A Scalable AutoML Approach Based on Graph Neural Networks

Mossad Helali
Concordia University
Montreal, Canada
mossad.helali@concordia.ca

Essam Mansour
Concordia University
Montreal, Canada
essam.mansour@concordia.ca

Ibrahim Abdelaziz
IBM T.J. Watson Research Center
New York, United States
ibrahim.abdelaziz1@ibm.com

Julian Dolby
IBM T.J. Watson Research Center
New York, United States
dolby@us.ibm.com

Kavitha Srinivas
IBM T.J. Watson Research Center
New York, United States
Kavitha.Srinivas@ibm.com

ABSTRACT

AutoML systems build machine learning models automatically by performing a search over valid data transformations and learners, along with hyper-parameter optimization for each learner. Many AutoML systems use meta-learning to guide search for optimal pipelines. In this work, we present a novel meta-learning system called KGpip which (1) builds a database of datasets and corresponding pipelines by mining thousands of scripts with program analysis, (2) uses dataset embeddings to find similar datasets in the database based on its content instead of metadata-based features, (3) models AutoML pipeline creation as a graph generation problem, to succinctly characterize the diverse pipelines seen for a single dataset. KGpip's meta-learning is a sub-component for AutoML systems. We demonstrate this by integrating KGpip with two AutoML systems. Our comprehensive evaluation using 121 datasets, including those used by the state-of-the-art systems, shows that KGpip significantly outperforms these systems.

PVLDB Reference Format:
Mossad Helali, Essam Mansour, Ibrahim Abdelaziz, Julian Dolby, and Kavitha Srinivas. A Scalable AutoML Approach Based on Graph Neural Networks. PVLDB, 15(11): 2428 - 2436, 2022.
doi:10.14778/3551793.3551804

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CoDS-GCS/kgpip-public.

1 INTRODUCTION

AutoML is the process by which machine learning models are built automatically for a new dataset. Given a dataset, AutoML systems perform a search over valid data transformations and learners, along with hyper-parameter optimization for each learner [18]. Choosing the transformations and learners over which to search is our focus. A significant number of systems mine from prior runs of pipelines over a set of datasets to choose transformers and learners that are effective with different types of datasets (e.g. [10], [33], [9]). Thus, they build a database by actually running different pipelines with a diverse set of datasets to estimate the accuracy of potential pipelines. Hence, they can be used to effectively reduce the search space. A new dataset, based on a set of features (meta-features) is then matched to this database to find the most plausible candidates for both learner selection and hyper-parameter tuning. This process of choosing starting points in the search space is called meta-learning for the cold start problem.

Other meta-learning approaches include mining existing data science code and their associated datasets to learn from human expertise. The AL [2] system mined existing Kaggle notebooks using dynamic analysis, i.e., actually running the scripts, and showed that such a system has promise. However, this meta-learning approach does not scale because it is onerous to execute a large number of pipeline scripts on datasets, preprocessing datasets is never trivial, and older scripts cease to run at all as software evolves. It is not surprising that AL therefore performed dynamic analysis on just nine datasets.

Our system, KGpip, provides a scalable meta-learning approach to leverage human expertise, using static analysis to mine pipelines from large repositories of scripts. Static analysis has the advantage of scaling to thousands or millions of scripts [1] easily, but lacks the performance data gathered by dynamic analysis. The KGpip meta-learning approach guides the learning process by a scalable dataset similarity search, based on dataset embeddings, to find the most similar datasets and the semantics of ML pipelines applied on them. Many existing systems, such as Auto-Sklearn [9] and AL [2], compute a set of meta-features for each dataset. We developed a deep neural network model to generate embeddings at the granularity of a dataset, e.g., a table or CSV file, to capture similarity at the level of an entire dataset rather than relying on a set of meta-features.

Because we use static analysis to capture the semantics of the meta-learning process, we have no mechanism to choose the **best** pipeline from many seen pipelines, unlike the dynamic execution case where one can rely on runtime to choose the best performing pipeline. Observing that pipelines are basically workflow graphs, we use graph generator neural models to succinctly capture the statically-observed pipelines for a single dataset. In KGpip, we formulate learner selection as a graph generation problem to predict optimized pipelines based on pipelines seen in actual notebooks.

KGpip does learner and transformation selection, and hence is a component of an AutoML systems. To evaluate this component, we integrated it into two existing AutoML systems, FLAML [31] and Auto-Sklearn [9]. We chose FLAML because it does not yet have any meta-learning component for the cold start problem and instead

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551804

allows user selection of learners and transformers. The authors of FLAML explicitly pointed to the fact that FLAML might benefit from a meta-learning component and pointed to it as a possibility for future work. For FLAML, if mining historical pipelines provides an advantage, we should improve its performance. We also picked Auto-Sklearn as it does have a learner selection component based on meta-features, as described earlier [8]. For Auto-Sklearn, we should at least match performance if our static mining of pipelines can match their extensive database. For context, we also compared KGpip with the recent VolcanoML [18], which provides an efficient decomposition and execution strategy for the AutoML search space. In contrast, KGpip prunes the search space using our meta-learning model to perform hyperparameter optimization only for the most promising candidates.

The contributions of this paper are the following:

- Section 3 defines a scalable meta-learning approach based on representation learning of mined ML pipeline semantics and datasets for over 100 datasets and 11K Python scripts.
- Sections 4 formulates AutoML pipeline generation as a graph generation problem. KGpip efficiently predicts optimized ML pipelines for unseen datasets using our meta-learning model. To the best of our knowledge, we are the first to formulate AutoML pipeline generation this way.
- Section 5 presents a comprehensive evaluation using a large collection of 121 datasets from major AutoML benchmarks and Kaggle. Our experimental results show that KGpip outperforms all existing AutoML systems and achieves state-of-the-art results on the majority of these datasets. KGpip significantly improves the performance of both FLAML and Auto-Sklearn in classification and regression tasks. We also outperformed AL in 75 out of 77 datasets and VolcanoML in 75 out of 121 datasets, including 44 datasets used only by VolcanoML [18]. On average, KGpip achieves scores that are statistically better than the means of all other systems.

2 RELATED WORK

In this section, we summarize the related work and restrict our review to meta-learning approaches for AutoML, dataset embeddings, and processing tabular structured data.

Learner and preprocessing selection. In most AutoML systems, learner and pre-processing selection for the cold start problem is driven by a database of actual executions of pipelines and data; e.g., [2], [9], [10]. This database often drives both learner selection and hyper parameter optimization (HPO), so we focus here more on how the database is collected or applied to either problem, since the actual application to learner selection or HPO is less relevant. For HPO, some have cast the application of the database as a multi-task problem (see [27]), where the hyperparameters for cold start are chosen based on multiple related datasets. Others, for instance, [9, 26], compute a database of dataset meta-features on a variety of OpenML [29] datasets, including dataset properties such as the number of numerical attributes, the number of samples or skewness of the features in each dataset.

These systems measure similarity between datasets and use pipelines from the nearest datasets based on the distance between the datasets' feature vectors as we do, but the computation of these vectors is different, as we describe in detail below. Auto-Sklearn 2.0 [8] defines instead a static portfolio of pipelines that work across a variety of datasets, and use these to cold-start the learner selection component - i.e., every new dataset uses the same set of pipelines. Others have created large matrices documenting the performance of candidate pipelines for different datasets and viewed the selection of related pipelines as a collaborative filtering problem [10].

Dataset embeddings. The most used mechanism to capture dataset features rely on the use of meta-features for a dataset such as [9, 26]. These dataset properties vary from simple, such as number of classes (see, e.g. [6]), to complex and expensive, such as statistical features (see, e.g. [30]) or landmark features (see, e.g. [24]). As pointed out in Auto-Sklearn 2.0 [8], these meta-features are not defined with respect to certain column types such as categorical columns, and they are also expensive to compute, within limited budgets. The dataset embedding we adopt is builds individual column embeddings, and then pools these for a table level embedding. Similar to our approach, Drori et al. [5] use pretrained language models to get dataset embeddings based on available dataset textual information, e.g. title, description and keywords. Given these embeddings, their approach tries to find the most similar datasets and their associated baselines. Unlike [5], our approach relies on embedding the actual data inside the dataset and not just their overall textual description, which in many cases is not available. OBOE [34] uses the performance of a few inexpensive, informative models to compute features of a model.

Pipeline generation. There is a significant amount of work viewing the selection of learners as well as hyperparameters as a bayesian optimization problem like [27, 28]. Other systems have used evolutionary algorithms along with user defined templates or grammars for this purpose such as TPOT [16] or Recipe [4]. Still, others have viewed the problem of pipeline generation as a probabilistic matrix factorization [10], an AI planning problem when combined with a user specified grammar [15, 32], a bayesian optimization problem combined with Monte Carlo Tree Search [25], or an iterative alternating direction method of multipliers optimization (ADMM) problem [20]. Systems like VolcanoML focus on an efficient decomposition of the search space [18]. To the best of our knowledge, KGpip is the first system to cast the actual generation of pipelines as a neural graph generation problem.

Some recent AutoML systems have moved away from the fairly linear pipelines generated by most earlier systems to use ensembles or stacking extensively. H2O for instance uses fast random search in combination with ensembling for the problem of generating pipelines [17]. Others rely on "stacking a bespoke set of models in a predefined order", where stacking and training is handled in a special manner to achieve strong performance [7]. Similarly, PIPER [21] uses a greedy best-first search algorithm to traverse the space of partial pipelines guided over a grammar that defines complex pipelines such as Directed Acyclic Graphs (DAGs). The pipelines produced by PIPER are more complex than the linear structures used in the current AutoML systems we use to test our ideas for historical pipeline modeling, and we do not use ensembling techniques yet in our approach. Neither is precluded, however, because KGpip meta-learning model can generate any type of structures, including complex structures that mined pipelines may have.

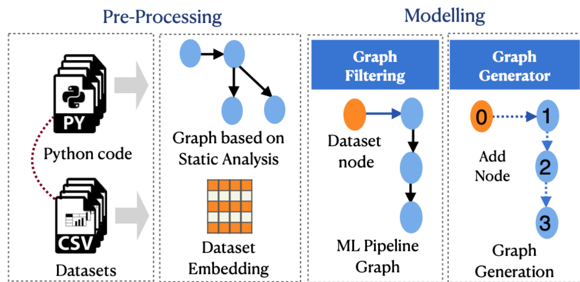


Figure 1: An overview of KGpip's meta-learning approach for mining a database of ML pipelines to train a graph generator model to predict ML pipeline skeletons in the form of graphs.

```
df = pd.read_csv('example.csv')
df_train, df_test = train_test_split(df)
X = df_train['X']
model = svm.SVC()
model.fit(X, df_train['Y'])
```

Figure 2: An example from a data science notebook.

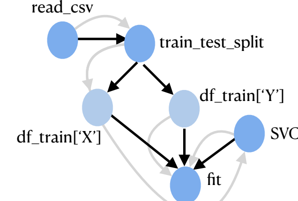


Figure 3: Code graph corresponding to Figure 2 obtained with GraphGen4Code. The graph shows control flow with gray edges and data flow with black edges. Numerous other nodes and edges are not shown for simplicity.

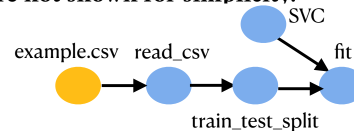


Figure 4: Our MetaPip graph of the graph from Figure 3, where the abstracted ML pipeline is linked to a dataset node (highlighted in Orange). MetaPip contains at least 96% less nodes and edges than the original graph while enhancing the overall quality of the graph generation process, as experimented in Section 5.5.

3 THE KGPIP SCALABLE META-LEARNING

Our meta-learning approach is based on mining large databases of ML pipelines associated with the used datasets, as illustrated in Figure 1. The mining process uses static program analysis instead of executing the actual pipeline scripts or preparing the actual raw data. The KGpip meta-learning component enhances the search strategy of existing AutoML systems, such as AutoSklearn and FLAML, and allows these systems to handle ad-hoc datasets, i.e., unseen ones. To retain a maximal degree of flexibility, KGpip captures metadata and semantics in a flexible graph format, and relies on graph generator models as the database of pipelines.

Unlike existing meta-learning approaches, our approach is designed to learn from a large scale database and achieve high degree of coverage and diversity. Several ML portals, such as Kaggle or OpenML [29], provide access to thousands of datasets associated with hundreds of thousands of public notebooks, i.e., ML pipelines/code. KGpip mines these large databases of datasets and pipelines using static analysis and filters them into ML pipelines customized for the learner selection problem. The KGpip meta-learning approach leverages [1] for code understanding via static analysis of scripts/code of ML pipelines. It extracts the semantics of these scripts as code and form an initial graph for each script.

KGpip cleans the graphs generated by [1] to keep the semantic required for the ML meta-learning process. Furthermore, our approach introduces dataset nodes and interlinks the relevant pipeline semantic to them. So, our meta-learning approach produces MetaPip, a highly interconnected graph of seen datasets and pipelines applied to them. We also developed a deep embedding model to find the closest datasets to an unseen one, i.e., to effectively prune MetaPip. We then train a deep graph generator model [19] using MetaPip. This model is the core of our meta-learning component as illustrated in Figure 1 and discussed in the next section.

3.1 Graph Representation of Code Semantics

Static and dynamic program analysis techniques could be used to abstract the semantics of programs and extract language-independent representations of code. A program source code is examined in the static analysis without running the program. In contrast, dynamic analysis examines the source code during runtime to collect memory traces and more detailed statistics specific to the analysis technique. Unlike static analysis, dynamic analysis helps in capturing more rich semantics from programs with the high cost of execution and storing massive memory traces. ML portals, such as Kaggle, have hundreds of thousands of ML pipelines with no instructions for running or managing the environments of these pipelines. KGpip combines dataset embedding with static code analysis tools, such as GraphGen4Code [1], to enrich the collected semantics of ML pipelines while avoiding the need to run them.

GraphGen4Code is optimized to efficiently process millions of Python programs, performing interprocedural data flow and control flow analysis to examine for instance, what happens to data that is read from a Pandas dataframe, how it gets manipulated and transformed, and what transformers or estimators get called on the dataframe. GraphGen4Code's graphs make it explicit what APIs and functions are invoked on objects without the need to model the used libraries themselves; hence GraphGen4Code can scale static analysis to millions of programs. Figures 2 and 3 show a small code snippet and its corresponding static analysis graph from GraphGen4Code, respectively. As shown in Figure 3, the graph captures control flow (gray edges), data flow (black edges), as well as numerous other nodes and edges that are not shown in the figure. Examples of these nodes and edges include those capturing location of calls inside a script file and function call parameters. For example, GraphGen4Code generates a graph of roughly 1600 nodes and 3700 edges for a Kaggle ML pipeline script of 72 lines of code. The number of nodes and edges dominate the complexity of training a graph generator model.

3.2 MetaPip: from Code to Pipeline Semantics

For AutoML systems, a pipeline is a set of data transformations, learner selection, and hyper-parameter optimization for each model that is selected. Mined data science notebooks often contain data analysis, data visualization, and model evaluation. Moreover, each

notebook is associated with one or more datasets. Thus, it is essential for our meta-learning model to distinguish between different types of pipelines and realize this association with datasets. Existing systems for static code analysis extract general semantics of code and cannot link pipeline scripts to the used datasets. Thus, the generated graphs by systems, such as GraphGen4Code, are scattered and unlinked, i.e., a graph per an ML pipeline script. Moreover, each graph will have nodes and edges that are not relevant for the meta-learning process. These irrelevant nodes and edges, i.e., triples, will add noise to the training data. Hence, a meta-learning model will not be able to learn from the abstracted graph pipelines generated by such tools, as shown in Table 4. We developed a method to filter out this kind of triples from GraphGen4Code's graph and analyze ML pipelines to prepare a training set interconnecting repositories of ML pipeline scripts with their associated datasets. Moreover, our method cleans the noisy nodes and edges and calls to modules outside the target ML libraries. For example, our method will extract triples related to libraries, such as Scikit-learn, XGBoost, and LGBM. These libraries are the most popular among the top-scoring ML pipelines in ML portals. The code for the cleaning method is available at the KGpip's repository.

Our meta-learning component aims to pick learners and transformer for unseen datasets. Thus, KGpip links the filtered ML pipelines with the used datasets. The result of adding these dataset nodes is a highly interconnected graph for ML pipelines, we refer to it as *MetaPip*. Our MetaPip graph captures both the code and data aspects of ML pipelines. Hence, we can populate the MetaPip graph with datasets from different sources, such as OpenML and Kaggle, and pipelines applied on these datasets. Figure 4 shows the MetaPip graph corresponding to the code snippet in Figure 2. KGpip utilizes MetaPip to train a model based on a large set of pipelines associated with similar datasets. For example, a *pandas.read_csv* node will be linked to the used table node, i.e., csv file. In some cases, the code, which reads a csv file, does not explicitly mention the dataset name. The pipelines are usually associated with datasets, such as Kaggle pipelines and datasets, as shown in Figure 1.

3.3 Dataset Representation Learning

Our approach efficiently guides the meta-learning process by linking the extracted semantics of pipelines to dataset nodes representing the used datasets. There is a sheer amount of datasets of variable sizes and we need to develop a scalable method for finding the most similar datasets for an unseen one. The pairwise comparison based of the actual content of datasets, i.e, tuples in CSV files, does not scale. Thus, we developed a dataset representation learning method to generate a fixed-size and dense embedding at the granularity of a dataset, e.g., a table or CSV file. The embedding of a dataset $\mathcal{D}$ is the average of its column embeddings, i.e.:

$$h_\theta(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{c \in \mathcal{D}} h_\theta(c) \quad (1)$$

where $|\mathcal{D}|$ is the number of columns in $\mathcal{D}$. Our work generalizes the approach outlined in [22] for individual column embeddings, where column embeddings are obtained by training a neural network on a binary classification task. The model learns when two columns represent the same concept, but with different values, as opposed to columns representing different concepts. Embeddings for an unseen dataset are produced by the last layer of the neural net.

KGpip reads datasets only once and leverages PySpark DataFrame to achieve high task and data parallelism. We use the embeddings of datasets to measure their similarity. With these embeddings, we build an index of vector embeddings for all the datasets in our training set. We utilize efficient libraries [14] for similarity search of dense vectors to retrieve the most similar dataset to a new input dataset based on its embeddings. Thus, our method scales well and leads to accurate results in capturing similarities between datasets.

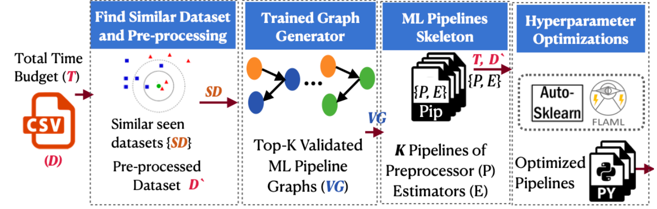


Figure 5: An overview of KGpip's workflows of ML pipeline generation for a given unseen dataset and certain time budget. KGpip utilizes systems for hyperparameter optimization, such as FLAML or Auto-Sklearn, to optimize KGpip's top-K predicted pipelines (VG), i.e., pruning the search space.

4 THE KGPIP PIPELINE AUTOMATION

The KGpip workflow for pipeline automation is based on our meta-leaning model, as illustrated in Figure 5. KGpip predicts the top-K pipeline skeletons, i.e., a specific set $\{P, E\}$ of Preprocessor (P) and Estimators (E), for an unseen dataset (D) based on the most similar seen dataset (SD), i.e., the nearest neighbour dataset. KGpip starts by finding SD based on the embedding of the unseen dataset. Then, KGpip generates the top-K validated ML pipeline graphs VG and converts them into ML pipeline skeletons $\{P, E\}$. Then, it performs hyperparameter optimization using systems, such as FLAML [31] and Auto-Sklearn [9], to find the optimum hyperparameters for each pipeline skeleton within a specific time budget.

4.1 Graph Generation for ML Pipelines

KGpip formulates the generation of ML pipelines as a graph generation problem. The intuition behind this idea is that a neural graph generator might capture more succinctly multiple pipelines seen in practice for a given dataset, and might also capture statistical similarities between different pipelines more effectively. To effectively use such a network, we add a single dataset node as the starting point for the filtered pipelines we generate from Python notebooks. The node is assumed to flow into a `read_csv` call which is often the starting point for the pipelines. For generating an ML pipeline, we simply pass in a dataset node for the nearest neighbour of the unseen dataset, i.e., the most similar dataset based on content similarity, as shown in Figure 5.

Our meta-learning model generates ML pipeline graphs in a sequential node-by-node fashion. Algorithm 1 illustrates the implementation of the graph generation model. For an empty graph G and the most similar dataset SD, the algorithm starts by adding an edge between SD and `pandas.read_csv`. Then, the graph neural network $f_{AddNode}$ decides whether to add a new node of a certain type. The network $f_{AddEdge}$ decides whether to add an edge

Algorithm 1: Graph Generation Process

Input: Graph G: $(E = \phi, V = \phi)$, Similar Dataset Node: SD,
Neural Networks: $f_{AddNode}, f_{AddEdge}, f_{ChooseNode}$

```
1  V ← V ∪ {SD, pandas.read_csv}
2  E ← E ∪ {(SD, pandas.read_csv)}
3  nodeToAdd = f_AddNode(V, E)
4  while nodeToAdd ≠ Null do
5      V ← V ∪ {nodeToAdd}
6      addEdge = f_AddEdge(V, E)
7      while addEdge do
8          nodeToLink = f_ChooseNode(V, E)
9          E ← E ∪ {(nodeToAdd, nodeToLink)}
10         addEdge ← f_AddEdge(V, E)
11     end
12     nodeToAdd ← f_AddNode(V, E)
13 end
14 VG = validate_pipeline_graph(G)
15 return VG
```

to the newly added node. Then, the network $f_{ChooseNode}$ decides the existing node to which the edge is to be added. The While loop at line 7 is repeated repeated until no more edges to be added. The While loop at line 4 is repeated until no more nodes to be added. The three neural networks, namely $f_{AddNode}$, $f_{AddEdge}$, and $f_{ChooseNode}$, utilize node embeddings that are learned throughout training via graph propagation rounds. These embeddings capture the structure of ML pipeline graphs.

The generated graph G is not guaranteed to be a valid ML pipeline. Thus, Algorithm 1 at line 14 checks that G is a valid ML pipeline graph. In KGpip, a graph G is valid if 1) it contains at least one estimator matching the task, i.e., regression or classification, and 2) the estimator is supported by the hyperparameter optimizer (AutoSklearn or FLAML in our case). With these modifications, it is possible to generate ML pipelines for unseen datasets using the closest seen dataset node – more specifically, its content embedding obtained from the dataset embedding module. We built Algorithm 1 on top of the system proposed in [19]. This system does not support conditional graph generation at test time by default, i.e., building a graph on top of a provided dataset node. We extended this system to generate valid ML pipeline graphs, as illustrated in Algorithm 1.

4.2 Hyperparameter Optimizion

KGpip maps the valid graphs into ML pipeline skeletons, where each skeleton is a set of pre-processors and an estimator with place holders for the optimal parameters. In KGpip, the hyperparameter optimizer is responsible for finding the optimal parameters for the pre-processors and learners on the target dataset. Then, KGpip replaces the place holders with these parameters. Finally, KGpip creates a python script using the pre-processors and estimator achieving the highest scores. KGpip is well designed to support both numerical and non-numerical datasets. Thus, KGpip applies different pre-processing techniques on the given dataset (D) and produces a pre-processed dataset (D'). Our pre-processing includes 1) detecting task type (i.e. regression or classification) automatically based on the distribution of the target column 2) automatically inferring accurate data types of columns, 3) vectorizing textual columns using word embeddings [3], and 4) imputing missing values in the dataset. In KGpip, the hyperparameter optimizer uses D'.

Similar to hyperparameter optimizers implemented in AutoML systems, such as FLAML or Auto-Sklearn, KGpip works within a provided time budget per dataset. We note here that the majority of the allotted time budget for ML pipeline generation is spent on the hyperparameter optimization; that is, if the user desires only to know what learners would work best for their dataset, KGpip can do that almost instantaneously. Given a time budget (T), KGpip calculates t, the time consumed in generating and validating the graphs. KGpip then divides the rest of the time budget between the K graphs. Hence, the hyperparameter optimizer has a time limit of $((T - t)/K)$ to optimize each graph independently.

The hyperparameter optimizer repeatedly applies the learners and pre-processors with different configurations while monitoring the target score metric throughout. KGpip keeps updating its output with the best pipeline skeleton, i.e., learners and pre-processors, and its score. For example, if the predicted learner is LogisticRegression, it searches for the best combination of regularization type (L1 or L2) and regularization parameter. The difference between hyperparameter optimizers is the search strategy followed to arrive at the best hyperparameters within the allotted time budget. A naive approach would be to perform an exhaustive grid search over all combinations, while a more advanced approach would be to start with promising configurations first. We integrate KGpip with the hyperparameter optimizers of both FLAML [31] and Auto-Sklearn [9] to demonstrate the generality of KGpip. The integration of a hyperparameter optimizer into KGpip needs a JSON document of the particular preprocessors and estimators supported by the hyperparameter optimizer. Thus, the integration is relatively easy. Finally, our neural graph generation produces a diverse set of pipelines across runs, allowing for exploration and exploitation.

5 EXPERIMENTS

5.1 Benchmarks

We evaluate KGpip as well as the other baselines on four benchmark datasets: 1) *Open AutoML Benchmark* [11], a collection of 39 binary and multi-class *classification* datasets (used by FLAML [31]). The datasets are selected such that they are representative of the real world from a diversity of problem domains and of enough difficulty for the learning algorithms. 2) *Penn Machine Learning Benchmark* (PMLB) [23]: Since Open AutoML Benchmark is limited to classification datasets, the authors of FLAML [31] evaluated their system on 14 more *regression* datasets selected from PMLB, such that the number of samples is more than 10,000. To demonstrate the generality of our approach, we include those datasets in our evaluation as well. 3) *AL's datasets*: We also evaluate on the datasets used for AL's [2] evaluation which include 6 Kaggle datasets (2 regression and 4 classification) and another 18 classification datasets (9 from PMLB and 9 from OpenML). Unlike other benchmarks, the Kaggle datasets include datasets with textual features. 4) VolcanoML's datasets: finally, we evaluate KGpip on 44 more datasets used by VolcanoML [18]. The authors of VolcanoML evaluate their system on a total of 66 datasets from OpenML and Kaggle, from which 11 datasets are not specified, 10 datasets overlap with ours, and 1 dataset consists of image samples. Table 1 includes a summary of

Table 1: Breakdown of all 121 datasets used in our evaluation, indicating those used by FLAML*, AL†, and VolcanoML§.

Task	Source			
	AutoML	PMLB	OpenML	Kaggle
Binary	22 ($18^{*}+1^{*\dagger}+3^{*\S}$)	5 ($4^{\dagger}+1^{\dagger\S}$)	27 ($3^{\dagger\S}+3^{\dagger}+21^{\S}$)	$2^{\dagger}$
Multi-class	17 ($15^{*}+1^{*\dagger}+1^{*\S}$)	$4^{\dagger}$	7 ($2^{\dagger\S}+1^{\dagger}$ + $4^{\S}$)	$2^{\dagger}$
Regression	0	14^{*}	$19^{\S}$	$2^{\dagger}$
Total	39	23	53	6

all 121 benchmark datasets. The detailed statistics of all datasets are shown in the appendix of [13]. These statistics include names, number of rows and columns, number of numerical, categorical, and textual features, number of classes, sizes, sources, and papers that evaluated on them.

5.2 Baselines

We empirically validate KGpip against three AutoML systems: (1) Auto-Sklearn (v0.14.0) [9] which is the overall winner of multiple challenges in the ChaLearn AutoML competition [12], and one of the top 4 competitors reported in the Open AutoML Benchmark [11]. (2) FLAML (v0.6.6) [31]: an AutoML library designed with both accuracy and computational cost in mind. FLAML outperforms Auto-Sklearn among other systems on two AutoML benchmarks using a low computational budget, (3) AL [2]: a meta-learning-based AutoML approach that utilizes dynamic analysis of Kaggle notebooks, an approach that has similarities to ours, and (4) VolcanoML (v0.5.0) [18], a recent AutoML approach which proposes efficient decomposition strategies for the large AutoML search spaces. In all our experiments, we used the latest code provided by the authors for existing systems, the same exact hardware, time budget, and the parameters recommended by the authors of these systems.

5.3 Training Setup

Because our approach to mining historical pipelines from scripts is relatively cheap, we can apply it more easily on a wider variety of datasets to form a better base as more and more scripts get generated by domain experts on Kaggle competitions. In this work, we performed program analysis on 11.7K scripts associated with 142 datasets, and then selected those with estimators from `sklearn`, `XGBoost` and `LightGBM` since those were the estimators supported by the most AutoML systems for classification and regression. This resulted in the selection of 2,046 notebooks for 104 datasets; a vast portion of the 11.7K programs were about exploratory data analysis, or involved libraries that were not supported by Auto-Sklearn [9] or FLAML (e.g., PyTorch and Keras) [31]. We used Macro F1 for classification tasks to account for data imbalance, if any, and use R^2 for regression tasks, as in FLAML [31]. We also varied the time budget given to each system between 1 hour and 30 minutes, to measure how fast can KGpip find an efficient pipeline compared to other approaches. The time budget is end-to-end, from loading the dataset till producing the best AutoML pipeline. In all experiments, we report averages over 3 runs.

5.4 Comparison with Existing Systems

In this section, we evaluate KGpip against state-of-the-art systems: FLAML [31] and Auto-Sklearn [9]. Figure 6 shows a radar graph of

Table 2: Average scores (mean and standard deviation) of KGpip compared to FLAML, Auto-Sklearn, and VolcanoML for binary classification (F1), multi-class classification (F1) and regression (R^2) tasks on 77 benchmark datasets. T-test values are for KGpip vs. FLAML and KGpip vs. Auto-Sklearn.

	Binary	Multi-class	Regression	T-Test
FLAML	0.74 (0.23)	0.70 (0.29)	0.65 (0.29)	0.0129
KGpipFLAML	0.81 (0.14)	**0.76 (0.24)**	**0.72 (0.24)**	-
Auto-Sklearn	0.76 (0.20)	0.65 (0.29)	0.71 (0.24)	0.0002
KGpipAutoSklearn	**0.83 (0.14)**	0.73 (0.28)	**0.72 (0.24)**	-
VolcanoML	0.55 (0.43)	0.51 (0.38)	0.56 (0.32)	-

all systems when given a time budget of 1 hour. It shows the performance of all systems on the three tasks in all benchmarks, namely, binary classification, multi-class classification, and regression. For every dataset, the figure shows the actual performance metric (F1 for classification and R^2 for regression) obtained from every system [1]. Therefore, the out most curve from the center of the radar graph has the best performance. In Figure 6, both variations of KGpip achieve the best performance across all tasks, outperforming both FLAML and Auto-Sklearn. We also performed a *two-tailed t-Test* between the performance obtained by KGpip compared to the other systems. The results show that KGpip achieves significantly better performance than both FLAML and Auto-Sklearn with a t-Test value of 0.01 and 0.0002, respectively (both have $p < 0.05$).

Table 2 also shows the average F1 and R^2 values for classification and regression tasks, respectively. The results show that both variations of KGpip achieve better performance compared to both FLAML and Auto-Sklearn over all tasks and datasets.

Scalability of KGpip's meta-learning against existing systems: The AL meta-learning approach [2] mines pipelines using dynamic code analysis, which has high cost as discussed in Section 3.1. Thus, the authors of AL provided a pre-trained meta-learning model on 500 pipelines and 9 datasets, which does not scale to cover various cases. In contrast, we trained our meta-learning model using 2000 pipelines and 142 datasets. None of these datasets were included in the 77 datasets used in testing. AL failed in 22 and timed out in 38 datasets. This shows that the KGpip meta-learning approach, which is based on pipelines semantics and dataset representation learning, is more effective. AL failed on many of the datasets during the fitting process. As the figure shows, KGpip still outperforms all other approaches, including AL, significantly. On these datasets, AL achieved the lowest F1 score on binary and multi-class classification tasks with values of 0.36 and 0.36, respectively. This compares to 0.74 and 0.75 by FLAML, 0.73 and 0.68 by Auto-Sklearn, 0.79 and 0.79 by KGpipFLAML, and 0.79 and 0.74 by KGpipAuto-Sklearn.

VolcanoML Datasets: VolcanoML used a variety of datasets that are not included in our 77 datasets of Figure 6. Some of these datasets are quite large which are meant to test the the system scalability. Therefore, we also collected all 49 the datasets we could find in their paper and tested the best version of KGpip (KGpipFLAML) against VolcanoML on these datasets with a time budget of 1 hour. The performances of KGpipFLAML and VolcanoML are shown in Figure 7. For brevity, we omitted from the figure all datasets on which the performance difference between both systems is ≤ 0.01

[1]The detailed scores for every system and dataset as well as the corresponding names of datasets are shown in the appendix of [13]

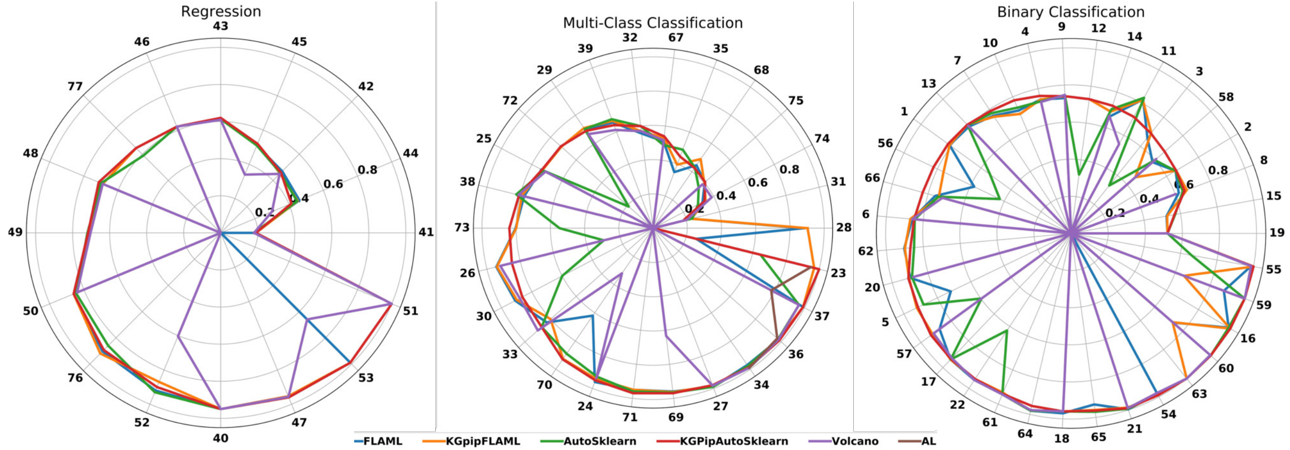


Figure 6: A radar diagram of the performance of KGpip vs. existing systems on multiple tasks (77 datasets) with a time budget of 1 hour for all systems. The outer numbers indicate different dataset IDs and the ticks inside the figure denote performance ranges of respective metrics; e.g., 0.2, 0.4, ..., etc. for F1 in binary classification. For any dataset, the system with the out most curve has the best performance. As an example, KGpipAutoSklearn and KGpipFLAML achieved 100% and 97% F1 on dataset #23 (multi-class classification) compared to 65% and 26% for AutoSklearn and FLAML, respectively.

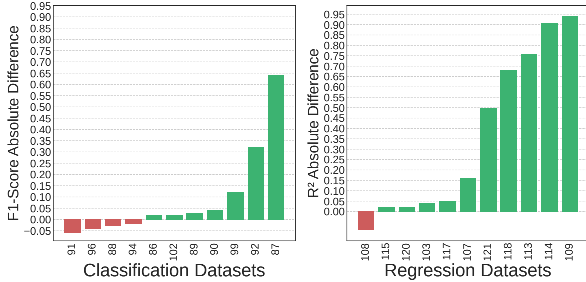


Figure 7: Score difference between KGpipFLAML and VolcanoML on the 44 classification and regression datasets from VolcanoML with a time budget of 1 hour. For brevity, we removed from this Figure the 22 datasets on which both systems perform comparably (within a difference of ≤ 0.01).

Table 3: Average scores (mean and standard deviation) of KGpipFLAML compared to VolcanoML on the 44 datasets from VolcanoML. Overall, KGpipFLAML achieves significantly better compared to VolcanoML, according to a statistical significance test of $p < 0.05$.

	Binary	Multi-class	Regression	T-Test
KGpipFLAML	**0.82 (0.14)**	**0.86 (0.16)**	**0.83 (0.13)**	-
VolcanoML	0.69 (0.23)	0.70 (0.31)	0.68 (0.25)	0.0001

and the 5 datasets overlapping with the ones shown in Figure 6. On those datasets, KGpipFLAML found a valid pipeline for all of them, sometimes with a decent absolute difference in F1 or R^2 scores of ≥ 0.90. Across all the 44 datasets, KGpipFLAML achieved significantly better average of scores compared to VolcanoML (statistical significance test of $p < 0.05$), see Table 3 for details.

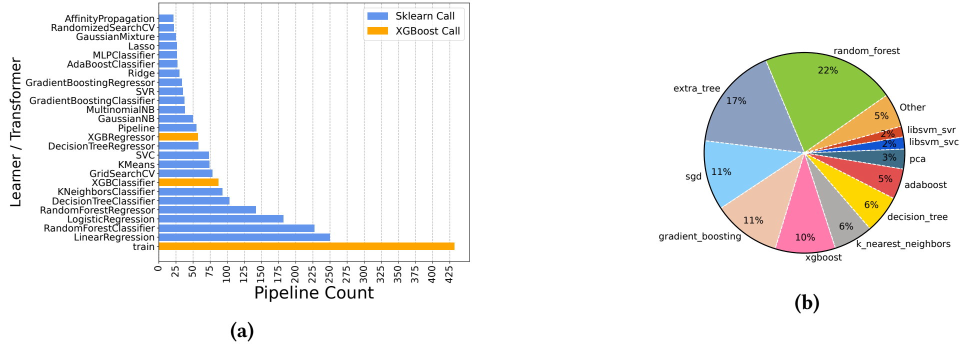
5.5 Ablation Study

5.5.1 The effectiveness of MetaPip. Our MetaPip approach manages to reduce dramatically the number of nodes and edges in the code graph. Using the original graph obtained from static analysis, it produces the MetaPip graph that focuses on the core aspects needed to train a graph generation model for ML pipelines, such as data transformations, learner selection, and hyper-parameter selection. This experiment investigates the scalability of our graph generation model based on two different training sets, i.e., the sets of MetaPip graphs described in section 3.2 vs. the original set of code graphs from static analysis for the same ML pipeline scripts.

Table 4: Different aspects comparing a model trained on a set of code graphs vs a model trained on a set of MetaPip graphs. The model based on original code graphs fails in trivial datasets to generate valid pipelines and limits KGpip's scalability to a larger set of ML pipelines scripts and KGpip's learning by using a fewer number of epochs.

Dataset/Aspect	Code Graph	MetaPip Graph
kr-vs-kp	0 (0)	1.00 (0)
nomao	0 (0)	0.96 (0)
cnae-9	0 (0)	0.95 (0.01)
mfeat-factors	0 (0)	0.98 (0)
segment	0 (0)	0.98 (0)
Avg. F1	0 (0)	0.97 (0.02)
No. Nodes	29,139	974
No. Edges	252,486	1,052
Training Time	175 (min)	2 (min)

For this experiment, we use a small-scale training set of 82 pipeline graphs pertaining to one classification dataset. The original code graphs for these 82 pipelines include 29,139 nodes and 252,486 edges. Our MetaPip graph, however, includes 974 nodes and 1052 edges. This is a graph reduction rate of at least 96.6%, Figure 4 shows these detailed statistics. The main investigation here is whether this huge reduction ratio will help improving the accuracy and scalability of our graph generation model. We train



(a) (b)

Figure 8: Top learner and transformers selected by KGpip (A) are with a wide range of coverage and diversity (B).

Table 5: Performance of KGpipFLAML (mean and standard deviation) as we vary the number of predicted pipeline graphs within 30 minutes time limit. We obtained similar results for KGpipAutoSklearn, and hence omitted its results.

	Binary	Multi-Class	Regression
Top-3 graphs	0.80 (0.14)	0.70 (0.31)	0.71 (0.23)
Top-5 graphs	0.81 (0.14)	0.73 (0.26)	0.70 (0.23)
Top-7 graphs	0.81 (0.14)	0.75 (0.24)	0.71 (0.24)

one model on the original code graph and another on the MetaPip graphs. Both models are trained for 15 epochs with the same set of hyperparameters. It is worth noting that due to the huge time required to process the nodes and edges in the code graph, we had to reduce the number of epochs from 400 to 15.

We test the performance of KGpip when trained on both graphs on the most trivial binary and multi-class classification datasets in the AutoML benchmark. These are the datasets where the F1 score of all the reported systems in section 5.4 is above 0.9. The result is a total of 5 datasets (1 binary and 4 multi-class). Both models use Auto-Sklearn as the hyperparameter optimizer with a time budget of 15 minutes and 3 graphs. We take the average of three runs. The results are summarized in Table 4. For these trivial datasets, the model trained using code graphs did not manage to generate any valid ML pipeline. This means the model failed to capture the core aspects of ML pipelines, i.e., valid transformation or learners. Moreover, our MetaPip approach helps KGpip to reduce the training time by 99%, as shown in Table 4.

5.5.2 The KGpip meta-learning quality. This experiment tests the quality of our meta-learning component. We test the performance as we vary the number of graphs selected from the graph generation phase before feeding it to the hyper-parameter optimization module. Table 5 shows the KGpipFLAML performance as we vary the number of predicted graphs between 3, 5 and 7.

With only 3 graphs, KGpip is still outperforming FLAML (second best system after KGpipFLAML) , although the effect is weaker (t-Test value = 0.06). Compared to Auto-sklearn (third best system after KGpipFLAML), all variations have similar or better performance, but the difference is insignificant. This experiment shows that even with three graphs, KGpip outperforms FLAML and KGpipAuto-Sklearn, i.e., the correct pipelines often appear in the top 3. As another assessment of the quality of our predictions, we measure where in our ranked list of predicted pipelines the best pipeline turned out to be. Ideally, the top pipeline would always be first, and we use Mean Reciprocal Rank (MRR) to measure how close to that our predictions are. Across all runs, the MRR is 0.71, indicating that the top pipeline is typically very near the top.

5.5.3 The KGpip meta-learning diversity. One question we address is whether KGpip produced different pipelines for the *same dataset* across different runs. This gives us a sense of whether KGpip is deterministic, or whether it produces different pipelines to help with pruning the AutoML search space. We took different runs for the exact same dataset, and created a list of learners and transformers produced for each dataset across runs. The list was limited by the shortest number of learners and transformers produced across runs. We then computed correlations for datasets across runs 1, 2, and 3. The correlations ranged from 0.60 - 0.64, suggesting that the runs did not produce the same transformers and learners across runs. We also examined the types of learners selected by KGpip for consideration. Figure 8a shows the learners and transformers found at least 20 times in the training pipelines. One can see from the figure that KGpip does not blindly output learners and transformers by counts. Figure 8b shows more diversity in what was selected overall. So, a variety of methods are covered by KGpip.

6 CONCLUSION

This paper proposed a novel formulation for the AutoML problem as a graph generation problem, where we can pose learner and pre-processing selection as a generation of different graphs representing ML pipelines. Hence, we developed the KGpip system based on mining large repositories of scripts, and leveraging recent techniques for static code analysis. KGpip utilized embeddings generated based on dataset contents to predict and optimize a set of ML pipelines based on the most similar seen datasets. KGpip is designed to work with AutoML systems, such as Auto-Sklearn and FLAML, to utilize their hyperparameter optimizers. We conducted the most comprehensive evaluation of 121 datasets, including the datasets used by FLAML, VolcanoML, and AL. Our comprehensive evaluation shows that KGpip significantly improves the performance of FLAML and Auto-Sklearn in classification and regression tasks. Moreover, KGpip outperformed AL, which is based on a more costly meta-learning process, in 97% of the datasets. This outstanding performance shows that the KGpip meta-learning approach is more effective and efficient. Finally, KGpip outperforms VolcanoML in 62% of the datasets and ties with it in 22%.

REFERENCES

[1] Ibrahim Abdelaziz, Julian Dolby, James P. McCusker, and Kavitha Srinivas. 2020. A Toolkit for Generating Code Knowledge Graphs. *ArXiv* (2020). https://arxiv.org/abs/2002.09440

[2] José P. Cambronero and Martin C. Rinard. 2019. AL: Autogenerating Supervised Learning Programs. In *Proceedings of the ACM on Programming Languages*, Vol. 3. https://doi.org/10.1145/3360601

[3] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder. *CoRR* abs/1803.11175 (2018). http://arxiv.org/abs/1803.11175

[4] Alex G. C. de Sá, Walter José G. S. Pinto, Luiz Otavio V. B. Oliveira, and Gisele L. Pappa. 2017. RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines. In *Genetic Programming*, James McDermott, Mauro Castelli, Lukas Sekanina, Evert Haasdijk, and Pablo García-Sánchez (Eds.). 246–261. https://doi.org/10.1007/978-3-319-55696-3_16

[5] Iddo Drori, Lu Liu, Yi Nian, Sharath C Koorathota, Jung-Shian Li, Antonio Khalil Moretti, Juliana Freire, and Madeleine Udell. 2019. AutoML using Metadata Language Embeddings. *ArXiv* (2019). https://arxiv.org/abs/1910.03698

[6] Robert Engels and Christiane Theusinger. 1998. Using a Data Metric for Preprocessing Advice for Data Mining Applications. In *In Proceedings of the European Conference on Artificial Intelligence (ECAI)*. 430–434. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.7414&rep=rep1&type=pdf

[7] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *ArXiv* (2020). https://arxiv.org/abs/2003.06505

[8] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-Sklearn 2.0: The Next Generation. *arXiv* (2020). https://arxiv.org/abs/2007.04074

[9] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 2962–2970. https://dl.acm.org/doi/10.5555/2969442.2969547

[10] Nicolo Fusi, Rishit Sheth, and Melih Elibol. 2018. Probabilistic Matrix Factorization for Automated Machine Learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 3352–3361. https://dl.acm.org/doi/10.5555/3327144.3327254

[11] P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. 2019. An Open Source AutoML Benchmark. In *AutoML Workshop at the International Conference on Machine Learning (ICML)*. https://arxiv.org/abs/1907.00909

[12] Isabelle Guyon, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Núria Macià, Bisakha Ray, Lukasz Romaszko, Michèle Sebag, Alexander Statnikov, Sébastien Treguer, and Evelyne Viegas. 2016. A brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning without Human Intervention. In *Proceedings of Machine Learning Research*, Vol. 64. 21–30. https://proceedings.mlr.press/v64/guyon_review_2016.html

[13] Mossad Helali, Essam Mansour, Ibrahim Abdelaziz, Julian Dolby, and Kavitha Srinivas. 2022. A Scalable AutoML Approach Based on Graph Neural Networks. *ArXiv* (2022). https://arxiv.org/abs/2111.00083

[14] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[15] Michael Katz, Parikshit Ram, Shirin Sohrabi, and Octavian Udrea. 2020. Exploring Context-Free Languages via Planning: The Case for Automating Machine Learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 403–411. https://ojs.aaai.org//index.php/ICAPS/article/view/6686

[16] Trang T Le, Weixuan Fu, and Jason H Moore. 2020. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics* 36, 1 (2020), 250–256. https://doi.org/10.1093/bioinformatics/btz470

[17] Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable Automatic Machine Learning. In *AutoML Workshop at the International Conference on Machine Learning (ICML)*. www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

[18] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Yaliang Li, Bolin Ding, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, and Bin Cui. 2021. VolcanoML: Speeding up End-to-End AutoML via Scalable Search Space Decomposition. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2167–2176. http://www.vldb.org/pvldb/vol14/p2167-li.pdf

[19] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning Deep Generative Models of Graphs. *ArXiv* (2018). https://arxiv.org/abs/1803.03324

[20] Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. 2020. An ADMM Based Framework for AutoML Pipeline Configuration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4892–4899. https://doi.org/10.1609/aaai.v34i04.5926

[21] Radu Marinescu, Akihiro Kishimoto, Parikshit Ram, Ambrish Rawat, Martin Wistuba, Paulito P. Palmes, and Adi Botea. 2021. Searching for Machine Learning Pipelines Using a Context-Free Grammar. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 8902–8911. https://ojs.aaai.org/index.php/AAAI/article/view/17077

[22] Jonas Mueller and Alex Smola. 2019. Recognizing Variables from Their Data via Deep Embeddings of Distributions. *International Conference on Data Mining (ICDM)* (2019), 1264–1269. https://doi.org/10.1109/ICDM.2019.00158

[23] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining* 10, 1 (2017), 36. https://doi.org/10.1186/s13040-017-0154-

[24] Bernhard Pfahringer, Hilan Bensusan, and Christophe Giraud-Carrier. 2000. Meta-Learning by Landmarking Various Learning Algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*. 743–750. https://dl.acm.org/doi/10.5555/645529.658105

[25] Herilalaina Rakotoarison, Marc Schoenauer, and Michèle Sebag. 2019. Automated Machine Learning with Monte-Carlo Tree Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 3296–3303. https://doi.org/10.24963/ijcai.2019/457

[26] Matthias Reif, Faisal Shafait, and Andreas Dengel. 2012. Meta-learning for evolutionary parameter optimization of classifiers. *Machine Learning* 87, 3 (2012), 357–380. https://doi.org/10.1007/s10994-012-5286-7

[27] Kevin Swersky, Jasper Snoek, and Ryan P. Adams. 2013. Multi-Task Bayesian Optimization. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 2004–2012. https://dl.acm.org/doi/10.5555/2999792.2999836

[28] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined election and hyperparameter optimization of classification algorithms. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 847–855. https://doi.org/10.1145/2487575.2487629

[29] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15 (2014), 49–60. https://doi.org/10.1145/2641190.2641198

[30] Ricardo Vilalta, Christophe Giraud-carrier, Pavel Brazdil, and Carlos Soares. 2004. Using Meta-Learning to Support Data Mining. *International Journal of Computer Science & Applications* 1 (2004). https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.1351&rep=rep1&type=pdf

[31] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: A Fast and Lightweight AutoML Library. In *Proceedings of Machine Learning and Systems (MLSys)*, Vol. 3. 434–447. https://proceedings.mlsys.org/paper/2021/file/92cc227532d17e56e07902b254dfad10-Paper.pdf

[32] Marcel Wever, Felix Mohr, and Eyke Hüllermeier. 2018. ML-Plan for Unlimited-Length Machine Learning Pipelines. In *AutoML Workshop at the International Conference on Machine Learning (ICML)*. https://ris.uni-paderborn.de/download/3852/3853/38.pdf

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

[34] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. 2019. OBOE: Collaborative Filtering for AutoML Model Selection. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. http://doi.org/10.1145/3292500.3330909