HyperFast: Instant Classification for Tabular Data

David Bonet[1,2], **Daniel Mas Montserrat**[1], **Xavier Giró-i-Nieto**[3,*], **Alexander G. Ioannidis**[1,†]
[1]Stanford University [2]Universitat Politècnica de Catalunya [3]Amazon

Abstract

Training deep learning models and performing hyperparameter tuning can be computationally demanding and time-consuming. Meanwhile, traditional machine learning methods like gradient-boosting algorithms remain the preferred choice for most tabular data applications, while neural network alternatives require extensive hyperparameter tuning or work only in toy datasets under limited settings. In this paper, we introduce HyperFast, a meta-trained hypernetwork designed for instant classification of tabular data in a single forward pass. HyperFast generates a task-specific neural network tailored to an unseen dataset that can be directly used for classification inference, removing the need for training a model. We report extensive experiments with OpenML and genomic data, comparing HyperFast to competing tabular data neural networks, traditional ML methods, AutoML systems, and boosting machines. HyperFast shows highly competitive results, while being significantly faster. Additionally, our approach demonstrates robust adaptability across a variety of classification tasks with little to no fine-tuning, positioning HyperFast as a strong solution for numerous applications and rapid model deployment. HyperFast introduces a promising paradigm for fast classification, with the potential to substantially decrease the computational burden of deep learning. Our code, which offers a scikit-learn-like interface, along with the trained HyperFast model, can be found at `https://github.com/AI-sandbox/HyperFast`.

1 Introduction

Many different machine learning (ML) methods have been proposed for the task of supervised classification [1], following a traditional two-stage methodology. The initial stage involves the optimization of a model using the training portion of a dataset. Several tuning iterations are performed with the aim of finding the hyperparameter configuration of the model that yields the best performance on the specific task. In the second stage, the model with the chosen hyperparameter setup is used for evaluation and inference on the test set. Training and tuning models for classification tasks is time-consuming, and it often requires extensive data pre-processing, expertise in selecting hyperparameters that could fit the task at hand, and a validation process. Further, the computational and temporal costs of the traditional process can be prohibitive, particularly in real-time applications (e.g., healthcare [2]) or when rapid model deployment is necessary [3]. In this work, we propose HyperFast, a novel method to solve classification tasks from multiple domains in a single forward pass. We substitute the slow training stage with a fixed hypernetwork that has been pre-trained (meta-trained) to predict the weights of a smaller neural network (i.e. main network) that can instantly solve the classification task with state-of-the-art performance. Recently, TabPFN [4] has been proposed, introducing a pre-trained Transformer that is able to perform classification without training. However, it is constrained to ≤ 1000 training examples, 100 features and 10 classes, which limits its application to real-world scenarios. Our model is designed for use with both large and small datasets, offering adaptability to different numbers of samples, features, and categories.

[*]Work done prior to joining Amazon.
[†]Correspondence to A.G.I. [ioannidis@stanford.edu].

Table Representation Learning Workshop at NeurIPS 2023.

During the meta-training stage, the hypernetwork parameters are learnt and the parameters of a main model are inferred, that is, we are "learning to learn" from a wide variety of datasets (meta-training datasets) from different modalities for which HyperFast generates a smaller neural model that performs the actual classification. During the meta-testing or inference stage, HyperFast receives a "support set" of an unseen dataset (both features and labels), and predicts a set of weights for the main model, which classifies the test samples of the dataset. In this way, the process of adapting the model to a new dataset is accelerated, and the model that does the meta-learning is decoupled from the model that does the actual inference on the data. Model size is also decoupled, which means that a very large model can be used for meta-learning to predict lighter models for inference. These properties are helpful to deploy models for mobile devices, to accelerate production, to improve privacy aspects, or for federated learning [5]. The hypernetwork is trained on a wide range of datasets. Due to large variability between dataset distributions, it is a challenge to learn relevant and general meta-features, such that during testing the hypernetwork is able to adapt and predict accurate set of weights for unseen datasets. We evaluate the performance of HyperFast across a set of 15 tabular datasets, including genomics datasets and a standardized suite of datasets from OpenML [6]. We also analyze the performance of HyperFast on larger time budgets by ensembling main networks generated with multiple forward passes and fine-tuning on inference. We compare our model to similar approaches and classical methods, both in terms of performance and time. Our method achieves competitive results compared to standard ML and AutoML algorithms tuned for up to one hour for each individual test dataset. We provide the code and the pre-trained HyperFast that can be used with a scikit-learn-like interface. Code is available in the Supplementary Material.

2 Related Work

Hypernetworks. Building from evolutionary algorithms, HyperNEAT [7] evolves Compositional Pattern-Producing Networks (CPPNs) to augment the weight structure for a larger main network. Based on this idea, [8] propose hypernetworks, where one neural network is used to generate weights for another neural network. The hypernetwork is trained end-to-end jointly with the main network to solve the task, producing weights in a deterministic way. [9] and [10] propose variational approximations for weight generation using normalizing flows, [11] use multilayer perceptrons (MLPs) and convolutions, and [12] use generative adversarial networks (GANs). [13] explores unsupervised weight generation through model datasets. [14] use neural representations in a similar fashion to NeRF [15] to reconstruct weights of a pre-trained neural network, also leveraging knowledge distillation. The HyperTransformer [16] is a few-shot learning hypernetwork based on the Transformer architecture [17] that generates weights of a convolutional neural network (CNN). Unlike our method, the HyperTransformer is only designed for image classification and also requires training image and activation feature extractors. HyperFast presents a novel approach by integrating hypernetworks with retrieval-based components, initial transformation modules, and different pooling operations into its architecture, offering feature permutation invariance and providing adaptability to new datasets while ensuring efficiency and speed.

Meta-learning. In the context of rapidly adapting to new tasks using limited data, meta-learning methods have emerged as powerful techniques. These approaches "learn to learn" by quickly integrating information at test time to make predictions for new, unseen tasks. A model $P_\theta(y|x, \mathcal{S})$ is learned for every new task, where y is the target, x is the test input, and $\mathcal{S} = \{X, Y\}$, is the support set. Metric-based learning methods such as Matching Networks [18] and Prototypical Networks [19] map a labelled support set $\mathcal{S}$ into an embedding space, where a distance is computed with the embedding of an unlabelled query sample to map it to its label. As in kernel-based methods, the model P_θ can be obtained through $P_\theta(y|x, \mathcal{S}) = \sum_{x_i, y_i \in \mathcal{S}} K_\theta(x, x_i) y_i$. Optimization-based methods such as Model-agnostic meta-learning (MAML) [20] learn an initial set of model parameters and perform an additional optimization through a function $f_{\theta(\mathcal{S})}$, where model weights θ are adjusted with one or more gradient updates given the support set of the task $\mathcal{S}$, i.e., $P_\theta(y|x, \mathcal{S}) = f_{\theta(\mathcal{S})}(x, \mathcal{S})$. Finally, model-based approaches such as Neural Processes (NPs) [21, 22] first process both support samples and query samples independently as in Deep Sets [23], and the predicted embeddings are aggregated with a permutation-invariant pooling operation, resulting in a dataset-level summary that is fed to a second stage network that predicts the output for the query sample. The overall model is defined by a function f and the process can be mathematically described as $P_\theta(y|x, \mathcal{S}) = f_\theta(x, \mathcal{S})$. Similarly, TabPFN [4] learns to learn Bayesian inference by using a Transformer network. In contrast,

our method directly obtains the model weights θ in a single forward step through an independent network, i.e., the hypernetwork h, such that $P_\theta(y|x, \mathcal{S}) = f_{h(\mathcal{S})}(x)$.

Deep Learning for Tabular Data. Although deep learning (DL) models achieve state-of-the-art results in many domains (e.g., language, computer vision, audio), this is not the case for tabular data. Tree-based models such as XGBoost [24], LightGBM [25] or CatBoost [26] are still the preferred choice in some tabular data applications [27, 28]. AutoML methods [29, 30, 31] are also a popular alternative, automatically selecting the most appropriate ML algorithm and its hyperparameter configuration. However, it has been shown that there is not a universal superior solution [32, 33], and many deep learning approaches for tabular data have been proposed [34]. [35] introduced Regularization Cocktails, where different regularization techniques are applied to simple MLPs to boost performance. Recent work has explored using attention mechanisms to improve performance on tabular data. TabNet [36] adopts sequential attention on subsets of features, SAINT [37] applies attention over rows and columns in a BERT-style fashion and uses contrastive pre-training with data augmentation, NPT [38] introduces attention between data points, and FT-Transformer [32] adapts a Transformer with embeddings for categorical and numerical features. Nevertheless, most of the proposed DL models for tabular data require slow training and custom hyperparameter tuning for every new dataset. In contrast, we focus on off-the-shelf models that do not need any tuning for a new task. In this direction, TabPFN [4] pre-trains a Transformer [17] on synthetic data given a prior to perform tabular data classification in a single forward pass with no hyperparameter tuning. However, TabPFN can only be applied to small tabular datasets, i.e., ≤ 1000 training examples, 100 features and 10 classes.

3 Background

3.1 Meta-learning problem setting

In our meta-learning experiments, we train a model h (i.e., the hypernetwork) that is able to quickly adapt to new tasks given some observations, and generate the weights of a main model f that solves the task for unseen datapoints. We consider a set of classification tasks $\mathcal{T}$ where each task $t \in \mathcal{T}$ is associated with a *support set* $\mathcal{S}_t$ of examples that are sufficient to find the optimal model f that solves the task, a loss function $\mathcal{L}_t$, and a *query set* $\mathcal{Q}_t$ to define $\mathcal{L}_t$. The first phase is the *meta-training*, where in each step a different training task $t \in \mathcal{T}_{\text{meta-train}}$ is selected. We compile a set of meta-training datasets $\mathcal{D}_{\text{meta-train}}$, where each dataset $d \in \mathcal{D}_{\text{meta-train}}$ is composed of a training set d_{train} and a test set d_{test}, as in the common machine learning setup. In each meta-training step, a task $t \in \mathcal{T}_{\text{meta-train}}$ is sampled by first randomly choosing a meta-training dataset d. Then, $\mathcal{S}_t$ and $\mathcal{Q}_t$ are generated by sampling examples from d_{train} and d_{test}, respectively. Meta-validation is also performed intermittently through meta-training, where a separate set of meta-validation datasets $\mathcal{D}_{\text{meta-val}}$ is used to generate validation tasks $\mathcal{T}_{\text{meta-val}}$ to evaluate our algorithm and select the best performing model.

Once HyperFast is trained, an independent set of meta-testing datasets $\mathcal{D}_{\text{meta-test}}$ are used to create the evaluation tasks $\mathcal{T}_{\text{meta-test}}$ in which the selected model is evaluated. This approach allows us to extend the classical "n-way-k-shot" few-shot learning setting to handle multiple datasets with varying distributions and categories, testing the robustness and generalization of our model on new data.

As opposed to the training tasks, where each $t \in \mathcal{T}_{\text{meta-train}}$ is randomly generated at every meta-training step, $\mathcal{T}_{\text{meta-val}}$ and $\mathcal{T}_{\text{meta-test}}$ are sets of partially fixed tasks, as the query set $\mathcal{Q}_t$ always covers all d_{test} samples, in order to evaluate and compare with other methods equally, which also tests their performance on the entire test subset d_{test} of a dataset d.

4 HyperFast

The traditional training process can be seen as a function $f(X, Y) = \theta$, that receives training instances $X \in \mathbb{R}^{N \times D}$ and corresponding labels $Y \in \mathbb{R}^N$, and produces a set of trained weights θ of a model. However, we substitute the training process with HyperFast, a pre-trained meta-model, i.e., hypernetwork [8] h, that gets a subset of the training data (i.e. support set $\mathcal{S}_t$) for a task $t \in \mathcal{T}_{\text{meta-train}}$ and predicts the weights of a main neural network f_θ for the given task t as $\theta^* = h(\mathcal{S}_t)$. The target model $f_{h(\mathcal{S}_t)}$ directly uses the predicted weights and makes predictions for test data points $x \in \mathcal{Q}_t$ in a single forward pass, such that $P_\theta(y|x, \mathcal{S}) = f_{h(\mathcal{S})}(x)$.

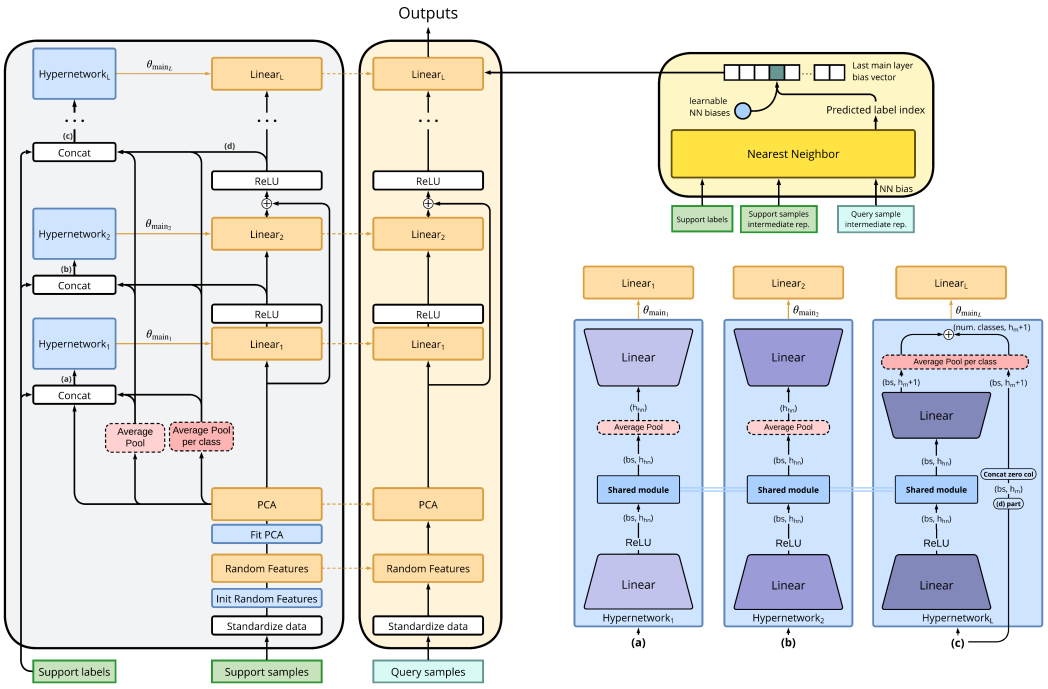

Figure 1: (left) HyperFast framework. (right) Architecture detail. Each hypernetwork module receives representations of the support set of *batch size* (bs) samples. The modules $l \in [1, L-1]$ compress the representations into a single embedding of *hypernetwork hidden size* (h_{hn}) to then generate the main network weights θ_{main_l}. Module L summarizes the representations per class with embeddings of *main hidden size* $(h_{\text{m}}) + 1$, directly obtaining the weights of the last classification layer θ_{main_L}.

The meta-model is learnt by observing a set of tasks $t \in \mathcal{T}_{\text{meta-train}}$ and minimizing $\mathcal{L}_t(f_{h(\mathcal{S})}(x))$. In this section, we detail the design and architecture of h, named HyperFast in analogy to Hypernetworks [8], and the ability to instantly adapt to new datasets in a single forward pass. Figure 1 illustrates the HyperFast framework and the main building blocks of the architecture. HyperFast is a multi-stage model with initial transformation layers that allows variable input size and permutation invariance, and a combination of linear layers and pooling operations that take both support samples and their associated labels to directly predict the weights θ_{main_l} (weight matrix and bias) of linear layers $l \in [1, L]$ of a target neural network. All trainable modules of HyperFast are learnt end-to-end by optimizing the classification loss of the main network evaluated on $\mathcal{Q}_t$.

The framework and HyperFast architecture proposed in this paper is a specific instance of a more general framework that could be easily extended to predict convolutional layers, batch normalization layers, recurrent layers, or deeper networks, for example. However, the architecture design depicted in Figure 1 selection has been driven by a global and simple approach to handle a wide range of multi-domain data, while seeking efficiency and speed.

4.1 Random Features and PCA

Properly dealing with datasets of differing dimensionality is a challenge, and one common solution is to apply padding [4] or to keep a subset of selected features up to a fixed size. We first perform a general data standardization stage by one-hot encoding categorical features, mean imputing missing numerical features, mode imputing missing categorical features, and feature-wise transforming to zero mean and unit variance. Then, HyperFast comprises initial layers that project datasets of different dimensionality to fixed-size and feature-permutation invariant representations. The kernel trick can be used to project data to a Reproducing Kernel Hilbert Space (RKHS) [39] when the number of dimensions tends to infinity. However, this would require computing all pair-wise kernel distance in every step of the training process. Instead, we use random features (RF) [40] to approximate a kernel with a fixed and finite number of dimensions. Random features are computed as $\phi(x) = a(Wx)$,

where $a(\cdot)$ is a non-linearity, and W is a random projection matrix that follows a pre-defined distribution. The approximated kernel depends on the distribution of W and the selected non-linearity. In our case, we sample W from a Gaussian distribution and use the ReLU activation as non-linearity, approximating an arc-cosine kernel. We choose to approximate the arc-cosine kernel because it captures sparse, neural network-like feature representations in a non-parametric kernel setting [41]. In contrast, polynomial kernel's features are neither sparse nor non-negative, and radial basis function (RBF) kernels capture localized similarities. In each forward step, the random features projection matrix is re-initialized and sampled. The number of rows is adjusted to match the dimensionality of the input dataset, while the number of columns remains fixed, determining the output size.

The combination of random features with Principal Component Analysis (PCA) provides an efficient low-rank randomized approximation of Kernel PCA [42, 43]. We estimate the PCA parameters ψ using the support set and project the data onto a specified number of components. Subsequently, both ϕ and ψ are applied to the query samples to transform the data. This transformed data is then forwarded through the L generated linear layers of the main network.

4.2 Hypernetwork modules

The process of generating the weights of the main network is done layer-by-layer, by multiple hypernetwork modules with both shared and layer-specific parameters, see Figure 1. The hypernetwork module that generates the weights for the main network layer l receives as input the representations of the support samples in the previous stage, concatenated with the one-hot encoded support labels, the global average, and the class average of the low-rank Kernel PCA projection of the support set. Note that each sample is concatenated with the class average corresponding to its associated label.

For predicting θ_{main_1}, the representations are the low-rank Kernel PCA projection of the support samples. For $\theta_{\text{main}_l} \in [2, L]$, the hypernetwork module receives the intermediate representations of the support set in the main network at the output of layer $l-1$, after non-linearities and residual connections are applied. Figure 1 represents a specific multi-layer perceptron (MLP) architecture with ReLU activations and residual connections, which we use for our experiments. However, the HyperFast framework can be easily extended to generate weights for other main network architectures.

The hypernetwork modules that predict the layers $l \in [1, L-1]$ are composed of MLPs with shared middle layers that take the support set representations and labels, and output embeddings for each sample in the support set. Then, permutation-invariant weights are obtained averaging all support embeddings in a similar fashion to Deep Sets [23], to obtain a single dataset embedding that is passed to a final linear layer which outputs the final weights θ_{main_l} of l. θ_{main_l} is then reshaped as weight matrix and bias vector to forward the data through the main network.

Layer L is the classification layer of the main layer that outputs the logits for the final prediction. In this case, the intermediate representations after the layer $L-1$ and labels information are encoded through a MLP hypernetwork but the weights θ_{main_L} are not directly predicted from a global embedding. Instead, we leverage the fact that the rows of the classification layer weight matrix correspond to the different categories of the task. We perform an average pooling per class, and obtain the rows of the classification weight matrix (and bias) as the average of representations for each category. This also allows a much lightweight implementation, instead of directly predict the weight matrix. Additionally, we add a residual connection [44] from the previous layer representations for which we also perform a per class average, which helps in retaining category information from the input. Finally, we consider a module based on Nearest Neighbors to add learnable parameters (NN biases) to the classification layer bias vector of the main network. The label of a query sample is predicted with NN using the support set and the intermediate representations of the data across the main network, such that $P_\theta(y|x, \mathcal{S}) = f_{h(\mathcal{S})}(x, \mathcal{S})$. We consider the representations after the PCA projection, and after each linear layer. The NN biases are added to the position of the bias of the last main classification layer corresponding to the predicted label.

Once the main network is fully generated, query samples can be forwarded to make predictions. During meta-training, the predictions for the query samples $\mathcal{Q}_t$ of $t \in \mathcal{T}_{\text{meta-train}}$ are used to compute the cross-entropy loss $\mathcal{L}_t$ and learn the parameters of HyperFast end-to-end. In evaluation, all hypernetwork parameters are frozen and generate weights for a main network in a single forward pass.

5 Experiments

In this section, we compare HyperFast to many standard ML methods, AutoML systems and DL methods for tabular data on a wide variety of tabular classification tasks, listed in the Appendix. We do not perform any hyperparameter tuning to HyperFast, as it can be used as an off-the-shelf model ready to generate networks to perform inference on new datasets. We then compare the performance and runtime of the generated network in a single forward pass, as well as the combination of multiple generated networks by ensembling and fine-tuning on inference.

Baselines We compare HyperFast to standard ML methods, AutoML systems and state-of-the-art DL methods for tabular data. We first consider simple and fast ML methods as K-Nearest Neighbors (KNN) and Logistic Regression (Log. Reg.), and a MLP matching the architecture of the main network. We also evaluate against tree-based boosting methods: XGBoost [24], LightGBM [25], and CatBoost [26]. As AutoML methods we incorporate Auto-Sklearn 2.0 (ASKL 2.0) [30], which uses Bayesian Optimization to efficiently discover a top-performing ML model or a combination of models by ensembling, and AutoGluon [31], which uses a selection of models such as neural networks, KNN, and tree-based models, combining them into a stacked ensemble. Finally, we include popular tabular DL methods: SAINT [37], and TabPFN [4]. All standard ML models, gradient boosting methods and SAINT are evaluated using 5-fold cross validation for hyperparameter adjustment. Hyperparameter configurations are drawn from search spaces (detailed in the Appendix) unitl 10 000 configurations are explored, a specified time budget is reached, or more than 32 GB of memory are required if GPU training is possible for the model. Then, the model is trained on the full training set with the best configuration between the hyperparameter search result and the default. For the AutoML methods, the time budget is given. Finally, both TabPFN and our HyperFast are pre-trained models with no hyperparameter tuning requirements, but with ensembling capabilities. Thus, we perform ensembling for each method until a given number of members are used (detailed in the Appendix) or until 32 GB of GPU memory are overloaded.

Data We collect a wide variety of datasets from different modalities. We use the 70 tabular datasets from the OpenML-CC18 suite [6] which, to the best of our knowledge, is the *largest* and most used standardized tabular dataset benchmark, composed of standard classification datasets (e.g., Breast Cancer, Bank Marketing). The collection of OpenML datasets is randomly shuffled and divided into meta-training, meta-validation and meta-testing sets, with a 75%-10%-15% split, respectively. We also include tabular genomics datasets sourced from distinct biobanks. Specifically, we utilize genome sequences of dogs [45] for dog clade (group of breeds) prediction in meta-training, European (British) humans from the UK Biobank (UKB) [46] for phenotype prediction in meta-validation, and HapMap3 [47] for subpopulation prediction in the meta-test. This strict separation ensures we meta-learn and evaluate on substantially different distributions and tasks. More details on the processing of these datasets are provided in the Appendix. The simple ML methods, implemented with scikit-learn [48], and the MLP, receive the numerical features standardized with zero mean and unit variance, and the categorical features are one-hot encoded. For the missing values, we perform mean imputation for numerical features and mode imputation for categorical features, as it was the configuration that yielded the best performance. We also perform imputation of missing values for SAINT. Boosting methods, AutoML systems, and TabPFN receive the raw data and the indices of categorical features when needed, as their documentation states that they pre-process inputs internally.

Apart from the large-sized original test datasets, we create a secondary small-sized tabular data version (*mini*-test) of the meta-testing datasets to compare to TabPFN, as it is only able to handle $\leq$ 1000 training examples, $\leq$ 100 features and $\leq$ 10 classes. We randomly select a subset of $\leq$ 1000 training samples and $\leq$ 100 features for each dataset. We do not perform any downsizing in terms of number of classes as the highest number of classes appearing in the meta-testing set is 10. However, HyperFast is pre-trained with datasets with higher number of classes and can be used in inference for datasets with $>$10 classes. The experiments, which are conducted for all models and both size versions of the 15 meta-testing datasets, considering all time budgets shown in Figure 2, require a total of 2 months to complete. Therefore, we show additional results with 10 repetitions of the experiments for a specific time budget of 5 minutes for each dataset in the Appendix.

Experimental set-up We perform supervised classification with HyperFast and all other baselines on the *mini*-test, a small-sized version of the meta-test datasets $\mathcal{D}_{\text{meta-test}}$, and in the original large-scale

datasets. To train HyperFast, we use a different set of meta-training datasets, $\mathcal{D}_{\text{meta-train}}$, and select the model with the best average performance on the meta-validation datasets, $\mathcal{D}_{\text{meta-val}}$. We report balanced accuracy, which is the mean of sensitivity and specificity. Balanced accuracy provides a more objective and robust evaluation across classes, especially in the context of imbalanced datasets. In contrast, standard accuracy can be misleading, often masking poor performance in minority classes. We evaluate the models on a time budget (including tuning, training, and prediction) to correctly assess computational complexity and performance. The average rank is also reported.

In order to transform the data to a fixed-size and permutation invariant representation, we apply Random Features and Principal Component Analysis to both support samples and query samples. We set a Random Features projection to 32 768 (2^{15}) features, sampled from a normal distribution following the He initialization [49], followed by a ReLU activation. Note that the random linear layer that computes the random features is not trained, and re-initialized in each HyperFast forward step. Then, we keep the principal components (PCs) associated to the 784 largest eigenvalues, as many of the datasets considered have this dimensionality, and it is a more than sufficient number of dimensions to retain the important information of higher dimensional datasets while preserving efficiency. After the PCA projection, most genomics datasets resemble a similar histogram distribution (i.e., zero mean, small deviation and no outliers). However, it is not the case for some OpenML datasets, which are also centered around zero but present many outliers. Thus, we clip the data after PCA at 4σ.

The hypernetwork modules receive a concatenation of intermediate representations of the support samples, and the support labels. Given that each dataset features a different number of categories and linear layers require a fixed input size, we one-hot encode the labels and apply zero padding up to the maximum number of categories considered in the experiments, which is 100. It is important to note that the number of categories that HyperFast can handle is easily extendable by expanding the input size of HyperFast and zero padding the remaining input dimensions. Such modifications have a negligible impact on efficiency or memory requirements, up to a reasonable number of categories. As a shared module we use 2 feed-forward layers with a hidden size of 1024 and ReLU activations. For the main network, we consider a 3-layer MLP with a residual connection [44], and a main network hidden size equal to the number of PCs (784 dimensions). We select this simple architecture to be able to obtain competitive performance on a wide variety of datasets with a single trained model while preserving efficiency. Other alternatives include predicting weights for CNN layers for only-image datasets, or recurrent layers, for sequential data. Instead, we create a general and simple meta-learning framework to perform fast lightweight inference. In the NN bias module, we randomly select a subset of a maximum of 2048 support samples, since computing all pairwise distances for a large number of datapoints causes high inefficiency and GPU memory overload. A maximum batch size of 2048 samples is used for training, and we make sure to have a sufficient number of samples per category in every case.

In evaluation, we show prediction results with HyperFast in a single forward pass, as well as predictions by ensembling main networks generated in multiple forward passes. We also experiment with performing gradient steps with the training data of the meta-testing datasets on the generated main networks, including the random features projection matrix, PCA parameters and linear layers.

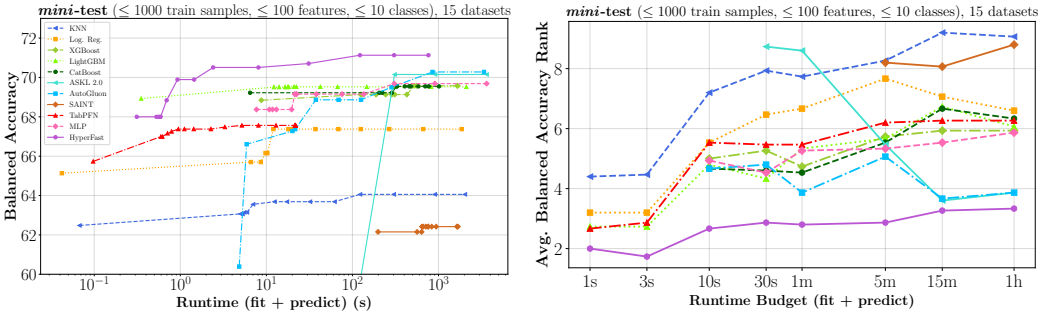

Figure 2: Runtime (fit + predict) vs. performance and average rank for given runtime budgets on the *mini*-test (small-sized version of the 15 meta-test datasets with $\leq$ 1000 training examples, $\leq$ 100 features and $\leq$ 10 classes restrictions).

Results on small-sized datasets. We first compare HyperFast to the other methods on a small-sized setting with datasets having ≤ 1000 training samples, ≤ 100 features and ≤ 10 classes, in order to compare to TabPFN. As shown in Figure 2, HyperFast delivers superior results in both performance and runtime, with better prediction capabilities up to 3 orders of magnitude faster than competing methods. Simple ML methods such as KNN and Log. Reg. also deliver instant predictions, but do not achieve remarkable performance. Interestingly, an MLP (with an architecture identical to the network generated by HyperFast, including the RF+PCA layers) performs on par with XGBoost. However, HyperFast surpasses gradient-boosting techniques in both runtime and performance. LightGBM stands out as the only boosting machine that achieves a higher balanced accuracy in a similar runtime to a single forward pass by HyperFast. Yet, an ensemble of networks generated by HyperFast outperforms all fine-tuned boosting machines in under 3 seconds. TabPFN is noted for its rapid predictions and outperforms SAINT. But on average, it falls behind gradient boosting machines and neural models, including HyperFast. AutoML systems are superior to the other baselines when given higher runtime budgets. However, HyperFast still outperforms both AutoGluon and ASKL 2.0 for runtimes up to 1h, obtaining the lowest rank throughout all the budgets in the mini test.

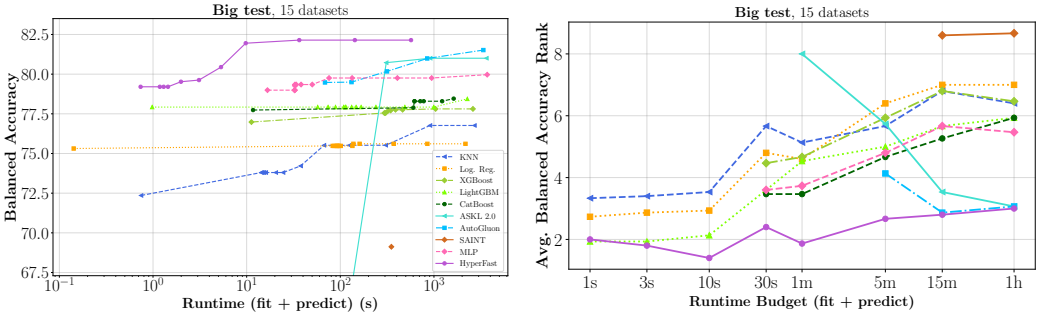

Figure 3: Runtime (fit + predict) vs. performance and average rank for given runtime budgets on the big test: 15 large/medium-sized meta-datasets.

Results on medium/large-scale datasets. Figure 3 benchmarks the algorithms on a large real-world collection of datasets. We observe that HyperFast achieves the overall best performance in a wide range of runtime budgets, ranging from 1 second to 5 minutes. For more extended budgets, up to 1h per dataset, HyperFast's performance is on par with other AutoML systems. Specifically, HyperFast, ASKL 2.0, and AutoGluon all achieve an average rank of approximately 3.0. In comparison, gradient-boosting machines plateau at a balanced accuracy of 78.4% and rank above 5.9, being outperformed by the MLP. SAINT obtains the lowest performance, using the hyperparameter configuration that the authors implement for the biggest datasets they consider in their benchmark, which are of similar size to the ones considered in this evaluation. No hyperparameter optimization is performed for SAINT in the big test since larger architectures do not fit in GPU memory for the larger datasets.

6 Conclusion

We present HyperFast, a meta-trained hypernetwork designed to perform instant classification of tabular data by encoding task information in the prediction of the weights of a target network in a single forward pass. Our experiments show that HyperFast consistently improves performance over traditional ML methods and tabular-specific DL architectures in a matter of seconds. Remarkably, it is able to replace the traditional training of a neural network, and achieves competitive results with state-of-the-art AutoML frameworks trained for 1h. HyperFast eliminates the necessity for hyperparameter tuning, making it a highly accessible, off-the-shelf model that can be specially useful for fast classification tasks. We also explore how we can leverage all training data by creating ensembles of generated networks and fine-tuning them on inference, significantly boosting performance at almost no additional computational cost. Future work should consider expanding this framework to a general architecture or multi-hypernetwork setting that is able to handle regression tasks, multi-domain and high-dimensional non-tabular settings such as audio streams, 3D, and video.

Acknowledgments and Disclosure of Funding

This work was partially supported by a grant from the Stanford Institute for Human-Centered Artificial Intelligence (HAI) and by NIH under award R01HG010140. This research has been conducted using the UK Biobank Resource under Application Number 24983.

References

[1] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, USA, 2000.

[2] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[3] Allison McCarn Deiana, Nhan Tran, Joshua Agar, Michaela Blott, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Scott Hauck, Mia Liu, Mark S Neubauer, et al. Applications and techniques for fast machine learning in science. *Frontiers in big Data*, 5:787421, 2022.

[4] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023.

[5] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13(3):1–207, 2019.

[6] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael Gomes Mantovani, Jan N van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[7] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[8] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[9] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian hypernetworks, 2018.

[10] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2218–2227. JMLR.org, 2017.

[11] Lior Deutsch. Generating neural networks with neural networks. *arXiv preprint arXiv:1801.01952*, 2018.

[12] Neale Ratzlaff and Li Fuxin. HyperGAN: A generative model for diverse, performant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5361–5369. PMLR, 09–15 Jun 2019.

[13] Konstantin Schürholt, Boris Knyazev, Xavier Giró i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[14] Maor Ashkenazi, Zohar Rimon, Ron Vainshtein, Shir Levi, Elad Richardson, Pinchas Mintz, and Eran Treister. Nern – learning neural representations for neural networks, 2022.

[15] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[16] Andrey Zhmoginov, Mark Sandler, and Maksym Vladymyrov. Hypertransformer: Model generation for supervised and semi-supervised few-shot learning. In *International Conference on Machine Learning*, pages 27075–27098. PMLR, 2022.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[18] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[19] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[21] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[22] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR, 2018.

[23] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[24] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[25] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[26] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[27] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[28] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Inf. Fusion*, 81(C):84–90, may 2022.

[29] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[30] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2020.

[31] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[32] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

[33] Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Ganesh Ramakrishnan, Micah Goldblum, Colin White, et al. When do neural nets outperform boosted trees on tabular data? *arXiv preprint arXiv:2305.02997*, 2023.

[34] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *CoRR*, abs/2110.01889, 2021.

[35] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 23928–23941. Curran Associates, Inc., 2021.

[36] Sercan Ö. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021.

[37] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[38] Jannik Kossen, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems*, 34:28742–28756, 2021.

[39] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

[40] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[41] Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

[42] Bharath Sriperumbudur and Nicholas Sterge. Approximate kernel pca using random features: Computational vs. statistical trade-off. *arXiv preprint arXiv:1706.06296*, 2017.

[43] David Lopez-Paz, Suvrit Sra, Alex Smola, Zoubin Ghahramani, and Bernhard Schölkopf. Randomized nonlinear component analysis. In *International conference on machine learning*, pages 1359–1367. PMLR, 2014.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[45] Emily R. Bartusiak, Míriam Barrabés, Aigerim Rymbekova, Julia Gimbernat-Mayol, Cayetana López, Lorenzo Barberis, Daniel Mas Montserrat, Xavier Giró-I-Nieto, and Alexander G. Ioannidis. Predicting dog phenotypes from genotypes. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 3558–3562, 2022.

[46] Cathie Sudlow, John Gallacher, Naomi Allen, Valerie Beral, Paul Burton, John Danesh, Paul Downey, Paul Elliott, Jane Green, Martin Landray, et al. Uk biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS medicine*, 12(3):e1001779, 2015.

[47] International HapMap 3 Consortium et al. Integrating common and rare genetic variation in diverse human populations. *Nature*, 467(7311):52, 2010.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[50] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4367–4375, 2018.

[51] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7229–7238, 2018.

[52] Junyang Qian, Yosuke Tanigawa, Wenfei Du, Matthew Aguirre, Chris Chang, Robert Tibshirani, Manuel A Rivas, and Trevor Hastie. A fast and scalable framework for large-scale and ultrahigh-dimensional sparse regression with application to the uk biobank. *PLoS genetics*, 16(10):e1009141, 2020.

[53] Yosuke Tanigawa, Junyang Qian, Guhan Venkataraman, Johanne Marie Justesen, Ruilin Li, Robert Tibshirani, Trevor Hastie, and Manuel A Rivas. Significant sparse polygenic risk scores across 813 traits in uk biobank. *PLoS Genetics*, 18(3):e1010105, 2022.

[54] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic gradient descent. In *International Conference on Learning Representations (ICLR)*, pages 1–15, 2015.

[55] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

[56] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, volume 9, page 50. Citeseer Austin, TX, 2014.

[57] John Novembre, Toby Johnson, Katarzyna Bryc, Zoltán Kutalik, Adam R Boyko, Adam Auton, Amit Indap, Karen S King, Sven Bergmann, Matthew R Nelson, et al. Genes mirror geography within europe. *Nature*, 456(7218):98–101, 2008.

A Further Motivations of HyperFast

In this study, we are particularly interested in ensuring adaptability to large dataset sizes, filling the gap present in the current landscape of pre-trained models for instant tabular data classification, where models like TabPFN [4] are promising but inherently limited to small datasets due to the constraints of the Transformer architecture. In contrast, we explore a different route with HyperFast, a pre-trained hypernetwork-based framework capable of rapid classification without such limitations. A key aspect of this adaptability comes from the architecture design, providing invariance to permutations of input samples with scalability with respect to the dataset size, as well as feature-permutation invariance.

HyperFast solves the classification task by taking a set of labeled datapoints (support set) and generating the weights of a neural model that can be directly used to classify new unseen datapoints. Previous work [50, 51] considered generating weights for specific layers (e.g., the last classification layer), while training the rest of the feature extractor. Here, we go one step further and consider generating all the weights of the model that performs the classification in a single forward pass.

We also focus on applications and scenarios where the computational budget can be limited. We address this concern by decoupling the complexity of specialized models that perform individual tasks from a general meta-model. In other words, we train a high-capacity meta-model to encode task-specific characteristics in the weights of a smaller model. This setting allows a large meta-learner to be trained just once, while many lightweight models generated by the meta-learner can be used for deployment in different applications such as edge computing and IoT devices, where computational resources are constrained, and fast inference is indispensable. Additionally, the meta-learner can be used in data streaming applications, where models need to be updated or trained frequently. The meta-learner can instantly generate a model that is ready for deployment, but the generated weights might not be optimal. Thus, we also explore further improvements to quickly boost the performance before deployment and leverage all the power of the framework. For example, ensembles of multiple generated models can be used, or the generated weights can be used as an initial point for fine-tuning. More detail on many of the possibilities to improve model performance can be found in the Additional Results Section.

B Experimental Setup

B.1 Datasets

OpenML We integrate the OpenML Curated Classification benchmarking suite 2018 (OpenML-CC18) [6]. 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 [47] 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 [45] 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 [46] is a large-scale biobank, from which we use the genotyping array data and full phenotypes as processed in [52]. 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 [53]. We split each dataset into 80% train and 20% test.

Table 1: 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
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 2: 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.

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 3: 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
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

The collection of OpenML datasets is randomly shuffled and divided into meta-training (Table 3), meta-validation (Table 1), and meta-testing (Table 2) 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.

B.2 HyperFast and Baselines Implementation

B.2.1 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. 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 [54], which required 20 hours on a single NVIDIA Tesla V100 SXM2 GPU.

B.2.2 Baselines Hyperparameter Selection

For hyperparameter tuning of the baselines, we use Hyperopt [55], a Python library for hyperparameter optimization that uses Bayesian optimization. For XGBoost and CatBoost we adapt the hyperparameter search spaces from [28] and [4], 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 [56]. For KNN and Logistic Regression we use the ranges used in [4], and for SAINT the search space implemented in [34]. For the MLP, we use the exact same architecture as the main network produced by HyperFast, 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 4 details the hyperparameter search spaces, as well as the sampling method used in every range for hyperparameter selection.

Table 4: Hyperparameter search spaces for baseline methods. Hyperparameter configurations are drawn using the sampling technique specified in every range.

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]

C Additional Results

C.1 Toy datasets

In Figure 4 we compare HyperFast to traditional ML methods on toy datasets from scikit-learn [48]: *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.

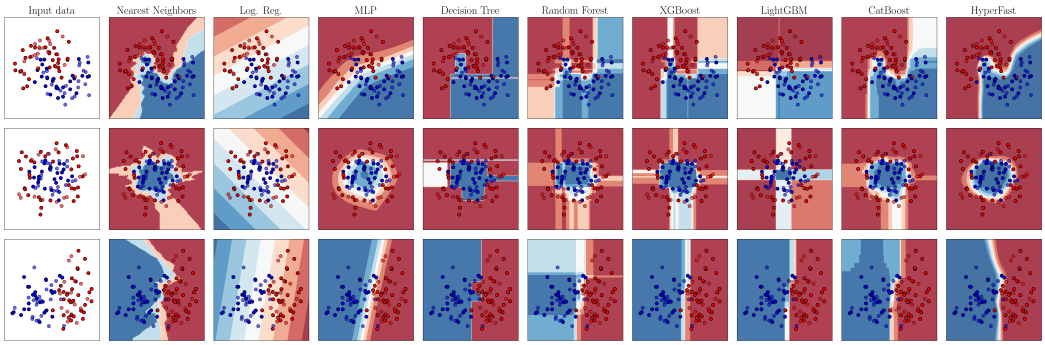

Figure 4: Classifiers comparison with the decision boundaries for toy binary classification datasets.

C.2 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.

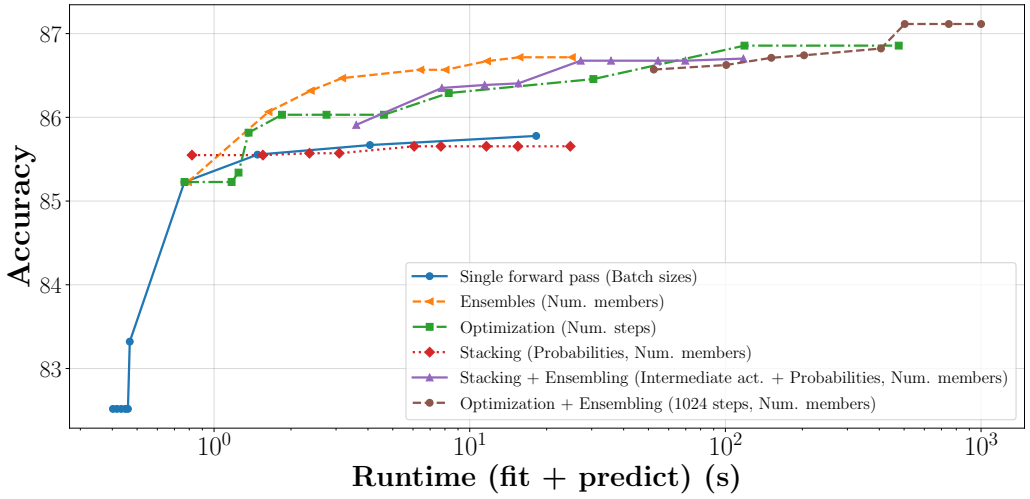

Figure 5: Performance as a function of runtime for different approaches to fully leverage all training data in the generation of the final inference model with HyperFast. Batch sizes considered in a single forward pass: [64, 128, 256, 512, 784, 1024, 2048, 4096, 8192, 16384]. Number of members considered 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].

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 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. In our sweep experiments, 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.

After validating these results with the meta-validation datasets, we recommend using HyperFast with a single forward pass for instant predictions, or using ensembles to obtain good predictions in a very short runtime. If best performance is required and runtime is not a priority, fine-tuning + ensembling is the best choice.

C.3 Ablation Experiments

Table 5: Ablation studies on HyperFast performing a single forward pass. Time results are shown for a single GPU. *HF size* denotes the number of trainable parameters of HyperFast, i.e., the meta-model, while *Model size* denotes the size of the generated model.

Variation	Bal. acc. (%)	Bal. acc. diff.	Fit time (s)	Pred. time (s)	HF size	Model size
Base model (784 PCs)	81.496	-	0.600	0.125	1.27 B	52.65 M
No RF	75.387	-6.108	0.126	0.114	1.27 B	1.85 M
No RF-PCA	73.704	-7.792	0.029	0.109	1.26 B	1.23 M
First 512 PCs only	81.347	-0.149	0.625	0.125	547 M	43.03 M
First 256 PCs only	81.235	-0.261	0.640	0.042	140 M	34.25 M
d_{RF}=16 384 (2^{14})	81.059	-0.436	0.510	0.116	1.27 B	26.95 M
No concat PCA to hypern. modules	80.727	-0.769	0.583	0.125	1.26 B	52.65 M
1 linear layer in shared module	80.790	-0.706	0.637	0.125	1.27 B	52.65 M
No residual conn. in hypern.$_L$	77.835	-3.660	0.620	0.125	1.27 B	52.65 M
No residual con. in main model	80.318	-1.178	0.633	0.125	1.27 B	52.65 M
No NN bias using PCA features	81.305	-0.191	0.625	0.125	1.27 B	52.65 M
No NN bias using interm. act.	80.703	-0.793	0.628	0.125	1.27 B	52.65 M
No NN biases	79.714	-1.781	0.628	0.125	1.27 B	52.65 M
Random init. linear layers main	72.229	-9.267	0.437	0.125	-	52.65 M

In Table 5, we present ablation studies for the HyperFast framework, exploring variations affecting both hypernetwork modules and the generated model. First, we consider removing the RF and both RF and PCA modules, obtaining a fixed-sized input by keeping the first 784 features or applying zero padding. The weight generation time is reduced from 0.6s to 0.12s and 0.03s, since the main time bottleneck is the RF matrix multiplication and SVD to obtain the PCs. Also, the main model size is greatly reduced as RFs account for most parameters, but the drop in performance is one of the most significant. This is because RF and PCA not only allow transforming any dataset to a fixed number of features, but also homogenize the input data to HyperFast and the generated network

across datasets. For example, the first feature post RF-PCA holds the most variance, with subsequent features capturing the maximum variance that is orthogonal to the previous dimensions, with minimal information loss. Also, histogram distributions are similar across datasets with zero mean. These properties help in learning important meta-features across different dataset distributions. If we scale down RF-PCA by reducing the number of PCs used and the RF dimensionality, we observe that model size is significantly reduced while the drop in performance is not critical, which shows that most dataset relevant information is preserved, even using 512 or 256 PCs. These observations can help create even more efficient HyperFast desings in the future. In addition, PCA representations concatenated to hypernetwork inputs retain key information without a major parameter increase. We also observe that reducing the shared hypernetwork module from 2 to 1 layer degrades performance, and residual connections in both the hypernetwork and main model are key to retain post-PCA and per class information, while not increasing model size and runtime. We also analyze the retrieval-based component of HyperFast. We observe that NN biases in the last classification layer improve predictions while maintaining model size, especially using the intermediate activations of the main network as features. Finally, if we replace the weights produced by HyperFast by random weights, and base the prediction solely on the Nearest Neighbor-based component, we observe the biggest drop in performance.

C.4 Extended Results of Experiments

Extending the results of the main paper, Table 6 shows per dataset results on the mini test, for a total runtime budget of 5 minutes and 10 repetitions. These results show that HyperFast is the best option for a rapid deployment setting, outperforming TabPFN, AutoML systems and other methods. Additionally, Table 7 shows the results for a total runtime budget of up to 1h for the mini test. Allowing a sufficient amount of time for hyperparameter tuning on such small datasets, the results are more diverse. AutoML systems and HyperFast are the best overall performing methods, followed by the MLP matching the architecture of HyperFast's generated main network, including the RF+PCA initial layers. HyperFast obtains the best average rank, while the ensemble of HyperFast and AutoGluon obtains the best mean balanced accuracy.

Table 6: 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.

	Log. Reg.	XGBoost	LightGBM	CatBoost	MLP	ASKL 2.0	SAINT	TabPFN	AutoGluon	HyperFast
$hapmap_1$	45.768 ± 2.09	45.492 ± 4.07	47.005 ± 2.45	44.437 ± 1.78	45.642 ± 3.94	45.140 ± 5.58	38.373 ± 2.45	40.270 ± 1.47	47.401 ± 2.28	**47.480 ± 0.99**
$hapmap_2$	**50.781 ± 3.51**	46.152 ± 2.86	48.096 ± 2.79	45.446 ±1.41	47.111 ± 1.62	49.923 ± 1.10	42.943 ± 3.57	41.816 ± 1.15	49.689 ± 1.77	50.392 ± 2.85
$hapmap_3$	48.002 ± 1.65	45.315 ± 2.34	46.020 ± 2.84	44.401 ± 2.00	47.324 ± 1.63	40.856 ± 17.40	38.199 ± 3.54	41.337 ± 0.89	46.678 ± 2.46	**48.002 ± 1.08**
$hapmap_4$	48.699 ± 1.64	45.444 ± 2.71	46.171 ± 1.08	45.621 ± 3.03	48.493 ± 1.15	47.801 ± 2.65	43.513 ± 1.85	43.303 ± 1.69	**50.601 ± 1.24**	48.787 ± 1.79
$hapmap_5$	49.071 ± 2.49	49.378 ± 2.52	48.636 ± 2.85	46.543 ± 1.25	48.977 ± 1.96	43.616 ± 18.94	42.290 ± 5.39	42.353 ± 2.35	49.741 ± 1.72	**50.303 ± 2.05**
phoneme	62.307 ± 3.76	80.854 ± 1.49	81.100 ± 1.59	82.223 ± 1.05	78.463 ± 1.20	**83.310 ± 0.88**	80.022 ± 1.12	80.956 ± 1.30	80.392 ± 1.14	80.073 ± 1.09
wilt	67.793 ± 10.57	82.902 ± 4.20	83.628 ± 3.68	84.600 ± 3.16	86.491 ± 4.17	88.123 ± 1.99	50.000 ± 0.00	**90.903 ± 3.71**	88.809 ± 1.69	88.508 ± 2.52
pendigits	93.396 ± 0.55	95.916 ± 0.52	96.593 ± 0.32	97.355 ± 0.54	97.454 ± 0.60	97.261 ± 0.24	77.635 ± 2.37	**98.447 ± 0.23**	97.766 ± 0.18	98.427 ± 0.43
satimage	78.745 ± 1.52	85.239 ± 0.88	85.435 ± 0.94	85.611 ± 1.51	84.179 ± 0.68	72.184 ± 31.04	85.310 ± 1.84	85.566 ± 1.00	85.796 ± 0.97	**86.153 ± 1.32**
credit-approval	83.928 ± 0.36	**85.691 ± 0.29**	85.152 ± 1.10	84.667 ± 0.37	82.925 ± 1.35	84.626 ± 0.47	83.553 ± 1.28	81.243 ± 0.00	84.865 ± 0.51	80.894 ± 1.33
banknote-auth.	98.693 ± 0.00	99.722 ± 0.30	99.738 ± 0.15	99.738 ± 0.15	**100**	**100**	99.869 ± 0.18	**100**	**100**	**100**
bank-marketing	65.340 ± 2.47	63.275 ± 6.20	65.891 ± 3.92	66.265 ± 3.40	64.711 ± 3.66	62.109 ± 8.55	54.375 ± 4.14	59.862 ± 2.59	60.173 ± 3.21	**67.770 ± 0.38**
pc4	67.834 ± 1.01	74.549 ± 2.40	73.316 ± 2.95	74.905 ± 1.34	69.753 ± 2.32	**78.255 ± 3.03**	69.136 ± 3.56	74.436 ± 1.63	71.215 ± 2.47	76.866 ± 2.67
kc2	62.656 ± 0.81	59.381 ± 0.96	61.561 ± 0.75	59.847 ± 2.30	63.899 ± 4.38	65.997 ± 1.89	62.684 ± 5.85	**69.715 ± 0.00**	63.713 ± 1.62	66.572 ± 2.64
diabetes	66.000 ± 0.00	**71.260 ± 1.78**	68.911 ± 0.03	68.748 ± 0.07	70.441 ± 2.07	69.145 ± 1.45	57.667 ± 4.94	70.204 ± 0.00	69.063 ± 1.01	69.681 ± 1.50
Mean rank	6.84 ± 0.3	5.28 ± 0.8	5.25 ± 0.4	5.71 ± 0.8	4.89 ± 0.6	3.41 ± 0.7	7.60 ± 0.7	5.27 ± 0.2	4.04 ± 0.5	**3.31 ± 0.6**
Mean bal. acc.	65.934 ± 0.53	68.705 ± 0.59	69.150 ± 0.23	68.694 ± 0.48	69.058 ± 0.63	68.556 ± 5.02	61.705 ± 1.13	68.027 ± 0.35	69.727 ± 0.64	**70.661 ± 0.32**

For the large datasets setting, Figure 6 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 8 for a 1h runtime budget. 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 a stronger predictor. The resulting ensemble is increasingly powerful when long fitting time budgets are allowed. Although our focus is on fast classification, HyperFast is still competitive in most datasets. In fact, if we combine the power of both HyperFast and AutoGluon in an ensemble we obtain the best results, which suggest that including HyperFast in AutoML systems could be very beneficial.

Table 7: Balanced accuracy results per dataset on the mini test for a runtime budget of 1h. The mean rank of each method is also shown. These extended experiments also include *HyperFast + AutoGluon*, an ensemble of HyperFast and AutoGluon. Thus, the average rank does not correspond to the main paper experiments. LR: Logistic Regression, XGB: XGBoost, LGBM: LightGBM, CatB: CatBoost, ASKL2: ASKL 2.0, AG: AutoGluon, HF: HyperFast.

	LR	XGB	LGBM	CatB	MLP	ASKL2	SAINT	TabPFN	AG	HF	HF + AG
hapmap$_1$	46.763	42.562	45.126	45.285	45.71	42.544	44.17	40.407	46.583	47.832	**48.511**
hapmap$_2$	45.395	47.472	45.091	47.103	47.668	47.055	35.981	41.787	50.164	49.230	**51.175**
hapmap$_3$	49.675	47.075	48.769	41.839	43.422	50.087	49.298	40.275	49.511	49.701	**50.251**
hapmap$_4$	49.063	49.222	45.415	43.37	46.376	48.371	44.757	40.572	48.516	**49.640**	46.664
hapmap$_5$	50.122	48.972	48.727	45.51	**52.133**	49.301	48.065	40.621	50.922	50.929	48.574
phoneme	62.433	80.566	81.535	81.265	78.891	80.646	80.716	80.033	**82.97**	80.715	80.622
wilt	73.656	78.682	72.006	86.32	85.359	**91.073**	50.0	86.429	90.964	**91.073**	89.259
pendigits	93.521	96.32	96.448	97.752	96.721	98.062	72.179	**98.78**	98.125	98.590	97.682
satimage	78.876	84.916	85.534	85.845	85.063	85.444	82.188	86.674	84.907	**87.093**	86.035
credit-appr.	79.806	**85.821**	84.831	83.362	80.392	85.31	84.181	81.243	83.67	82.063	81.414
banknote-auth.	98.693	**100.0**	99.673	99.673	**100.0**	**100.0**	98.693	**100.0**	**100.0**	**100.0**	**100.0**
bank-marketing	65.898	61.686	64.215	53.457	**70.508**	62.458	57.073	62.027	60.153	68.195	69.282
pc4	68.273	71.05	74.826	70.269	69.076	74.24	50.0	74.436	71.658	**78.212**	78.016
kc2	64.704	62.897	60.624	60.022	62.897	62.897	66.238	**69.715**	62.897	62.760	67.442
diabetes	66.0	70.981	65.574	67.5	69.259	65.0	50.0	70.204	69.778	70.907	**72.259**
Mean rank	7.35	5.59	6.82	7.41	5.59	4.88	8.41	6.06	4.29	**2.77**	3.35
Mean bal. acc.	66.192	68.548	67.893	67.238	68.898	69.499	60.902	67.547	70.055	71.129	**71.146**

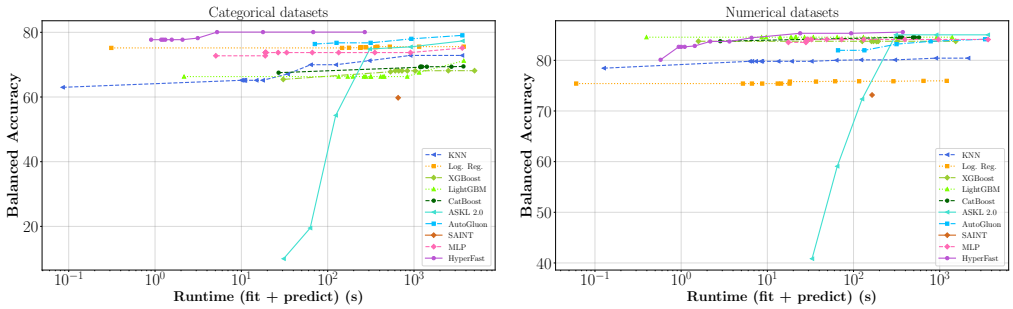

Figure 6: (left) Categorical datasets of the big test. (right) Numerical datasets of the big test.

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 [57] 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.

D 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 datapoints used for the generation might not fully represent the entire dataset. However, rapid improvements can be obtained with the optimization and ensembling techniques detailed previously, enabling any dataset size to be used. Regarding the number of features, there is also no limitation, as HyperFast projects the original data with the random features and PCA module to a fixed size, and feature selection can be used with

Table 8: Balanced accuracy results per dataset on the big test for a runtime budget of 1h. The mean rank of each method is also shown. These extended experiments also include *HyperFast + AutoGluon*, an ensemble of HyperFast and AutoGluon. Thus, the average rank does not correspond to the main paper experiments. LR: Logistic Regression, XGB: XGBoost, LGBM: LightGBM, CatB: CatBoost, ASKL2: ASKL 2.0, AG: AutoGluon, HF: HyperFast.

	LR	XGB	LGBM	CatB	MLP	SAINT	ASKL2	AG	HF	HF + AG
$hapmap_1$	73.445	68.117	70.75	68.445	72.976	61.874	76.397	76.712	74.490	**77.529**
$hapmap_2$	73.357	62.651	70.622	68.028	73.268	62.326	75.869	76.782	75.855	**79.633**
$hapmap_3$	75.32	66.742	71.436	70.031	75.949	60.51	76.784	**79.548**	74.338	77.743
$hapmap_4$	75.45	72.692	71.346	68.536	73.462	52.228	76.815	79.894	79.282	**80.070**
$hapmap_5$	79.105	67.059	72.009	71.0	80.106	61.833	80.797	82.374	80.158	**83.637**
phoneme	64.553	84.717	86.633	84.94	83.247	81.384	**87.437**	86.422	82.839	84.568
wilt	69.974	89.15	86.32	90.166	90.221	50.0	91.128	93.051	**93.903**	92.996
pendigits	94.563	98.998	99.14	99.184	**99.457**	94.622	99.232	99.454	99.410	**99.548**
satimage	80.822	89.39	89.824	89.67	88.584	86.325	89.332	**90.759**	90.731	90.616
credit-appr.	79.806	**86.949**	84.831	85.48	83.362	81.105	84.66	83.702	81.584	82.883
banknote-auth.	98.693	99.673	**100.0**	99.673	**100.0**	99.673	**100.0**	**100.0**	**100.0**	**100.0**
bank-marketing	64.951	70.335	72.853	73.771	72.334	71.687	**75.186**	70.406	72.363	71.410
pc4	68.273	75.022	74.631	69.466	72.656	55.36	68.468	72.049	**77.821**	73.850
kc2	64.704	57.749	60.624	61.692	61.692	67.908	65.17	62.897	**69.578**	**69.578**
diabetes	66.0	72.056	65.574	68.204	**72.185**	50.0	64.574	68.704	70.407	71.833
Mean rank	7.59	6.65	5.41	6.24	4.88	8.35	3.71	3.06	3.29	**3.00**
Mean bal. acc.	75.268	77.420	78.440	77.886	79.967	69.122	80.790	81.517	81.511	**81.736**

ensembling for 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.