Experimental Setup

Datasets

OpenML We integrate the OpenML Curated Classification benchmarking suite 2018 (OpenML-CC18) (Bischl et al. 2021). OpenML-CC18 consists of 72 diverse and curated classification tasks, and we keep 70 datasets, excluding the vision datasets Fashion-MNIST and CIFAR-10. We split each dataset into 80% train and 20% test.

HapMap3 HapMap3 (Consortium et al. 2010) is a publicly available dataset that contains single-nucleotide polymorphisms (SNPs) sequences of whole-genome data from humans with subpopulation annotations. Samples are filtered for the 10 largest human subpopulations, which are used as categories for the *hapmap* datasets. Individuals are split into 75% for train and 25% for test. SNPs with missing values for any sample are discarded. Finally, 5 different datasets are created by randomly sampling 784 SNP positions from different sections of the chromosomes, which are encoded as binary values. For every created dataset, labels are permuted to avoid overfitting to the positions of the labels of each subpopulation.

Dogs Similarly, Dogs (Bartusiak et al. 2022) is a dataset of dog DNA sequences. The dataset consists of the genotyping array of purebred dogs from 75 breeds. Dog breeds can be organized into clades, which are groups of dog breeds that share a common ancestor. Since the number of samples per breed in the dataset is very low, breeds are clustered into clades, and the 10 most common clades are kept and used as categories for the *dogs* datasets. Samples are split into 75% for train and 25% for test. SNPs with missing values for any sample are discarded. Finally, 30 different datasets are created by randomly sampling 784 positions from different sections of the chromosomes. For every created dataset, labels are permuted to avoid overfitting to the positions of the labels of each clade.

UK Biobank The UK Biobank (Sudlow et al. 2015) is a large-scale biobank, from which we use the genotyping array data and full phenotypes as processed in (Qian et al. 2020). We include 8 of the most predictive binary phenotypes according to their polygenic risk score (PRS) model:

- Hair colour (natural, before greying) red: *red hair*
- Hair colour (natural, before greying) blonde: *blonde hair*
- Hair colour (natural, before greying) dark brown: *dark brown hair*
- Hair colour (natural, before greying) black: *black hair*
- Ease of skin tanning (Never tan, only burn): *skin burn*
- Ease of skin tanning (Get very tanned): *skin tan*
- Hair colour (natural, before greying) brown: *brown hair*
- Malabsorption/coeliac disease: *malabsorption-coeliac*

In order to allow proper phenotype prediction modeling, it is a standard practice to stick to a single population, to avoid the prediction being biased by other factors. In this case, we filter by the majority population in the UK Biobank, which is British individuals with European ancestry. Then, we create a balanced dataset for each phenotype by selecting all samples of the minority class (presence of the phenotype), and randomly selecting the same number of samples from the majority class. Variants (features) are selected based on the PRS model weights reported (Tanigawa et al. 2022). We split each dataset into 80% train and 20% test.

The collection of OpenML datasets is randomly shuffled and divided into meta-training (Table 2), meta-validation (Table 3), and meta-testing (Table 4) sets, with a 75%-10%-15% split, respectively. Dogs datasets for dog clade (group of breeds) prediction are used in meta-training, British humans datasets from the UK Biobank (UKB) for phenotype prediction are used in meta-validation, and HapMap3 datasets for subpopulation prediction are used in the meta-test. This strict separation ensures we meta-learn and evaluate on substantially different distributions and tasks.

HyperFast and Baselines Implementation

HyperFast Training Details In the meta-training stage, HyperFast weights are learnt by generating the weights of a smaller model that solves a different training task $t \in \mathcal{T}_{\text{meta-train}}$ at each training step. t is derived from a randomly selected dataset d from the collection of meta-training datasets $\mathcal{D}_{\text{meta-train}}$. However, the gradient signal is too noisy for weight updates at every training step. We fix this issue by accumulating gradients across different tasks before performing an optimization step. We experiment with gradient accumulation of 2, 3, 5, 10, 25, 50, and 100 steps. In our experiments we find that, in general, a larger number of accumulation steps always yields a more stable loss curve. That is, the meta-model learns better from observing the variations across different datasets, rather than solving one task at a time. We use a total of 25 gradient accumulation steps, which already allows a stable training, without excessively prolonging convergence. Despite this, during meta-validation, we observe a tendency to overfit to the meta-training datasets over very long training times. We select the HyperFast model that achieves the best average performance across the meta-validation datasets. We also experimented with solving multiple tasks in a single pass, but it was not possible in many cases due to memory constraints. Another key architectural design choice that significantly stabilizes the training process is sharing the core parameters between hypernetwork modules. As a shared module we use 2 feed-forward layers with a hidden dimensionality of 1024 and ReLU activations. We also experimented with deeper shared modules and different architectures based on attention mechanisms and convolutions, however, training stability and model generalization were inferior. The HyperFast used in this work has 1.27 B parameters (4.7 GB of memory), which generates the weights of smaller models of 52.65 M parameters (200.8 MB). The model is trained for 100,000 steps with a learning rate of 0.0003 with the AdamW optimizer (Loshchilov and Hutter 2018), which required 20 hours on a single NVIDIA Tesla V100 SXM2 GPU.

HyperFast Inference Details Once HyperFast is trained, the hypernetwork weights are frozen and HyperFast can be used as an off-the-shelf model to generate target networks. Significant improvements in performance can be achieved

Dataset name	Train size	Test size	Feature size	Categorical	Classes
dogs$_{1..30}$ (30)	1372	458	784	784	10
sick	3017	755	29	22	2
Bioresponse	3000	751	1776	0	2
splice	2552	638	60	60	3
qsar-biodeg	844	211	41	0	2
MiceProtein	864	216	77	0	8
isolet	6237	1560	617	0	26
connect-4	54045	13512	42	42	3
analcatdata_authorship	672	169	70	0	4
kr-vs-kp	2556	640	36	36	2
optdigits	4496	1124	64	0	10
analcatdata_dmft	637	160	4	4	6
churn	4000	1000	20	4	2
mfeat-karhunen	1600	400	64	0	10
mfeat-factors	1600	400	216	0	10
kc1	1687	422	21	0	2
texture	4400	1100	40	0	11
Internet-Advertisements	2623	656	1558	1555	2
har	8239	2060	561	0	6
jungle_chess_2pcs_raw_endgame_complete	35855	8964	6	0	3
car	1382	346	6	6	4
credit-g	800	200	20	13	2
adult	39073	9769	14	8	2
nomao	27572	6893	118	29	2
jm1	8708	2177	21	0	2
numerai28.6	77056	19264	21	0	2
first-order-theorem-proving	4894	1224	51	0	6
dna	2548	638	180	180	3
Devnagari-Script	73600	18400	1024	0	46
mfeat-morphological	1600	400	6	0	10
madelon	2080	520	500	0	2
pc3	1250	313	37	0	2
blood-transfusion-service-center	598	150	4	0	2
vehicle	676	170	18	0	4
vowel	792	198	12	2	11
balance-scale	500	125	4	0	3
segment	1848	462	16	0	7
pc1	887	222	21	0	2
tic-tac-toe	766	192	9	9	2
semeion	1274	319	256	0	10
letter	16000	4000	16	0	26
electricity	36249	9063	8	1	2
GesturePhaseSegmentationProcessed	7898	1975	32	0	5
cnae-9	864	216	856	0	9
ozone-level-8hr	2027	507	72	0	2
ilpd	466	117	10	1	2
wall-robot-navigation	4364	1092	24	0	4
mfeat-fourier	1600	400	76	0	10
spambase	3680	921	57	0	2
mnist_784	56000	14000	784	0	10
PhishingWebsites	8844	2211	30	30	2
climate-model-simulation-crashes	432	108	18	0	2
steel-plates-fault	1552	389	27	0	7
mfeat-pixel	1600	400	240	0	10

Table 2: Meta-training datasets $\mathcal{D}_{\text{meta-train}}$. Train size is the number of training instances in d_{train}, and Test size is the number of test instances in d_{test}. Subscripts $i..j$ and $(\cdot)$ denote the interval of indices and the total number of datasets of the same group used, respectively.

Dataset name	Train size	Test size	Feature size	Categorical	Classes
cylinder-bands	432	108	37	19	2
wdbc	455	114	30	0	2
eucalyptus	588	148	19	5	5
mfeat-zernike	1600	400	47	0	10
cmc	1178	295	9	7	3
dresses-sales	400	100	12	11	2
breast-w	559	140	9	0	2
red hair	24638	6160	1621	1621	2
blonde hair	62297	15575	6968	6968	2
dark brown hair	202459	50615	5662	5662	2
black hair	23001	5751	1649	1649	2
skin burn	94972	23744	3158	3158	2
skin tan	108592	27148	4130	4130	2
brown hair	114502	28626	4024	4024	2
malabsorption-coeliac	3672	918	423	423	2

Table 3: Meta-validation datasets $\mathcal{D}_{\text{meta-val}}$. Train size is the number of training instances in d_{train}, and Test size is the number of test instances in d_{test}.

Dataset name	Train size	Test size	Feature size	Categorical	Classes
hapmap$_{1..5}$ (5)	1660	554	784	784	10
phoneme	4323	1081	5	0	2
wilt	3871	968	5	0	2
pendigits	8793	2199	16	0	10
satimage	5144	1286	36	0	6
credit-approval	552	138	15	9	2
banknote-authentication	1097	275	4	0	2
bank-marketing	36168	9043	16	9	2
pc4	1166	292	37	0	2
kc2	417	105	21	0	2
diabetes	614	154	8	0	2

Table 4: Meta-testing datasets $\mathcal{D}_{\text{meta-test}}$. Train size is the number of training instances in d_{train}, and Test size is the number of test instances in d_{test}. Subscripts $i..j$ and $(\cdot)$ denote the interval of indices and the total number of datasets of the same group used, respectively.

when selecting the optimal target model configuration for the task at hand by ensembling and fine-tuning the generated networks. In other words, the meta-model is fixed and ready to generate weights for a support set, without needing any hyperparameter tuning. For the fastest inference, predictions can be obtained by directly using the target network generated by HyperFast in a single forward pass. For slower but most accurate predictions, one can optimize the inference model configuration for each dataset by ensembling generated networks and fine-tuning them, using the recommended search space from Table 5.

Parameter	Range
n_ensemble	[1, 4, 8, 16, 32]
batch_size	[1024, 2048]
nn_bias	[True, False]
optimization	[None, “optimize”, “ensemble_optimize”]
optimize_steps	[1, 4, 8, 16, 32, 64, 128]
seed	[0, 1, ..., 9]

Table 5: Recommended search space for the inference framework of HyperFast.

Baselines Hyperparameter Selection For hyperparameter tuning of the baselines, we use Hyperopt (Bergstra et al. 2015), a Python library for hyperparameter optimization through Bayesian optimization. For XGBoost and CatBoost we adapt the hyperparameter search spaces from (Shwartz-Ziv and Armon 2022) and (Hollmann et al. 2023), which also tried other search spaces fixing the number of iterations and yielded suboptimal performance. For LightGBM we use the default hyperparameter search space defined in Hyperopt-sklearn (Komer, Bergstra, and Eliasmith 2014). For KNN and Logistic Regression we use the ranges used in (Hollmann et al. 2023), while for SAINT, the search space implemented in (Borisov et al. 2021). We benchmark DANet according to the configuration detailed in (Chen et al. 2022b), and for Net-DNF (Katzir, Elidan, and El-Yaniv 2020), we follow the search space suggested by the authors. For NODE (Popov, Morozov, and Babenko 2019), FT-Transformer (Gorishniy et al. 2021), and T2G-Former (Yan et al. 2023), we conducted experiments with the search

Model	Hyperparameter	Sampling	Range
KNN	n_neighbors	randint	[1, 16]
Log. Reg.	penalty	choice	[l1, l2, none]
	max_iter	randint	[50, 500]
	fit_intercept	choice	[True, False]
	C	loguniform	[e^{-5}, 5]
XGBoost	learning_rate	loguniform	[e^{-7}, 1]
	max_depth	randint	[1, 10]
	subsample	uniform	[0.2, 1]
	colsample_bytree	uniform	[0.2, 1]
	colsample_bylevel	uniform	[0.2, 1]
	min_child_weight	loguniform	[e^{-16}, e^{5}]
	alpha	loguniform	[e^{-16}, e^{2}]
	lambda	loguniform	[e^{-16}, e^{2}]
	gamma	loguniform	[e^{-16}, e^{2}]
	n_estimators	randint	[100, 4000]
LightGBM	num_leaves	randint	[5, 50]
	max_depth	randint	[3, 20]
	learning_rate	loguniform	[e^{-3}, 1]
	n_estimators	randint	[50, 2000]
	min_child_weight	loguniform	[e^{-5}, e^{4}]
	subsample	uniform	[0.2, 0.8]
	colsample_bytree	uniform	[0.2, 0.8]
	reg_alpha	choice	[0, 0.1, 1, 2, 5, 7, 10, 50, 100]
	reg_lambda	choice	[0, 0.1, 1, 5, 10, 20, 50, 100]
CatBoost	learning_rate	loguniform	[e^{-5}, 1]
	random_strength	randint	[1, 20]
	l2_leaf_reg	loguniform	[1, 10]
	bagging_temperature	uniform	[0, 1]
	leaf_estimation_iterations	randint	[1, 20]
	iterations	randint	[100, 4000]
SAINT	dim	choice	[32, 64, 128, 256]
	depth	choice	[1, 2, 3, 6, 12]
	heads	choice	[2, 4, 8]
	dropout	choice	[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
MLP	learning rate	loguniform	[e^{-9}, e^{-3}]
	batch size	uniform	[10, 2048]
	optimizer	choice	[Adam, AdamW, SGD, RMSprop]
	patience	uniform	[10, 50]
	validation split	uniform	[0.05, 0.5]
Net-DNF	number of formulas	choice	[64, 128, 256, 512, 1024, 2048, 3072]
	feature selection beta	choice	[1.6, 1.3, 1., 0.7, 0.4, 0.1]

Table 6: Hyperparameter search spaces for baseline methods. Hyperparameter configurations are drawn using the sampling technique specified in every range.

spaces provided in the original papers. However, the computational and runtime costs associated with these methods is very high, making it impractical to thoroughly explore the search spaces within the 48-hour limit set for the evaluation corpus. In every instance, configurations explored below this limit yielded inferior results compared to the default configuration of each method, with the default implementations, surpassing the time limit on the big test. As an alternative to these challenges, we use the default configuration of the models on the mini test, and also perform a sweep of epochs until early stopping is performed (default case) to assess the improvement in performance within the runtime ranges of the models comparison. For the MLP, we replicate the exact same architecture as the main network produced by Hyper-

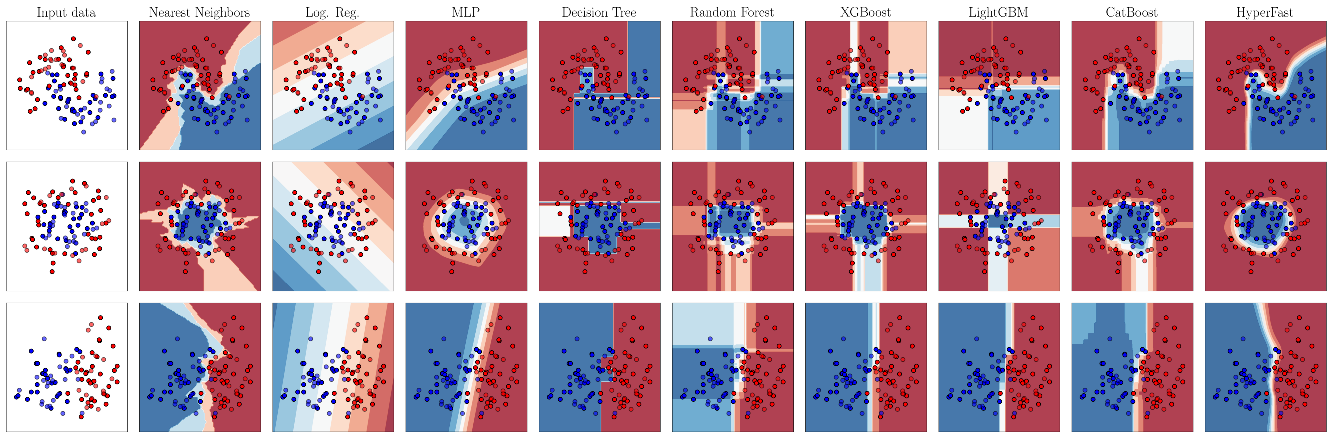

Figure 4: Classifiers comparison with the decision boundaries for toy binary classification datasets.

Fast, including the initial RF and PCA transformation layers. We perform hyperparameter tuning on the training hyperparameters, fixing the number of epochs to 1,000,000 and performing early stopping based on the validation loss. For TabPFN we consider up to 4096 data permutations for ensembling. Table 6 details the hyperparameter search spaces, as well as the sampling method used in every range for hyperparameter selection.

Additional Results

Toy datasets

In Figure 4 we compare HyperFast to traditional ML methods on toy datasets from scikit-learn (Pedregosa et al. 2011): *make_moons* in the top row, *make_circles* in the middle row, and a linearly separable dataset in the bottom row, all with Gaussian noise added. We can see how HyperFast models correctly the *moons* and *circles* without overfitting to the outliers, and creates a reasonably linear decision boundary for the bottom case. In contrast, tree-based methods overfit to the training data and fail to model accurately the distributions, creating abrupt and inaccurate decision boundaries in most cases.

How Can We Leverage All Labeled Data of a Large Dataset?

In a single forward pass, HyperFast can generate a set of weights for a smaller model ready for inference using a set of labeled samples. However, for datasets with large training sets it is not possible to use all available labeled data in a single forward pass due to memory and efficiency constraints, thus possibly losing relevant information from the dataset that could be valuable for the generation of weights to solve the task. We compare different options to leverage all labeled data in the generation of the final inference model in Figure 5.

We first experiment with increasing the batch size in a single forward pass. As we can expect, larger batch sizes yield significantly better performance, but at the cost of a much slower runtime. This is mainly due to the singular value decomposition (SVD) performed in the PCA module, although implemented and optimized for GPU, the computation time scales rapidly with the number of input samples when an excessively large batch size is used. Thus, for the trained HyperFast and for the rest of experiments, we use a fixed maximum batch size of 2048 samples, which yields very good results in less than a second.

Multiple models can be generated from different subsets of datapoints, each capturing different variations between samples. Additionally, the random features projection matrix is reinitialized in every forward pass of HyperFast, injecting more variability in all the following generated layers across models, even if the same subset of samples is used in different forward passes. We combine the predictions of multiple generated models with soft-voting ensembles, and we observe that bigger ensembles make more accurate predictions. Another alternative we experiment with is stacking the predictions of multiple main models using a Logistic Regression as the meta-learner. However, performance stagnates and does not improve with more stacking members. We also try a variant of stacking, where instead of stacking predictions from multiple models and fitting a single meta-learner, we stack the predictions and all intermediate activations from a single model and fit a meta-learner. We repeat the process for several main models and meta-learners, creating an ensemble of meta-learners. Although it is a more expensive process, we find that it yields better results than traditional stacking, performing on par with ensembling but with higher runtimes. Furthermore, we consider the weights generated by HyperFast as an starting point for fine-tuning the model on all training data. Note that in this case, all model weights are optimized: random features, PCA parameters, and linear layer weights. In Figure 5 we see that optimizing the generated model in a single forward pass with all the training data, results are worse than ensembling for a small runtime budget. But for larger runtimes, optimization outperforms ensembling and stacking on their own. Finally, we combine the two fastest and best performing options, i.e., *Optimization* + *Ensembling*, where we generate models in

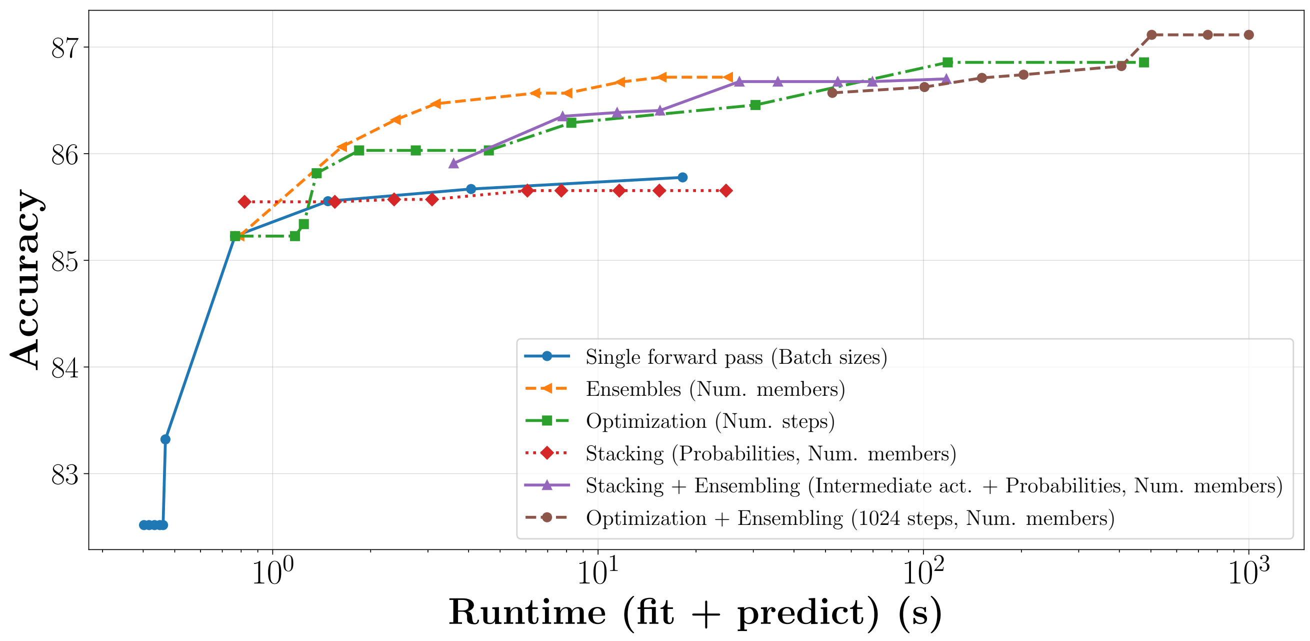

Figure 5: Performance as a function of runtime for different approaches to fully leverage all training data in the generation of the final inference model with HyperFast. Batch sizes considered in a single forward pass: [64, 128, 256, 512, 784, 1024, 2048, 4096, 8192, 16384]. Number of members considered in options involving ensembling or stacking: [1, 2, 3, 4, 8, 10, 15, 20, 32]. Optimization steps trials: [0, 2, 3, 4, 8, 16, 32, 64, 256, 1024, 4096].

	Log. Reg.	XGBoost	LightGBM	CatBoost	MLP*	ASKL 2.0	SAINT	DANet	Net-DNF	TabPFN	AutoGluon	HyperFast
hapmap_1	47.899 ± 2.618	45.598 ± 2.412	46.037 ± 1.668	44.433 ± 1.555	46.870 ± 1.409	47.816 ± 1.928	39.750 ± 2.619	31.043 ± 8.923	39.288 ± 1.625	41.339 ± 1.942	**47.992 ± 2.664**	47.758 ± 1.437
hapmap_2	49.675 ± 2.149	45.754 ± 2.242	47.502 ± 2.175	46.654 ± 1.709	48.564 ± 1.324	**50.60 ± 1.924**	40.534 ± 3.689	30.241 ± 5.934	39.776 ± 1.737	42.807 ± 1.650	49.598 ± 2.198	48.490 ± 1.736
hapmap_3	**49.702 ± 1.422**	45.862 ± 2.421	47.613 ± 2.777	45.028 ± 2.389	48.221 ± 1.713	49.142 ± 2.082	38.380 ± 2.648	29.952 ± 8.464	38.917 ± 1.602	41.109 ± 2.151	48.920 ± 2.044	48.261 ± 2.020
hapmap_4	**50.331 ± 1.990**	47.911 ± 2.854	46.988 ± 2.645	45.706 ± 3.018	48.473 ± 2.209	49.065 ± 2.190	40.792 ± 2.958	32.588 ± 6.857	39.580 ± 2.525	43.118 ± 1.945	50.077 ± 1.605	49.216 ± 2.082
hapmap_5	50.221 ± 2.092	47.567 ± 3.638	48.385 ± 2.332	47.733 ± 2.917	49.761 ± 2.173	**50.884 ± 3.262**	41.271 ± 3.099	36.132 ± 7.936	39.925 ± 2.461	42.011 ± 3.045	50.668 ± 2.50	50.173 ± 2.156
phoneme	65.172 ± 3.023	80.824 ± 1.631	80.968 ± 1.513	**81.926 ± 1.634**	79.780 ± 1.331	81.831 ± 1.542	79.548 ± 1.154	75.092 ± 4.147	50.0 ± 0.0	81.080 ± 1.274	81.748 ± 1.128	80.794 ± 0.887
wilt	69.774 ± 7.629	85.155 ± 3.311	85.535 ± 2.320	85.957 ± 2.979	91.225 ± 2.727	88.956 ± 1.718	50.0 ± 0.0	50.0 ± 0.0	50.0 ± 0.0	**91.420 ± 3.576**	87.351 ± 3.730	89.054 ± 4.225
pendigits	92.974 ± 0.594	96.348 ± 0.847	96.805 ± 0.484	97.714 ± 0.385	97.936 ± 0.245	97.433 ± 0.447	77.179 ± 2.048	96.152 ± 2.768	80.425 ± 3.754	**98.657 ± 0.241**	97.783 ± 0.276	98.498 ± 0.378
satimage	79.126 ± 0.955	85.451 ± 1.025	85.654 ± 0.705	85.481 ± 0.748	85.372 ± 1.647	85.736 ± 0.949	84.739 ± 1.713	79.597 ± 8.511	79.048 ± 1.482	85.361 ± 1.066	85.984 ± 0.853	**86.437 ± 0.542**
credit-appr.	84.352 ± 0.0	**87.290 ± 0.0**	87.0 ± 0.082	84.352 ± 0.0	84.059 ± 0.850	84.257 ± 0.708	83.334 ± 0.891	80.892 ± 1.836	80.777 ± 1.264	80.594 ± 0.0	84.694 ± 0.987	80.594 ± 0.0
bank.-auth.	98.693 ± 0.0	99.739 ± 0.138	99.837 ± 0.172	99.673 ± 0.0	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	98.733 ± 1.519	93.727 ± 2.938	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**
bank-mkt.	65.952 ± 2.422	64.973 ± 2.773	64.507 ± 2.980	65.928 ± 2.215	59.738 ± 3.748	62.806 ± 5.105	57.260 ± 4.854	50.376 ± 1.180	51.802 ± 2.549	61.591 ± 2.699	61.054 ± 3.546	**76.669 ± 1.330**
pc4	68.351 ± 0.210	75.161 ± 1.205	74.963 ± 1.554	73.099 ± 1.918	71.497 ± 1.979	74.737 ± 3.870	66.743 ± 4.820	50.0 ± 0.0	59.314 ± 6.702	72.170 ± 1.249	71.359 ± 2.406	**75.662 ± 2.736**
kc2	65.170 ± 0.0	67.908 ± 0.0	67.442 ± 0.0	63.946 ± 1.761	66.265 ± 2.503	64.540 ± 0.758	65.115 ± 2.194	63.245 ± 5.950	50.0 ± 0.0	**69.113 ± 0.0**	62.878 ± 1.905	67.442 ± 0.0
diabetes	66.926 ± 0.0	**72.204 ± 0.0**	71.280 ± 0.327	69.630 ± 1.263	70.280 ± 1.161	67.593 ± 2.703	58.713 ± 3.241	63.261 ± 5.322	58.565 ± 6.963	68.778 ± 0.0	68.935 ± 1.431	70.907 ± 0.0
Mean rank	5.953 ± 0.191	4.747 ± 0.514	4.660 ± 0.488	5.347 ± 0.514	4.720 ± 0.478	4.30 ± 0.542	8.533 ± 0.439	9.373 ± 0.524	10.113 ± 0.252	6.093 ± 0.308	4.273 ± 0.341	**3.547 ± 0.408**
Mean bal. acc.	66.955 ± 0.525	69.850 ± 0.316	70.034 ± 0.461	69.151 ± 0.328	69.869 ± 0.372	70.360 ± 0.492	61.557 ± 0.647	57.820 ± 1.470	56.743 ± 0.827	67.943 ± 0.435	69.936 ± 0.455	**71.330 ± 0.445**

Table 7: Balanced accuracy results per dataset on the mini test for a runtime budget of 5 minutes. The mean rank of each method is also shown, for 10 repetitions with different selection of samples and features to subset and create the mini test. MLP*: MLP with the exact same architecture as the main network produced by HyperFast, including the initial RF and PCA transformation layers.

different forward passes, optimize them, and combine the fine-tuned models by ensembling. We perform 1024 fine-tuning steps in each generated network with a batch size of 2048, using the AdamW optimizer with a learning rate of 1e-4, and a scheduler that reduces the learning rate by a factor of 0.1 when the loss stagnates for 10 steps. We observe that although this combination requires more runtime, a single fine-tuned model matches the performance of large ensembles of non-optimized models, and a large ensemble of fine-tuned models yields the best results. We show results by starting with a single forward pass, then increasing the ensemble size by performing multiple forward passes until GPU memory is overloaded. Then, we restart the sweep by optimizing each generated model and ensembling the fine-tuned networks.

Extended Results of Experiments

Extending the results of the main paper, Table 7 shows per dataset results on the mini test, for a total runtime budget of 5 minutes, and 10 repetitions for different sample and feature subsetting to create the small-sized mini test. These results show that HyperFast is the best option for a rapid deployment setting, outperforming TabPFN, AutoML systems and other methods. Additionally, Table 8 shows the results on the mini test for a 1h budget, but we allow an extended total runtime of 48h on the 15 datasets. With a sufficient amount

	LR	XGB	LGBM	CatB	MLP*	ASKL2	SAINT	DANet	Net-DNF	NODE	TabPFN	FT-T	T2G	AG	HF
hapmap_1	48.195	45.353	46.649	47.209	47.490	45.278	47.049	19.868	43.832	43.303	41.262	44.304	43.787	47.769	**49.640**
hapmap_2	51.434	51.372	47.968	48.139	51.529	49.552	49.332	27.122	48.572	49.535	42.778	50.242	48.085	**51.741**	51.197
hapmap_3	50.294	47.162	48.821	44.927	48.348	50.087	**50.384**	28.614	44.549	47.806	40.464	49.230	47.070	49.511	49.701
hapmap_4	51.347	49.222	50.676	47.784	49.808	51.299	46.988	31.779	44.880	45.332	40.572	45.256	47.604	**53.314**	49.640
hapmap_5	**52.170**	52.122	50.934	51.282	52.133	49.512	50.058	28.311	47.017	46.791	41.189	48.647	47.195	51.131	51.190
phoneme	66.526	82.420	81.685	83.601	82.035	**83.769**	80.716	70.754	50.0	80.403	80.703	82.953	81.691	82.970	81.146
wilt	77.284	89.259	90.003	90.166	91.019	91.073	50.0	50.0	50.0	50.0	88.352	92.833	**95.587**	90.964	91.073
pendigits	93.944	97.038	97.155	98.085	97.978	98.062	76.753	97.897	90.571	96.250	**98.823**	97.486	97.541	98.291	98.652
satimage	80.791	86.493	86.721	86.857	86.317	87.155	86.420	84.769	83.480	87.016	87.358	86.073	86.274	87.074	**87.416**
credit-appr.	84.352	86.30	**87.119**	86.30	85.650	85.480	84.831	78.720	83.532	84.288	81.893	86.438	86.268	85.650	82.063
bank.-auth.	98.693	**100.0**	**100.0**	99.673	**100.0**	**100.0**	**100.0**	98.854	99.673	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
bank-mkt.	70.475	69.063	67.949	70.859	70.508	62.490	61.104	56.968	54.101	64.596	65.0	70.648	64.214	65.529	**75.656**
pc4	68.663	78.798	77.604	76.411	76.432	**83.181**	72.873	49.609	61.914	66.102	74.631	70.877	66.710	73.438	78.212
kc2	65.170	64.567	67.442	64.567	69.113	67.908	66.840	61.829	65.772	65.170	69.715	69.113	66.375	65.635	**72.453**
diabetes	67.0	75.759	71.130	71.981	71.981	**76.889**	63.444	67.074	71.278	68.352	70.204	68.074	64.944	70.556	70.907
Mean rank	7.467	5.933	6.20	5.667	4.533	4.533	8.60	12.467	11.067	9.733	8.80	6.667	8.667	4.60	**3.933**
Mean bal. acc.	68.423	71.662	71.457	71.189	72.023	72.116	65.786	56.811	62.611	66.330	68.196	70.812	69.556	71.572	**72.596**

Table 8: Balanced accuracy results per dataset on the mini test with extended runtime. The mean rank of each method is also shown. LR: Logistic Regression; XGB: XGBoost; LGBM: LightGBM; CatB: CatBoost; MLP*: MLP with the exact same architecture as the main network produced by HyperFast, including the initial RF and PCA transformation layers; ASKL2: ASKL 2.0; FT-T: FT-Transformer; T2G: T2G-Former; AG: AutoGluon; HF: HyperFast.

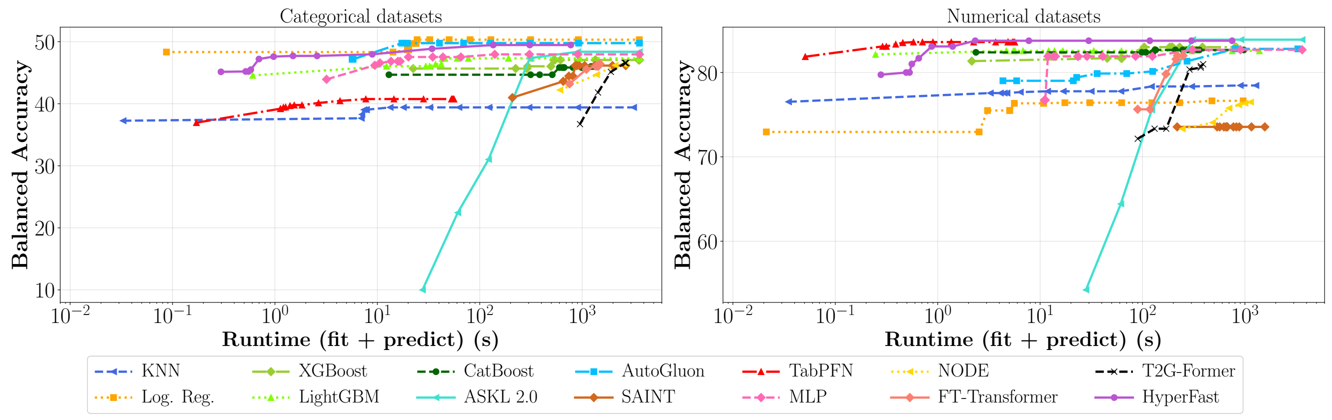

Figure 6: (left) Categorical datasets of the mini test. (right) Numerical datasets of the mini test.

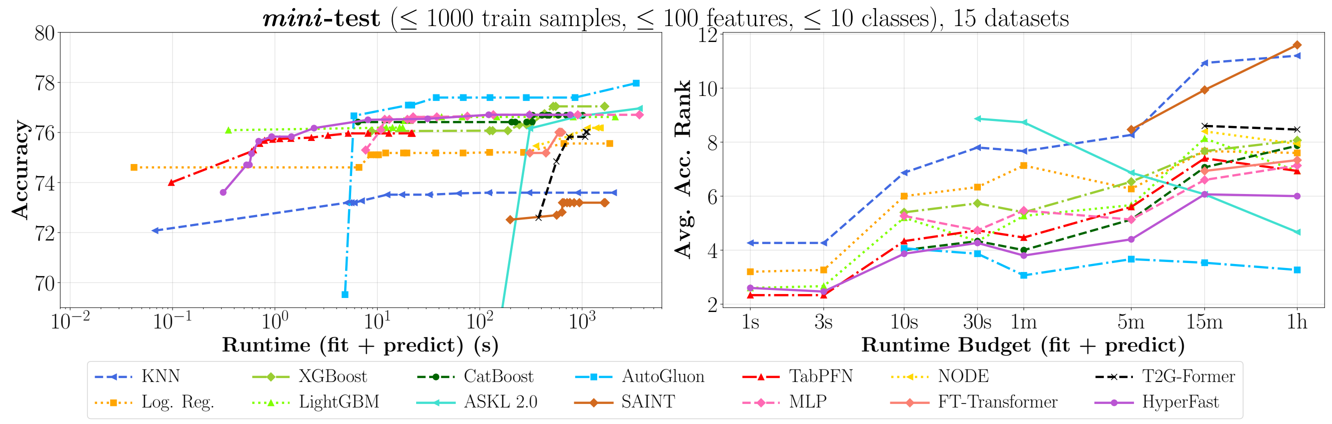

Figure 7: Runtime (fit + predict) vs. regular accuracy and average rank for given runtime budgets on the mini test: 15 small-sized meta-datasets.

	LR	XGB	LGBM	CatB	MLP*	ASKL2	SAINT	DANet	Net-DNF	AG	HF
hapmap$_1$	74.299	71.396	70.750	68.608	74.169	76.397	61.874	16.029	58.501	76.712	**80.497**
hapmap$_2$	73.507	63.447	70.622	70.695	73.509	75.869	62.326	29.873	63.359	77.210	**78.693**
hapmap$_3$	76.007	66.742	71.436	72.079	75.949	78.481	60.510	65.724	62.977	**79.548**	78.828
hapmap$_4$	75.60	74.043	71.346	68.860	75.275	77.388	52.228	66.506	62.718	79.894	**81.782**
hapmap$_5$	79.435	67.059	72.009	72.045	80.106	80.797	61.833	32.847	64.505	82.374	**83.364**
phoneme	64.553	84.717	86.633	85.467	83.482	**87.975**	81.384	70.323	81.116	86.422	83.863
wilt	69.974	89.150	88.189	91.019	91.980	**93.958**	50.0	50.0	69.231	93.051	93.903
pendigits	94.737	98.998	99.184	99.275	**99.596**	99.232	94.622	98.721	91.945	99.505	99.501
satimage	81.057	89.390	89.824	89.708	89.157	89.655	86.325	89.319	82.235	**90.942**	90.813
credit-appr.	84.352	86.949	**87.119**	86.30	84.490	84.831	81.105	80.424	81.754	85.480	82.063
bank.-auth.	98.693	99.673	**100.0**	99.673	**100.0**	**100.0**	99.673	99.673	93.453	**100.0**	**100.0**
bank-mkt.	66.174	71.593	73.814	73.771	72.952	75.426	71.687	65.436	66.523	71.491	**77.019**
pc4	68.273	75.022	76.606	76.606	74.240	**82.986**	55.360	50.0	66.688	72.049	80.599
kc2	65.170	61.090	68.045	64.567	71.249	67.908	67.908	67.442	50.0	63.499	**72.453**
diabetes	67.0	72.056	71.130	71.981	72.407	**74.185**	50.0	65.426	67.352	69.852	73.185
Mean rank	6.867	5.867	4.667	5.0	4.333	2.733	8.667	9.0	9.0	3.467	**2.267**
Mean bal. acc.	75.922	78.088	79.780	79.377	81.237	83.006	69.122	63.183	70.824	81.868	**83.771**

Table 9: Balanced accuracy results per dataset on the big test with extended runtime. The mean rank of each method is also shown. LR: Logistic Regression; XGB: XGBoost; LGBM: LightGBM; CatB: CatBoost; MLP*: MLP with the exact same architecture as the main network produced by HyperFast, including the initial RF and PCA transformation layers; ASKL2: ASKL 2.0; AG: AutoGluon; HF: HyperFast.

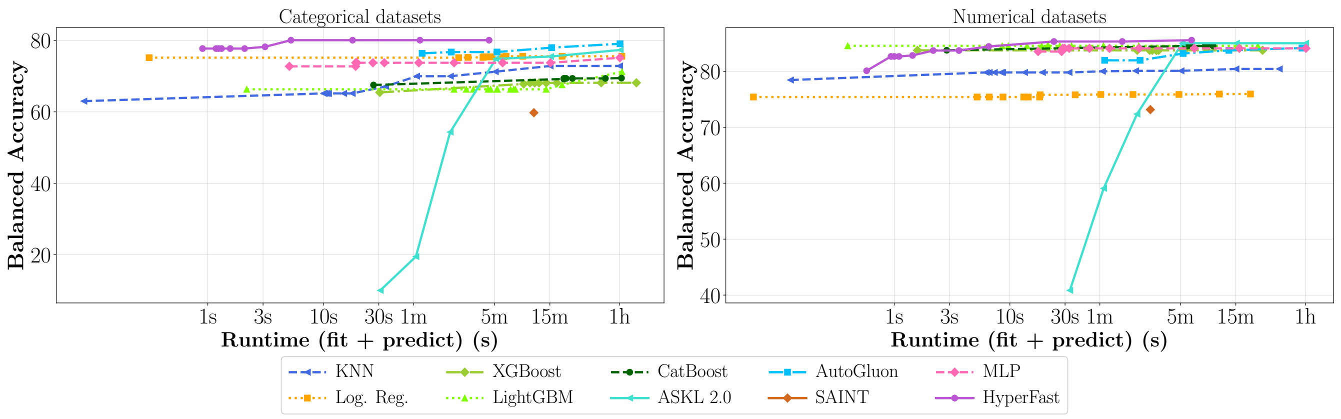

Figure 8: (left) Categorical datasets of the big test. (right) Numerical datasets of the big test.

of time for hyperparameter tuning on such small datasets, HyperFast is the best overall performing method, followed by AutoML systems and the MLP matching the architecture of HyperFast's generated main network, including the RF+PCA initial layers. In Figure 6 we can see that TabPFN underperforms for categorical datasets but obtains competitive performance for numerical datasets in a very low runtime. However, it is outperformed by HyperFast with a runtime of 2 seconds. Figure 7 shows the mini-test results in terms of regular accuracy, where TabPFN surpasses HyperFast for low runtimes, and AutoGluon also obtains better accuracy for high runtimes. This is contrary to the balanced accuracy results, which indicates that these models may underperform in accurately predicting minority classes on imbalanced datasets.

For the large datasets setting, Figure 8 shows the results of the different classifiers separated for fully categorical and numerical datasets. HyperFast obtains the best balanced accuracy results across all runtime regimes for categorical datasets. In contrast, gradient-boosting machines obtain better results for small time budgets on numerical datasets, but AutoML systems and HyperFast rapidly match and surpass their performance when more time is given to create larger ensembles and fine-tune each member. We show detailed results per dataset in Table 9 for the big test on a 1h budget per dataset, with a total extended runtime limit of 48h. In a large-scale setting, tree-based gradient-boosting machines and the MLP are outperformed by AutoML systems which, in fact, train multiple instances of these gradient boosting algorithms and neural networks (among other models) to build

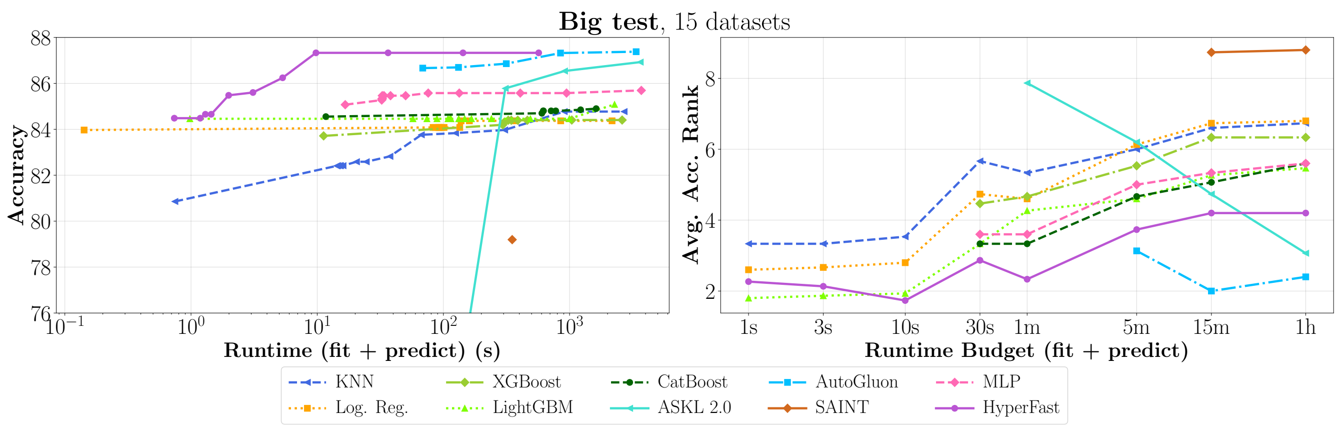

Figure 9: Runtime (fit + predict) vs. regular accuracy and average rank for given runtime budgets on the big test: 15 large/medium-sized meta-datasets.

a stronger predictor. The resulting ensemble is increasingly powerful when long fitting time budgets are allowed. Figure 9 shows that HyperFast is outperformed by AutoML systems for long runtimes in terms of regular accuracy. However, HyperFast obtains the best balanced accuracy results, which suggests that HyperFast is the model that performs more consistently across different classes, particularly in datasets where some classes are underrepresented.

When it comes to individual datasets, HyperFast outperforms other methods, especially in fully categorical datasets. One reason is the use of PCA projections before the neural layers, and the concatenation of the global average and per class average of the PCA projections to each hypernetwork module. Previous work on genetic datasets (Novembre et al. 2008) demonstrated the capability of PCA-based methods to capture the variation of samples and structure of the data. In the case of more diverse tabular datasets that have no underlying structure, the use of PCA does not have a negative impact. As a result, we have observed HyperFast also outperforming other baselines in diverse OpenML tabular datasets. When using a large number of principal components (PCs) (784) there is no information loss for datasets with d features if $d \leq 784$, which is the case for most datasets considered. Information loss in datasets with $d > 784$ is minimal, since we keep the first 784 PCs associated with the largest eigenvalues, while the remaining components explain the least amount of variance in the data. The ablation studies show that even decreasing the number of PCs to 512, performance is not very affected, while removing the PCA transformation results in the largest drop in performance. The volume of support samples also has an effect on HyperFast's performance, as larger datasets provide more robust statistical basis for the hypernetwork to accurately predict weights. In contrast, a limited number of support samples may restrict the hypernetwork's ability to capture the statistical properties of the dataset, consequently affecting the accuracy of the generated main network.

High-Dimensional Biomedical Datasets

In real-world biomedical applications, many tabular datasets exhibit very high dimensionality, making gradient-boosted trees computationally expensive and prone to overfitting, while traditional linear methods fail to capture non-linear interactions, leading to subpoptimal modeling performance. Additionally, current deep learning methods can struggle with scalability and present unfeasible training challenges when applied to datasets of such scale.

A meta-trained and scalable model, such as HyperFast, offers a new approach to address these issues and provides an alternative classification approach for real-world applications. In this work, we conduct additional experiments on two high-dimensional biomedical datasets. First, we use *hapmap-100k*, a HapMap3 (Consortium et al. 2010) dataset following the steps described in the Experimental Setup, but randomly selecting a total of 100,000 SNPs from all the available SNPs without missing data. Next, we utilize *diabetes-31k*, a UK Biobank diabetes prediction dataset including underrepresented populations (Bonet et al. 2024). While such biobanks include more diverse genetic backgrounds, the majority group in the UK Biobank includes individuals with European (British) ancestry, and other groups are still highly underrepresented. This dataset includes 31,153 SNPs for 66,302 individuals with European, South Asian, African, and East Asian ancestry.

Model	hapmap-100k	diabetes-31k	
		All	Underrep. only
Lasso	93.564	54.365	53.376
Elastic Net	94.208	53.454	52.708
LightGBM	83.301	50.412	50.419
XGBoost	82.475	50.561	50.351
HyperFast	**95.889**	**64.327**	**54.023**

Table 10: Balanced accuracy on high-dimensional biomedical datasets.

In the case of high-dimensional datasets where the support set including all features does not fit in GPU memory, we adapt HyperFast to perform feature bagging. For these experiments, we create ensembles of 32 networks and perform fine-tuning. For each ensemble member, HyperFast samples a subset of 3,000 SNPs from a multinomial distribution weighted by their standard deviation.

The results in Table 10 highlight HyperFast's robustness in high dimensional settings, achieving the best performance in both biomedical datasets, followed by the linear models. Remarkably, HyperFast obtains a 10% increase in balanced accuracy in *diabetes-31k* for test samples of all populations, and also obtains stronger predictions when only analyzing the performance for test individuals that are under-represented in the training data.

Limitations

In terms of number of samples, HyperFast takes a fixed number of training samples (support set) to predict a single set of weights. For very large datasets, the generated main network in a single forward pass will not deliver optimal results, as the sample of data points used for the generation might not fully represent the entire dataset distribution. However, rapid improvements can be obtained with the optimization and ensembling techniques detailed previously, enabling the use of any dataset size. Regarding the number of features, the input size is not fixed, as HyperFast projects the original data of any given feature size with the random features and PCA module to a fixed size. In a single forward pass, the number of input features is only restricted by the amount of GPU memory available. To address this issue, feature bagging can be used together with ensembling for dealing with very high-dimensional datasets. Note that if the number of selected features for an ensemble member is much larger than the number of PCs used, some information might be lost. To address this, one can train larger versions of HyperFast by increasing the number of retained PCs.

Our work prioritizes a simple yet effective method suitable for most tabular datasets within a constrained computational environment. Future work could explore expanding HyperFast to regression tasks and transitioning to a large-scale setup utilizing multiple GPUs for the meta-training of the model, where most information could be retained for very large numbers of features or different modalities (e.g., high resolution images). We also leave as future work the study of dataset distribution differences from meta-training and how it affects generalization performance. Lastly, in terms of number of categories, the HyperFast version discussed in this paper supports up to 100 classes. Nonetheless, training a HyperFast to accommodate more categories would increase linearly the complexity of the initial layers of the hypernetwork modules, which accounts for very small memory and computational requirements.