Quick-Tune: Quickly Learning Which Pretrained Model to Finetune and How

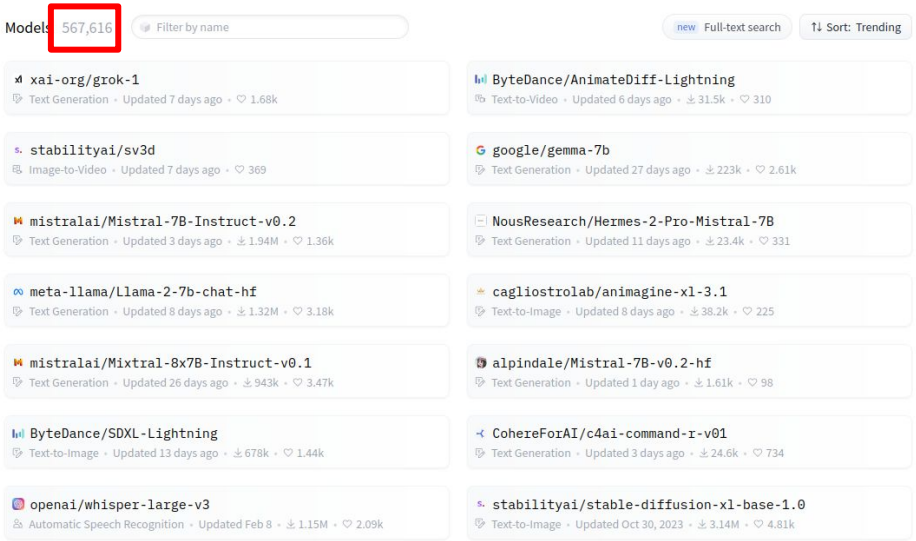
Sebastian Pineda Arango, Fabio Ferreira, Arlind Kadra,
Frank Hutter, Josif Grabocka

University of Freiburg

Model Hubs

Nowadays, communities make available a large number of pretrained models. They have different sizes with different type of inductive biases.

- HuggingFace
- Pytorch Hub
- TensorFlow Hub
- Timm Library
- …

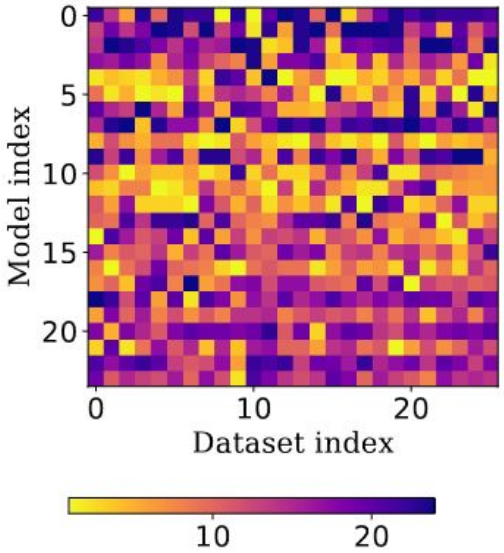


Model Hubs vs One-Fits-All models

There is not a one-fits-all model that perform well across different setups. For instance: different metrics, dataset size, compute or time requirements.

If compute the rank among models with optimized HPs, **we get different patterns across datasets**. The figure shows the model ordered by size (largest model has index = 0).

Some small model are competitive.



Every cell is considering the best model out of a random search

How to select quickly the model and hyperparameters?

To achieve efficiency, we propose to apply:

- **Gray Box Optimization.** Early iterations are correlated with the performance in later iterations.
- **Cost Awareness.** Time is critical and model hubs include architectures with different sizes, thus it is important to be aware of the time.
- **Transfer HPO.** Information from auxiliary tasks can be leveraged. Similar tasks might need similar hyperparameters.

Quick-Tune

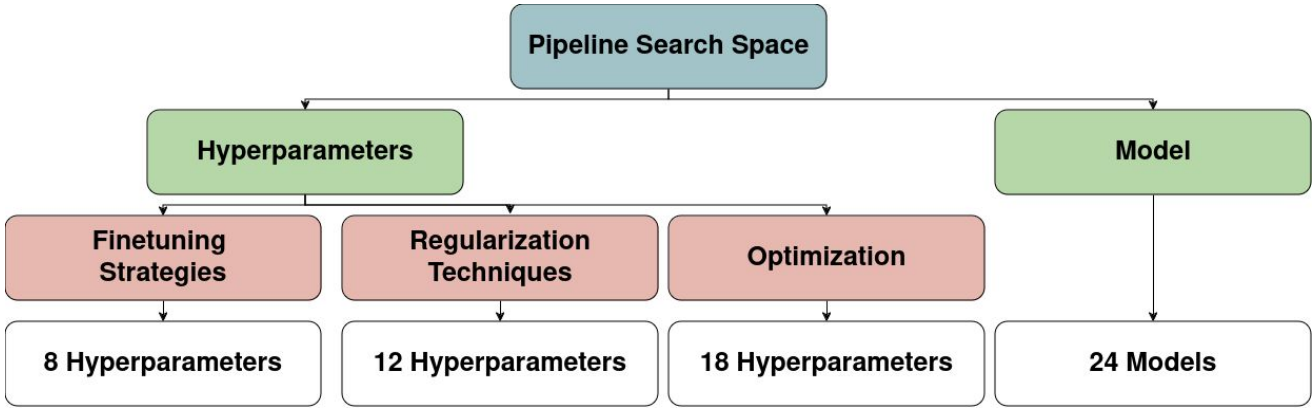
Our work comprises two parts:

- **Metadataset:**
  - Design search space and finetuning pipeline
  - Generate metadata

- **Algorithm:**
  - Formalize algorithm
  - Empirical evaluation

Search Space: Overview

- We call **pipeline** the combination of hyperparameter and models.
- We use the **Timm library*** for our experiments. We take some of the hyparameters (such as data augmentations, optimizers), and integrate additional finetuning strategies.



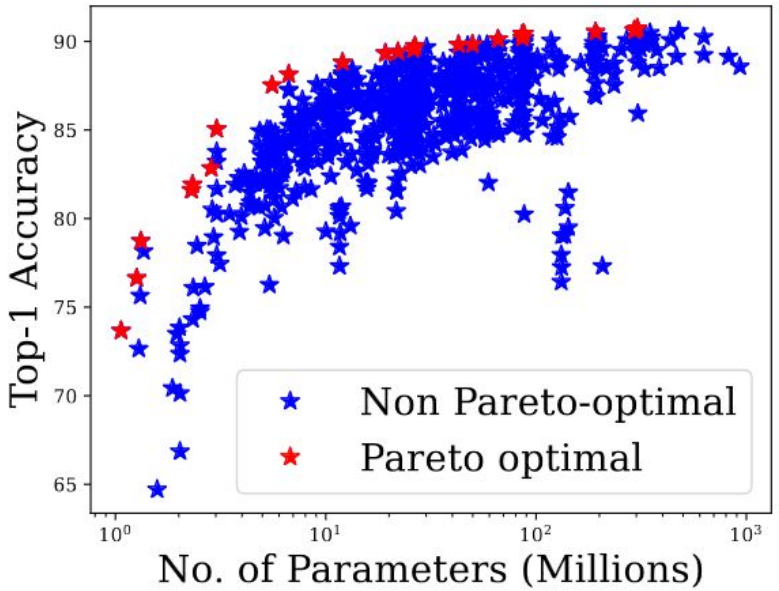
* https://github.com/huggingface/pytorch-image-models

Search Space: Models

We selected the efficient models by accounting for its performance on ImageNet $f_{\text{ImageNet}}(m)$ and its size $S(m)$.

We included the models in the pareto front of the following objective:

$$\mathcal{M} = \left\{ m^* \,|\, m^* \in \underset{m \in \mathcal{M}_{\text{Timm}}}{\arg\max} \; [f_{\text{ImageNet}}(m),\; -S(m)] \right\}$$



Meta-Dataset Generation

- We generated learning curves by finetuning models on **Meta-Album datasets***. It has three groups/versions with different sizes: *micro, mini* and *extended.* In total, we included 86 datasets.
- Since the search space is so large, we did not evaluate exhaustively but **sample randomly configurations**.
- We run every configuration on a single GPU, up to 50 epochs.
- We also runtime limits for every dataset version:
  - Micro: 1 hour
  - Mini: 4 hours
  - Extended: 16 hours

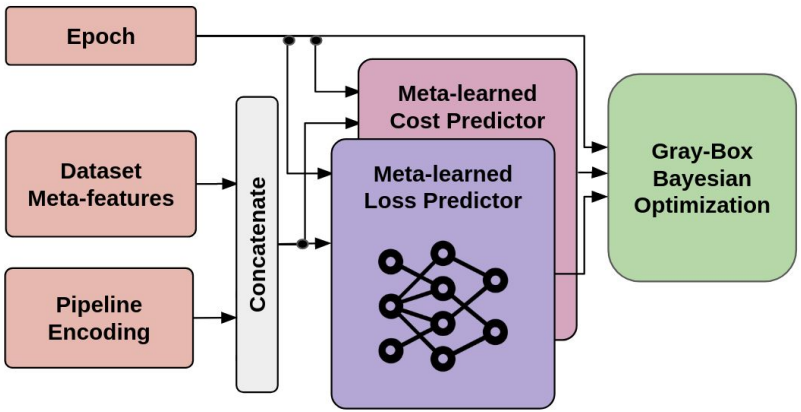
* Ullah, I., Carrión-Ojeda, D., Escalera, S., Guyon, I., Huisman, M., Mohr, F., ... & Vu, P. A. (2022). *Meta-album: Multi-domain meta-dataset for few-shot image classification.*

Meta-Dataset Composition

- 86 tasks
- 20.108 learning curves
- 743.644 observed epochs
- 23.991 total execution time

Meta-Dataset	Number of Tasks	Number of Curves	Total Epochs	Total Run Time
Micro	30	8.712	371.538	2.076 GPU Hours
Mini	30	6.731	266.384	6.049 GPU Hours
Extended	26	4.665	105.722	15.866 GPU Hours

Quick-Tune Method



Our method relies on gray box optimization with two main components:

- (Meta-learned) Cost predictor
- (Meta-learned) Loss/performance predictor

They use dataset meta-features, the budget in epochs and an encoding of the pipeline as input.

Quick-Tune Algorithm

Algorithm 1: Quick-Tune Algorithm

Input: Search space of pipelines $x \in \mathcal{X}$, Epoch step Δt
Output: Pipeline with the smallest observed loss

Select randomly a pipeline $x' \in \mathcal{X}$ and evaluate it for Δt epochs ;
Initialize the history $\mathcal{H} \leftarrow \{(x', \Delta t, \ell(x', \Delta t), c(x', \Delta t))\}$
while *budget* **do**
    Update the performance predictor $\hat{\ell}$ from $\mathcal{H}$ using Equation 1;
    Update the cost estimator $\hat{c}$ from $\mathcal{H}$ using Equation 2;
    Select the next pipeline x^* using Equation 3;
    Evaluate the performance $\ell\left(x^*, \tau(x^*)\right)$ and measure the cost $c\left(x^*, \tau(x^*)\right)$;
    Update the history $\mathcal{H} \leftarrow \mathcal{H} \cup \{(x^*, \tau(x^*), \ell\left(x^*, \tau(x^*)\right), c\left(x^*, \tau(x^*)\right))\}$;
end
return $\arg\min_{x \in \mathcal{X}} \{\ell(x, t) \mid (x, t, \ell(x, t), \cdot) \in \mathcal{H}\}$;

Update predictors using NLL and MSE

Select pipeline

Update history

The implementation can be found in: https://github.com/releaunifreiburg/QuickTune.

Cost-Sensitive Acquisition Function

We use an acquisition function that uses cost information to explore partially the gray-box function for Δt epochs.

$$\arg\max_{x \in \mathcal{X}} \frac{\mathbb{E}_{\hat{\ell}(x,\tau(x))}\left[\max\left(\ell^{\min}_{\tau(x)} - \hat{\ell}(x,\tau(x)), 0\right)\right]}{\hat{c}\Big(x,\tau(x)\Big) - c\Big(x,\tau(x) - \Delta t\Big)}$$

EI with incumbent relative to the next query budget

Cost of finetuning the pipeline until the next query budget

$\tau(x)$: denotes the next query budget (epochs) for the pipeline.

$$\tau(x) := \max\{t' | (x, t', \cdot, \cdot) \in \mathcal{H}\} + \Delta t$$

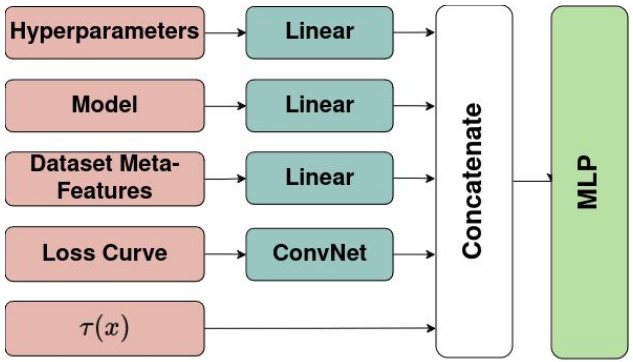
$\ell^{\min}_{\tau(x)}$: denotes the incumbent relative to the query budget*.

$$\ell^{\min}_{\tau(x)} := \min\left(\{\ell(x,\tau(x)) | (x, \tau(x), \ell\left(x,\tau(x)\right), \cdot) \in \mathcal{H}\}\right)$$

* Wistuba, M., Kadra, A., & Grabocka, J. (2022). *Supervising the multi-fidelity race of hyperparameter configurations.*

Pipeline Embeddings for Deep Kernel Surrogate

- Since the search space is relatively large (38 hyperparameters, 24 models), we use a Deep Kernel GP.
- The embedding network use separate encoders for different stages*, followed by a MLP.
- This network can be also metatrained using the metadata.



* Pineda Arango, S., & Grabocka, J. (2023, August). *Deep Pipeline Embeddings for AutoML*.

Experiments and Results

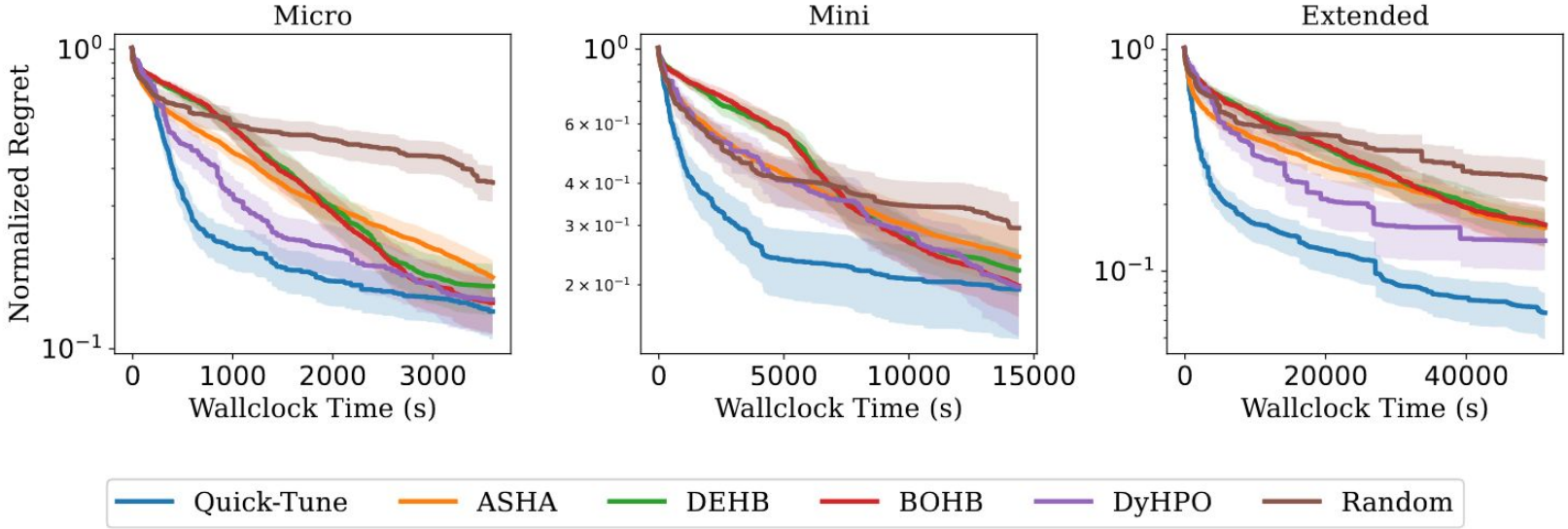
Experiment 1: Finetuning without HPO

Finetuning without HPO is not enough. We tried three models with different sizes and default hyperparameters, but **QuickTune outperformed in the three versions.**

	Normalized Regret			Rank		
	Micro	**Mini**	**Extended**	**Micro**	**Mini**	**Extended**
BEiT+Default HP	$0.229_{\pm 0.081}$	$0.281_{\pm 0.108}$	$0.225_{\pm 0.059}$	$2.583_{\pm 0.829}$	$2.611_{\pm 0.465}$	$3.136_{\pm 0.215}$
XCiT+Default HP	$0.223_{\pm 0.075}$	$0.290_{\pm 0.107}$	$0.199_{\pm 0.057}$	$2.500_{\pm 0.751}$	$2.694_{\pm 0.264}$	$2.522_{\pm 0.344}$
DLA+Default HP	$0.261_{\pm 0.074}$	$0.325_{\pm 0.111}$	$0.219_{\pm 0.076}$	$3.062_{\pm 0.770}$	$3.138_{\pm 0.248}$	$2.977_{\pm 0.284}$
Quick-Tune	$\mathbf{0.153_{\pm 0.054}}$	$\mathbf{0.139_{\pm 0.112}}$	$\mathbf{0.052_{\pm 0.031}}$	$\mathbf{1.854_{\pm 1.281}}$	$\mathbf{1.555_{\pm 0.531}}$	$\mathbf{1.363_{\pm 0.376}}$

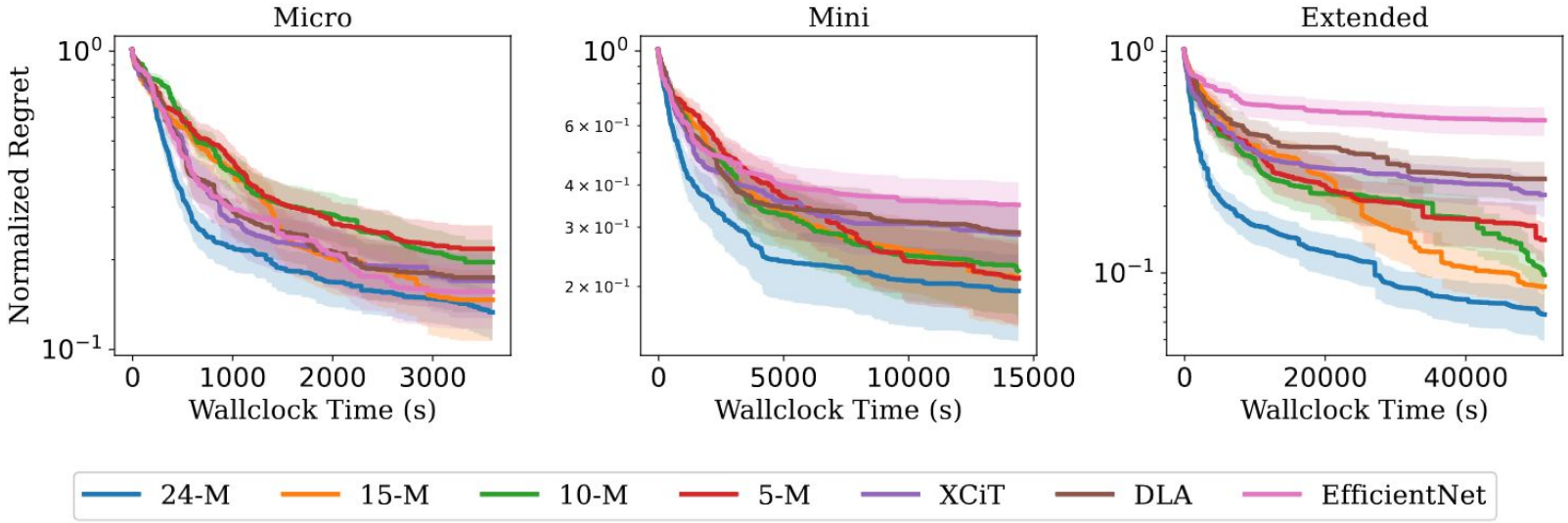
Experiment 2: CASH with other optimizers

QuickTune is competitive with respect to other optimizers, especially in early iterations. We observe that the largest gap is in the large datasets (*extended* version).



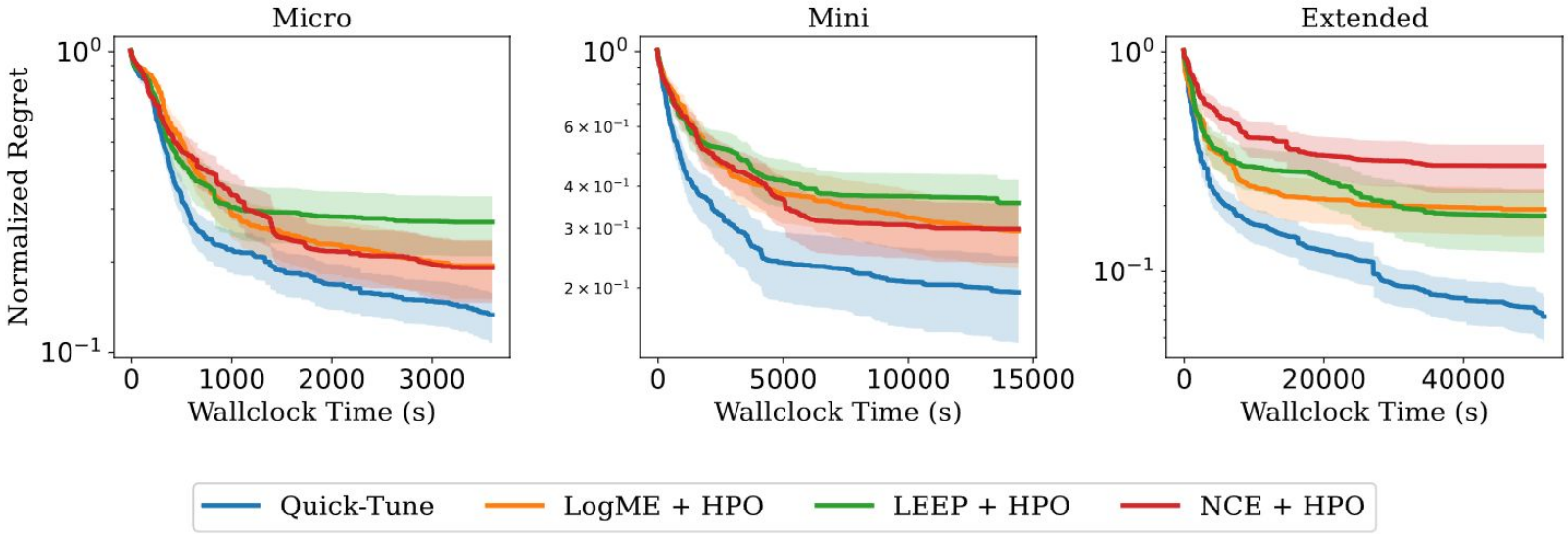
Experiment 3: Varying the size of the model hub

We vary the number of models, and find that having the whole model hub is the best approach in all the versions. The benefit of increasing the size of model hub is clearer in large datasets.



Experiment 4: Separated model selection and HPO

By selecting jointly the pretrained model and its hyperparameters, **QuickTune is more effective than selecting independently a model (by using a score function) and its hyperparameters.**



Experiment 5: Efficient finetuning of large models

A common alternative is to use large model by default, and then performing parameter efficient finetuning (PEFT). In a simple experiment, we show that QuickTune outperforms this strategy. However, we observe that LoRA is competitive in large datasets. Still, these strategies can be included in an augmented search space.

	4 Hours			24 Hours		
	Micro	**Mini**	**Extended**	**Micro**	**Mini**	**Extended**
Dinov2 + LoRA	$0.541_{\pm 0.093}$	$0.049_{\pm 0.018}$	$0.055_{\pm 0.004}$	$0.332_{\pm 0.095}$	$0.014_{\pm 0.021}$	$0.004_{\pm 0.012}$
Dinov2 + Linear Probing	$0.081_{\pm 0.041}$	$0.067_{\pm 0.021}$	$0.081_{\pm 0.012}$	$0.067_{\pm 0.038}$	$0.017_{\pm 0.019}$	$0.042_{\pm 0.011}$
QuickTune	$\mathbf{0.072_{\pm 0.024}}$	$\mathbf{0.039_{\pm 0.014}}$	$\mathbf{0.042_{\pm 0.016}}$	$\mathbf{0.018_{\pm 0.012}}$	$\mathbf{0.012_{\pm 0.008}}$	$\mathbf{0.003_{\pm 0.008}}$

Conclusion and future work

- We present an efficient method for finetuning architectures in a model hub, by applying gray-box optimization, meta-learning and cost-awareness.
- We created a meta-dataset with more than 20.000 curves and used it to evaluate our method.
- We demonstrated the effectiveness of our method by comparing it against different optimizers and lightweight alternatives.
- As future work our method can be extended:
    - By exploring more complex acquisition functions.
    - By applying it to other modalities in model hubs e.g. NLP.
    - Efficient warm-starting.

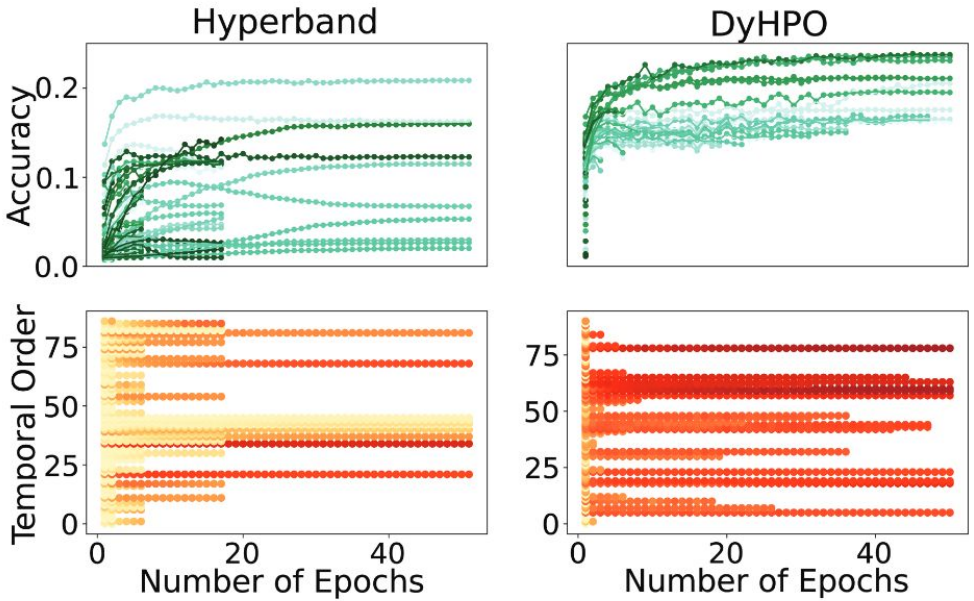
Thank you

Appendix

DyHPO*

Hyperband is a gray-box approach that statically pre-allocates the budget for a set of candidates (Hyperband bracket) according to a predefinedpolicy. However, DYHPO dynamically adapts the allocation of budgets for configurations after every HPO step (a.k.a. a dynamic race).

As a result, DYHPO invests only a small budget on configurations that show little promise as indicated by the intermediate scores.



* Wistuba, M., Kadra, A., & Grabocka, J. (2022). Supervising the multi-fidelity race of hyperparameter configurations.

QuickTune Configuration

- MLP: 32 neurons, 2 layers
- ConvNet: 8 filters, 2 layers
- Metatraining epochs: 1000
- Learning rate: 0.0001
- BO (test) epochs: 100
- Metafeatures:
  - Dataset size, number of classes, number of samples

QuickTune Search Space

Hyperparameter Group	Hyperparameters
Finetuning Strategies	Percentage of the Model to Freeze, Layer Decay, Linear Probing, Stochastic Norm, SP-Regularization, DELTA Regularization, BSS Regularization, Co-Tuning
Regularization Techniques	MixUp, MixUp Probability*, CutMix, Drop-Out, Label Smoothing, Gradient Clipping
Data Augmentation	Data Augmentation Type (Trivial Augment, Random Augment, Auto-Augment), Auto-Augment Policy*, Number of operations*, Magnitude*
Optimization	Optimizer type (SGD, SGD+Momentum, Adam, AdamW, Adamp), Beta-s*, Momentum*, Learning Rate, Warm-up Learning Rate, Weight Decay, Batch Size
Learning Rate Scheduling	Scheduler Type (Cosine, Step, Multi-Step, Plateau), Patience*, Decay Rate*, Decay Epochs*
Model	24 Models on the Pareto front (see Appendix 8)

Models in Quick-Tune

Model Name	No. of Param.	Top-1 Acc.
beit_large_patch16_512	305.67	90.691
volo_d5_512	296.09	90.610
volo_d5_448	295.91	90.584
volo_d4_448	193.41	90.507
swinv2_base_window12to24_192to384_22kft1k	87.92	90.401
beit_base_patch16_384	86.74	90.371
volo_d3_448	86.63	90.168
tf_efficientnet_b7_ns	66.35	90.093
convnext_small_384_in22ft1k	50.22	89.803
tf_efficientnet_b6_ns	43.04	89.784
volo_d1_384	26.78	89.698
xcit_small_12_p8_384_dist	26.21	89.515
deit3_small_patch16_384_in21ft1k	22.21	89.367
tf_efficientnet_b4_ns	19.34	89.303
xcit_tiny_24_p8_384_dist	12.11	88.778
xcit_tiny_12_p8_384_dist	6.71	88.101
edgenext_small	5.59	87.504
xcit_nano_12_p8_384_dist	3.05	85.025
mobilevitv2_075	2.87	82.806
edgenext_x_small	2.34	81.897
mobilevit_xs	2.32	81.574
edgenext_xx_small	1.33	78.698
mobilevit_xxs	1.27	76.602
dla46x_c	1.07	73.632

Transfer HPO

There are different ways to leverage previous information:

- **Zero Shot HPO:**
  - Train a model to select hyperparameters directly.
- **Surrogate Transfer**
  - Pretrain a surrogate or ensemble of surrogates.
- **Acquisition Transfer**
  - Pretrain or create an ensemble of functions.

Real-time Executions