Efficient Automatic Tuning for Data-driven Model Predictive Control via Meta-Learning

Baoyu Li[1], William Edwards[1], Kris Hauser[1]

***Abstract*— `AutoMPC`[2] is a Python package that automates and optimizes data-driven model predictive control. However, it can be computationally expensive and unstable when exploring large search spaces using pure Bayesian Optimization (BO). To address these issues, this paper proposes to employ a meta-learning approach called *Portfolio* that improves `AutoMPC`'s efficiency and stability by warmstarting BO. Portfolio optimizes initial designs for BO using a diverse set of configurations from previous tasks and stabilizes the tuning process by fixing initial configurations instead of selecting them randomly. Experimental results demonstrate that Portfolio outperforms the pure BO in finding desirable solutions for `AutoMPC` within limited computational resources on 11 nonlinear control simulation benchmarks and 1 physical underwater soft robot dataset.**

I. Introduction

Data-driven model predictive control (MPC) is a robust and flexible framework for robotic controller design [1]. By leveraging knowledge of the data-learned dynamics system, it can predict a robot's behavior over a specified time horizon and determines the optimal control actions that will achieve the desired objectives while satisfying constraints. Due to its capacity to handle intricate nonlinear systems, constraints, and uncertainties in real-time, data-driven MPC has broad applications in robotics, including trajectory tracking [2], obstacle avoidance [3], and manipulation tasks [4].

However, successfully implementing data-driven MPC requires expert knowledge to make various design choices such as objective functions, system identification (SysID), and optimization techniques. Additionally, simple data-driven MPC necessitates intricate manual hyperparameter tuning for these design components, which can prove to be time-consuming and susceptible to errors.

To address these issues, the research community has made efforts to democratize data-driven MPC for users with no expert-level knowledge in the delicate MPC design process [5]–[7]. Most recently, Edwards et al. [7] proposed an end-to-end automatic tuning pipeline for data-driven MPC, named `AutoMPC`, which optimizes hyperparameters by Bayesian Optimization (BO) and evaluates performance via a cross-validation-like technique using surrogate dynamics. However, `AutoMPC` has a significant limitation: it begins from random initialization on every new problem and does not leverage knowledge from previous encountered tasks, leading to unstable tuning processes and sub-optimal performance for dynamics modeling and control.

Inspired by recent advances in AutoML [8], [9], we propose to apply an alternate meta-learning-based paradigm called *Portfolio* in this paper. Portfolio allows `AutoMPC` to adjust to new tasks rapidly, similar to how it warms up the AutoML system. It provides an optimized initial design for pure BO, drawn a diverse and dependable set of configurations from previous runs. By replacing the random initialization of pure BO with a method which leverages prior knowledge, Portfolio improves the stability and effectiveness of the tuning process. We employ portfolio-based meta-learning on model tuning, which serves as a major component in `AutoMPC` pipeline.

We conduct extensive experiments on various benchmarks to build a comprehensive Portfolio and validate the contributions of our proposed method. Besides the three benchmarks described in [7], we perform experiments on more nonlinear control benchmarks from the OpenAI Gym MuJoCo [10], Gym extensions [11], and underwater soft robot [12], [13]. Experimental results on 11 nonlinear control simulation benchmarks and 1 dataset collected from physical underwater soft robot demonstrate the efficacy of Portfolio.

II. Preliminaries

A. AutoMPC Tuning

As in [7], we consider full-observable, discrete-time dynamical systems of the form

$$x_{t+1} = f(x_t, u_t),$$

with state $x_t \in \mathbb{R}^n$ and control $u_t \in \mathbb{R}^m$. In general, we do not assume access to the ground-truth dynamics f, so the function must be approximated with a SysID algorithm to obtain a data-learned dynamics model $\hat{f}$.

`AutoMPC` manages design components and hyperparameters using configurations with the `ConfigSpace` library [14] and conducts Bayesian Optimization (BO) with the `SMAC3` library [15]. Tuning in `AutoMPC` can be performed either in a decoupled fashion, where model tuning is performed prior to control tuning, or fully end-to-end, with all hyperparameters being tuned simultaneously. Model tuning[3] only optimizes the SysID hyperparameters for model accuracy, while control tuning optimizes the control performance by tuning the hyperparameters of objective function and optimizer.

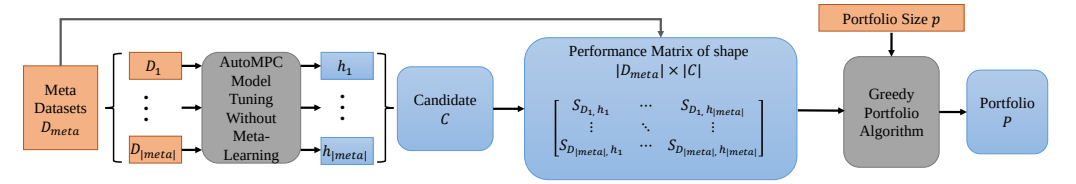
B. Meta Learning

Meta learning, also known as *learning to learn*, is aimed at facilitating effective learning by utilizing prior knowledge

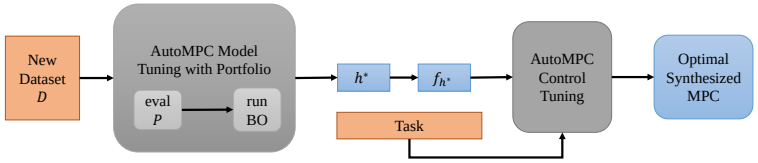
[1]B. Li, W. Edwards, and K. Hauser are with the Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA. {baoyul2,wre2,kkhauser}@illinois.edu

[2]The code and documentation for `AutoMPC` and its improved version are available at https://github.com/uiuc-iml/autompc.

[3]Note that model tuning and SysID tuning are synonymous in our paper.



(a) Build the Portfolio from meta datasets offline.



(b) Apply Portfolio to the AutoMPC pipeline.

Fig. 1: **Overview of AutoMPC with Portfolio.** Orange rectangular boxes refer to the input data and the blue rounded boxes refer to the output component in each process. (a) We build the Portfolio in a similar way that [8] does for AutoML. We first perform AutoMPC model tuning without meta-learning for each meta dataset D_i, obtaining the corresponding optimal model configuration h_i. These configurations form the candidate set C. Then, we evaluate the performance of C on D_{meta} and construct a performance matrix where S_{D_i,h_j} represents the score of configuration h_j on meta dataset D_i. Finally, we employ the Greedy Portfolio algorithm proposed by [8] to obtain the Portfolio P. (b) We utilize P as the initial configurations for BO in AutoMPC model tuning. It will return the best model configuration h^* and correspondent surrogate dynamics model f_{h^*}, which will be used for control tuning in AutoMPC and helps to achieve the optimal synthesized MPC.

of various datasets or tasks [16]. It has been applied in the context of MPC to enable fast fine-tuning for unseen tasks [17]. However, previous work only considers single dynamics model class such as Gaussian process regression, while in our work, we perform automatic selection of the model class. The search space for model tuning is thus very large, including both the choice of model class and the model hyperparameters for each class. By using meta learning, the efficiency of hyperparameter search can be significantly improved by transferring knowledge between tasks, allowing promising areas in the search space to be quickly identified [18]. This can accelerate the automatic tuning and synthesis process of the overall `AutoMPC` pipeline.

III. Method

Efficiently exploring the extensive search space of potential MPC pipelines is crucial to obtain an effective controller within limited computational resources. `AutoMPC` [7] performs hyperparameter tuning using pure BO, which can be inefficient as it initiates each new problem from scratch and does not take advantage of prior knowledge to identify promising areas in the search space. Pure BO can also cause problem of instability for the tuning process due to the random selection of initial configurations.

To overcome the limitations of pure BO and enhance the efficiency and effectiveness of `AutoMPC`, we employ a meta-learning approach called *Portfolio*. This method is inspired by the meta-learning technique proposed in `Auto-Sklearn 2.0` [8] for AutoML. Portfolio selects a fixed set of configurations from meta datasets to warmstart BO, giving priority to configurations that demonstrate strong performance across a broad range of scenarios. Subsequently, configurations with more specialized applicability are added to the Portfolio. We leverage Portfolio for system ID tuning, which plays a pivotal role in the `AutoMPC` pipeline.

In the following, we detail the problem statement of SysID tuning in `AutoMPC` and describe how Portfolio connects to our problem. Figure1 shows the overview of Portfolio building and how it applies to the `AutoMPC` pipeline.

A. System ID Tuning Problem Statement

`AutoMPC` provides several system ID algorithms which can be used to learn a dynamics model from data. The SysID tuning generates a data-learned dynamics model $\hat{f}_h : (x_t, u_t) \to x_{t+1}$ with a model configuration[4] $h \subseteq \mathcal{H}_S$. The learned dynamics model predicts the state in the next time step based on the current observation and control input.

A nonlinear control benchmark offers a set of state x_t and control u_t. Given the number of trajectories N and the length of a trajectory L, we can construct a dataset $D = \{((x_{i,j}, u_{i,j}), x_{i,j+1})\}$ where $i \in \{0, \dots, N\}$ and $j \in \{0, \dots, L-1\}$, which can be used to learn a dynamics model and evaluate the performance of system ID tuning.

Given a model configuration h and a dataset D, we can empirically approximate the model score[5]:

$$\hat{\epsilon}(h, D) = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{j=1}^{L} \mathcal{L}(\hat{f}_h(x_{i,j}, u_{i,j}), x_{i,j+1}),$$

where $\mathcal{L}$ is the evaluation metric.

During the search, we perform K-fold cross-validation to estimate the model score $\hat{\epsilon}_{CV}$ for each model configuration. The goal of SysID tuning is to find an optimal model configuration h^*, which can achieve the best model score in the training set:

$$h^* \in \underset{h \subseteq \mathcal{H}_S}{\operatorname{argmin}} \ \hat{\epsilon}_{CV}(h, D_{train}).$$

[4]A model configuration contains a model class and its corresponding hyperparameters.

[5]The smaller the model score, the better the performance.

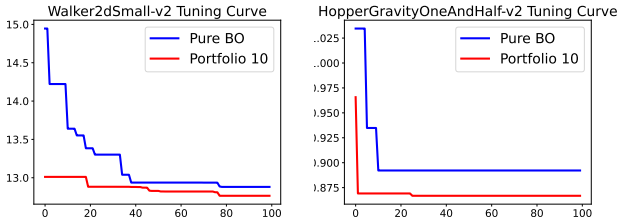


Fig. 2: **Comparison between Portfolio with size 10 and Pure BO.** We evaluate the model tuning performance on Walker2dSmall and HopperGravityOneAndHalf datasets within 100 iterations. The blue lines represent the tuning curve for Pure BO, while the red lines denote the tuning curve for Portfolio with size 10. Portfolio can lead to a faster convergence of model tuning and outperform pure BO within limited time.

B. Portfolio on AutoMPC

Figure 1 shows the construction of a Portfolio and how it connects to the `AutoMPC` system, inspired by `Auto-Sklearn 2.0` [8] for AutoML. First, we conduct the AutoMPC model tuning separately for each meta dataset without meta-learning. This gives us the optimal configuration for each dataset, forming the candidate set C. Next, we evaluate the performance of configurations in C on the meta dataset D_{meta} to create a performance matrix. Using the Greedy Portfolio algorithm proposed by [8], we construct a Portfolio P by considering the performance matrix and the specified portfolio size. Once we have P, we initialize the model tuning using P as the initial configurations for BO. This process yields the best model configuration h^* and its corresponding surrogate dynamics model f_{h^*}, which are then used for control tuning in `AutoMPC`, ultimately helping us achieve the optimal synthesized MPC. Portfolio works by warmstarting BO in SysID tuning and results in a better initialization when encountering new tasks. More details about Greedy Portfolio algorithm can be found in [8].

IV. Experiments and Results

In this section, we investigate the effectiveness of Portfolio on `AutoMPC`. In particular, we aim to answer the following three questions through the experiments. (1) Can Portfolio enhance the efficiency and stability of SysID tuning? (2) What is the impact of portfolio size p on the tuning performance? (3) Can a superior model obtained through Portfolio lead to a better controller?

A. Benchmarks and Datasets

We consider benchmark systems and tasks from OpenAI Gym [10], such as Cartpole and HalfCheetah, and their extensions [11] by modifying some parameters including gravity and morphological size. The details of our benchmarks can be found in Appendix A.

We generate each dataset by executing 1,000 trajectories with uniform random controls selected at each time step. Each trajectory has a duration of 10s and a time step of 0.05s. We also include smaller datasets consisting of only 100 trajectories. We choose 20 datasets for Portfolio training and 11 for evaluation. The details of meta training and testing datasets can be found in Appendix B.

TABLE I: **Comparison between Portfolio with size 10 and pure BO on 12 datasets.** Averaged results and standard deviation on 5 independent runs are reported. * indicates the statistically significant differences, with p-value< 0.1. The best results are in boldface.

Dataset	RMSE				Gains (%)
	BO		Portfolio		
	mean	std	mean	std	
HopperSmall	0.5987	0.0107	**0.5944**	0.0099	*0.7182*
Walker2dSmall	12.9722	0.2790	**12.7447***	0.1073	*1.7538*
InvertedPendulumSmall	1.2506	0.0555	**1.2395**	0.0378	*0.8876*
HalfCheetahGravityHalf	2.5899	0.0214	**2.5499***	0.0234	*1.5445*
HopperGravityOneAndHalf	0.8803	0.0361	**0.8573***	0.0171	*2.6127*
HumanoidGravityHalf	5.5249	0.0849	**5.4808**	0.1070	*0.7982*
HalfCheetahSmallLeg	2.5504	0.0340	**2.5358**	0.0077	*0.5725*
HopperBigThigh	0.8194	0.0233	**0.8090***	0.0085	*1.2692*
HopperSmallTorso	0.8656	0.0429	**0.8085***	0.0116	*6.5966*
Out-of-distribution Data					
Cartpole	0.0101	0.0007	**0.0097**	0.0012	*3.9604*
Pendulum	**0.3174**	0.0000	**0.3174**	0.0001	*0.0000*
Underwater Soft Robot	**0.1165**	0.0000	**0.1165**	4.19e-15	*0.0000*

B. System ID Tuning with Portfolio

In this study, we aim to evaluate the performance of the learned Portfolio and compare it with the pure BO. Each system ID dataset is divided into a training set, a validation set, and a testing set, which contain 70%, 15%, and 15% of the trajectories, respectively. Each method is tuned for 100 iterations using SMAC3 library [15], based on the 1 step rollout RMSE prediction error on the validation set.

To account for randomness, we report results averaged over 5 repetitions and also provide the standard deviation over these repetitions. To assess the statistical significance of performance differences, we conduct t-test with $\alpha = 0.1$ wherever possible. We set the portfolio size to **10** and tested our approach on 12 meta testing datasets. We present the results in Table I.

As anticipated, Portfolio-based meta-learning generally enhances the model performance within 100 iterations. Additionally, we observe that Portfolio can lead to smaller standard deviations in most cases, improving the stability of the model tuning and reducing the need for multiple attempts to obtain an effective solution.

Furthermore, Figure 2 shows the tuning curves for the median results of 5 independent runs on 2 datasets, which compares the tuning progress between the Portfolio with size 10 and Pure BO. They demonstrate that the initialization provided by Portfolio leads to faster convergence of `AutoMPC` tuning, which can yield a substantial improvement in model performance when tuning time is limited. For instance, the Walker2dSmall model error can be reduced to below 13 within 20 iterations with Portfolio as opposed to 40 iterations for pure BO. Similarly, for HopperGravityOneAndHalf, Portfolio can achieve a score below 0.875 within 5 iterations, while pure BO cannot achieve this score within 100 iterations. This reduced tuning time can greatly accelerate the `AutoMPC` workflow since a full 100 iterations of tuning can take more than 24 hours to run.

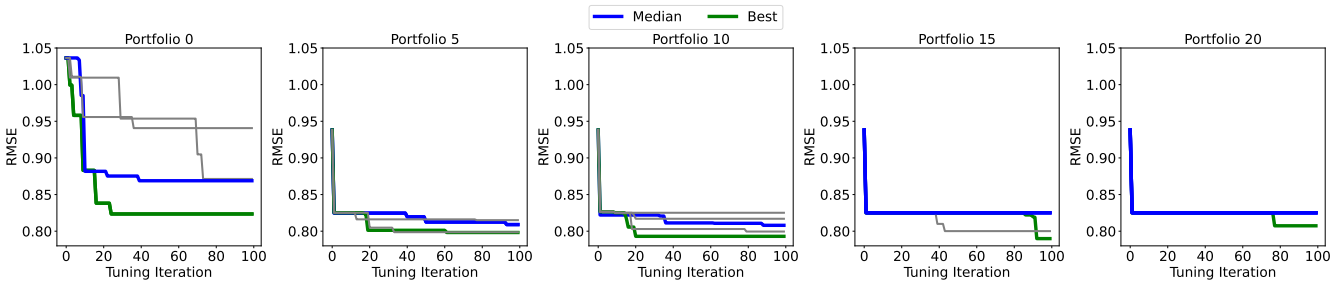


Fig. 3: **Comparison among different Portfolio sizes.** We evaluate the model tuning performance on HopperSmallTorso dataset with pure BO (Portfolio 0) and Portfolio with sizes $5, 10, 15, 20$ on 5 independent runs, plotted in grey. The median and best result are highlighted in blue and green, respectively. In this case, Portfolio consistently outperforms the pure BO regardless of the size. The tuning process becomes more stable as the Portfolio size increases.

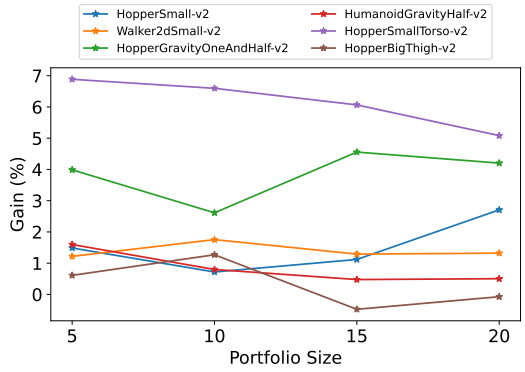


Fig. 4: **Gains among different Portfolio sizes.** We compare the gains of Portfolio with sizes $5, 10, 15, 20$ over the pure BO on 6 datasets. The best Portfolio size varies for different datasets and an inappropriate Portfolio size may worsen the performance.

However, the Portfolio-based meta-learning approach may not be advantageous for out-of-distribution data. The Portfolio is primarily trained on the OpenAI Gym MuJoCo datasets and its extensions, which perform well on datasets with the same data distribution. However, we do not observe any improvement in performance for the out-distribution tasks such as Pendulum and Underwater Soft Robot.

C. Comparison for Different Portfolio Sizes

In this section, we investigate the impact of portfolio size p, a new hyperparameter introduced by Portfolio, on model tuning performance. We conduct experiments with four different portfolio sizes, namely, 5, 10, 15, and 20, and evaluated their performance on the testing datasets.

As depicted in Figure 3, the initial configurations for pure BO approach are random, resulting in large deviation for tuning performance across five attempts. However, Portfolio fixes the initial p configurations and makes the tuning process more stable. The larger the portfolio size is, the more stable the tuning process becomes.

Although the Portfolio with different sizes usually exhibit better performance within a limited number of iterations, there is no correlation between the final result and portfolio size. As shown in Figure 4, the best portfolio size varies depending on the dataset and sometimes an inappropriate portfolio size will worsen the performance of model tuning. Therefore, our study shows that portfolio size is an important hyperparameter that affects the stability and effectiveness of the model tuning, which should be selected carefully.

D. Control Performance with Portfolio

TABLE II: Control performance on Cartpole.

Portfolio Size	Control Score	Gain (%)
0	8.4000	-
5	7.7100	*8.2143*
10	**7.1600**	*14.7619*

Finally, in this section, we evaluate the impact of model tuning with Portfolio on control performance. We pick the Cartpole benchmark as a representative task and perform model tuning with portfolio size 0 (Pure BO), 5, and 10, with five seeds each. We then combine the tuned models with a previously tuned optimizer and objective function to obtain a set of controllers. (The optimizer and objective hyperparameters were taken from the experiment in Sec. 6.2.1 of [19]). The average control performance for each portfolio size is reported in Table II. The control scores range from 0 to 10 with lower being better (see [7] for details). Both portfolio sizes improve over pure BO and the Portfolio of size 10 achieves the best control score with a 14.76% improvement over the pure BO. It demonstrates that a superior model obtained through Portfolio can lead to a better controller.

V. Conclusion and Future Work

Our paper is built upon the recent advance of a meta-learning approach named *Portfolio* to enhance the efficiency and robustness of `AutoMPC`. We leveraged Portfolio to optimize the initial designs for pure Bayesian Optimization, utilizing a diverse and reliable set of configurations from previous tasks. Our experiments demonstrated that Portfolio significantly improves the capacity of `AutoMPC` to achieve desirable dynamics models and controllers within limited computational resources.

This work also offers avenues for future extensions. We are interested in exploring Portfolio selection, such as choosing appropriate portfolio size and training datasets, based on the characteristics of particular testing dataset, making `AutoMPC` more adaptive to out-of-distribution data and open-world robotics.

REFERENCES

[1] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*. Springer, 1999, pp. 207–226.

[2] M. Abdollahyan, A. M. Sardari, and S. M. E. Alamdari, "A data-driven mpc approach for trajectory tracking and obstacle avoidance of mobile robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 3581–3586.

[3] R. Palma, T. Menegatti, and L. Bascetta, "Data-driven predictive control for autonomous navigation and obstacle avoidance in unstructured environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2036–2043, 2020.

[4] J. A. Ortiz, A. F. T. Martins, R. N. Calheiros, and R. Carelli, "A data-driven mpc approach for trajectory tracking and manipulation tasks," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 146–151, 2021.

[5] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomiin, "Goal-driven dynamics learning via Bayesian optimization," in *Conference on Decision and Control (CDC)*, 2017, pp. 5168–5173.

[6] D. Piga, M. Forgione, S. Formentin, and A. Bemporad, "Performance-oriented model learning for data-driven MPC design," *Control Systems Letters*, vol. 3, no. 3, pp. 577–582, 2019.

[7] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, "Automatic tuning for data-driven model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7379–7385.

[8] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Auto-sklearn 2.0: Hands-free automl via meta-learning," *arXiv:2007.04074 [cs.LG]*, 2020.

[9] H. Jin, F. Chollet, Q. Song, and X. Hu, "Autokeras: An automl library for deep learning," *Journal of Machine Learning Research*, vol. 24, no. 6, pp. 1–6, 2023. [Online]. Available: http://jmlr.org/papers/v24/20-1355.html

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *Arxiv preprint arXiv:1606.01540*, 2016.

[11] P. Henderson, W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek, "Benchmark environments for multitask learning in continuous domains," *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop*, 2017.

[12] W. D. Null, J. Menezes, and Y. Z, "Development of a modular and submersible soft robotic arm and corresponding learned kinematics models," in *2023 IEEE International Conference on Soft Robotics (RoboSoft)*, 2023, pp. 1–6.

[13] W. D. Null, W. Edwards, D. Jeong, T. Tchalakov, J. Menezes, K. Hauser, and Y. Z, "Automatically-tuned model predictive control for an underwater soft robot," 2023, iEEE Robotics and Automation Letters (submitted).

[14] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter, "Boah: A tool suite for multi-fidelity bayesian optimization & analysis of hyperparameters," *arXiv:1908.06756 [cs.LG]*.

[15] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*, 2011, pp. 507–523.

[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1126–1135. [Online]. Available: https://proceedings.mlr.press/v70/finn17a.html

[17] E. Arcari, A. Carron, and M. Zeilinger, "Meta learning mpc using finite-dimensional gaussian process approximations," *arXiv preprint arXiv:2008.05984*, 08 2020.

[18] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Neural Information Processing Systems (NeurIPS)*, 2015, pp. 2962–2970.

[19] W. Edwards, "Data-driven methods for design of model predictive controllers," Master's thesis, University of Illinois at Urbana-Champaign, 2022.

APPENDIX

A. Extensive Benchmarks

Our extensive benchmark systems are based on OpenAI Gym MuJoCo and their extensions by modifying some parameters. We detail the extensive benchmarks in the following:

- **Open AI Gym MuJoCo:** We consider 10 robotics tasks based on the v-2 environments in Gym MuJoCo including *Ant-v2, HalfCheetah-v2, Hopper-v2, Humanoid-v2, HumanoidStandUp-v2, InvertedDoublePendulum-v2, InvertedPendulum-v2, Reacher-v2, Swimmer-v2, Walker2D-v2, and Pusher-v2.*
- **Modified Gravity:** We adjust the simulated gravity to different scales that range from 0.5 to 1.5 times the normal gravity level, such as *HopperGravityOneAndHalf-v2* and *HalfCheetahGravityHalf-v2*.
- **Morphological Modifications:** We alter the morphology of a particular body part, such as the foot, leg, thigh, or torso. The body parts that are scaled by 1.25 times their mass and width are labeled as "Big" while those scaled by 0.75 are classified as "Small", such as *HopperSmallTorso-v2*.
- **PusherMovingGoal:** We randomly move the goal position for the Pusher task, namely *PusherMovingGoal-v2*.

B. Meta Training and Testing Datasets

As mentioned in Section IV-A, the datasets are generated by executing trajectories. The normal datasets consists of 1,000 trajectories while the small datasets only have 100 trajectories, such as *HopperSmall-v2* and *Walker2dSmall-v2*.

We select **20** datsets for Portfolio training:
[*HalfCheetah-v2, Hopper-v2, Walker2d-v2, Swimmer-v2, InvertedPendulum-v2, Reacher-v2, Pusher-v2, InvertedDoublePendulum-v2, Ant-v2, Humanoid-v2, HalfCheetahSmall-v2, ReacherSmall-v2, SwimmerSmall-v2, HopperGravityThreeQuarters-v2, Walker2dGravityOneAndHalf-v2, HalfCheetahGravityOneAndQuarter-v2, HalfCheetahBigThigh-v2, HopperSmallLeg-v2, Walker2dSmallTorso-v2, PusherMovingGoal-v2*].

And **12** datasets for evaluation:
[*HopperSmall-v2, Walker2dSmall-v2, InvertedPendulumSmall-v2, HalfCheetahGravityHalf-v2, HopperGravityOneAndHalf-v2, HumanoidGravityHalf-v2, HalfCheetahSmallLeg-v2, HopperBigThigh-v2, HopperSmallTorso-v2, CartpoleSwingup, PendulumSwingup, SoftRobot*].

C. More Experimental Results for System ID Tuning with Portfolio

We provide four additional tuning curves for the median results of 5 independent runs to compare Portfolio with 10 and Pure BO in Figure 5. They all demonstrate that Portfolio can provide better initialization than the pure BO, which leads to faster convergence of `AutoMPC` tuning and yields a substantial improvement in model tuning with limited computational resources.

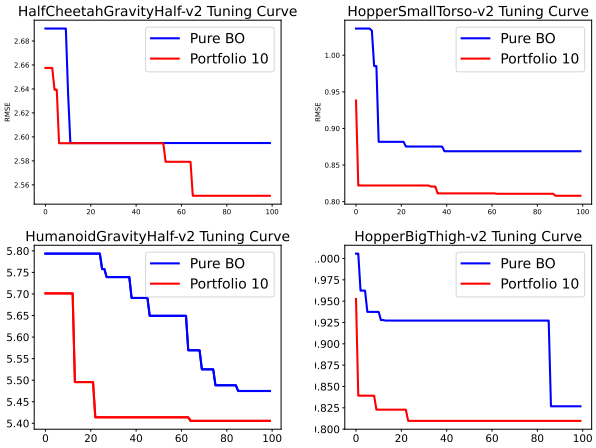


Fig. 5: Tuning curves for Portfolio with size **10** and Pure BO.

D. More Experimental Results for Different Portfolio Sizes

Table III shows the results for Portfolio with various sizes (5, 10, 15, 20) and pure BO on 12 testing datasets. For datasets excluding "Pendulum" and "Under Soft Robot", Portfolio generally enhances the model performance and stability within 100 iterations. However, the best portfolio size varies depending on the dataset and sometimes an inappropriate portfolio size will worsen the performance of model tuning.

E. Other Experimental Details

All experiments are conducted on computer clusters including Illinois Campus Cluster Program[6] and Digital Environments for Learniung, Teaching, and Agency (DELTA)[7], which consist of 12 compute nodes providing 296 CPU cores and 1152GB RAM.

The experiments also have time-out setting, which will automatically terminate the tuning of each iteration if it exceeds 10 minutes, in which case no result will be provided.

[6]https://campuscluster.illinois.edu/resources/docs/user-guide/
[7]https://education.illinois.edu/ci/programs-degrees/delta

TABLE III: Comparison between Portfolio with various sizes (5, 10, 15, 20) and pure BO (0) on 12 datasets. Averaged results and standard deviation on 5 independent runs are reported. The best results are in boldface and the second best results are underlined.

Dataset	**RMSE**										**Best Gains (%)**
Portfolio Size	**0**		**5**		**10**		**15**		**20**		
	mean	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	
HopperSmall	0.5987	0.0107	0.5898	0.0093	0.5944	0.0099	0.5920	0.0074	**0.5825**	0.0059	*2.7059*
Walker2dSmall	12.9722	0.2790	12.8142	0.0990	**12.7447**	0.1073	12.8049	0.0317	12.8008	0.0843	*1.7538*
InvertedPendulumSmall	1.2506	0.0555	**1.2182**	0.0205	1.2395	0.0378	1.2775	0.0452	1.2324	0.0470	*2.5908*
HalfCheetahGravityHalf	2.5899	0.0214	2.6418	0.0228	**2.5499**	0.0234	2.5783	0.0263	2.5806	0.0126	*1.5445*
HopperGravityOneAndHalf	0.8803	0.0361	0.8452	0.0033	0.8573	0.0171	**0.8402**	0.0159	0.8433	1.11e-16	*4.5553*
HumanoidGravityHalf	5.5249	0.0849	**5.4366**	0.0589	5.4808	0.1070	5.4986	0.0549	5.4971	0.0400	*1.5982*
HalfCheetahSmallLeg	2.5504	0.0340	2.6279	0.0508	**2.5358**	0.0077	2.5404	0.0013	2.5413	0.0	*0.5725*
HopperBigThigh	0.8194	0.0233	0.8144	0.0084	**0.8090**	0.0085	0.8233	0.0002	0.8200	0.0041	*1.2692*
HopperSmallTorso	0.8656	0.0429	**0.8060**	0.0066	0.8085	0.0116	0.8131	0.0151	0.8216	0.0071	*6.8854*
Out-of-distribution Data											
Cartpole	0.0101	0.0007	**0.0088**	0.0012	0.0097	0.0012	0.0101	0.0003	0.0097	0.0011	*12.8713*
Pendulum	**0.3174**	0.0	**0.3174**	4.15e-16	**0.3174**	0.0001	**0.3174**	6.25e-05	**0.3174**	5.75e-06	*0*
Underwater Soft Robot	**0.1165**	4.19e-15	**0.1165**	0.0001	**0.1165**	4.19e-15	**0.1165**	0.0001	**0.1165**	4.19e-15	*0*