

883 Appendix

884 A Additional results

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

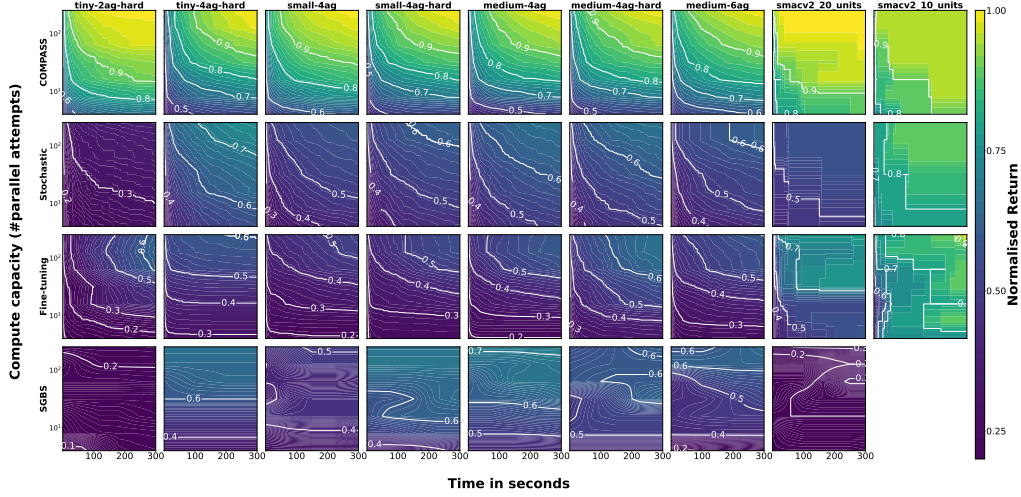


Figure 8: **Performance contour plots of Sable with different inference strategies on remaining 9 tasks** on a range of time budget (x -axis) and compute capacities (y -axis). Colours indicates performance, with brighter colours indicating higher values.

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

893 B Details about the tasks

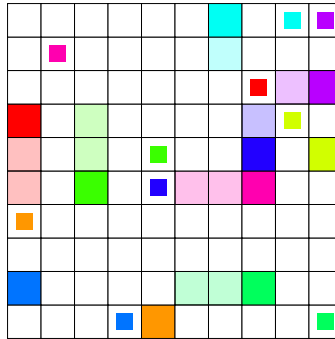


Figure 9: Environment rendering for Connector. Task name: con-10x10-10a. Image from [Mahjoub et al. \(2025\)](#).

894 **Connector** is an environment designed to model the routing of a printed circuit board (PCB). Agents
 895 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.

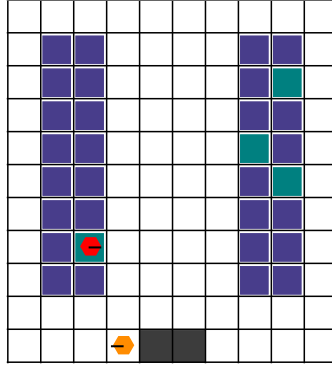


Figure 10: Environment rendering for Robot Warehouse. Task name: `tiny-2ag`. Image from [Mahjoub et al. \(2025\)](#).

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.

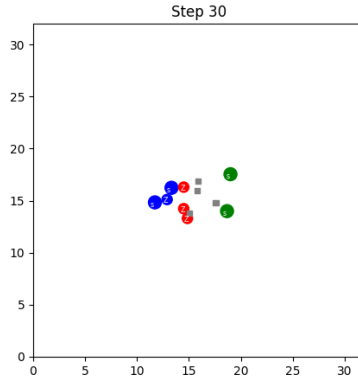


Figure 11: Environment rendering for SMAX. Task name: `2s3z`. Image from [Mahjoub et al. \(2025\)](#).

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. \(2025\)](#).

918 C Hyper-parameters

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

921 **Training - base policies** Hyper-parameters used to train the base policies for this benchmark are
 922 taken from Mahjoub et al. (2025). These hyper-parameters were tuned on each task with a budget of
 923 40 trials using the Tree-structured Parzen Estimator (TPE) Bayesian optimisation algorithm, and are
 924 reported in Mahjoub et al. (2025).

925 **Hyper-parameters - COMPASS Training** Hyper-parameters used to train all the COMPASS
 926 checkpoints can be found in Table 1. Most of them were kept close to the original hyper-parameters
 927 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

928 **Hyper-parameters - Inference-time search** Most values are directly taken from original works
 929 or kept close to these. When different, we tried to use guidance from original work to decide how
 930 to set them. The values for the CMA-ES search used with COMPASS, for all base policies, can
 931 be found in Table 2. The values for online fine-tuning (all base policies) are reported in Table 3.
 932 Hyper-parameters for SGBS (all base policies) are reported in Table 4, and for stochastic sampling
 933 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

934 D Experimental stack

935 All algorithmic implementations are built by extending the JAX-based research library, Mava (de Kock
936 et al., 2021). Our version of CMA-ES used for searching COMPASS’s latent space comes from the
937 JAX-based quality diversity library, QDAX (Chalumeau et al., 2024a). Both these libraries are built to
938 leverage the DeepMind JAX ecosystem (DeepMind et al., 2020) and use Flax (Heek et al., 2024) for
939 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

```

1: Input: policy  $\pi_\theta$ , instance  $\rho$ , time budget  $T_{\max}$ , compute capacity  $B_{\max}$ , reward function  $\mathcal{R}$ 
2: Initialize best trajectory  $\tau^* \leftarrow \emptyset$ , best score  $R^* \leftarrow -\infty$ 
3: while elapsed time  $< T_{\max}$  do
4:   for parallel rollout  $b = 1$  to  $B_{\max}$  do
5:     Initialize trajectory  $\tau_b \leftarrow \emptyset$ 
6:     for step  $t = 1$  to  $H$  do
7:       Observe  $\mathbf{o}_t$  from environment copy  $\rho_b$ 
8:       Sample  $\mathbf{a}_t \sim \pi_\theta(\cdot \mid \mathbf{o}_t)$ 
9:       Execute  $\mathbf{a}_t$  in  $\rho_b$ , observe  $\mathbf{o}_{t+1}$ 
10:      Append  $(\mathbf{o}_t, \mathbf{a}_t)$  to  $\tau_b$ 
11:       $R_b \leftarrow \mathcal{R}(\tau_b)$ 
12:      if  $R_b > R^*$  then
13:         $\tau^* \leftarrow \tau_b$ ,  $R^* \leftarrow R_b$ 
14: return  $\tau^*$ 

```

Algorithm 2 Simulation-Guided Beam Search

```

1: Input: policy  $\pi_\theta$ , instance  $\rho$ , time budget  $T_{\max}$ , compute capacity  $B_{\max}$ , beam width  $K$ , reward function  $\mathcal{R}$ 
2: Initialize beam  $\mathcal{B} \leftarrow \{\emptyset\}$ , best trajectory  $\tau^* \leftarrow \emptyset$ , best score  $R^* \leftarrow -\infty$ 
3: while elapsed time  $< T_{\max}$  do
4:    $\mathcal{B}_{\text{new}} \leftarrow \emptyset$ 
5:   for partial trajectory  $\tau$  in  $\mathcal{B}$  do
6:     for sample  $b = 1$  to  $B_{\max}/|\mathcal{B}|$  do
7:       Let  $\mathbf{o}_t$  be the observation at the end of  $\tau$ 
8:       Sample  $\mathbf{a}_t \sim \pi_\theta(\cdot \mid \mathbf{o}_t)$ 
9:       Simulate greedy rollout from  $\tau \circ \mathbf{a}_t$  using  $\pi_\theta$ 
10:      Let  $\hat{\tau}_b$  be the resulting full trajectory
11:       $R_b \leftarrow \mathcal{R}(\hat{\tau}_b)$ 
12:      Add  $(\tau \circ \mathbf{a}_t, R_b)$  to  $\mathcal{B}_{\text{new}}$ 
13:      if  $R_b > R^*$  then
14:         $\tau^* \leftarrow \hat{\tau}_b$ ,  $R^* \leftarrow R_b$ 
15:   Prune top- $K$  trajectories from  $\mathcal{B}_{\text{new}}$  into  $\mathcal{B}$ 
16: return  $\tau^*$ 

```

Algorithm 3 Online Fine-Tuning

```
1: Input: base policy  $\pi_\theta$ , instance  $\rho$ , time budget  $T_{\max}$ , compute capacity  $B_{\max}$ , reward function  $\mathcal{R}$ , learning rate  $\alpha$ 
2: Initialize best trajectory  $\tau^* \leftarrow \emptyset$ , best score  $R^* \leftarrow -\infty$ 
3: Initialize adapted parameters  $\theta' \leftarrow \theta$ 
4: while elapsed time  $< T_{\max}$  do
5:   for parallel rollout  $b = 1$  to  $B_{\max}$  do
6:     Rollout  $\tau_b \sim \pi_{\theta'}$  on  $\rho$ 
7:      $R_b \leftarrow \mathcal{R}(\tau_b)$ 
8:     if  $R_b > R^*$  then
9:        $\tau^* \leftarrow \tau_b$ ,  $R^* \leftarrow R_b$ 
10:    Compute gradient:  $\nabla_\theta \leftarrow \frac{1}{B_{\max}} \sum_{b=1}^{B_{\max}} \nabla_\theta \log \pi_{\theta'}(\tau_b) \cdot R_b$ 
11:    Update parameters:  $\theta' \leftarrow \theta' + \alpha \nabla_\theta$ 
12: return  $\tau^*$ 
```

Algorithm 4 COMPASS + CMA-ES search

```
1: Input: latent-conditioned policy  $\pi_\theta(\cdot \mid \mathbf{o}, z)$ , instance  $\rho$ , time budget  $T_{\max}$ , compute capacity  $B_{\max}$ , reward function  $\mathcal{R}$ 
2: Initialize CMA-ES: mean  $\mu$ , covariance  $\Sigma$ 
3: Initialize best trajectory  $\tau^* \leftarrow \emptyset$ , best score  $R^* \leftarrow -\infty$ 
4: while elapsed time  $< T_{\max}$  do
5:   Sample  $\{z_b\}_{b=1}^{B_{\max}} \sim \mathcal{N}(\mu, \Sigma)$ 
6:   for each  $z_b$  in parallel do
7:     Rollout  $\tau_b \sim \pi_\theta(\cdot \mid \mathbf{o}, z_b)$  on  $\rho$ 
8:      $R_b \leftarrow \mathcal{R}(\tau_b)$ 
9:     if  $R_b > R^*$  then
10:       $\tau^* \leftarrow \tau_b$ ,  $R^* \leftarrow R_b$ 
11:   Update  $(\mu, \Sigma)$  using CMA-ES with  $\{(z_b, R_b)\}_{b=1}^{B_{\max}}$ 
12: return  $\tau^*$ 
```

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