2406.17949.

A. Rutherford, B. Ellis, M. Gallici, J. Cook, A. Lupu, G. Ingvarsson, T. Willi, A. Khan, C. S. de Witt, A. Souly, et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023.

M. Samvelyan, T. Rashid, C. S. De Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

C. V. Snell, J. Lee, K. Xu, and A. Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano. Learning to summarize from human feedback. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

C. Wang, A. Rahman, I. Durugkar, E. Liebman, and P. Stone. N-agent ad hoc teamwork. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=q7TxGUWlhD.

R. Wang, E. Zelikman, G. Poesia, Y. Pu, N. Haber, and N. Goodman. Hypothesis search: Inductive reasoning with language models. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=G7UtIGQmjm.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Y. Wu, Z. Sun, S. Li, S. Welleck, and Y. Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2025. URL https://arxiv.org/abs/2408.00724.

E. S. Yourdshahi, T. Pinder, G. Dhawan, L. S. Marcolino, and P. Angelov. Towards large scale ad-hoc teamwork. In *2018 IEEE International Conference on Agents (ICA)*, pages 44–49, 2018. doi: 10.1109/AGENTS.2018.8460136.

C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=YVXaxB6L2Pl.

Appendix

A Additional results

Section 4.2 presents the contour plots of Sable with all inference strategies on the hardest 8 tasks of the benchmark. We report the contour plots of the other 9 remaining tasks on Fig. 8.

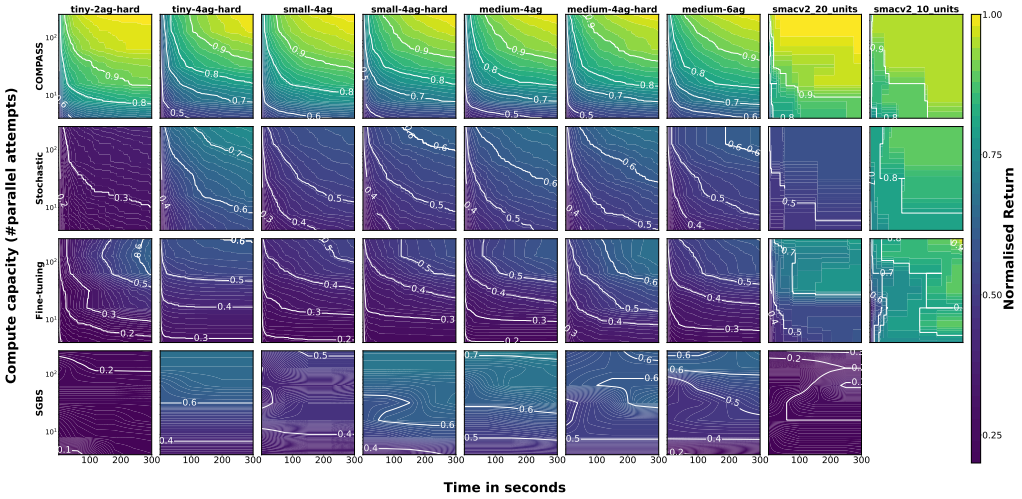

Figure 8: **Performance contour plots of Sable with different inference strategies on remaining 9 tasks** on a range of time budget (x-axis) and compute capacities (y-axis). Colours indicates performance, with brighter colours indicating higher values.

Fig. 8, shows similar trends to those illustrated in Section 4.2. SABLE +COMPASS leads to the best overall performance. Stochastic sampling provides a good and robust baseline overall. Finally, SGBS and online fine-tuning suffer from variance: despite being able to compete close to COMPASS on a couple tasks, they often get outperformed by stochastic sampling. We do not report the results for SABLE +SGBS on `smacv2_10_units` because we found an issue, but the figure will be updated shortly with new results.

B Details about the tasks

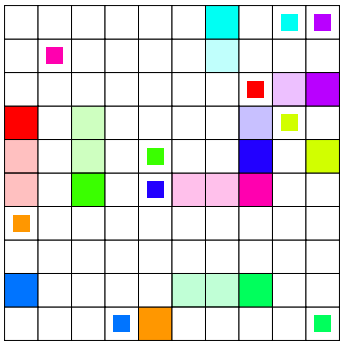

Figure 9: Environment rendering for Connector. Task name: `con-10x10-10a`. Image from Mahjoub et al. (2025).

Connector is an environment designed to model the routing of a printed circuit board (PCB). Agents start at randomly generated initial positions and have to reach a goal position without overlapping

with each other's paths (indicated as lower opacity coloured cells in Figure 9). Each agent has a partial view with a fixed field of view around itself as well as its current (x, y)-coordinate on the grid and the (x, y)-coordinate of its goal location. At each timestep, agents receive a small negative reward -0.03 and receive a reward of 1 for successfully connecting to a goal location. The particular difficulty in this environment stems from the fact that agents have to select actions carefully so that they not only reach their goal greedily but so that all agents can ultimately reach their goals.

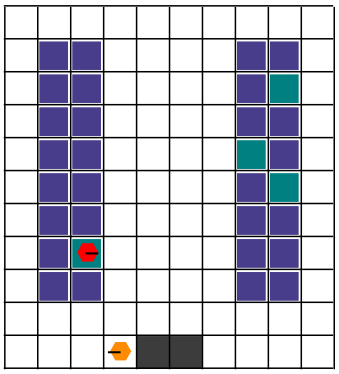

Figure 10: Environment rendering for Robot Warehouse. Task name: `tiny-2ag`. Image from Mahjoub et al. (2025).

RWARE is an environment where agents need to coordinate to deliver shelves (green cells in Figure 10) to a goal location (dark grey cells in Figure 10) in a warehouse. The reward is sparse since agents only receiving a reward of 1 for a successful shelf delivery to the goal location and 0 otherwise. This sparsity makes the environment particularly difficult since agents have to complete a long sequence of correct actions in order to receive any reward signal.

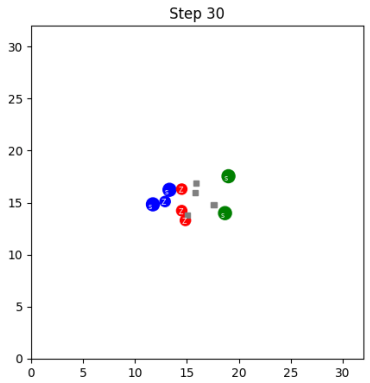

Figure 11: Environment rendering for SMAX. Task name: `2s3z`. Image from Mahjoub et al. (2025).

SMAC is an environment where agents need to coordinate and develop the correct micro strategies in order to defeat an enemy team. It was shown that the original benchmark was overly simplistic and for this reason SMACv2 (Ellis et al., 2023) was developed in order to address the shortcomings of the original benchmark. SMACv2 randomly generates a team of allied units by sampling from a selection of unit types and randomly generates ally starting positions at the start of each episode. The need for generalisation is what gives this benchmark its difficulty.

JAX-based implementations. Since Connector was developed as part of Jumanji (Bonnet et al., 2023), we make use of the JAX-based implementation directly. For RWARE we use the JAX-based reimplementation from Jumanji and for SMAC and SMACv2 we make use of the JAX-based reimplementation from JaxMARL (Rutherford et al., 2023), named SMAX. For a detailed discussion on task naming conventions we refer the reader to the Appendix of Mahjoub et al. (2025).

C Hyper-parameters

In this section, we report the hyper-parameters used in our experiments. They can also be found in the submitted code files.

Training - base policies Hyper-parameters used to train the base policies for this benchmark are taken from Mahjoub et al. (2025). These hyper-parameters were tuned on each task with a budget of 40 trials using the Tree-structured Parzen Estimator (TPE) Bayesian optimisation algorithm, and are reported in Mahjoub et al. (2025).

Hyper-parameters - COMPASS Training Hyper-parameters used to train all the COMPASS checkpoints can be found in Table 1. Most of them were kept close to the original hyper-parameters used in Chalumeau et al. (2023b).

Table 1: Hyper-parameters used for COMPASS' training phase. The same values are used for all three base policies (i.e., SABLE, IPPO, MAPPO).

Parameter	Value
Instances batch size	128
Latent space sample size	64
Latent space dimension size	16
Latent amplifier	1
Padding with random weights	True
Weight noise amplitude	0.01

Hyper-parameters - Inference-time search Most values are directly taken from original works or kept close to these. When different, we tried to use guidance from original work to decide how to set them. The values for the CMA-ES search used with COMPASS, for all base policies, can be found in Table 2. The values for online fine-tuning (all base policies) are reported in Table 3. Hyper-parameters for SGBS (all base policies) are reported in Table 4, and for stochastic sampling in Table 5.

Table 2: Hyper-parameters of the CMA-ES process used to search COMPASS' latent space at inference time. The same parameters are used for all tasks and all base policies (i.e., SABLE, MAPPO and IPPO).

Parameter	Number of attempts						
	4	**8**	**16**	**32**	**64**	**128**	**256**
Latent sample size	4	8	16	32	64	128	256
Number of elites	2	4	8	16	32	64	128
Covariance matrix step size	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Initial sigma	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Num. components	1	1	1	1	1	1	1

Table 3: Hyper-parameters used when doing online fine-tuning at inference-time with SABLE, MAPPO and IPPO.

Parameter	Task Name	Sable	IPPO	MAPPO
Max. trajectory size	All tasks	64	64	64
Learning rate	con-10x10x10a	0.001	0.00025	0.0001
	con-15x15x23a	0.001	0.0001	0.0001
	large-4ag	0.001	0.0005	0.0001
	large-4ag-hard	0.001	0.0005	0.00025
	large-8ag	0.001	0.00025	0.0001
	large-8ag-hard	0.001	0.00025	0.0005
	medium-4ag	0.001	0.00025	0.00025
	medium-4ag-hard	0.0005	0.0005	0.00025
	medium-6ag	0.0005	0.0001	0.00025
	smacv2_10_units	0.00025	0.0005	0.00025
	smacv2_20_units	0.001	0.0005	0.0005
	smacv2_5_units	0.001	0.00025	0.00025
	small-4ag	0.0005	0.0005	0.0005
	small-4ag-hard	0.001	0.0001	0.00025
	tiny-2ag-hard	0.001	0.00025	0.00025
	tiny-4ag-hard	0.0005	0.0005	0.0005
	xlarge-4ag	0.001	0.0005	0.0005
	xlarge-4ag-hard	0.0001	0.0005	0.0001
Entropy coefficient	Connector	0.05	0.00	0.00
	RWARE	0.0	0.0	0.0
	SMAX	0.0	0.0	0.0

Table 4: Hyper-parameters for Simulation Guided Beam Search (SGBS). The same values are used for SABLE, IPPO and MAPPO. Values are chosen such that beam width and top-k use fully the number of attempts allowed by batch.

Parameter	Number of attempts					
	4	16	32	64	128	256
Beam width	4	4	4	4	4	4
Top k	1	4	8	16	32	64

Table 5: To sample stochastically from a policy (SABLE, IPPO or MAPPO), we always use the same temperature as the one used during training: 1.0.

Parameter	Value
Evaluation greedy	False
Temperature	1.0

D Experimental stack

All algorithmic implementations are built by extending the JAX-based research library, `Mava` (de Kock et al., 2021). Our version of CMA-ES used for searching COMPASS's latent space comes from the JAX-based quality diversity library, `QDAX` (Chalumeau et al., 2024a). Both these libraries are built to leverage the DeepMind JAX ecosystem (DeepMind et al., 2020) and use `Flax` (Heek et al., 2024) for building neural networks, `optax` for optimisers and `orbax` for model checkpointing.

E Additional details about training

E.1 Base policy training

In Section 3, we explain that in our setting, it is assumed that all methods can be trained until convergence. For transparency and comparison with previous results in the literature, we report in Table 6 the training budget given to the methods to reach convergence, in environment steps.

Table 6: Training budgets (in environment steps) for each method across tasks.

Task Name	Sable	MAPPO	IPPO
con-10x10x10a	100M	200M	200M
con-15x15x23a	100M	200M	200M
large-4ag	600M	200M	200M
large-4ag-hard	600M	400M	200M
large-8ag	600M	400M	200M
large-8ag-hard	600M	200M	200M
medium-4ag	400M	200M	200M
medium-4ag-hard	600M	200M	200M
medium-6ag	600M	200M	200M
small-4ag	200M	200M	200M
small-4ag-hard	400M	200M	200M
tiny-2ag-hard	100M	200M	200M
tiny-4ag-hard	200M	200M	200M
xlarge-4ag	200M	400M	200M
xlarge-4ag-hard	200M	200M	200M
smacv2_10_units	100M	200M	200M
smacv2_20_units	600M	200M	200M

E.2 COMPASS training

In this section, we provide details about COMPASS and our implementation. For a more in-depth explanation, we refer the reader to the original paper (Chalumeau et al., 2023b). Our implementation is available in the submitted code files.

COMPASS is a method, which (i) uses a pre-trained base policy and modifies it to be conditioned by a latent vector (ii) re-trains it to enable latent vectors to create policies that are diversified and specialised for distinct training sub-distributions. At inference-time, this enables to search for the latent vector which performs best on the new instance to be solved.

Reincarnating a pre-trained base policy COMPASS checkpoints are not trained from scratch, they reincarnate an existing base policy, add parameters to this policy in order to process the latent vector which can be given as additional input. Hence, one must choose where to inject the latent vector in the existing architecture, and must modify the existing parameters of the neural policy to account for the new shape of the input. The latent vector must be input in a way that enables to diversify the actions produced for the same observation (i.e., making sure that different latent vectors create different policies). In practice, we concatenate the latent vector to the observations. When relevant, we make sure that this is done after observation normalisation layers.

In the original work, the additional weights added to the base neural network are initialised with zeros. In our case, we observed that this could not provide enough diversification in the first training step, preventing any emergence of specialisation, leading to no benefit. We found that initialising these parameters with random uniform values between -0.01 and 0.01 fixed the issue across all tasks.

Creating diversity and specialisation in the latent space Similarly to the original method, the latent space is not learned, it is a fixed prior, and the network's weights are learned to create diversity from this prior space. We also use a similar prior, which is a uniform distribution between -1 and 1, over 16 dimensions. We do not need any amplification of the latent vector (i.e., we multiply by 1, whereas the original work multiplies it by 100). At each training step, a batch of instances is

sampled from the training distribution, a batch of latent vectors is sampled from that latent space, the conditioned policy is evaluated for each instance, for each latent vector, and only the best performing latent vectors, for each instance, are used in the computation of the loss. This creates the specialisation (diversification). For additional motivation and mathematical formulation, we refer the reader to Chalumeau et al. (2023b). Like all training processes in our experimental study, the method is allowed to train until convergence. In practice, each COMPASS checkpoint was trained for 100 million steps.

F Pseudo-algorithms

In this section, we provide algorithmic descriptions for the inference strategies used in the paper, showing how the policy is used at inference time, under the time and budget constraints. Stochastic sampling is explained in Algorithm 1, SGBS in Algorithm 2, online fine-tuning in Algorithm 3 and COMPASS in Algorithm 4.

Algorithms use the terminology *trajectory* τ, which is equivalent to *solution*. The symbol $\circ$ is used for the concatenation of a new step transition to a partial trajectory.

Algorithm 1 Stochastic Sampling

Input: policy $\boldsymbol{\pi}_\theta$, instance ρ, time budget $T_{\max}$, compute capacity $B_{\max}$, reward function $\mathcal{R}$
Initialize best trajectory $\tau^* \leftarrow \emptyset$, best score $R^* \leftarrow -\infty$
while elapsed time $< T_{\max}$ **do**
  for parallel rollout $b = 1$ to $B_{\max}$ **do**
    Initialize trajectory $\tau_b \leftarrow \emptyset$
    for step $t = 1$ to H **do**
      Observe $\boldsymbol{o}_t$ from environment copy ρ_b
      Sample $\boldsymbol{a}_t \sim \boldsymbol{\pi}_\theta(\cdot \mid \boldsymbol{o}_t)$
      Execute $\boldsymbol{a}_t$ in ρ_b, observe $\boldsymbol{o}_{t+1}$
      Append $(\boldsymbol{o}_t, \boldsymbol{a}_t)$ to τ_b
    $R_b \leftarrow \mathcal{R}(\tau_b)$
    if $R_b > R^*$ **then**
      $\tau^* \leftarrow \tau_b, \quad R^* \leftarrow R_b$
return τ^*

Algorithm 2 Simulation-Guided Beam Search

Input: policy $\boldsymbol{\pi}_\theta$, instance ρ, time budget $T_{\max}$, compute capacity $B_{\max}$, beam width K, reward function $\mathcal{R}$
Initialize beam $\mathcal{B} \leftarrow \{\emptyset\}$, best trajectory $\tau^* \leftarrow \emptyset$, best score $R^* \leftarrow -\infty$
while elapsed time $< T_{\max}$ **do**
  $\mathcal{B}_{\text{new}} \leftarrow \emptyset$
  for partial trajectory τ in $\mathcal{B}$ **do**
    for sample $b = 1$ to $B_{\max}/|\mathcal{B}|$ **do**
      Let $\boldsymbol{o}_t$ be the observation at the end of τ
      Sample $\boldsymbol{a}_t \sim \boldsymbol{\pi}_\theta(\cdot \mid \boldsymbol{o}_t)$
      Simulate greedy rollout from $\tau \circ \boldsymbol{a}_t$ using $\boldsymbol{\pi}_\theta$
      Let $\hat{\tau}_b$ be the resulting full trajectory
      $R_b \leftarrow \mathcal{R}(\hat{\tau}_b)$
      Add $(\tau \circ \boldsymbol{a}_t, R_b)$ to $\mathcal{B}_{\text{new}}$
      if $R_b > R^*$ **then**
        $\tau^* \leftarrow \hat{\tau}_b, \quad R^* \leftarrow R_b$
  Prune top-K trajectories from $\mathcal{B}_{\text{new}}$ into $\mathcal{B}$
return τ^*

Algorithm 3 Online Fine-Tuning

Input: base policy $\boldsymbol{\pi}_\theta$, instance ρ, time budget $T_{\max}$, compute capacity $B_{\max}$, reward function $\mathcal{R}$, learning rate α
Initialize best trajectory $\tau^* \leftarrow \emptyset$, best score $R^* \leftarrow -\infty$
Initialize adapted parameters $\theta' \leftarrow \theta$
while elapsed time $< T_{\max}$ **do**
  for parallel rollout $b = 1$ to $B_{\max}$ **do**
    Rollout $\tau_b \sim \boldsymbol{\pi}_{\theta'}$ on ρ
    $R_b \leftarrow \mathcal{R}(\tau_b)$
    if $R_b > R^*$ **then**
      $\tau^* \leftarrow \tau_b, \quad R^* \leftarrow R_b$
  Compute gradient: $\nabla_\theta \leftarrow \frac{1}{B_{\max}} \sum_{b=1}^{B_{\max}} \nabla_\theta \log \boldsymbol{\pi}_{\theta'}(\tau_b) \cdot R_b$
  Update parameters: $\theta' \leftarrow \theta' + \alpha \nabla_\theta$
return τ^*

Algorithm 4 COMPASS + CMA-ES search

Input: latent-conditioned policy $\boldsymbol{\pi}_\theta(\cdot \mid \boldsymbol{o}, z)$, instance ρ, time budget $T_{\max}$, compute capacity $B_{\max}$, reward function $\mathcal{R}$
Initialize CMA-ES: mean μ, covariance Σ
Initialize best trajectory $\tau^* \leftarrow \emptyset$, best score $R^* \leftarrow -\infty$
while elapsed time $< T_{\max}$ **do**
  Sample $\{z_b\}_{b=1}^{B_{\max}} \sim \mathcal{N}(\mu, \Sigma)$
  for each z_b in parallel **do**
    Rollout $\tau_b \sim \boldsymbol{\pi}_\theta(\cdot \mid \boldsymbol{o}, z_b)$ on ρ
    $R_b \leftarrow \mathcal{R}(\tau_b)$
    if $R_b > R^*$ **then**
      $\tau^* \leftarrow \tau_b, \quad R^* \leftarrow R_b$
  Update (μ, Σ) using CMA-ES with $\{(z_b, R_b)\}_{b=1}^{B_{\max}}$
return τ^*

G Discussion about the inference-time budget

Choice of the budget used in the experiments We create our experimental setting to be similar to practical use cases. Hence, we use time as the main budget constraint, instead of number of attempts (contrary to most literature on the topic). This enables to account for the cost of search and adaptation. Even though we always evaluate methods on 128 instances for each task, we ensure that each instance is solved fully independently, to avoid that the vectorisation process creates a bias towards one method or another. We choose to study methods with time budgets ranging from 30 to 300 seconds. The lower bound is taken to be small, to show that even short inference-time searches can provide significant benefit. The higher bound is chosen to be high enough to see which method can really benefit from a longer search budget. Concerning the compute capacity, we choose a range of values that seems achievable in practice. In real-world scenarios, simulations can get more complex and require one CPU core for one simulation. We hence believed that most use cases would fit within the limit of 256 simulations in parallel. Nevertheless, our code works for higher values and can be used to obtain results in new settings (both in terms of time and parallel attempts).

Total number of attempts achieved within time budget To remain consistent with previous literature, and provide a better understanding of the search and adaptation cost of each inference strategy and each base policy, we provide the number of attempts achieved within the time limit, for all tasks and compute capacity (i.e., number of parallel attempts allowed). Values for SABLE within 300 seconds are reported on Table 7 for COMPASS, Table 8 for online fine-tuning, Table 9 for stochastic sampling. SGBS builds new solutions from existing partial solutions, hence the count of the attempted solution cannot be obtained in the same way as other methods.

Values for all base policies within 30 seconds are reported on Table 10, Table 12 and Table 11.

Table 7: Number of attempts (solutions generated) when using SABLE + COMPASS during 300 seconds of inference-time search.

Task Name	Number of attempts						
	4	**8**	**16**	**32**	**64**	**128**	**256**
con-10x10x10a	9960	19448	37520	67296	131456	248320	422144
con-15x15x23a	1608	3112	7744	14144	27456	53248	80896
large-4ag	2608	5208	9360	16896	30464	44032	59392
large-4ag-hard	2596	5000	9360	16832	29568	44288	59904
large-8ag	1612	3200	6240	10304	17280	23424	30976
large-8ag-hard	1708	3144	6096	10624	18880	29056	39680
medium-4ag	2088	4072	7904	15104	29376	48256	87296
medium-4ag-hard	2072	4040	7888	16992	29632	48384	87552
medium-6ag	1536	3016	5728	10816	21440	37760	74240
smacv2_10_units	7628	13248	26176	45088	74304	122624	174336
smacv2_20_units	5824	10428	18528	34112	64064	111232	180992
small-4ag	1964	3880	8672	14112	28096	61568	83456
small-4ag-hard	1876	4216	7104	15488	30464	52096	93696
tiny-2ag-hard	2736	6072	11856	22496	44544	77440	147200
tiny-4ag-hard	1384	2648	5024	10624	16640	32256	45568
xlarge-4ag	1816	3568	7952	15264	24320	45440	81152
xlarge-4ag-hard	1180	2264	5344	9696	17792	28544	51200

Table 8: Number of attempts (solutions generated) when using online fine-tuning and SABLE during 300 seconds of inference-time search.

Task Name	Number of attempts						
	4	**8**	**16**	**32**	**64**	**128**	**256**
con-10x10x10a	10608	20256	37744	64448	115392	226560	395776
con-15x15x23a	2136	3952	6864	10496	16592	32000	60928
large-4ag	1792	3296	6560	9920	13376	24448	49664
large-4ag-hard	2064	3416	6448	8832	13248	24448	46592
large-8ag	1416	2336	4592	7552	11136	19200	34304
large-8ag-hard	1140	2576	4064	6752	12736	26624	44032
medium-4ag	2000	4176	7472	9920	15744	28160	48384
medium-4ag-hard	2024	3776	7232	10816	14848	28160	48384
medium-6ag	1700	2872	6448	8704	21120	32896	65792
smacv2_10_units	6848	12576	20176	37536	68160	156672	235776
smacv2_20_units	5700	10584	18176	25216	47488	104382	155904
small-4ag	1940	3688	7072	9600	14272	30336	49408
small-4ag-hard	1940	3688	7072	9600	14272	30336	49408
tiny-2ag-hard	2572	5600	9824	13344	22848	48768	86272
tiny-4ag-hard	2572	5568	9808	13344	22784	48768	86272
xlarge-4ag	1768	3856	6368	9792	13760	24448	46080
xlarge-4ag-hard	1280	2024	3776	6368	11456	21504	40192

Table 9: Number of attempts (solutions generated) when sampling stochastically from SABLE during 300 seconds of inference-time search.

Task Name	Number of attempts						
	4	**8**	**16**	**32**	**64**	**128**	**256**
con-10x10x10a	10632	20504	39524	70720	138624	264960	448512
con-15x15x23a	2048	4056	7840	14144	27648	54144	101376
large-4ag	2092	3624	7808	15456	30336	46464	82944
large-4ag-hard	2108	3696	8112	13344	26816	46592	82176
large-8ag	1508	2552	5824	9568	17984	30720	52480
large-8ag-hard	1324	2256	4432	8416	16576	34048	32208
medium-4ag	2388	4576	8064	15296	29760	53120	81664
medium-4ag-hard	2144	4616	7968	16992	29696	52992	88576
medium-6ag	1344	3496	6768	10848	25280	66048	66048
smacv2_10_units	7168	11056	25376	47088	94720	148608	261888
smacv2_20_units	4848	9144	17584	34688	55232	102144	228608
small-4ag	1472	3896	7072	16448	27840	61056	84224
small-4ag-hard	1812	3704	7072	16448	26880	59776	84224
tiny-2ag-hard	2824	6168	10432	20192	45504	77952	165632
tiny-4ag-hard	1600	2672	5888	9184	16960	33152	58880
xlarge-4ag	1832	3624	7888	13248	27584	50944	81152
xlarge-4ag-hard	1164	2280	4480	8064	15296	33152	58880

Table 10: Number of attempts (solutions generated) when using COMPASS with SABLE, IPPO and MAPPO during 30 seconds of inference-time search.

Task Name	Number of attempts - Sable							PPO
	4	**8**	**16**	**32**	**64**	**128**	**256**	**64**
con-10x10x10a	996	1872	3744	6720	13120	24832	42240	65152
con-15x15x23a	160	384	768	1408	2752	5376	8192	36608
large-4ag	260	464	928	1664	3072	4480	5888	5056
large-4ag-hard	256	464	928	1664	2944	4480	6144	4928
large-8ag	160	312	624	1024	1728	2304	3072	3520
large-8ag-hard	168	304	608	1056	1856	2944	4096	3584
medium-4ag	208	392	784	1504	2944	4864	8704	5120
medium-4ag-hard	208	392	784	1696	2944	4864	8704	5056
medium-6ag	152	288	576	1088	2112	3840	7424	4416
smacv2_10_units	760	1304	2608	4480	7424	12288	17408	30464
smacv2_20_units	584	928	1856	3392	6400	11136	18176	26880
small-4ag	196	432	864	1408	2816	6144	8320	5248
small-4ag-hard	188	352	704	1536	3072	5120	9216	5248
tiny-2ag-hard	272	592	1184	2240	4480	7680	14592	6848
tiny-4ag-hard	136	248	496	1056	1664	3200	4608	5312
xlarge-4ag	180	392	784	1536	2432	4480	8192	4160
xlarge-4ag-hard	116	264	528	960	1792	2816	5120	4288

Table 11: Number of attempts (solutions generated) when using stochastic sampling with SABLE, IPPO and MAPPO during 30 seconds of inference-time search.

Task Name	Number of attempts - Sable							PPO
	4	8	16	32	64	128	256	64
con-10x10x10a	1060	2048	3940	7040	13760	26240	44800	103680
con-15x15x23a	204	400	780	1408	2752	5376	10112	51456
large-4ag	208	360	768	1536	3008	4608	8192	5312
large-4ag-hard	208	368	800	1312	2688	4608	8192	5184
large-8ag	148	248	576	928	1792	3072	5120	3904
large-8ag-hard	132	224	432	832	1664	3328	3200	3712
medium-4ag	236	456	800	1504	2944	5248	8128	5120
medium-4ag-hard	212	456	784	1696	2944	5248	8832	5312
medium-6ag	132	344	672	1056	2496	6528	6528	4480
smacv2_10_units	716	1104	2528	4704	9408	14848	26112	33408
smacv2_20_units	484	912	1744	3456	5504	10176	22784	31808
small-4ag	144	384	704	1632	2752	6016	8384	5184
small-4ag-hard	180	368	704	1632	2688	5888	8384	5248
tiny-2ag-hard	280	608	1040	2016	4544	7808	16512	4544
tiny-4ag-hard	160	264	576	896	1664	3328	5888	5312
xlarge-4ag	180	360	784	1312	2752	5120	8128	4416
xlarge-4ag-hard	116	224	448	800	1536	3328	5888	4352

Table 12: Number of attempts (solutions generated) when using online fine-tuning with SABLE, IPPO and MAPPO during 30 seconds of inference-time search.

Task Name	Number of attempts - Sable							PPO
	4	8	16	32	64	128	256	64
con-10x10x10a	1060	2024	3760	6432	11520	22656	39424	23680
con-15x15x23a	212	392	672	1024	1600	3200	5888	5888
large-4ag	176	328	656	992	1280	2432	4864	3136
large-4ag-hard	204	336	640	864	1280	2432	4608	2944
large-8ag	140	232	448	736	1088	1920	3328	2304
large-8ag-hard	112	256	400	672	1216	2560	4352	2112
medium-4ag	200	416	736	992	1536	2816	4608	2880
medium-4ag-hard	200	376	720	1056	1472	2816	4608	2880
medium-6ag	168	280	640	864	2112	3200	6400	4160
smacv2_10_units	684	1256	2016	3744	6784	15616	23552	8960
smacv2_20_units	568	1056	1808	2496	4736	10368	15360	7936
small-4ag	192	368	704	960	1408	2944	4864	2816
small-4ag-hard	192	368	704	960	1408	2944	4864	2816
tiny-2ag-hard	256	560	976	1312	2240	4864	8448	3904
tiny-4ag-hard	256	552	976	1312	2240	4864	8448	3904
xlarge-4ag	176	384	624	960	1344	2432	4608	2944
xlarge-4ag-hard	128	200	368	608	1088	2048	3840	2432