

# SUPPLEMENTARY MATERIAL FOR REAL-TIME AUTOML

**Anonymous authors**

Paper under double-blind review

## A ABLATION STUDIES

We performed different ablation studies changing a factor while keeping others equal: (i) we compared our approach using both the dataset description and data meta-features with using only one of these components; (ii) we compared several options of constructing the datasets graph: distances measured with Euclidean distance or cosine similarity, and threshold and k-nearest neighbors (k-NN) based edge assignments; (iii) we compared using zero, one, and two fusion networks; (iv) we compared our approach predicting only the estimator and an entire pipeline. We also compared our approach using each individual AutoML system separately while keeping all other factors equal.

### A.1 DESCRIPTION EMBEDDING VS. DATA META-FEATURES

In order to verify the effectiveness of our framework, we build the datasets graph and train our framework with both description embedding and data meta-features, with only description embedding, and with only data meta-features respectively. Figure 1 compares performance between training our framework with both description embedding and data meta-features and with only one of them. All of these results are obtained within 3 seconds using our zero-shot approach. We found that embedding the dataset description alone often produces results that are nearly as good as using both the dataset description and data meta-features.

Table 1 shows that using both data meta-features and description performs best or very close to the best result on all datasets. However, to recommend an initial pipeline in constant time, independent of the size of the dataset, we can use just the dataset descriptions, which performs surprisingly well, as shown in Table 1, opening the door to real-time AutoML with very large datasets.

### A.2 GRAPH CONSTRUCTION

We tried using several options to construct a graph: distances measured with Euclidean distance or cosine similarity, and threshold and k-nearest neighbors (k-NN) based edge assignments. Euclidean distance outperformed cosine similarity. We found thresholds created isolated nodes in graphs; kNN assignments performed more robustly. We experimented with various values of  $k$ . If  $k$  is too large, most nodes are connected and the neighborhood of a dataset is uninformative: in the extreme case the entire graph forms a clique, leading to inaccurate predictions. In contrast, if  $k$  is too small, we

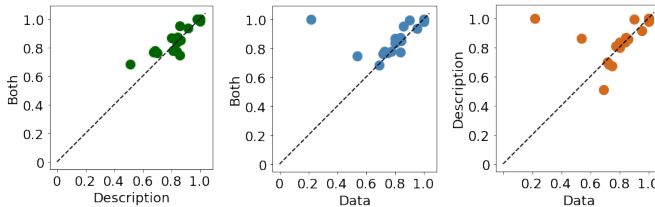


Figure 1: Comparison of accuracy of test set with machine learning algorithm generated using both description embedding and data meta-features, only description embedding, and only data meta-features respectively. Embedding the dataset description alone often produces results nearly as good as using both the description and the data. By embedding only the description, our system can select an AutoML pipeline in constant time, independent of the size of the dataset.

Datasets	Both	Data	Description
Lymph	<b>0.867</b>	0.800	0.800
Heart-C	0.774	<b>0.839</b>	<b>0.839</b>
Vehicle	0.776	0.776	<b>0.812</b>
Hayes-Roth	<b>0.769</b>	0.750	0.673
Colleges	<b>0.838</b>	0.803	<b>0.838</b>
KC1	<b>0.872</b>	0.839	0.844
Banana	0.745	0.538	<b>0.860</b>
Cardi	<b>1.000</b>	0.216	<b>1.000</b>
Cnae-9	0.935	<b>0.954</b>	0.917
Seeds	<b>0.952</b>	0.857	0.857
Wall-Robot	<b>1.000</b>	<b>1.00</b>	0.980
Cardi-Multi	<b>0.995</b>	0.901	<b>0.995</b>
BachChoral	<b>0.776</b>	0.727	0.684
Cjs	0.982	<b>1.000</b>	<b>1.000</b>
LED-Display	<b>0.760</b>	0.720	0.700
Wine-Quality	0.686	<b>0.692</b>	0.512
SpeedDating	0.851	0.846	<b>0.866</b>
Mofn	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>

Table 1: Comparison of performance between both description embedding and data meta-features and with only one of them.

lose information from potential surrounding nodes, which results in lower prediction accuracy. We found  $k = 20$  worked well.

### A.3 FUSION NETWORKS

In this work we combine three different types of data that live in different spaces: (i) an embedding of text describing the data; (ii) meta-features of the data; (iii) and an embedding of text describing the pipeline. A meaningful representation and similarity measure demands more than simply concatenating these feature vectors together: experimentally, we found that fusion networks produces superior results compared to concatenation.

### A.4 ESTIMATOR VS. PIPELINE

We compared our approach predicting only the estimator and an entire pipeline. Table 2 compares testing performance within 3 seconds between our zero-shot approach for predicting machine learning pipelines using only a single AutoML system and the 3 second baselines.

Dataset	Zero-Shot	Linear	Random Forest
Lymph	0.733	<b>0.800</b>	0.670
Heart-C	0.774	0.710	<b>0.806</b>
Vehicle	<b>0.765</b>	0.612	0.729
Hayes-Roth	0.654	0.404	<b>0.731</b>
Colleges	<b>0.803</b>	0.726	0.786
KC1	<b>0.853</b>	0.848	0.829
Banana	<b>0.892</b>	0.551	0.881
Cardi	<b>1.000</b>	0.432	<b>1.000</b>
Cnae-9	0.889	<b>0.954</b>	<b>0.954</b>
Seeds	<b>0.905</b>	<b>0.905</b>	<b>0.905</b>
Wall-Robot	0.991	0.907	<b>1.000</b>
Cardi-Multi	0.526	0.869	<b>0.986</b>
BachChoral	<b>0.792</b>	0.580	0.786
Cjs	<b>0.993</b>	0.846	0.971
LED-Display	0.720	<b>0.740</b>	0.680
Wine-Quality	<b>0.688</b>	0.451	0.678
SpeedDating	0.869	<b>0.870</b>	0.847
Mofn	<b>1.000</b>	<b>1.000</b>	<b>1.000</b>

Table 2: Comparison of testing performance within 3 seconds between our zero-shot approach for predicting machine learning pipelines using only a single AutoML system and the 3 second baselines.

## B DATASETS

Our dataset consists of 178 tabular OpenML datasets. OpenML datasets were filtered (using the OpenML API) by the number of instances in the datasets between 100 and 50,000 examples which

includes 1,807 datasets. These were filtered again by those that include descriptions (1,157 datasets), by those that are supervised classification (289 datasets), and by those that are tabular with descriptions longer than ten words while removing duplicates ( $n = 178$  datasets). Each dataset  $\mathcal{D}$  is associated with a (binary or multi-class) classification task and a dataset description  $\mathcal{M}(\mathcal{D})$ .

## C COMPUTATION TIME

Training time is two hours on Google Colab GPUs (NVIDIA Tesla K80). Total testing time is under 3 seconds and includes the time of embedding, meta-feature extraction, neural network prediction, hyper-parameter tuning, and running the predicted algorithm. Meta-features and runtime dominate. See the supplemental material for a break-down of the testing time on individual dataset. Embedding time is around 11 milliseconds independent of data size. Prediction time is around 5 milliseconds, also independent of dataset size. In contrast, data meta-feature extraction time depends linearly on dataset size. At run-time, the predicted algorithm performs a single fit to the data to compute performance, so runtime also depends on dataset size. For datasets of up to 50,000 instances, this processing time is less than 3 seconds. Table 3 gives the break-down of the running times for our approach.

Dataset	Total	Features	Predict	Run
Lymph	0.570s	0.068s	0.044s	0.348s
Heart-C	0.215s	0.052s	0.036s	0.017s
Vehicle	0.234s	0.055s	0.040s	0.028s
Hayes-Roth	0.410s	0.022s	0.037s	0.241s
Colleges	0.249s	0.083s	0.036s	0.020s
KC1	0.212s	0.042s	0.035s	0.025s
Banana	2.071s	0.022s	0.039s	1.899s
Cardi	2.835s	0.062s	0.040s	2.623s
Cnae-9	2.058s	1.870s	0.035s	0.042s
Seeds	0.495s	0.024s	0.036s	0.325s
Wall-Robot	0.205s	0.024s	0.039s	0.032s
Cardi-Multi	0.296s	0.057s	0.108s	0.021s
BachChoral	0.716s	0.245s	0.036s	0.325s
Cjs	0.534s	0.180s	0.175s	0.209s
LED-Display	0.210s	0.030s	0.038s	0.034s
Wine-Quality	0.284s	0.039s	0.040s	0.095s
SpeedDating	1.054s	0.728s	0.036s	0.179s
Mofn	0.347s	0.042s	0.036s	0.160s

Table 3: Test time breakdown of our zero-shot approach.

While these computation times may seem trivially small, consider the impact for real-time AutoML on large datasets. To process massive tables with millions of rows and columns in real-time, we can use sampling to compute meta-features efficiently.

### C.1 PERFORMANCE COMPARISONS

Figure 2 compares the accuracy on the test set between our zero-shot approach and a regularized linear model and random forest baselines, all given 3 seconds.

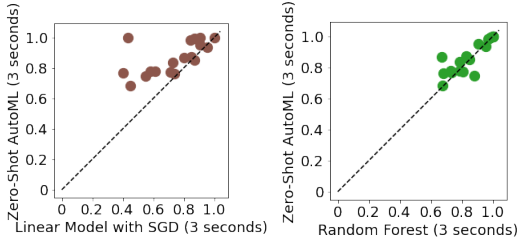


Figure 2: Comparison of accuracy on the test set between our zero-shot approach and a regularized linear model baseline (left) and random forest baseline (right), all given 3 seconds of computation. Our zero-shot approach systematically outperforms baselines.

Figures 3 and 4 compare the performance on each dataset and normalized cumulative performance between our zero-shot approach given 3 seconds and AutoML systems AutoSklearn, OBOE, TPOT, and AlphaD3M given 1 minute and random forest baseline given 1 minute of computation. Normalized cumulative performance is the sum of the performance of all previous datasets, ranked by performance, divided by the number of datasets. These other AutoML systems are unable to perform within the regime of 3 seconds.

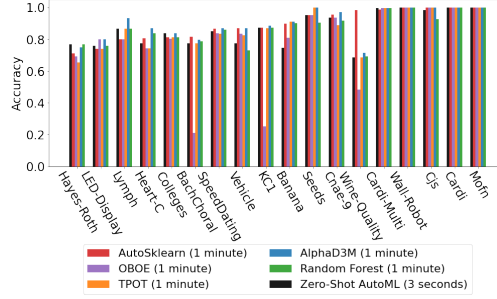


Figure 3: Comparison of performance on each dataset between our zero-shot AutoML given 3 seconds and AutoSklearn, OBOE, TPOT, AlphaD3M, and random forest given 1 minute.

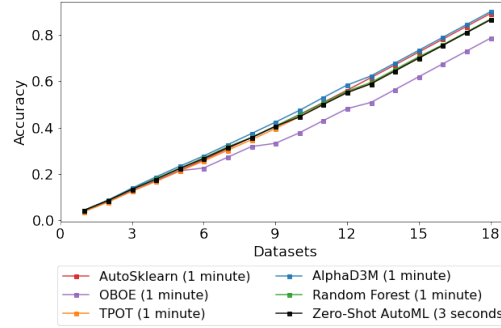


Figure 4: Comparison of normalized cumulative performance between our zero-shot AutoML given 3 seconds and AutoML systems AutoSklearn, OBOE, TPOT, and AlphaD3M given 1 minute and random forest baseline given 1 minute of computation.

Figure 5 and 6 compare performance on each dataset and normalized cumulative performance between our zero-shot approach given 3 seconds and a regularized linear model and random forest baselines given 3 seconds of computation.

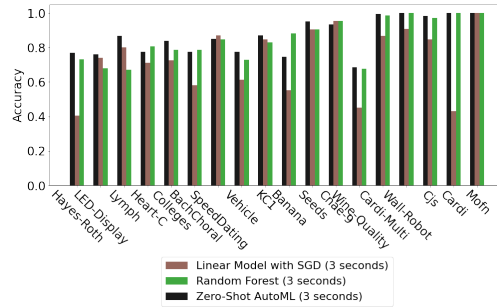


Figure 5: Comparison of performance on each dataset between our zero-shot approach given 3 seconds and a regularized linear model and random forest baselines given seconds of computation.

Figure 7 compares performance on each test dataset between our zero-shot AutoML given 3 seconds and AutoML systems AutoSklearn, OBOE, TPOT, and AlphaD3M given 1 minute and random

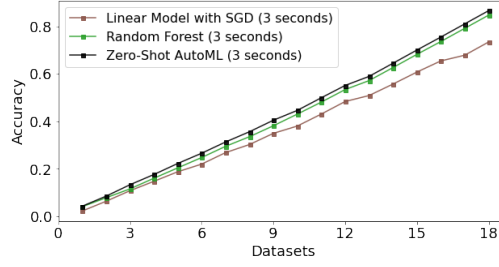


Figure 6: Comparison of normalized cumulative performance between our zero-shot approach given 3 seconds and a regularized linear model and random forest baselines given seconds of computation.

forest baseline given 1 minute of computation. Figure 7 shows that AutoSklearn performs better on the wine quality dataset than other systems. Figure 8 compares performance on each test dataset between our zero-shot AutoML given 3 seconds and regularized linear model and random forest baselines, given 3 seconds of computation.

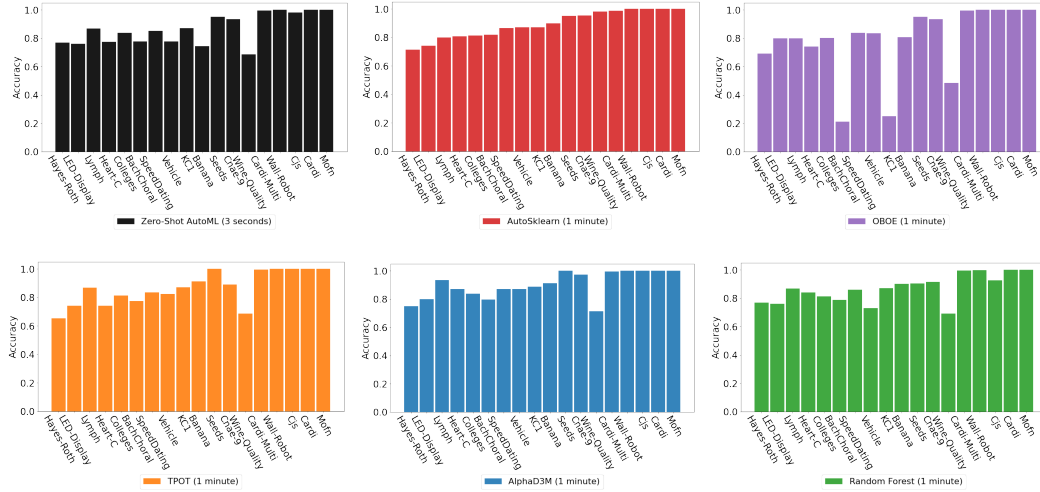


Figure 7: Comparison of performance on each test dataset between our zero-shot AutoML given 3 seconds and AutoML systems AutoSklearn, OBOE, TPOT, and AlphaD3M given 1 minute and random forest baseline given 1 minute of computation.

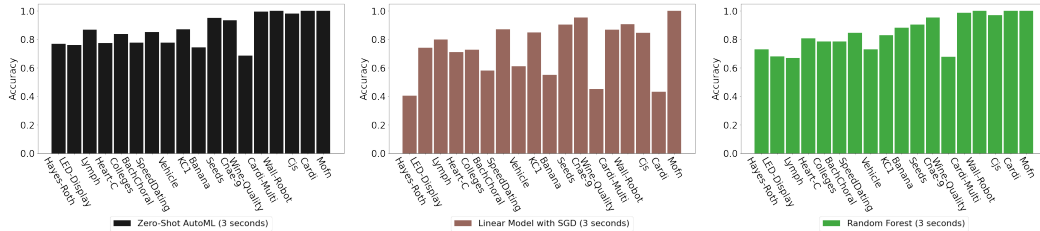


Figure 8: Comparison of performance on each test dataset between our zero-shot AutoML given 3 seconds and regularized linear model and random forest baselines, given 3 seconds of computation.

## D HYPER-PARAMETER OPTIMIZATION

During training, we use the hyper-parameters of the best pipelines among the AutoML systems. Initially, we store the value of the hyper-parameters for each pipeline primitive that worked best across

all training datasets. We then use these hyper-parameters as defaults when we select a particular primitive on the test set. Our system then recommends the pipeline together with these hyper-parameters. Notice the system is zero-shot: no models have been trained on the test dataset. After recommending a pipeline, we further tune its hyper-parameters using random search as implemented in scikit-learn, while time remains. Notice this step does fit the parameters of the model, which can be slow on large datasets; regardless, we terminate the search after 3 seconds have elapsed, even if no parameter fits have yet completed. With this short time budget, we found random search to be as effective as Bayesian optimization.

## E MACHINE LEARNING PIPELINE PREDICTION

We extend our framework from estimators to recommending pipelines. We compute the best pipelines predicted by any AutoML systems on each training dataset. To embed a pipeline, we concatenate the embedding of each type of primitive (eg, pre-processor, features selector, or estimator) in the pipeline. The remaining processes are the same as described in the pre-processing, training, and testing Algorithms. In our experiments, we select among 15 different types of pre-processors and feature selectors in addition to 16 different types of estimators. Perhaps most importantly, we share the GNN weights for predicting different pipeline primitives.

## F IMPLEMENTATION DETAILS

We use stochastic gradient descent with a constant learning rate of 0.0002 and momentum of 0.9 for training all neural networks. The fusion network  $f_\phi$  has 2 layers and 786,432 parameters. The fusion network  $g_\theta$  has a single layer and 786,432 parameters. The GNN  $h_{W,z}$  uses 3 GAT layers with input and output dimensions of (512, 512), (512, 256), and (256, 256), and a total of 462,848 parameters.

## G DESCRIPTIONS

Tables 7, 5, and 6 show examples of estimator, pre-processor, and feature selector descriptions embedded using our approach.

Dataset	Description
Bank Marketing	The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (or not) subscribed. The classification goal is to predict if the client will subscribe a term deposit (variable y).
Dresses Sales	This dataset contain attributes of dresses and their recommendations according to their sales. Sales are monitor on the basis of alternate days. The attributes present analyzed are: Recommendation, Style, Price, Rating, Size, Season, NeckLine, SleeveLength, waiseline, Material, FabricType, Decoration, Pattern, Type. In this dataset they are named Class(target) and then subsequently.
Amazon Reviews	Dataset are derived from the customers reviews in Amazon Commerce Website for authorship identification. Most previous studies conducted the identification experiments for two to ten authors. But in the online context, reviews to be identified usually have more potential authors, and normally classification algorithms are not adapted to large number of target classes. To examine the robustness of classification algorithms, we identified 50 of the most active users (represented by a unique ID and username) who frequently posted reviews in these newsgroups. The number of reviews we collected for each author is 30. Attribute Information: attribution includes authors' linguistic style such as usage of digit, punctuation, words and sentences' length and usage frequency of words and so on.

Table 4: Examples of dataset descriptions.

Pre-Processor	Description
Robust Scaler	This Scaler removes the median and scales the data according to the quantile range. The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).
PCA	Principal component analysis (PCA). Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.
Normalizer	Normalize samples individually to unit norm. Each sample (i.e. each row of the data matrix) with at least one non zero component is rescaled independently of other samples so that its norm (l1 or l2) equals one.

Table 5: Examples of pre-processor descriptions.

Selector	Description
Select Percentile	Select features according to a percentile of the highest scores.
Select from Model	Meta-transformer for selecting features based on importance weights.
RFE	Feature ranking with recursive feature elimination. Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features.

Table 6: Examples of feature selector descriptions.

Estimator	Description
Random Forest	A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).
Decision Tree	Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
Linear SVC	Linear Support Vector Classification is similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.
Gradient Boosting	Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
SVC	C-Support Vector Classification. The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples.
Extra Trees	This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
Adaboost	An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
KNN	Classifier implementing the k-nearest neighbors vote. KNN is a non-parametric lazy learning algorithm. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.
Gaussian NB	GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian.
SGD	This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch learning, see the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.
QDA	Quadratic Discriminant Analysis (QDA). A classifier with a quadratic decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.
LDA	A classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix. The fitted model can also be used to reduce the dimensionality of the input by projecting it to the most discriminative directions.
Logit	Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable.
Multinomial NB	The multinomial Naive Bayes classifier is suitable for classification with discrete features. The multinomial distribution normally requires integer feature counts.
Bernoulli NB	Naive Bayes classifier for multivariate Bernoulli models. This classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features.
Bagging	A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

Table 7: Examples of a subset of estimator descriptions.