Using Pipeline Performance Prediction to Accelerate AutoML Systems

Haoxiang Zhang
New York University
haoxiang.zhang@nyu.edu

Roque López
New York University
rlopez@nyu.edu

Aécio Santos
New York University
aecio.santos@nyu.edu

Jorge Piazentin Ono
New York University
jorgehpo@nyu.edu

Aline Bessa
New York University
aline.bessa@nyu.edu

Juliana Freire
New York University
juliana.freire@nyu.edu

ABSTRACT

Automatic machine learning (AutoML) systems aim to automate the synthesis of machine learning (ML) pipelines. An important challenge these systems face is how to efficiently search a large space of candidate pipelines. Several strategies have been proposed to navigate and prune the search space, from the use of grammars to deep learning models. However, regardless of the strategy used, a major overhead lies in the evaluation step: for each synthesized pipeline p, these systems must both train and test p to guide the search and to identify the best pipelines. Given a time budget and computing resources, the evaluation cost limits how much of the search space can be explored. As a result, these systems may miss good pipelines. We propose ML4ML, an approach that aims to reduce the evaluation overhead for AutoML systems. ML4ML leverages the provenance of prior pipeline runs to predict performance without having to re-train and test the pipelines. We present results of an experimental evaluation which demonstrates that not only can ML4ML build a reliable predictive model with low mean absolute error, but the integration of this model with AutoML systems leads to substantial speedups, enabling the systems to explore a larger number of pipelines and primitive combinations and derive pipelines at a much lower cost.

ACM Reference Format:
Haoxiang Zhang, Roque López, Aécio Santos, Jorge Piazentin Ono, Aline Bessa, and Juliana Freire. 2023. Using Pipeline Performance Prediction to Accelerate AutoML Systems. In *Data Management for End-to-End Machine Learning (DEEM '23), June 18, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3595360.3595856

1 INTRODUCTION

Given a prediction problem, a data scientist must assemble and end-to-end pipeline that executes multiple steps, from data cleaning and transformation to estimation. There are many options for each step. For example, there are several variations of imputers and scalers for data cleaning, encoders and dimensionality reduction approaches for data transformation, and a plethora of learning methods for estimation. These, together with various hyperparameter choices, lead to a very large search space a data scientist must navigate in a trial-and-error process to design an effective pipeline.

Automatic machine learning (AutoML) systems have been proposed as a means to both make data scientists more efficient and democratize ML by automating the construction of ML models [21]. While early approaches automated model selection, more recent systems automate the synthesis of end-to-end pipelines which, as illustrated in Figure 1, include steps for data cleaning, data transformation and estimation [11, 13, 29, 34]. We refer to these as *AutoML-EEP*. Given a problem (e.g., binary classification, regression) and a dataset, AutoML-EEP systems automatically synthesize pipelines that solve the problem. Figure 2 shows two pipelines generated for a binary classification problem that differ in the scaling methods and estimation models used.

To assemble pipelines, *AutoML-EEP* systems must explore different combinations of primitives. This is expensive and can take a substantial amount of time when the search involves a large primitive collection, and also for problems that use *large datasets* or *complex models* (e.g., deep learning). The cost incurred by *AutoML-EEP* systems consists of two key components: 1) navigating the search space and synthesizing pipelines; and 2) evaluating each derived pipeline. Approaches have been proposed that apply different strategies to control and prune the search space, such as genetic/evolutionary algorithms [33, 34], Bayesian optimization [13, 45, 46], matrix factorization/tensor decomposition [52, 53], reinforcement learning [10, 11] and many others [16, 27, 29].

However, regardless of the search strategy, a major overhead lies in the evaluation step: each derived pipeline must be trained and tested on the input data. The evaluation results are used to guide the search process and to help users select the most suitable pipeline. In practice, this overhead has limited a wider adoption of AutoML systems by requiring powerful machines with large memory or computing clusters. In fact, a recent study on the usability of AutoML systems reported a number of technical challenges users face, including running out of memory, failures for complex pipelines, and the compute-intensive nature of these systems – some users even mentioned having to revert to manually designing models [49]. Even for users that have access to powerful computing infrastructure, the increasing amounts of energy expended for artificial intelligence (AI) and its environmental implications have raised concerns, leading to a push towards "Green AI" and energy-efficient AI systems [30, 41, 44].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DEEM '23, June 18, 2023, Seattle, WA, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0204-4/23/06…$15.00
https://doi.org/10.1145/3595360.3595856

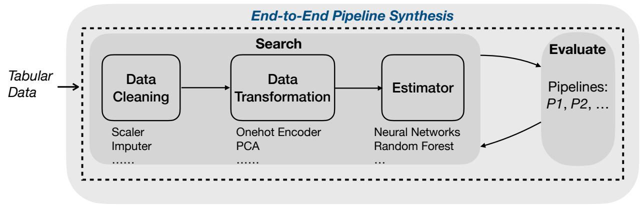


Figure 1: Synthesis of end-to-end ML pipelines. AutoML-EEP systems search over a large corpus of primitives for data cleaning, data transformation, and estimation and derive pipelines that combine these primitives in different ways. Each derived pipeline must be evaluated – trained and tested, which is often costly both in terms of time and use of computational resources.

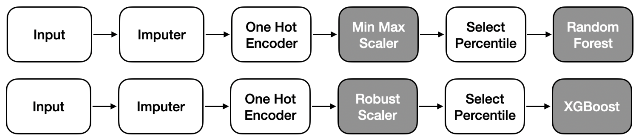


Figure 2: Pipelines synthesized by an AutoML-EEP system for a binary classification problem.

Pipeline Provenance as an Enabler for Efficient AutoML. Given the growing use of AutoML-EEP systems and the fact that, increasingly, people make their provenance available in public repositories [23, 31, 35], we asked the following question: *can we use provenance information and learn how to predict pipeline performance?* If this is possible, by eliminating the need to re-train and test each derived pipeline, the overall cost of pipeline synthesis can be reduced. This, in turn, brings important benefits: AutoML systems would require less computational resources (and thus consume less energy) and could be used by a wider audience. Moreover, by reducing the evaluation cost, AutoML systems can dedicate more resources to exploring the search space, opening the opportunity to derive pipelines that might otherwise be overlooked given a budget.

Learning to Predict the Performance of ML Pipelines. We explore the problem of learning to predict the performance of end-to-end ML pipelines using provenance of prior pipeline runs as the training data. This includes prospective provenance – the pipeline definition – and retrospective provenance – information about the pipeline execution, input data, and its results [37]. For example, the D3M MetaLearning Database [31] maintains information about pipeline runs, which include information about the training and test data, the pipeline definition, and its performance on the test data. In addition, the AutoML systems themselves can be used to automatically derive provenance as they generate and test pipelines.

While pipeline provenance data is abundant, an important open question is how to define the learning task. The performance of a pipeline is influenced both by the actual pipeline (and its primitives) and the input data. Therefore, these are key inputs for the learning task. Another important aspect to consider is the structure of the pipeline supported by an AutoML-EEP system. Some systems use a linear sequence of primitives, while others make use of directed acyclic graphs (DAGs) to represent pipelines. Thus, to design a general method to predict pipeline performance, we need to take these different structures into account and devise appropriate encoding strategies. Furthermore, the performance of a given pipeline can vary for different datasets. Thus, an important challenge is to identify features of datasets that capture the interaction between pipeline and data and that, at the same time, are good predictors for pipeline performance.

Last, but not least, pipeline provenance repositories often contain information for many different classes of learning problems (e.g., binary and multi-class classification, time-series classification) and cover a plethora of datasets that differ in size, structure, and contents. This *heterogeneity* can introduce noise in the learning process and negatively impact prediction accuracy. Therefore, we need to devise mechanisms to identify relevant entries to be used in the training process.

Our Approach: ML4ML. We propose ML4ML, a system that leverages provenance and **M**eta**L**earning to accelerate Auto**ML** systems. This acceleration is achieved in part by a pipeline performance prediction model (P3M), which can be used to replace the costly evaluation step during the AutoML synthesis process (see Figure 1). To construct P3M, ML4ML utilizes provenance information stored in a Pipeline Provenance Store (PPS) that contains triples consisting of a dataset D, a pipeline P, and the performance of P when applied on D. At prediction time, given a unseen pipeline and dataset, P3M outputs an estimate for the pipeline performance.

To support multiple AutoML systems and the different pipeline structures they use, ML4ML considers multiple pipeline and dataset encoding methods to systematically select the best combination of encoding methods for a given system. In addition, to properly handle the heterogeneous nature of pipeline provenance, we propose a strategy to select from the provenance store a set of entries to train the prediction model that uses the notion of dataset difficulty number (DDN) [40]. The intuition behind the DDN is that a pipeline should have similar performance when tested on datasets with similar DDN. Thus, by using the DDN as the criteria to select a subset of provenance entries, we can build a robust pipeline performance prediction model.

Contributions. To the best of our knowledge, this paper proposes the first approach to accelerate AutoML-EEP systems by replacing evaluation with prediction in the pipeline synthesis process. Our main contributions can be summarized as follows:

- We introduce an approach that takes both data and pipeline structure into account to construct a reliable pipeline performance prediction model (P3M);
- We study the effectiveness of different data and pipeline encoding methods, and propose a systematic approach for selecting the best combination of methods for a given scenario;
- We introduce a strategy to derive a dataset difficulty number (DDN) for tabular datasets that is used to select appropriate training entries used to construct the P3M and experimentally show its effectiveness;
- We perform an extensive experimental evaluation using multiple datasets and different AutoML systems. Our results demonstrate that ML4ML derives a reliable prediction model for tabular datasets on classification tasks that has low mean absolute error. Furthermore, when combined with AutoML systems, ML4ML leads to substantial speedups.

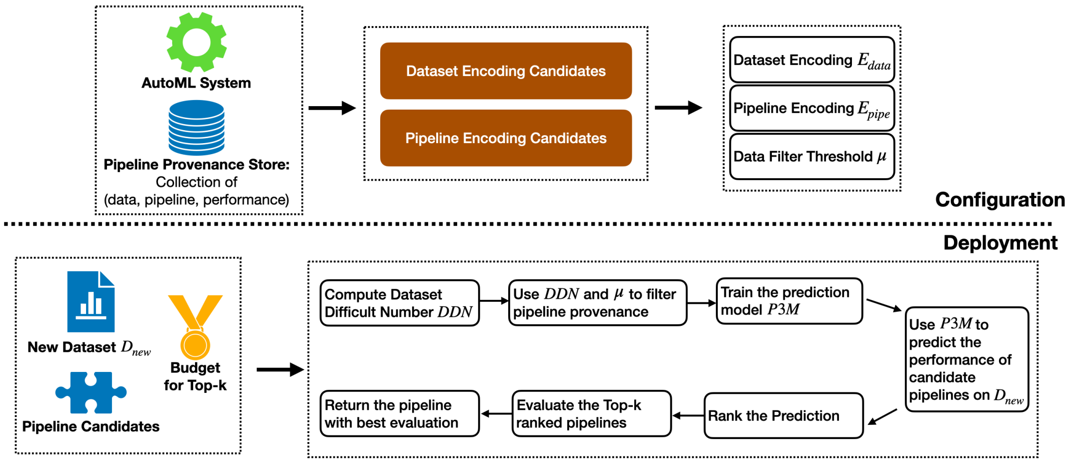

AutoML System
Pipeline Provenance Store:
Collection of
(data, pipeline, performance)
Dataset Encoding Candidates
Pipeline Encoding Candidates
Dataset Encoding E_{data}
Pipeline Encoding E_{pipe}
Data Filter Threshold μ
Configuration
Deployment
New Dataset D_{new}
Budget for Top-k
Pipeline Candidates
Compute Dataset Difficult Number DDN
Use DDN and μ to filter pipeline provenance
Train the prediction model $P3M$
Use $P3M$ to predict the performance of candidate pipelines on D_{new}
Rank the Prediction
Evaluate the Top-k ranked pipelines
Return the pipeline with best evaluation

Figure 3: ML4ML works in two steps: configuration and deployment. To automatically select an effective configuration for a given AutoML system, ML4ML takes as input the pipeline provenance, and tests different combinations of dataset and pipeline encodings as well as identify a suitable DDN threshold. The configuration step is executed only once and in an offline fashion. When the system is deployed, given a new dataset, pipeline candidates, and the desired number of pipelines (k), ML4ML uses P3M to rank and evaluate the performance of candidate pipelines. The pipeline with the best evaluation is returned.

2 ML4ML

Background and Definitions. Before presenting our approach, we first introduce concepts and background information needed to formally define the problem we address in this paper.

Definition 1. ***Primitive***. *A primitive $\underline{m}$ is an algorithm that is used as a computational step in a machine learning (ML) pipeline.*

Definition 2. ***Pipeline***. *A pipeline $\underline{p}$ is directed acyclic graph (DAG) that combines a set of primitives.*

Figure 2 shows pipelines that orchestrate primitives for imputing values (Imputer), encoding (One Hot Encoder), and estimation (Random Forest, XGBoost).

Definition 3. ***Pipeline Provenance Store (PPS)***. *As an AutoML-EEP derives and tests pipelines, it stores their provenance in Pipeline Provenance Store DB_{pps}, which consists of a set of tuples*

$$DB_{pps} = \{(d_1, p_1, s_1), (d_2, p_2, s_2), ..., (d_n, p_n, s_n)\}$$

where n is the number of triples (d_i, p_i, s_i) that consist of a dataset d_i, a pipeline p_i, and a pipeline performance score s_i obtained by calculating an evaluation metric using the output obtained after running the pipeline p_i on dataset d_i.

Note that in this paper we use term *provenance* to refer to logs recorded during previous executions of an AutoML system. These logs contain information that can be used as training data to construct a pipeline performance prediction model (P3M). In order to train this model, we must define its inputs and outputs. More specifically, from each provenance tuple (d_i, p_i, s_i), we use d_i and p_i to derive the model inputs that are used to predict the performance s_i. Next, we formally define the pipeline performance prediction problem.

Definition 4. ***Dataset Encoding***. *A dataset encoding $\underline{E_{data}(D)}$ is a feature vector generated by encoding method E_{data} that represents characteristics of a given dataset D.*

Definition 5. ***Pipeline Encoding***. *A pipeline encoding $\underline{E_{pipe}(p)}$ is a feature vector generated by encoding method E_{pipe} that represents characteristics of a given pipeline p.*

Definition 6. ***Pipeline Performance Prediction***. *Given an AutoML system and a pipeline provenance store DB_{pps}, we aim to build a reliable pipeline performance prediction model Φ that takes as input the dataset and pipeline encodings $[E^*_{data}, E^*_{pipe}]$ and outputs an accurate pipeline performance estimate $\hat{s} = \Phi([E^*_{data}, E^*_{pipe}])$.*

2.1 Solution Overview

As illustrated in Figure 3, ML4ML operates in two stages: *configuration* and *deployment*.

Configuring ML4ML. ML4ML was designed to work with different AutoML systems. To properly support multiple systems, we include a configuration stage that determines the appropriate parameters for the selected AutoML system. Note that this configuration step is only executed *once* and offline.

To accelerate AutoML-EEP systems, ML4ML leverages the *Pipeline Performance Prediction Model (P3M)*: instead of retraining and evaluating a model, systems can use P3M to predict the performance of the synthesized pipelines.

To generate these predictions, P3M relies on information about the dataset and the pipeline being used. Different encoding methods can be applied to represent this information. The effectiveness of encoding methods varies depending on the design of a specific AutoML system and the pipeline provenance records available. Consequently, to ensure that the most effective encoding methods are selected, ML4ML tests various combinations of dataset and pipeline

Algorithm 1 ML4ML Configuration

```
1:  Input: AutoML system $S$, Pipeline Provenance Store database $DB_{pps}$
2:  Default Parameters: combination of dataset and pipeline encoding methods $\{(E_{data}, E_{pipe})_i\}_{i=1}^{4}$
3:  $DDN_{interval} = DB_{pps}.max\_ddn - DB_{pps}.min\_ddn$
4:  $\{\mu_j\}_{j=1}^{m} = \{0.05, 0.1, ..., DDN_{interval}\}$
5:  Initialize 2D array $Err$ with 0s
6:  $D_{pps} = DB_{pps}.tabular\_datasets$
7:  for $i = 1$ to 4 do
8:      for $j = 1$ to $m$ do
9:          for $k = 1$ to $|D_{pps}|$ do
10:             Initialize $err = 0$
11:             $exp\_test = DB_k$
12:             $exp\_train$ = SelectSimilarExperiments($D_k$, $DB$, $\mu_j$)
13:             $err$ += test_P3M($exp\_train$, $exp\_test$, $(E_{data}, E_{pipe})_i$)
14:         end for
15:         $Err_{i,j} = \frac{err}{k}$
16:     end for
17: end for
18: $i^*, j^*$ = argmax $Err$
19: return $(E_{data}, E_{pipe})_{i^*}$ and $\mu_{j^*}$
```

encoding candidates during the configuration phase. This is outlined in Algorithm 1 and illustrated in the top part of Figure 3. This process takes the pipeline provenance of an AutoML system as input and produces a dataset encoding (E_{data}), a pipeline encoding (E_{pipe}), and a data filter threshold (μ) as output. The threshold μ is a crucial component of the configuration process, as it determines which pipeline provenance records are selected for use in the prediction model. The selection of an appropriate threshold value is discussed in detail in Section 2.3. In Section 2.4, we present various dataset and pipeline encoding methods that we considered for ML4ML.

Deployment: Deriving Predictions. The performance of a pipeline can vary significantly depending on the characteristics of the dataset it is applied to. Therefore, when constructing the Pipeline Performance Prediction Model (P3M) for a given dataset D_{new}, it is crucial to select pipeline provenance entries that include datasets D_i that are *similar* to D_{new}, in the sense that a given pipeline will have similar performance for D_{new} and D_i. To capture this notion of similarity, ML4ML computes a Dataset Difficulty Number (DDN) for the datasets, and uses the DDN to filter the pipeline provenance prior to training the prediction model P3M. The details of this process are discussed in subsequent sections. Before delving into the specifics, we provide an overview of the workflow associated with the *deployment* phase.

As demonstrated in the lower part of Figure 3 and outlined in Algorithm 2, the *deployment* phase takes as input a new dataset, pipeline candidates, and the desired number (k) of pipelines. First, a subset of the tuples is selected from the provenance store DB_{pps} to serve training data for building the P3M model (Algorithm 2, line 2). To do so, we first compute the dataset difficulty number for the input dataset D_{new}. Then, we use $DDN_{D_{new}}$ to find tuples that have a *similar difficulty level* in the provenance store DB_{pps}

Algorithm 2 ML4ML Deployment

```
1: Input: Pipeline Provenance Store database $DB_{pps}$, dataset encoding method $E_{data}$, pipeline encoding method $E_{pipe}$, filter threshold $\mu$, unseen dataset $D_{new}$, pipeline candidates $P$, budget $k$
2: $exp\_sim$ = SelectSimilarExperiments($D_{new}$,$DB_{pps}$,$\mu$)
3: $P3M$ = Train($exp\_sim$, $E_{data}$, $E_{pipe}$)
4: $P_{pred} = P3M.predict(E_{data}, E_{pipe}, D_{new}, P)$
5: $P_{rank}$ = Rank($P_{pred}$)
6: for $i$ in range($k$) do
7:     Evaluate($P_{rank-i}$)
8: end for
9: return $best\_pipeline$
```

(i.e., tuples that have dataset difficulty difference smaller than the threshold μ). Once we select the training data, the next steps are: 1) build the P3M model; 2) use it to predict the performance of candidate pipelines on D_{new}; and 3) rank the pipelines according to their predicted performance (Algorithm 2, line 3-5). Finally, the top-k predicted pipelines are evaluated and the pipeline with the best evaluation is then returned as the output (Algorithm 2, lines 6-9).

In the following sections, we define the dataset difficulty number and how it is used (Section 2.2), describe in detail the process of selecting entries from the pipeline provenance store (Section 2.3), introduce different encoding methods (Section 2.4), and present the P3M model (Section 2.5).

2.2 Dataset Difficulty Number (DDN)

The Dataset Difficulty Number (DDN) plays a critical role in he selection of entries from the PPS to train the prediction model P3M. Intuitively, the DDN provides a score that approximates the performance of the best model on a dataset (e.g., classification accuracy in the context of this paper), and it should be computed much faster than actually applying the model to the data. Consider for example, an *easy* dataset for which the best model, such as a Random Forest or a 10-layer multi-layer perceptron (MLP), takes 10 minutes to run and achieves a classification accuracy of 96%. The DDN for this dataset is 0.93 (corresponding to the classification accuracy of 93%) and can be computed using a 3-layer MLP in just 1 minute. For a more *difficult* dataset, where the best model takes 30 minutes and achieves a classification accuracy of 84%, the DDN value of 0.8 can be computed in 1 minute. In summary, the DDN value effectively represents the difficulty level of a given dataset and it is easy to compute.

Another benefit of using the DDN is efficiency: it provides a concise representation (a one-dimensional scalar) that significantly enhances the computational efficiency of the pipeline provenance filtering process (Section 2.3). For instance, comparing the distances between various scalars (achieved by subtracting two numbers) is considerably cheaper than comparing the distances between high-dimensional vectors (such as using the L^2 norm distance). By simplifying the representation and reducing it to a single scalar value, the DDN allows for quicker and more manageable comparisons,

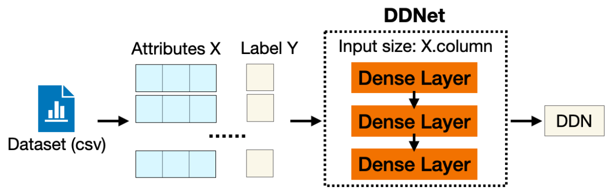


Figure 4: Computing the Dataset Difficulty Number (DDN). The configuration phase supports the search of different parameters for a dataset difficulty networks (DDNet). The default setting is a 3-layer MLP. Given a tabular dataset, numerical, categorical and target columns are handled with different methods. Then DDNet takes the processed data and computes the Dataset Difficulty Number. The input size to the DDNet is the same as the number of attribute columns, or X.column in this figure.

Algorithm 3 Compute Dataset Difficulty Number (DDN)

```
1: **Input:** Collection of datasets $D = \{D_1, D_2, ..., D_n\}$, MLP parameter settings $ParaSet$
2: **Output:** Collection of DDNs $A = \{A_1, A_2, ..., A_n\}$
3: for $i = 1$ to $n$ do
4:     $(X_i, Y_i) \leftarrow$ process($D_i$)
5:     initialize a DDNet $M$ with $ParaSet$
6:     $A_i \leftarrow$ train_and_evaluate($M$, $X_i$, $Y_i$)
7: end for
8: return $A$
```

ultimately resulting in more efficient filtering and processing of experimental data in pipeline provenance store.

Computing Dataset Difficulty Number. To encode a dataset D, we compute a scalar $DDN \in \mathbb{R}^1$ using a multi-layer perceptron (MLP) [19]. The calculated difficulty score is a scalar number within the range [0,1], which represents an approximation to the best accuracy score of a given dataset.

We selected MLP to compute DDN due to its strong approximation capability, as per the Universal Approximation Theorem [20], and its ability to perform automatic feature engineering [18]. These properties make MLPs an efficient and robust choice for calculating dataset difficulty.

As shown in Figure 4, the configuration phase of our system supports the search of different hyperparameters for the Dataset Difficulty Network (DDNet), including the number of layers, hidden units, and activation functions. By allowing for the tuning of hyperparameters, the configuration phase helps to optimize the performance of DDNet for a given dataset. The default implementation of DDNet is as follows: a three-layer Multi-Layer Perceptron (MLP) with 256 hidden units in each layer, cross entropy as the loss function, and a dropout rate of 0.5. The Adam optimizer [25] is used to minimize the loss during training. Keras library [7] is the implementation library for MLP.

The input to DDNet is data processed from a CSV file (i.e., a tabular dataset). The processed data contains feature attributes X and target variables Y. Numerical attributes are preserved, while categorical attributes are encoded using an ordinal encoder. The target column Y is also transformed using an ordinal encoder if it is categorical. As outlined in Algorithm 3, after processing an input dataset, a DDNet model (MLP) is initialized with same parameters to compute the difficulty number DDN.

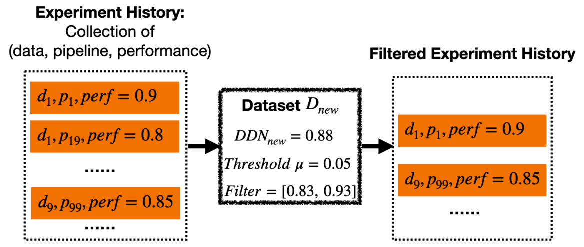


Figure 5: Filtering pipeline provenance. For the threshold $\mu = 0.05$, given a new dataset with $DDN_{new} = 0.88$, provenance entries will be selected if they include datasets with DDN between 0.83 ($DDN_{new} - \mu$) and 0.93 ($DDN_{new} + \mu$).

2.3 Provenance Selection for Training the P3M

Provenance stores (PPS) are heterogeneous, encompassing pipelines that use datasets with different levels of difficulty. Thus, to construct a reliable prediction model for a new dataset D_{new}, we evaluate the difficulty level (DDN) of D_{new} and select previous experiments from the provenance store that include datasets with difficulty levels comparable to that of D_{new}.

Filtering Pipeline Provenance. Algorithm 4 outlines the procedure for selecting tuples from the provenance store DB_{pps}. Given datasets D_1 and D_2, we first compute the DDN difference $\delta = |\text{DDN}(D_1) - \text{DDN}(D_2)|$. Next, we use a threshold μ to filter out entries that have $\delta > \mu$. In other words, the parameter μ represents the maximum acceptable difference between the DDNs of two datasets.

Algorithm 4 SelectSimilarExperiments

```
1: **Input:** target dataset $D_t$, Pipeline Provenance Store database $DB_{pps}$, threshold value $\mu$
2: Initialize $exp\_sim = \emptyset$
3: $D_{pps} = DB_{pps}.tabular\_datasets$
4: for $D_i$ in $D_{pps}$ do
5:     if $D_i \neq D_t$ then
6:         if $|DDN(D_i) - DDN(D_t)| \leq \mu$ then
7:             $exp\_sim = exp\_sim \cup DB_i$
8:         end if
9:     end if
10: end for
11: return $exp\_sim$
```

For instance, as illustrated in Figure 5, if we set the threshold $\mu = 0.05$ and have a new dataset with dataset difficulty number $DDN_{new} = 0.88$, the filter will create a range from 0.83 ($DDN_{new} - \mu$) to 0.93 ($DDN_{new} + \mu$). Consequently, only the entries with performance between 0.83 and 0.93 will be selected, ensuring that the chosen provenance entries are relevant to the input dataset.

Experimentally Selecting a Suitable μ Threshold Value. As discussed earlier, we select all entries with DDN difference smaller than μ. This brings a new question: what threshold value should we use? ML4ML uses a data-driven approach to automatically derive this value during the configuration stage. To do so, it evaluates different threshold values as described in Algorithm 1. Specifically, the pipeline provenance store is first used to calculate the interval $DDN_{interval}$ between the maximum DDN and the minimum DDN (Algorithm 1, line 3), and this $DDN_{interval}$ is utilized to generate threshold value candidates (Algorithm 1, line 4). Next, different threshold values are tested (Algorithm 1, line 7-17) and, finally, the best threshold value is selected to be used in the deployment stage (Algorithm 1, line 18-19). We discuss the effectiveness of the threshold selection process in Section 3.2.

2.4 Encoding Data and Pipelines

To make a prediction, the prediction model P3M considers both the encoding of the input dataset and the encoding of the pipeline generated by the AutoML pipeline search algorithm. Below, we describe our choices for dataset and pipeline encoding methods.

2.4.1 ***Dataset Encoding: Multi-Dimensional Meta-Features (MF).*** Besides the *Uni-Dimensional* dataset encoding method using dataset difficulty number, ML4ML also includes a *Multi-Dimensional* method. One intuitive way to encode a given dataset is to compute its meta-features, which serve as a representation of dataset properties [1]. The properties used to characterize the dataset include general information such as the number of rows and columns, as well as specific features derived from the data. We use a set of meta-features extracted from each dataset using the MFE framework [5]. These meta-features capture statistical characteristics like mean, variance, and skewness for numerical attributes, as well as information-theoretic measures such as entropy and mutual information for categorical attributes. Furthermore, they include features like the count of missing values and outliers, and structural features like the extent of correlation among attributes and the existence of clusters or subgroups in the data. The resulting dataset encoding is a 231-dimensional vector $E_{data}(D) \in \mathbb{R}^{231}$.

2.4.2 ***Dataset Encoding: Uni-Dimensional Dataset Difficulty Number (DDN).*** We also consider the DDN as the dataset encoding method. To encode a dataset D with DDN, we compute a one-dimensional vector $E_{data}(D) \in \mathbb{R}^1$ using a Multi-Layer Perceptron (MLP) as described in Section 2.2 and Figure 4.

2.4.3 ***Pipeline Encoding: Structure-Agnostic OneHot Encoder.*** One straightforward and *structure-agnostic* method to represent ML pipeline information is using OneHot Encoder. For a given AutoML system, the dimension of the one-hot encoding vector for pipeline p is $E_{pipe}(p) \in \mathbb{R}^{|M|}$, where $|M|$ is the cardinality of the set M of unique primitives used by the system: $M = \{m_1, m_2, ..., m_{|M|}\}$. For example, given a pipeline p consisting of primitives

$$\{m_2, m_4, ..., m_{|M|-1}\}$$

the one hot encoding for p is $E_{pipe}(p) = [0, 1, 0, 1, ..., 1, 0]$. In other words, the encoding vector has value 1 on the positions where corresponding primitive exists in the given pipeline and has value 0 otherwise.

2.4.4 ***Pipeline Encoding: Structure-Aware Graph Neural Network (GNN).*** Different AutoML systems use different types of directed acyclic graphs (DAG) to represent pipelines. For example, Auto-Sklearn [12] derives simple linear pipelines while AlphaD3M [10] is able to generate graph-structure pipelines. To account for these complex structures, ML4ML also uses graph neural networks (GNNs) as an encoding method for pipeline. GNNs uses deep learning to transform a graph structure into embeddings for various tasks, from node classification, link prediction to graph classification [28]. We choose the powerful Graph Isomorphism Network (GIN) [50] based GNN layers to encode our pipelines. In the case of GNN, each node represents a primitive, and two nodes are connected by an edge only if the corresponding primitives coexist and are connected within the same pipeline. The optimizer is Adam. The PyTorch Geometric library [14] is used to implement the graph neural network.

2.4.5 ***Selecting the Best Encoding Combination.*** As discussed above, it is not clear what combination of encodings leads to the best prediction performance for a given AutoML system and PPS. Before deployment, ML4ML determines which combination is optimal for an given AutoML system and pipeline provenance (Figure 3 upper part and Algorithm 1). There are four combinations of encoding methods: (1) DDN for dataset encoding and OneHot for pipeline encoding; (2) DDN for dataset and GNN for pipeline; (3) MetaFeature for dataset and Onehot for pipeline; (4) MetaFeature for dataset and GNN for pipeline. Given an AutoML system and its pipeline provenance store, ML4ML first examines these combinations of encoding methods (specifically Algorithm 1, line 7-17). Then the combination with the least mean absolute error on pipeline performance prediciton is selected (Algorithm 1, line 18-19) for deployment stage. The experiment results of this configuration phase is presented in Section 3.2.

2.5 P3M Model

Just like the MLP used for computing the dataset difficulty number (DDN), the P3M prediction model also employs an MLP regressor architecture to tackle the pipeline performance prediction problem. The MLP regressor for P3M is a powerful and flexible neural network model that consists of an input layer, three hidden layers with 256 neurons of each layer, and an output layer. The rectified linear unit (ReLU) activation function and the Adam optimizer are used. The learning rate is 0.001 and the number of epochs for training is 200. Through automatic feature engineering and non-linear activation functions, the MLP in P3M is able to learn and generalize from the given input dataset and pipeline encodings to make reliable performance predictions.

As illustrated in Figure 6, the P3M input is a concatenation of the given dataset and pipeline encodings. The output is the predicted performance of the specified pipeline on the input dataset. The filtered pipeline provenance, a collection of (dataset, pipeline, pipeline performance) tuples, is utilized for training P3M. During training, the dataset and pipeline encodings serve as the training data (X_{train}), while the pipeline performance acts as the training label (y_{label}). In the prediction phase, the dataset encoding and pipeline encoding constitute the testing data (X_{test}), and the corresponding performance is predicted (y_{pred}).

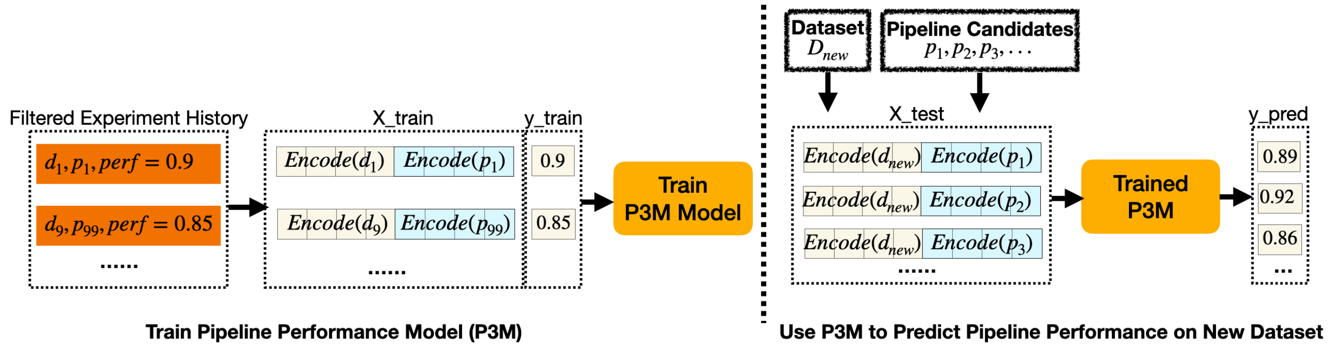

Filtered Experiment History
$d_1, p_1, perf = 0.9$
$d_9, p_{99}, perf = 0.85$
X_train
$Encode(d_1)$ $Encode(p_1)$
$Encode(d_9)$ $Encode(p_{99})$
y_train
0.9
0.85
Train P3M Model
Train Pipeline Performance Model (P3M)
Dataset D_{new}
Pipeline Candidates $p_1, p_2, p_3, \dots$
X_test
$Encode(d_{new})$ $Encode(p_1)$
$Encode(d_{new})$ $Encode(p_2)$
$Encode(d_{new})$ $Encode(p_3)$
Trained P3M
y_pred
0.89
0.92
0.86
Use P3M to Predict Pipeline Performance on New Dataset

Figure 6: Training P3M and using it to predict pipeline performance for a new dataset.

3 EXPERIMENTAL EVALUATION

3.1 Experiment Setup

Our approach is designed to expedite AutoML-EEP systems for novel, previously unseen datasets. Since this acceleration occurs during the *Deployment* stage (refer to Figure 3), we evaluate P3M's *Deployment* performance. Additionally, we assess the effectiveness of the *Configuration* stage, which determines the optimal encoding methods for constructing P3M.

Data. We conduct experiments on 33 ***tabular*** datasets from the AutoML Benchmark on ***classification*** task, encompassing datasets of varying sizes, domains, and difficulty levels [15]. To build PPS, we ran the AutoML system for each dataset until all pipelines in the search space were derived and evaluated. When a dataset is selected as the new dataset in the *deployment* phase experiment, the (data, pipeline, pipeline performance) history of that dataset is excluded from the pipeline provenance store. The history of the other 38 datasets is considered for selecting pipeline running history to train P3M (see Figure 5). In this way, both the new dataset and its pipeline performance remain completely unseen.

AutoML Systems. For our experimental evaluation, we tested ML4ML with two open-source AutoML systems: ***AlphaD3M*** [2] and ***TPOT*** [34]. These systems use different search strategies: AlphadD3M uses *deep reinforcement learning* to search and derive end-to-end pipelines; while TPOT optimizes machine learning pipelines using *genetic programming*. TPOT was one of the first AutoML systems developed and it is widely used in the data science community. ML4ML requires the underlying AutoML system to record pipeline provenance. AlphaD3M automatically outputs pipeline provenance when searching pipelines, and we modified TPOT to record this information. For both systems, the input consists of a subsampled version of a given tabular dataset (with a maximum of 5,000 rows), and the default optimization settings are employed for pipeline search.

3.2 Configuration Phase: Evaluating Encoding Strategies

Given an AutoML system and its pipeline provenance store, ML4ML initially performs a configuration experiment to identify the optimal combination of dataset and pipeline encoding methods, as well as the threshold value for filtering pipeline provenance according to a target dataset (Section 2.4). This search examines all possible combinations exhaustively. Note that this process is only executed once (and offline) for a given AutoML system and PPS. Thus, the execution of the configuration stage does not impact the user experience.

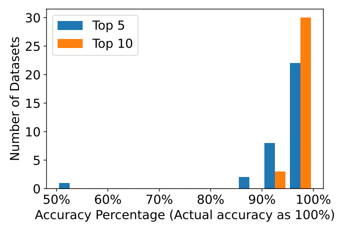
We examined the following combinations: (1) **DDN,OneHot**, (2) **DDN,GNN**, (3) **MF,OneHot** and (4) **MF,GNN**, where the first is dataset encoding and the second is pipeline encoding. For example, the combination **DDN,OneHot** has dataset difficulty number as the dataset encoding, and onehot encoder as the pipeline encoding method. The range of threshold values is determined by the maximum and minimum DDNs in the pipeline provenance store, as described in Section 2.3 and Algorithm 1. To encode a dataset D with meta-features (MF), we computed a vector $E_{data}(D) \in \mathbb{R}^{231}$ using an open-source meta-feature extractor [5].

Table 1 presents the results for *AlphaD3M* and shows the mean absolute error of the predicted pipeline performance (averaged on all datasets) for different encoding combinations and different threshold μ values. The combinations using *MF* as dataset encoding are consistently worse for all thresholds, while the combination *DDN* with *OneHot* is uniformly better for all values. This suggests that although the meta-features include a larger number of features it is not as informative as the dataset difficulty number – this is an issue we plan to explore in future work. Among all configuration, *DDN* is the best method to represent a dataset, *OneHot* is the best method to represent a pipeline, and $\mu = 0.1$ is the best threshold to filter pipeline provenance.

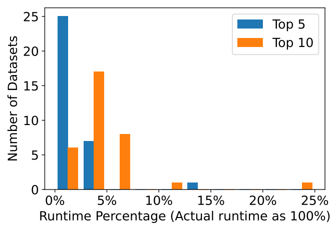
We were initially surprised to find that OneHot encoding works best for pipeline representation. Since OneHot encoding is structure-agnostic, intuitively, it should not be able to capture pipeline connectivity, which could impact the accuracy of the P3M prediction model. A close examination of the actual pipelines derived by the AutoML systems showed that they have a simple (often linear) structure and they are also short – consisting of few primitives. This observation confirms the findings in [38] that most explicit pipelines (those defined using a single library, such as SCIKIT-LEARN or AlphaD3M) have a length of 2 to 5 primitives. This suggests that the information represented by individual primitives is more significant than the information conveyed by their connectivity. Consequently,

Table 1: Configuration Phase: Mean Absolute Error of Pipeline Performance Prediction for Different Encoding Combinations and Different Thresholds. Cells with darker blue color show better performance (i.e., lower error). Among all configurations, the combination *DDN,OneHot* with threshold $\mu = 0.1$ has the best performance.

Threshold μ	*0.05*	*0.1*	*0.15*	*0.2*	*0.25*	*0.3*	*0.35*	*0.4*	*0.45*
DDN,OneHot	0.113	**<u>0.089</u>**	0.110	0.111	0.119	0.122	0.120	0.118	0.119
DDN,GNN	0.118	0.094	0.100	0.119	0.144	0.158	0.169	0.176	0.174
MF,OneHot	0.354	0.368	0.347	0.337	0.374	0.356	0.296	0.273	0.255
MF,GNN	0.194	0.169	0.150	0.162	0.188	0.187	0.168	0.175	0.189



(a) Relative Accuracy



(b) Relative Runtime

Figure 7: The histogram summarizes accuracy and runtime for *Top5* and *Top10* budgets in comparison to the *Actual* accuracy. The results show that the majority of *Top5* datasets achieve an accuracy within 90% of the actual accuracy, while taking less than 5% of the actual runtime. For more generous budgets, most *Top10* datasets achieve an accuracy within 95% of the actual accuracy and require less than 10% of the runtime. Figure 8 provides a detailed view of the performance metrics for each dataset.

the concise OneHot encoding outperforms graph encoding in this context.

Note that we have also run this experiment using *TPOT* and observed similar trends. The optimal encoding combination and the dataset filter threshold are used at the deployment stage experiment.

3.3 Deployment Phase: P3M Quality and Speed

An important goal of P3M is to accelerate the AutoML pipeline synthesis process with minimal negative impact on the quality of the derived pipelines. To assess the effectiveness of P3M, we compare the quality of the derived pipelines and the run times using the following three approaches:

Actual: the original AutoML system that evaluates each synthesized pipeline (i.e., training and testing pipelines with the actual dataset).

Top 5: ML4ML that skips the evaluation of pipelines by using the P3M prediction with a budget to retrieve the top-5 pipelines. In other words, only the top-5 ranked pipelines from P3M predictions are evaluated by training and testing with the actual dataset (see the Deployment part in Figure 3).

Top 10: Same configuration as **Top 5** but that evaluates the top-10 ranked pipelines from P3M predictions.

For classification accuracy, the best one is reported from evaluated pipelines: all synthesized pipelines for the *Actual* method, the top 5 pipelines for the *Top* 5 method and the top 10 for the *Top* 10 method. For runtime, the total time required for deriving and evaluating synthesized pipelines is reported for the *Actual* method. For the *Top* 5 and *Top* 10 methods, the reported time includes the following components: DDN computation, provenance filtering, P3M training, P3M predicting, P3M ranking, plus the time required for evaluating the top-ranked pipelines.

Figure 8 illustrates the performance of P3M using different budgets compared to the *Actual* method in terms of prediction quality and runtime speedup. The first plot, displaying the classification accuracy of P3M with top 5 and top 10 budget sizes, indicates that both budgets can achieve high classification accuracy, closely matching the Actual method. Moreover, the top 10 budget results in marginally better accuracy. The second plot, showcasing the runtime of P3M with top 5 and top 10 budget sizes, reveals that for the majority of datasets, P3M with both budget sizes significantly reduces runtime compared to the Actual method. These findings suggest that P3M can maintain high classification accuracy while greatly accelerating the pipeline synthesis process, demonstrating the effectiveness of P3M in optimizing the performance of machine learning pipeline synthesis.

3.3.1 P3M Speedup in Achieving the Best Accuracy. In the above section, we take an overview that P3M can have a substantial speedup while achieving close-optimal classification accuracy. In this section, let's take a detailed look of P3M acceleration of individual datasets. In addition to comparing P3M with baselines of original AutoML systems, we also compare with random predictions.

AlphaD3M/TPOT - Baseline: AutoML system with its default pipeline search strategy. Each pipeline is evaluated.

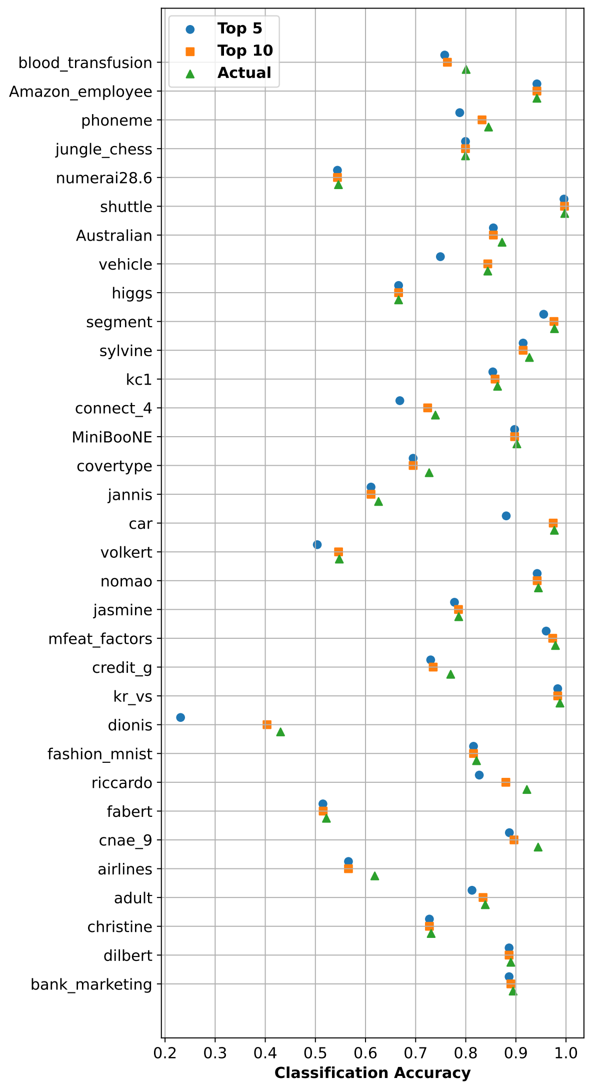
P3M: ML4ML that skips the evaluation of pipelines by using the P3M prediction with budget for top 20. In other words, only the top 20 ranked pipelines from P3M predictions will be evaluated (training and testing with the actual dataset).

Random: similar as P3M with budget for top 20. But instead of using P3M to predict the pipeline performance, a random classification accuracy between 0 and 1 is assigned as the predicted performance.

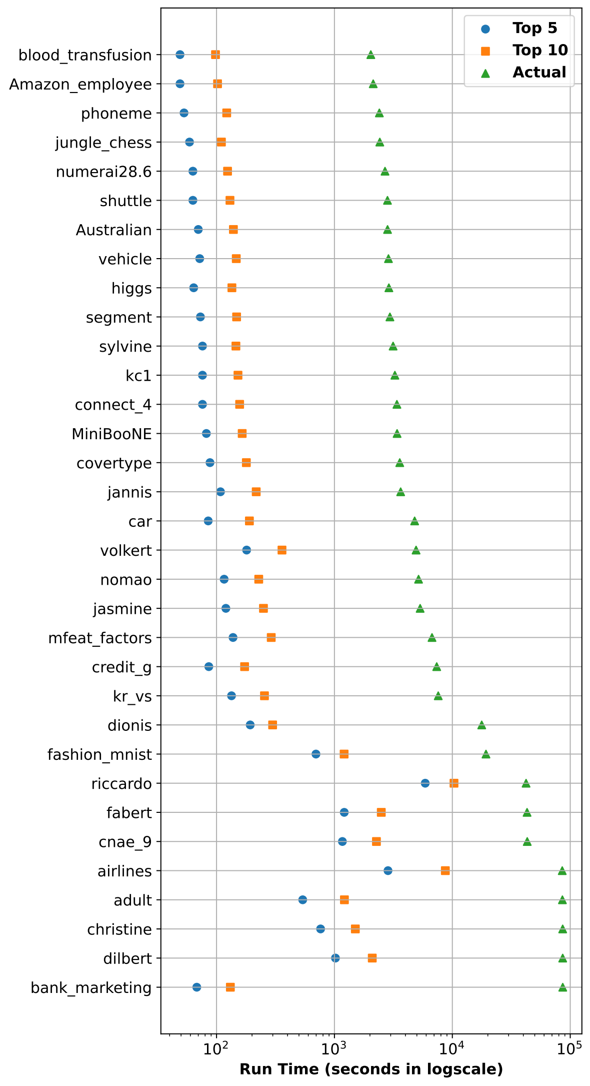
Figure 9 plots the best accuracy over the time from above methods. Two datsets *Airlines* and *Fashion MNIST* are presented, as they represent the diversity of run time and the diversity that different AutoML systems handle the same dataset. For example, *Airlines* dataset runs slowly on AlphaD3M, but fast on TPOT. With the help of P3M, the time to find the optimal pipeline can speed up to 22.9x.

4 RELATED WORK

AutoML for Pipeline Synthesis. AutoML systems have received substantial attention in the recent literature. However, the large search space and the need to evaluate each derived configuration on



(a) P3M Prediction Quality

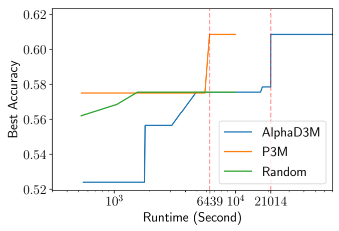


(b) P3M Speedup

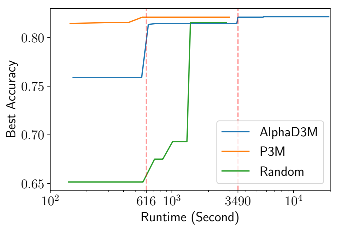
Figure 8: The two plots show the performance of P3M with different budgets (*Top 5* and *Top 10*) compared to the *Actual* method. Y-axis is different dataset names. Both plots are generated by ranking the runtime of the *Actual* method increasingly. The first plot presents the classification accuracy of P3M compared to the Actual method. The results show that P3M with both budgets achieves classification accuracy close to the Actual method, and the top 10 budget is slightly better than top 5. The second plot shows the runtime comparison. P3M has a much faster speed compared to the Actual method. In summary, the results suggest that P3M can achieve a much faster speed with a small trade-off of classification accuracy. Figure 7 presents a summary of relative accuracy and relative runtime.

actual datasets makes AutoML systems computationally expensive. Many approaches have been proposed to make the search process more efficient. These optimization methods include grid/random search [16, 27], genetic/evolutionary algorithms [33, 34], Bayesian optimization [13, 45, 46], matrix factorization/tensor decomposition [52, 53], and reinforcement learning [10, 11]. Several surveys are available that cover these in detail [17, 21, 24, 54, 56, 57]. Meta-learning has also been used to further speed up the search [12, 51]. There have also been approaches that reduce the evaluation cost by training and testing using a small sample of the data [42, 47, 51]. But all these approaches still require each derived pipeline to be evaluated. In this paper, we aim to reduce the cost of the evaluation by replacing the costly training-testing step with prediction.

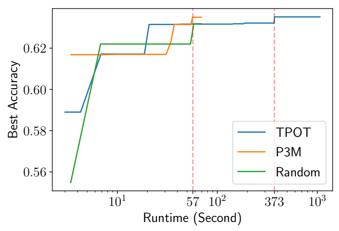
AutoML for Neural Architecture Search. Neural Architecture Search (NAS) aim to identify good configurations for neural networks. Genetic evolution and reinforcement learning have been



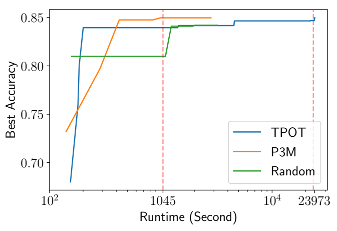
(a) Airlines (AlphaD3M)



(b) Fashion MNIST (AlphaD3M)



(c) Airlines (TPOT)



(d) Fashion MNIST (TPOT)

Figure 9: Deployment Phase: P3M Speedup Effectiveness in Achieving the Best Accuracy. These figures present the run time to find the best pipeline. Each figure has two red vertical lines that mark the time of P3M and the time of Original AutoML system to find the best pipeline. The x-axis represents run time in seconds and in log scale, and y-axis represents the best accuracy. With the help of P3M, a 3.2X to 22.9X speedup can be achieved to find the optimal pipeline.

employed to search different neural architectures [3, 6, 32, 36, 39, 48, 55, 58, 59]. All of these methods require a large amount of training time when exploring the search space. To reduce the training cost, approaches have been proposed to predict the performance of a given architecture [4, 9, 26, 43]. *Peephole* [8] utilizes the network structure information to predict the network accuracy, but it does not leverage information from similar datasets. Thus, given a new dataset, it needs to train hundreds of networks to build a reliable prediction model. *TAPAS* [22] constructs network performance prediction model on unseen dataset without expensive training. This is achieved by learning from previous experiments and predicting based on the difficulty of the dataset. Similar to ML4ML, these approaches replace evaluation with prediction, but there are important differences: (1) While NAS approaches have focused on image or language data, ML4ML has to support tabular datasets; and (2) Neural network performance prediction is essentially *primitive-level* prediction (i.e., the neural network can be viewed as an *Estimation* algorithm). In contrast, our approach derives *pipeline-level* performance predictions and each pipeline consists of multiple primitives for data pre-processing, feature engineering, and estimation. These require new approaches for encoding data and the pipeline, as well as for learning which we describe in detail in Section 2.

5 CONCLUSION

We propose ML4ML, an approach that significantly reduces the evaluation overhead for AutoML systems. ML4ML leverages the provenance of prior pipeline runs and the data they use – stored in a pipeline provenance store– to predict pipeline performance without having to train and test them. We present results of an experimental evaluation which demonstrate that not only can ML4ML build a reliable predictive model with low mean absolute error, but the integration of this model with an AutoML system leads to substantial speedups, enabling the system to not only explore a larger number of pipelines and primitives, but also derive effective pipelines at a much lower cost.

Our experimental evaluation provided valuable insights and showed that it is possible to predict pipeline performance for classification tasks over tabular datasets. For future research, we plan to explore additional learning tasks such as regression and time series problems, various types of datasets or data formats like audio and video, and a broader range of open-source AutoML systems. Simultaneously, we plan to investigate and incorporate more dataset and pipeline encoding methods to further enhance the performance of P3M.

Acknowledgements. This work was supported by the DARPA D3M program and NSF award IIS-2106888. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, DARPA.

REFERENCES

[1] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís P. F. Garcia, Jefferson T. Oliva, and André C. P. L. F. de Carvalho. 2020. MFE: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research* 21, 111 (2020), 1–5. http://jmlr.org/papers/v21/19-348.html
[2] AlphaD3M. 2022. AlphaD3M. https://github.com/ViDA-NYU/ache. (accessed September 12, 2022).
[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167* (2016).
[4] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823* (2017).
[5] BYU-DML. 2019. BYU's python library of useable tools for metalearning. (2019). https://github.com/byu-dml/metalearn
[6] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[7] François Chollet. 2015. Keras. https://github.com/fchollet/keras.
[8] Boyang Deng, Junjie Yan, and Dahua Lin. 2017. Peephole: Predicting Network Performance Before Training. (2017). arXiv:1712.03351 http://arxiv.org/abs/1712.03351
[9] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence.*
[10] Iddo Drori, Yamuna Krishnamurthy, Raoni Lourenco, Remi Rampin, Kyunghyun Cho, Claudio Silva, and Juliana Freire. 2019. Automatic machine learning by pipeline synthesis using model-based reinforcement learning and a grammar. *arXiv preprint arXiv:1905.10345* (2019).
[11] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni Lourenço, Jorge One, Kyunghyun Cho, Claudio Silva, and Juliana Freire. 2018. AlphaD3M: Machine learning pipeline synthesis. In *AutoML Workshop at ICML.*
[12] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. (2020), 1–56. arXiv:2007.04074 http://arxiv.org/abs/2007.04074
[13] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-Sklearn 2.0: The Next Generation. (jul 2020). arXiv:2007.04074 http://arxiv.org/abs/2007.04074
[14] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds.*
[15] P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. 2019. An Open Source AutoML Benchmark. *arXiv preprint arXiv:1907.00909 [cs.LG]* (2019). https://arxiv.org/abs/1907.00909 Accepted at AutoML Workshop at ICML 2019.
[16] H2O.ai. 2021. H2O AutoML. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html. (accessed September 12, 2022).

[17] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems* 212 (2021), 106622.

[18] Jeff Heaton. 2008. The number of hidden layers. *Heaton Research Inc* (2008).

[19] Geoffrey E Hinton. 1990. Connectionist learning procedures. In *Machine learning*. Elsevier, 555–610.

[20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[21] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

[22] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, and A. C.I. Malossi. 2019. TAPAS: Train-less accuracy predictor for architecture search. *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019* i (2019), 3927–3934. https://doi.org/10.1609/aaai.v33i01.33013927 arXiv:1806.00250

[23] Kaggle. 2022. Kaggle. https://www.kaggle.com/. (accessed September 12, 2022).

[24] David Jacob Kedziora, Katarzyna Musial, and Bogdan Gabrys. 2020. AutonoML: Towards an Integrated Framework for Autonomous Machine Learning. Ml (2020). arXiv:2012.12600 http://arxiv.org/abs/2012.12600

[25] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[26] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. 2016. Learning curve prediction with Bayesian neural networks. (2016).

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

[28] Jure Leskovec. 2021. CS224W: Machine Learning with Graphs. (2021). http://web.stanford.edu/class/cs224w/

[29] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, and Bin Cui. 2021. VolcanoML: Speeding up End-to-End AutoML via Scalable Search Space Decomposition. 14, 11 (2021). https://doi.org/10.14778/3476249.3476270 arXiv:2107.08861

[30] Steve Lohr. 2019. At tech's leading edge, worry about a concentration of power. *The New York Times* 26 (2019), 2019.

[31] MARVIN. 2022. MARVIN: An Open Machine Learning Corpus and Environment for Automated Machine Learning. https://marvin.datadrivendiscovery.org/. (accessed September 12, 2022).

[32] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 293–312.

[33] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. 2018. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning* 107, 8 (2018), 1495–1515.

[34] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.

[35] OpenML. 2022. OpenML. https://www.openml.org/. (accessed September 12, 2022).

[36] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*. PMLR, 4095–4104.

[37] João Felipe Pimentel, Juliana Freire, Leonardo Murta, and Vanessa Braganholo. 2019. A survey on collecting, managing, and analyzing provenance from scripts. *ACM Computing Surveys (CSUR)* 52, 3 (2019), 1–38.

[38] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Jordan Henkel, Matteo Interlandi, Subru Krishnan, Brian Kroth, Venkatesh Emani, Wentao Wu, Ce Zhang, et al. 2022. Data Science Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML. NET Pipelines. *ACM SIGMOD Record* 51, 2 (2022), 30–37.

[39] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*. PMLR, 2902–2911.

[40] Florian Scheidegger, Roxana Istrate, Giovanni Mariani, Luca Benini, Costas Bekas, and Cristiano Malossi. 2020. Efficient image dataset classification difficulty estimation for predicting deep-learning accuracy. *Visual Computer* (2020). https://doi.org/10.1007/s00371-020-01922-5 arXiv:1803.09588

[41] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Green ai. *Commun. ACM* 63, 12 (2020), 54–63.

[42] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing data science through interactive curation of ML pipelines. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2019), 1171–1188. https://doi.org/10.1145/3299869.3319863

[43] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. 2014. Freeze-thaw Bayesian optimization. *arXiv preprint arXiv:1406.3896* (2014).

[44] Ameet Talwalkar. 2021. Ai in the 2020s must get greener-and here's how. https://spectrum.ieee.org/energy-efficient-green-ai-strategies. (accessed September 12, 2022).

[45] Janek Thomas, Stefan Coors, and Bernd Bischl. 2018. Automatic gradient boosting. *arXiv preprint arXiv:1807.03873* (2018).

[46] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Vol. Part F1288. Association for Computing Machinery, 847–855. https://doi.org/10.1145/2487575.2487629 arXiv:1208.3719

[47] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: A Fast and Lightweight AutoML Library. *Proceedings of Machine Learning and Systems* 3 (2021).

[48] Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*. 1379–1388.

[49] Doris Xin, Eva Yiwei Wu, Doris Jung-Lin Lee, Niloufar Salehi, and Aditya Parameswaran. 2021. Whither automl? understanding the role of automation in machine learning workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–16.

[50] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[51] Anatoly Yakovlev, Hesam Fathi Moghadam, Ali Moharrer, Jingx-Iao Cai, Nikan Chavoshi, Venkatanathan Varadarajan, Sandeep R Agrawal, Sam Idicula, Tomas Karnagel, Sanjay Jinturkar, and Nipun Agarwal. 2020. Oracle AutoML. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3166–3180. https://doi.org/10.14778/3415478.3415542

[52] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. 2019. OBoe: Collaborative filtering for automl model selection. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 1173–1183. https://doi.org/10.1145/3292500.3330909 arXiv:1808.03233

[53] Chengrun Yang, Jicong Fan, Ziyang Wu, and Madeleine Udell. 2020. AutoML Pipeline Selection: Efficiently Navigating the Combinatorial Space. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Virtual Event, USA. ACM, 1446–1456. https://doi.org/10.1145/3394486.3403197

[54] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).

[55] Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. 2017. Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552* 6 (2017).

[56] Marc-André Zöller and Marco F Huber. 2019. Survey on automated machine learning. *arXiv preprint arXiv:1904.12054* 9 (2019).

[57] Marc-André Zöller and Marco F Huber. 2021. Benchmark and survey of automated machine learning frameworks. *Journal of Artificial Intelligence Research* 70 (2021), 409–472.

[58] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[59] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.