Database Native Model Selection: Harnessing Deep Neural Networks in Database Systems

Naili Xing
National University of Singapore
xingnl@comp.nus.edu.sg

Shaofeng Cai
National University of Singapore
shaofeng@comp.nus.edu.sg

Gang Chen
Zhejiang University
cg@zju.edu.cn

Zhaojing Luo
National University of Singapore
zhaojing@comp.nus.edu.sg

Beng Chin Ooi
National University of Singapore
ooibc@comp.nus.edu.sg

Jian Pei
Duke University
j.pei@duke.edu

ABSTRACT

The growing demand for advanced analytics beyond statistical aggregation calls for database systems that support effective model selection of deep neural networks (DNNs). However, existing model selection strategies are based on either training-based algorithms that deliver high-performing models at the expense of high computational cost, or training-free algorithms that enhance computational efficiency with reduced effectiveness. These strategies often disregard computational cost and response time Service-Level Objectives (SLOs), which are of concern to average or budget-conscious machine learning users. In addition, they lack a well-designed integration of the model selection algorithms with DBMSs, which hinders efficient in-database model selection. This paper presents TRAILS, a resource-efficient and SLO-aware in-database model selection system. To leverage the strengths of both training-free and training-based model selection, we first characterize nine state-of-the-art training-free model evaluation metrics and propose a more effective one named JacFlow, and then, restructure the conventional model selection procedure into two phases: filtering and refinement. A novel coordinator is also introduced to strike a balance between the high efficiency of train-free algorithms and the high effectiveness of training-based algorithms, ensuring high-performing model selection while adhering to target SLOs. Moreover, we incorporate the proposed algorithm into PostgreSQL to develop TRAILS, thereby both enhancing resource efficiency and reducing model selection latency. This integration establishes a foundation for declarative model definition and selection within DBMSs. Empirical results demonstrate that our TRAILS reduces model selection time and computational expenses considerably by up to 24.38x and 29.32x respectively compared to existing model selection systems.

PVLDB Reference Format:
Naili Xing, Shaofeng Cai, Gang Chen, Zhaojing Luo, Beng Chin Ooi, Jian Pei. Database Native Model Selection: Harnessing Deep Neural Networks in Database Systems. PVLDB, 17(5): 1020 - 1033, 2024.
doi:10.14778/3641204.3641212

Proceedings of the VLDB Endowment, Vol. 17, No. 5 ISSN 2150-8097.
doi:10.14778/3641204.3641212

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nusdbsystem/Trails.

1 INTRODUCTION

Database Management Systems (DBMSs) store and manage data for business daily operations and support analytical processes through query execution [6, 19, 26, 50, 54]. In-database analytical processes offer many advantages, such as reducing data management overhead in separate systems and ensuring data provenance and security [22, 60]. Machine Learning (ML) has shown great potential in data analytics, exhibiting superior performance compared to traditional statistical aggregation methods [12, 13, 27, 37, 39, 43, 58]. Therefore, not surprisingly, recent works [16, 23, 28, 33, 38, 47] have proposed to integrate machine learning into DBMSs for complex analytics via declarative languages such as SQL.

With the continuing trend of devising new Deep Neural Network (DNN) models in the research community [1–3, 24, 68], automatic selection of a well-performing model for a specific analytics task becomes increasingly imperative. This motivates the integration of model selection into DBMSs as an in-database analytical process, such that the model selection can be effectively supported by Model Selection Queries (MSQs) within DBMSs.

Unlike hyper-parameter optimization (HPO), which focuses on finding the optimal hyper-parameters for a given model, e.g., learning rate, batch size, and activation functions [55, 61], model selection is dedicated to searching for the best-performing model with the optimal network topology in a search space of candidate models. Typically, the network topology of a DNN model can be represented as a Directed Acyclic Graph (DAG), where nodes and edges represent certain computation operations. A search space of network topologies with multiple layers of nodes and connections can comprise up to 10^{18} different candidate models [57, 62]. Fully evaluating these models to select the best one is computationally prohibitive in both time and resources [40]. Therefore, both the effectiveness and efficiency of the search process are critical in real-world model selection, where a strict response-time threshold is typically predefined by users and must be met by the model selection task. Such requirements are often specified as response-time Service-level Objectives (SLOs) [7, 66]. For instance, a user may submit a model selection query for CTR-Prediction on the Criteo dataset and expect to receive a higher-performing model within a two-hour response-time threshold, as illustrated in Figure 1. Failing to meet the SLOs can lead to reduced query service quality or even

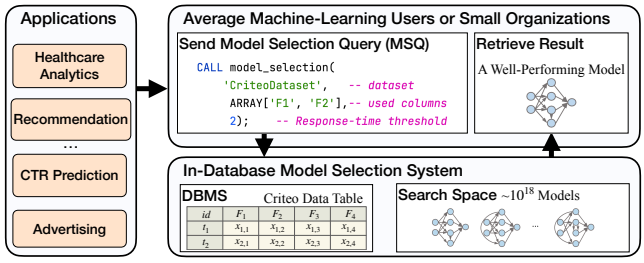


Figure 1: An in-database model selection example.

financial losses [66]. Therefore, an in-database model selection system must execute MSQs in an SLO-aware manner and minimize resource consumption, particularly in terms of GPU and memory.

Achieving SLO-aware model selection that can consistently produce a well-performing model within any given response-time threshold presents great difficulty. Training-based model selection algorithms can effectively measure the model performance [42, 68], but they typically require training and evaluating hundreds to thousands of candidate models, which are computationally heavy and cannot satisfy the SLO when the time budget is limited. To improve efficiency, recent works on training-free model selection algorithms [5, 49, 56] seek to estimate model performance without training by computing certain statistics of candidate models, namely **T**raining-**F**ree **M**odel **E**valuation **M**etrics (TFMEMs). Leveraging TFMEMs enables efficient evaluation of a large number of models. However, relying solely on TFMEMs restricts the potential for finding higher-performing models compared to more accurate training-based algorithms within a given response-time threshold constraint. Indeed, a model selection algorithm that is both effective and efficient, and can adhere to SLOs consistently is still lacking.

Additionally, efficiently integrating the model selection algorithm into a database to reduce data retrieval and preprocessing overhead, enhance scalability, and reduce memory consumption presents further challenges from a system perspective. One primary challenge is how to coordinate data management within a DBMS with the execution of model selection algorithms. Conventional approaches typically support model selection using two separate systems [17, 53] as illustrated in Figure 2, which requires transferring the entire data from a DBMS to a separate model selection system. This process is inefficient and prone to errors and breaches of privacy and security [19, 60]. Further, loading data into a separate system results in considerable memory usage. Recently, there have been preliminary attempts to directly integrate model selection algorithms into DBMSs via User Defined Functions (UDFs) [19, 60]. However, such integration complicates the system implementation and misses the opportunities to scale out the model selection algorithms with more computational resources, e.g., leveraging dedicated hardware such as GPUs to accelerate model training [32].

To address the above challenges, we build an SLO-aware and resource-efficient in-database model selection system. First, we propose an SLO-aware and GPU-resource-conserving two-phase model selection algorithm that harnesses the advantages of both training-free and training-based paradigms. Specifically, we benchmark nine state-of-the-art TFMEMs regarding their theoretical characterization of either the trainability or expressivity of the models. Second, we combine the best performing TFMEMs by a learned weighted average of each property to enjoy the benefits of both and obtain a more effective TFMEM termed as *JacFlow*. Lastly, we restructure the model selection into two separate phases, the filtering and refinement phases, to leverage the efficiency of training-free and the effectiveness of training-based model selection algorithms. The filtering phase is designed to quickly explore a large set of candidate models and approximately derive a set of promising models based on JacFlow. The refinement phase seeks to accurately pick the best-performing model from the promising model set through slightly more expensive training-based model evaluation. To balance the two phases and achieve SLO awareness, we investigate the inherent trade-offs in their interaction and introduce a coordinator to deliver higher-performing model selection while adhering to the target response-time threshold.

We integrate the proposed two-phase model selection algorithm into PostgreSQL non-intrusively via stored procedure and build fil**T**e**R**ing **A**nd ref**I**nement in-database mode**L** **S**election system (TRAILS), which is designed to improve efficiency, enhance scalability, and reduce memory consumption. Particularly, we execute the computationally light but I/O intensive model filtering on the CPUs and within the DBMS via UDFs to minimize data operation overhead. For the computationally intensive refinement phase, we execute it through an external execution engine separate from the DBMS to enhance scalability. Additionally, we design a data cache service to facilitate on-the-fly data transformation between the DBMS and this execution engine. The cache service enables pipeline parallelism between batch data operations and model training, thus eliminating the waiting time associated with retrieving and preprocessing data before training and reducing overall memory usage by avoiding loading the entire dataset into memory.

In summary, we make the following contributions:

- We characterize nine state-of-the-art TFMEMs and comprehensively examine their effectiveness. Based on the analysis, we propose a more effective TFMEM termed JacFlow to estimate model performance efficiently and effectively.
- We propose an SLO-aware and resource-efficient two-phase model selection algorithm with a novel coordinator to leverage the strengths of both efficient training-free and effective training-based model selection.
- We build an end-to-end in-database model selection system called TRAILS by non-intrusively integrating the two-phase model selection algorithm on top of PostgreSQL with optimization on scalability, efficiency, and memory consumption. The TRAILS also functions as a component of NeuRDB to select well-performing DNN models for downstream tasks, such as model inference.
- We construct an extensive benchmark dataset with 160,000 candidate models and their detailed training and evaluation statistics for benchmarking model selection algorithms on structure data. Experiments show that TRAILS significantly reduces the model selection time and the computational cost by up to 24.38x and 29.32x respectively, and supports SLO-aware model selection efficiently.

The remainder of this paper is organized as follows: Section 2 presents the preliminaries, followed by the characterization and analysis of TFMEMs in Section 3 and the two-phase model selection algorithm in Section 4. We discuss the integration of the proposed algorithm on DBMS in Section 5. Experimental results are provided in Section 6, and related work is summarized in Section 7. Finally, we conclude the paper in Section 8.

2 PRELIMINARIES

In this section, we introduce the background, problem definition, and existing model selection systems, followed by two key techniques that are central to our in-database model selection system, namely training-free model evaluation and fixed-budget algorithms.

2.1 Background and Problem Definition

Model Selection aims to automatically search for the optimal neural network topology from a vast search space with the highest performance, measured in certain metrics such as accuracy or AUC, on a given dataset, which typically involves three key components: search space, search strategy, and model evaluation.

Search space, denoted as M, refers to a collection of possible models m, i.e., $M = \{m\}$, each with a unique topology [18, 65]. Its design defines the solution space, influencing the final model's quality and problem complexity.

Search strategy iteratively samples candidate models from the search space for model evaluation, as denoted by $m = f_s(M, S_i)$, where S_i represents the state of the current search strategy at iteration i. It aims to explore the search space effectively by incorporating knowledge from previous iterations [9, 53], such as the model and its performance p (AUC or accuracy), to update S accordingly.

Model evaluation involves assessing the performance of models on the validation dataset, i.e., $p_m = E(m, D_{valid})$, where $E(\cdot,\cdot)$ is the model evaluation function and can be either training-based or training-free, and D_{valid} represents the validation dataset.

Problem Definition. The objective of *in-database model selection* is to perform model selection in a DBMS via query execution, i.e., Model Selection Queries (MSQs). To achieve SLO awareness and resource efficiency, the MSQ must be completed within a user-specified response-time threshold T_{max}, i.e., $T_{MSQ} \leq T_{max}$, and conform to the resource constraints R_{max}. The overall objective functions can be formulated as follows:

$$\begin{aligned} &\text{Maximize:} && p_m = E(m, D_{\text{valid}}) \\ &\text{Subject to:} && T_{MSQ} \leq T_{max} \\ & && \text{Resource}(MSQ(M, D, T_{max})) \leq R_{max} \end{aligned}$$

Traditional Model Selection Systems in DBMSs are mostly based on the full model training [42, 53, 67], and can be structured into four stages: data retrieval, data preprocessing, model sampling, and training-based model evaluation, as illustrated in Figure 2. Data retrieval typically exports the entire data table to external storage, often as a CSV file or multiple partitioned files. Upon model selection initiation, the entire data is loaded once into memory for preprocessing. The search strategy then samples models from the search space. Evaluation workers evaluate these models which require extensive full training of hundreds to thousands of iterations on the preprocessed data. The evaluation results then guide the subsequent model sampling.

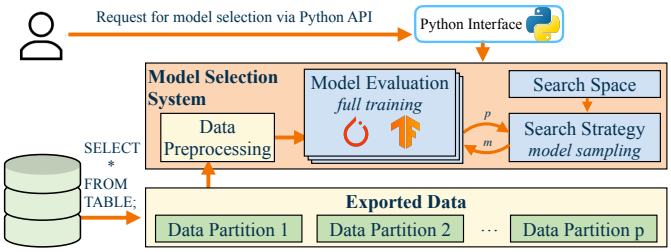


Figure 2: Data access pattern in traditional model selection.

Traditional model selection systems require data transfer from a DBMS to external storage and preprocessing of the entire dataset before training, causing a long waiting time for training. Also, it requires a substantial amount of memory. Since they are based on full-model training, they are unaware of the T_{max} and SLO requirements. All these limitations call for the design of an in-database resource-efficient and SLO-aware model selection system.

2.2 Training-Free Model Evaluation

In principle, training-free model evaluation techniques estimate model performance by calculating certain statistics of the model using a single forward or backward computation on a single mini-batch of data without full training. Given a model m with parameters a θ, a TFMEM [5, 49, 56] computes a score s_a to quantitatively estimate the model performance p. A larger absolute value of s_a typically indicates better model performance. A TFMEM can be formally defined as:

$$s_a = E(m(\theta), \mathcal{B}_{data}) \tag{1}$$

where E denotes the training-free model evaluation function of the a TFMEM, and $\mathcal{B}_{data}$ refers to a batch of data. The main advantage of TFMEMs is the extremely high computational efficiency, which requires computation on only one batch of data and thus enables the exploration of a wider range of candidate models. Different TFMEMs are designed considering various properties, including the evaluation metrics, computational complexity, methods of computation, and access to the data and labels (data/label agnostic). We summarize these properties in Table 1 and present more comprehensive benchmarking and discussion in Section 3.

2.3 Fixed-Budget Algorithm

Fixed-budget algorithms aim to balance exploration and exploitation by allocating resources judiciously across models. These algorithms [8, 21, 25] seek to find the best-performing model without exceeding the overall budget constraint, such as the T_{max}. Notable algorithms include Uniform Allocation (UNIFORM)[25], Successive Rejects (SUCCREJCT)[8], and Successive Halving (SUCCHALF) [25]. SUCCHALF commences by apportioning an equal, minimal time budget for training each model. Following each round, a fraction of the top-performing models (typically half) are retained, and the time budget for the next training round is doubled. This procedure

Table 1: A Comparison of Different Training-Free Model Evaluation Metrics (TFMEMs).

TFMEM	Evaluation Metric	Complexity	Computation	Data/Label Agnostic	Characterization
GradNorm [5]	Frobenius norm	1F+1B	$s_a = \|\frac{\partial L}{\partial \theta}\|_F$	Not	Trainability
NASWOT [41]	Hamming distance	1F	$s_a = log\|K_H\|$	Label	Expressivity
NTKCond [15]	Neural tangent kernel	1F+1B	$s_a = \frac{\lambda_{max}(\Theta)}{\lambda_{min}(\Theta)}$	Not	Trainability
NTKTrace [49]	Neural tangent kernel	1F+1B	$s_a = \|\Theta\|_{trace}$	Not	Trainability
NTKTraceAppx [48]	Neural tangent kernel	1F+1B	$s_a = \|\Theta_{appx}\|_{trace}$	Not	Trainability
Fisher [5]	Hadamard product	1F+1B	$s_a = \sum_{l=1}^{L}(\frac{\partial \mathcal{L}}{\partial ac_l} ac_l)^2$	Not	Trainability
GraSP [52]	Hessian vector product	1F+1B	$s_a = \sum -(H\frac{\partial \mathcal{L}}{\partial \theta}) \bigodot \theta$	Not	Trainability
SNIP [31]	Hadamard product	1F+1B	$s_a = \sum \|\frac{\partial \mathcal{L}}{\partial \theta} \bigodot \theta\|$	Not	Trainability
SynFlow [51]	Hadamard product	1F+1B	$s_a = \sum \frac{\partial \mathcal{L}}{\partial \theta} \bigodot \theta$	Data/Label	Trainability

$\mathcal{L}$: loss function. θ: model parameters. Θ: Neural Tangent Kernel (NTK) matrix of the model. $\bigodot$: Hadamard product. s_a: the score of a model. λ: the eigenvalue of NTK matrix. H: Hessian vector. L: number of model layers. $\|.\|_F$: Frobenius norm. ac_l: activation saliency of one layer. F: forward computation. B: backward computation.

is reiterated until a single model prevails. Similarly, SUCCREJCT initially allocates an equal, small time budget to each model. After each evaluation round, the least-performing models are discarded, and their time budget is redistributed among the remaining models. In contrast, UNIFORM distributes the budget equitably among all models, where each candidate model undergoes training and evaluation using the same fixed-time budget.

3 CHARACTERIZATION OF TFMEMs

3.1 Comparison of TFMEMs

We present the comparison of nine TFMEMs in Table 1. Each TFMEM is classified based on its ability to capture one of two key theoretical properties of the models under evaluation: *trainability* and *expressivity*, as summarized in the last column in Table 1. Trainability measures the extent to which the model can be effectively trained via gradient descent. It has been explored in the context of network pruning [31, 51, 52]. They primarily focus on identifying highly trainable subnetworks by evaluating the importance of individual parameters within the DNN, followed by pruning less important connections. A concept known as *synaptic saliency* [31, 51] is proposed to quantify the importance of each parameter, formally defined as follows:

$$\Phi(\theta) = f(\frac{\partial \mathcal{L}}{\partial \theta}) \bigodot g(\theta) \tag{2}$$

where $\mathcal{L}$ represents the loss function, and $\bigodot$ denotes the Hadamard product. Approaches in this category such as SNIP [31], GraSP [52], and SynFlow [51] mainly differ in how $f(\cdot)$ and $g(\cdot)$ are defined: e.g., $\Phi(\theta) = |\frac{\partial \mathcal{L}}{\partial \theta}| \bigodot |\theta|$ in SNIP; $\Phi(\theta) = -(H\frac{\partial \mathcal{L}}{\partial \theta}) \bigodot \theta$ in GraSP; and $\Phi(\theta) = \frac{\partial \mathcal{L}}{\partial \theta} \bigodot \theta$ in SynFlow. [5] evaluates the architecture performance by aggregating the synaptic saliency of all parameters in a DNN: $s_a = \sum \Phi(\theta)$. Here H is the Hessian vector, and $\sum$ represents the sum over all elements in the $\Phi(\theta)$.

Expressivity measures the complexity of the functions that the model can represent. TE-NAS [15] evaluates it by calculating the number of linear regions the model can divide. NASWOT [41] evaluates it by measuring the distance between the activation pattern vectors generated by the model for any sample pair in a mini-batch of data. A larger distance indicates a strong capacity to distinguish between different samples. Formally, the NASWOT is calculated as $log|K_H|$, where K_H is an $N \times N$ kernel matrix with $K_{H_{(i,j)}} = N_a - d_H(c_i, c_j)$, N_a is the number of rectified linear units in the model and d_H is the Hamming distance between two samples' activation pattern vectors c_i and c_j.

3.2 Benchmarking of TFMEMs

To benchmark the effectiveness of different TFMEMs, we quantitatively measure the *evaluation correlation* between the score computed by them and the actual performance of models across different search spaces and datasets. The results are shown in Table 2. A TFMEM with consistent signs across different search spaces and datasets and large absolute correlation values is regarded as more effective (marked as bold in Table 2). First, expressivity-focused TFMEMs, e.g., NASWOT, shows a consistent positive correlation across all experimental settings. However, it disregards the synaptic saliency information associated with each parameter, resulting in suboptimal correlation values compared to the trainability-focused TFMEMs such as SNIP and SynFlow on the Criteo and Frappe datasets. Second, while the trainability-focused TFMEMs can achieve a higher positive correlation, they involve more intricate computations, which can introduce inconsistencies of signs when applied across different search spaces, such as NTKTrAppx and GraSP. In summary, NASWOT and SynFlow demonstrate superior effectiveness, as indicated by consistently positive and higher absolute SRCC values across different search spaces and datasets.

3.3 JacFlow TFMEM

To derive a more effective TFMEMs for our system, we propose to combine the strengths of both expressivity-based and trainability-based TFMEMs for an enhanced TFMEM termed *JacFlow*:

$$s_a = w_1 \sum \frac{\partial \mathcal{L}}{\partial \theta} \bigodot \theta + w_2 log|K_H| \tag{3}$$

where we combine the best-performing TFMEMs of respective properties, i.e., SynFlow and NASWOT, into JacFlow by calculating a weighted average of their scores. The weights w_1 and w_2 are learned via regression on the actual performance of the models [29].

Table 2: Spearman Rank Correlation Coefficient (SRCC) of TFMEMs measured on six datasets across three search spaces.

TFMEM	NB101+C10	NB201+C10	NB201+C100	NB201+IN16	DNN+Frappe	DNN+Diabete	DNN+Criteo	Average Rank
GradNorm	-0.34	0.64	0.64	0.57	0.45	0.39	0.32	N/A
NASWOT	0.37	0.79	0.80	0.78	0.61	0.63	0.69	2.71
NTKCond	-0.28	-0.48	-0.39	-0.41	**-0.77**	-0.56	-0.66	3.71
NTKTrace	-0.42	0.37	0.38	0.31	0.54	0.37	0.46	N/A
NTKTrAppx	-0.53	0.34	0.38	0.36	0.13	0.31	0.01	N/A
Fisher	-0.37	0.38	0.38	0.32	0.48	0.21	0.41	N/A
GraSP	0.14	0.53	0.54	0.52	-0.27	-0.23	-0.18	N/A
SNIP	-0.27	0.64	0.63	0.57	0.68	0.62	0.78	N/A
SynFlow	0.39	0.78	0.76	0.75	**0.77**	0.68	0.74	2.57
JacFlow	**0.42**	**0.83**	**0.83**	**0.81**	**0.77**	**0.69**	**0.75**	**1.00**

In the last column of Table 2, we calculate the average rank of only TFMEMs with consistent signs across six datasets and three search spaces, i.e., JacFlow, SynFlow, NASWOT, and NTKCond. The results show JacFlow clearly outperforming other TFMEMs with an average rank of 1.00. This suggests that the proposed combination strategy is simple yet effective in augmenting model evaluation by using complementary TFMEMs for improving the effectiveness of fast model evaluation.

4 TWO-PHASE MODEL SELECTION ALGORITHM

With the computationally efficient JacFlow, a straightforward approach of model selection is to randomly score a large number of models based on Equation 3 and then select the model with the highest score. However, such a random sampling approach is not efficient as compared to more advanced search strategies [45]. Moreover, scores estimated by JacFlow only indicate the estimated performance of models as discussed in Section 2.2 and supported by Table 2 (SRCC is less than 1). Therefore, we propose a two-phase model selection algorithm to enhance the effectiveness of our model selection algorithm. First, we introduce a filtering phase, which employs advanced search strategies to efficiently explore promising models with higher scores in the search space. Second, instead of simply returning the highest-scored model, we rank the models by their scores and retain only a small subset of promising models. Then, we introduce a refinement phase, which employs training-based model evaluation to more effectively determine the optimal model from this small subset. Lastly, to coordinate the two phases and achieve SLO-aware model selection, we design a coordinator to jointly optimize the two phases via an objective function that maximizes the performance of the selected model given T_{max}.

4.1 Filtering Phase - Efficient Exploration

Filtering Phase Algorithm. We adopt the Asynchronous Regularized Evolution Algorithm (AREA) [35] in the filtering phase, which supports efficient search space exploration [18, 35]. AREA maintains a diverse population, and then iteratively samples, mutates, and evaluates them. Better-performing models have a higher probability of selection and producing offspring. Specifically, as outlined in Algorithm 1, for each evaluation circle, a worker first acquires the model encoding (also named genotype) as shown in Figure 3 from the search strategy, constructs the model (lines 14-15), and then scores it using JacFlow and sends the score back to the shared message queue (lines 16-17). The search strategy maintains a pool of model encodings and the corresponding scores. The model pool is initialized randomly at the initialization of the search strategy. Once the search strategy receives the request from workers, it employs AREA to read a small set of model encodings from the local pool, and then mutates the encoding of the highest score and returns it to the worker (lines 5-7). Simultaneously, another thread keeps reading from the shared message queue and updating the current model pool (lines 10-11). Finally, the search strategy keeps the top K out of the N scored models for the refinement phase.

Discussion. The proposed filtering strategy offers several key advantages. It achieves high efficiency by utilizing AREA for fast exploration and adopting the JacFlow metric for training-free model evaluation. Also, the exploration process is readily schedulable. Given that scoring each model is performed using a fixed-size batch of data and takes only a single forward and backward computation, the time required to evaluate a model is relatively constant and can be measured via the profiling stage as demonstrated in Figure 3. This predictability allows for easy scheduling based on a given time budget. In conclusion, TRAILS can efficiently and effectively explore the large search space and derive a small set of promising models for further training-based evaluation.

4.2 Refinement Phase - Effective Exploitation

The refinement phase is introduced to employ a training-based model evaluation to accurately identify the best-performing model from the K models returned by the filtering phase. Given a user-defined T_{max}, training all K models till convergence suffers from heavy consumption of both time and GPU resources, and may violate SLOs. The challenge here is how to effectively identify the optimal model from the K models within a fixed time budget.

Refinement Phase Algorithm. We first experimentally analyze three widely adopted fixed-budget optimization algorithms for the refinement phase [10]: Uniform Allocation (UNIFORM) [25], Successive Rejects (SUCCREJCT) [8] and Successive Halving (SUCCHALF) [25]. The results summarized in Figure 13 demonstrate that SUCCHALF consistently achieves high model performance given a time budget. We thus adopt SUCCHALF in two-phase model selection algorithm. The refinement phase is outlined in Algorithm 2: the budget controller allocates the initial training epochs U_{init} to train and evaluate the performance of each of the K models (lines 2-4), keeps only $\frac{1}{\eta}$ models (line 5) after the first round, and increases the U_{init} to $U_{init} \cdot \eta$ such that each kept model is trained with

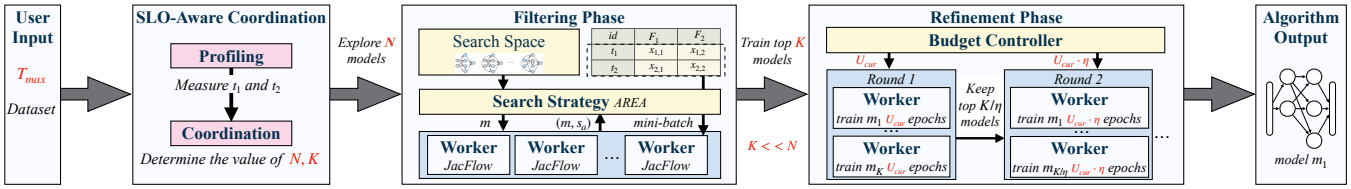


Figure 3: Two-Phase model selection algorithm, which consists of coordinator, filtering, and refinement phases.

Algorithm 1: Filtering Phase - Efficient Exploration

Input : M: Search space. T_{max}: Time budget. Q_m: Model queue. Q_s: Model score pair queue. $\mathcal{B}_{data}$: Mini-batch. n: Initial model pool size.
Output: K models.

1 **Async Aging Evolution** (M, T_{max})
2 $Pool \leftarrow$ RandomGenotype(n)
3 **while** *Receive worker's request /* Thread-1 */*
4 **do**
5 genotype $\leftarrow$ GeneticSelection($Pool$)
6 genotype' $\leftarrow$ Mutate(genotype)
7 $Q_m \leftarrow Q_m \bigcup$ (genotype')
8 **while** *True /* Thread-2 */*
9 **do**
10 (genotype, s_a) $\leftarrow Q_s$.FetchScore()
11 UpdateLocalPool(genotype, s_a)
12 **Evaluation Worker** (D_B)
13 **while** *True* **do**
14 genotype' $\leftarrow Q_m$.FetchGenotype()
15 $m \leftarrow$ ModelConstruct(genotype')
16 $s_a \leftarrow$ Score(m, $\mathcal{B}_{data}$)
17 $Q_s \leftarrow Q_s \bigcup$ (genotype', s_a)

Algorithm 2: Refinement Phase - Effective Exploitation

Input : K: Top models. U_{init}: initial training epochs. η: $\frac{1}{\eta}$ of the models to keep per-round.
Output: The final selected model.

1 **Evaluation Worker** (K)
2 $U_{cur} \leftarrow U_{init}$
3 **while** $K.length() > 1$ **do**
4 $\{p\} \leftarrow \{E(m_i, U_{cur}); m_i \in K\}$
5 $K \leftarrow Top_{\frac{1}{\eta}}(\{p\})$ /* Only keep top $\frac{1}{\eta}$ models. */
6 $U_{cur} \leftarrow U_{cur} \cdot \eta$ /* Increase U_{cur} per-model. */

more budget in the next round for more accurate evaluation (line 6). SUCCHALF stops when only one model remains.

Discussion. The refinement phase is budget-aware, resource-efficient, and easy to scale. With the SUCCHALF algorithm, the refinement phase can adhere to the predefined time budgets. Additionally, it performs training-based evaluation in an epoch-wise manner and can direct more budgets toward more promising models, which thus avoids wasting resources in training unpromising models. Lastly, the independent training and evaluation of models in each evaluation round make the refinement phase easy to scale.

4.3 Coordinator

In practice, both phases are governed by predefined parameters, namely the number of models explored in the filtering phase (N), the number of promising models to retain for further refinement (K), and the initial training epochs required to train and evaluate each model during the refinement phase (U_{init}). Given the T_{max}, the coordinator is responsible for determining the values of these parameters to balance between exploring a diverse range of models and concentrating on the most promising models for further evaluation while meeting SLOs.

Formally, we denote the time required to score a model based on JacFlow using a single mini-batch of data as t_1 and the time to train a model for one epoch as t_2, both determined via profiling (Figure 3). The filtering phase time T_1 then equals the product of N models and t_1, i.e., $T_1 = N \cdot t_1$. The refinement phase employs SUCCHALF to train and retains only the top $1/\eta$ models at each round. Given K models, it spends $K \cdot U_{init} \cdot t_2$ evaluating all K models at the first round, each being trained for U_{init} epochs. Such process iterates for $\lfloor \log_\eta K \rfloor$ rounds until one model remains, with each round allocated an equal amount of time. i.e., $K \cdot U_{init} \cdot t_2$. Therefore, the total time for the refinement phase is $T_2 = K \cdot U_{init} \cdot t_2 \cdot \lfloor \log_\eta K \rfloor$. We can then define the objective function and constraints as follows:

$$\begin{aligned} \max \quad & p = MSQ(\text{filtering}(N, t_1), \text{refinement}(K, U_{init}, t_2, \eta)) \\ \text{s.t.} \quad & T_1 + T_2 \leq T_{max} \\ \text{where} \quad & T_2 = K \cdot U_{init} \cdot t_2 \cdot \lfloor \log_\eta K \rfloor; \\ & T_1 = t_1 \cdot N; \quad K < N \end{aligned} \tag{4}$$

The objective function aims to maximize the performance of the selected model and guarantee that the overall time usage $T_1 + T_2$ does not exceed the T_{max}. The primary challenge of achieving the object lies in striking a balance between N and K. On the one hand, exploring more models while neglecting the refinement phase (e.g., $K = 1$) is highly efficient but may result in reduced effectiveness of the model being selected. On the other hand, evaluating each explored model in training-based methods (e.g., $K = N$) leads to excessive time and GPU resource consumption when training more unpromising models in initial rounds of SUCCHALF. Additionally, once the T_2 is determined, another challenge arises in trading off between η and U_{init}, as more models will compete for the time budget provided by U_{init} in a SUCCHALF round with a smaller η.

To address these challenges, we conduct extensive experiments across four datasets to find a balance between the trade-offs. As

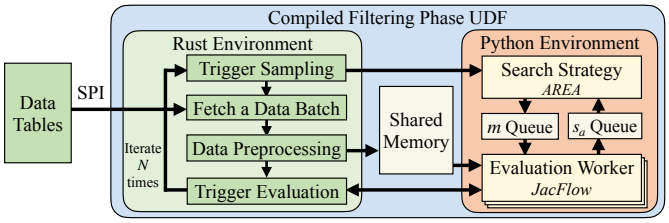


Figure 4: Filtering phase UDF design and implementation.

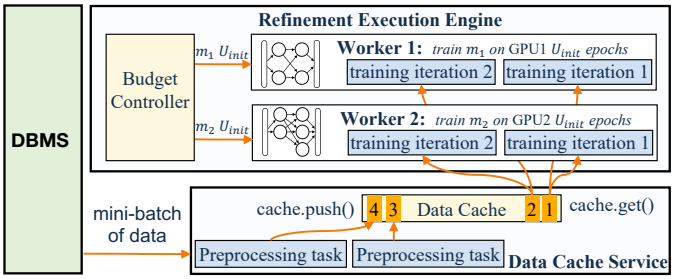


Figure 5: Pipeline parallelism between model training and data preprocessing in refinement phase through caching.

shown in Section 6.6, we have two key observations: (1) a small U_{init}, e.g., U_{init}=1 epoch, is sufficient to differentiate the performance of the K models; (2) $r = N/K \approx 100$ yields a good-performing final model across different T_{max}. Accordingly, the coordinator can thus set the value of N and K according to Equation 4.

Discussion. Our coordinator jointly optimizes the two phases, which maximizes the effectiveness of the model selection and meanwhile guarantees adherence to the T_{max}. Both t_1 and t_2 are dynamically obtained for each dataset by an on-the-fly profiling mechanism, as illustrated in Figure 3, and thus the coordinator can readily adapt to the given prediction task. Furthermore, the parameters r and U_{init} are also empirically demonstrated to be general across datasets as discussed in Section 6.6 and shown in Figure 14.

5 IN-DATABASE MODEL SELECTION

In our two-phase model selection algorithm, only a small subset of promising models need to be trained, which is thus efficient in both time and GPU resource usage. We incorporate the two-phase algorithm into DBMSs to further reduce memory consumption and data operation overhead, i.e., data retrieval and preprocessing, and enhance the scalability. Specifically, we adopt PostgreSQL [4] and PolarDB [14] in our implementation. We note that other databases such as MySQL can also be readily adopted.

5.1 Efficient Integration to DBMSs

Resource-Efficient Runtime Placement. As introduced in Section 2.2, JacFlow requires only one forward and one backward computation on a single mini-batch of data. The scoring of each model involves four distinct stages: initializing the model parameters θ, optionally transferring the model parameters to specific hardware, calculating a score following Equation 3; and finally, releasing the memory allocated to the model. The filtering phase typically involves exploring thousands of models. Although GPUs facilitate faster computation, they introduce significant overhead due to repeated transfers of model parameters to the GPUs and the subsequent release of resources after computation. Hence, the benefits of GPUs, mainly optimized for highly parallelizable tasks, become less pronounced, as shown in Figure 9a. Considering the similar latencies in filtering phase performances on both CPUs and GPUs as well as the limited GPU availability, TRAILS therefore executes the filtering phase on CPUs. For the refinement phase, which involves training-based evaluation of promising models, TRAILS executes this phase on GPUs to accelerate the training process.

Efficient Data Retrieval and Preprocessing. During the filtering phase, TRAILS needs to obtain a batch of data, formatted as multi-dimensional tensors suitable for feeding directly into DNN models [44]. Likewise, the refinement phase requires the dataset to be represented as an iterative object capable of generating multi-dimensional tensors for the training process.

However, DBMSs typically store data in formats not inherently compatible with machine learning algorithms. For example, PostgreSQL stores data in data types such as characters or text, which can not be directly used for model scoring or training. Therefore, data preprocessing is required to convert the data retrieved from DBMSs into multi-dimensional tensors. Traditional approaches retrieve data from the database to external storage for preprocessing, which is often inefficient due to significant execution latency and memory consumption, as discussed in Section 2.1.

To minimize these overheads in the filtering phase, characterized by computationally light but I/O intensive model exploration, we implement the process within the DBMS using UDFs. This approach integrates both Rust and Python environments (Figure 4). Specifically, Rust invokes Python functions to sample new models, which are stored in the Python environment's model queue (m queue). Then, Rust retrieves data from the DBMS using the Server Programming Interface (SPI) [1]. After retrieval, data is preprocessed and transferred to shared memory. Finally, Rust activates a Python evaluation worker for model evaluation. This worker directly reads the preprocessed data from shared memory into a multi-dimensional tensor and retrieves the sampled model from the m queue. The worker calculates the JacFlow score, which is stored in the score queue (s_a queue). This iterative process is repeated N times to explore N models. This design offers several advantages. First, TRAILS utilizes UDF, which enables access to more efficient and lower-level data retrieval APIs provided by DBMSs. Second, TRAILS employs Rust for data preprocessing, which utilizes shared memory for more efficient data loading, as empirically demonstrated in Figure 11.

To facilitate data operation and memory usage in the refinement phase, we introduce a data cache service as shown in Figure 5. Specifically, the data cache service utilizes a predetermined cache size to effectively control memory utilization. Upon instantiation, the service adopts a pull-based mechanism, fetching batches of data from the DBMS for preprocessing. Then, the service responds to data retrieval requests from the refinement phase in a First-In-First-Out (FIFO) sequence. Notably, a background thread is employed to regularly check the availability of cache space and fetch batches

[1] https://www.postgresql.org/docs/current/spi.html

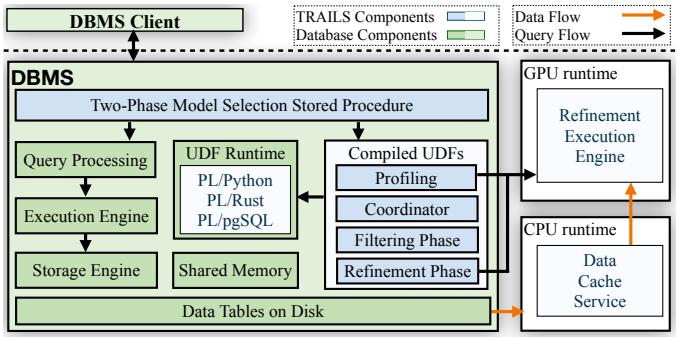


Figure 6: Overview of TRAILS System.

of data from the DBMS, which ensures a steady supply of data for model training. The data cache service facilitates pipeline parallelism, enabling concurrent data retrieval and preprocessing with the ongoing model training on preprocessed data batches. This mechanism significantly reduces the waiting time typically associated with data retrieval and preprocessing before model training. Consequently, this service effectively decreases both the overall model selection execution latency and memory usage, as validated empirically in Section 6.4.1.

High Scalability. The filtering and refinement phases are designed to be scalable and can operate in parallel with more evaluation workers. For the filtering phase, which operates on CPUs, we can employ more CPUs to support parallel computation of JacFlow scores for different models as illustrated in Figure 3. The refinement phase involves several evaluation rounds, each requiring training multiple models independently, which allows for parallelized model training when more GPUs are available as illustrated in Figure 5. Our experimental results shown in Figure 12 confirm the scalability of TRAILS with respect to computational resources in both phases.

5.2 System Overview

We present the core components of the system TRAILS in Figure 6, which consists of three types of components: refinement execution engine, data cache service, and a set of pre-compiled UDFs that cover the entire model selection pipeline. These components operate within three dedicated runtimes. The UDF runtime corresponds to the runtime environment provided by a DBMS server process, such as "PL/Rust"[2] for Rust-based UDFs, the GPU runtime refers to the execution environment on GPU hardware, and the CPU runtime represents an isolated process separate from the DBMS, dedicated explicitly to CPU-based computations.

To initiate the two-phase model selection procedure, users can submit a Model Selection Query (MSQ) via a DBMS client using the declarative syntax provided in the following template:

```
CALL model_selection(params);
```

where the params include dataset name, selected columns, and response-time threshold T_{max}. The two-phase model selection stored procedure involves spawning a data cache service on the CPU runtime and orchestrates the execution of all defined UDFs.

[2]https://plrust.io/plrust.html

Profiling UDF determines t_1 and t_2 for two phases as introduced in Section 4.3. It computes the JacFlow in the UDF runtime and triggers the model training in the refinement execution engine.
Coordinator UDF retrieves profiling results and generates runtime decisions, i.e., N and K, based on the Equation 4.
Refinement Phase UDF and Refinement Execution Engine. This UDF triggers the execution of the refinement phase within the refinement execution engine in the GPU runtime. The engine incorporates a budget controller and many training-based model evaluation workers as introduced in Section 4.2. When receiving requests from profiling or refinement UDF, it retrieves the necessary mini-batch of data from the data cache service and proceeds with the model training process.

6 EXPERIMENTS

In this section, we evaluate the effectiveness and efficiency of our proposed system TRAILS on six datasets. Furthermore, we analyze the scalability and components of TRAILS, detailing the algorithm choices for the refinement phase and coordinator design.

6.1 System Implementation

We have implemented TRAILS with 8.7K lines of code (LoC). Specifically, there are 6K LoC for implementing the three search spaces, ten TFMEMs, the coordinator, and the main searching process, 1.5K LoC for exhaustive experiments and analysis, and 1k LoC for building stored procedures and integrated with PostgreSQL 14.

6.2 Evaluation Setup

We conduct experiments on a cluster of three servers, each equipped with Intel(R) Xeon(R) Silver 4214R CPU (12 cores), 128 GB memory and 8 GeForce RTX 3090 GPUs. All servers are running on CUDA 11.0 and Ubuntu 20.04.2 LTS.

6.2.1 Datasets. We employ six datasets encompassing both structured and unstructured data across various real-world domains: the UCI Diabetes dataset (Diabetes)[3] for healthcare analytics, the Frappe dataset[4] for app recommendations, the Criteo dataset[5] for click-through rate prediction, CIFAR-10 (C10), CIFAR-100 (C100) [30] and ImageNet-16-120(IN16) [18] for image classification. A summary of the dataset statistics can be found in Table 3.

6.2.2 Search Space. We incorporate three search spaces into our study: NAS-Bench-101 (NB101) [65], NAS-Bench-201 (NB201) [18], and DNN search space. NB101 and NB201 are directly adopted from existing work for vision tasks, and we construct DNN search space for structured datasets, which is based on a Fully-connected Feed-forward Network comprising four hidden layers, each offering 20 size choices, resulting in 160,000 distinct and independent models.

6.2.3 Evaluation Metric. We employ the Spearman Rank Correlation Coefficient (SRCC) to evaluate the performance of TFMEMs as discussed in Section 3. To assess the final selected model's performance, we use AUC as the performance metric for structured data tasks and accuracy for vision tasks. To evaluate the efficiency of executing Model Selection Queries (MSQs) within TRAILS, we

[3]https://archive.ics.uci.edu/ml/datasets
[4]https://www.baltrunas.info/research-menu/frappe
[5]https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

Table 3: Dataset Statistics.

Dataset	Data Type	Classes	Samples	Fields/Resolution	Task	Adopted Search Space
Frappe	Structured Data	2	288,609	10	App Recommendation	DNN search space
Diabetes	Structured Data	2	101,766	43	Healthcare Analytics	DNN search space
Criteo	Structured Data	2	45,840,617	39	CTR Prediction	DNN search space
CIFAR-10 (C10)	Unstructured Data	10	60,000	32x32x3	Computer Vision	NB101, NB201
CIFAR-100 (C100)	Unstructured Data	100	60,000	32x32x3	Computer Vision	NB201
ImageNet (IN16)	Unstructured Data	120	151,700	16x16x3	Computer Vision	NB201

use query latency as the key performance indicator. Query latency measures the duration which is from a user submits an MSQ to when they receive the results. To evaluate GPU resource consumption, we record the duration of GPU usage throughout the entire model selection process, which is referred to as *GPU time*.

6.2.4 Baselines. We benchmark the effectiveness of our two-phase model selection algorithm (*2Phase-MS*) against three other categories: training-based, weight-sharing, and training-free.

Training-Based Model Selection (*Training-Based MS*) . We adopt AREA as the search strategy and fully train each model to evaluate its performance. For structured data, we use validation AUC after training for 14 epochs on Frappe, 1 epoch on UCI, and 10 epochs on Criteo. For unstructured data, we use test accuracy after 200 epochs of training in NB201 and 108 epochs in NB101.

Weight-Sharing-based Model Selection. We adopt DARTS [36] and ENAS [11] as baselines. Specifically, for unstructured data, we implement them in a modified supernet search space derived from NB201 [18]. We use five operations and four nodes, as compared to the original eight operations and seven nodes in the original DARTS search space. For structured data, since there is only one operation between each neuron connection in the DNN search space, and the DNN search space only includes 160,000 distinct independent models, DARTS and ENAS are not directly applicable.

Training-Free Model Selection. We use only the filtering phase of our two-phase algorithm and denote it as *Training-Free MS*. We also compare with training-free algorithms such as KNAS [59] and TE-NAS [15]. For DNN search space of structured data, since TE-NAS is based on a supernet where each connection has multiple operator options, it cannot be directly applicable.

We evaluate the efficiency of TRAILS against two baselines: the training-based model selection system depicted in Figure 2 and a decoupled variant of TRAILS, named TRAILS (Decoupled). This variant executes the two-phase model selection algorithm outside the PostgreSQL and without cache service. TRAILS (Decoupled) retrieves data in batches from PostgreSQL via psycopg[6] for scoring models during the filtering phase, and loading all data for model training during the refinement phase.

6.3 Effectiveness

6.3.1 Effective Combination of Training-free and Training-based Model Evaluation. To verify the effectiveness of different combinations of JacFlow and SUCCHALF, we construct nine unique combinations using three top-ranked TFMEMs: JacFlow, SynFlow, and SNIP based on the results in Table 2, with three training-based algorithms: SUCCREJECT, SUCCHALF, and full model training. We report the test accuracy of selected models and time usage after exploring N=10000 and training top K=100 models.

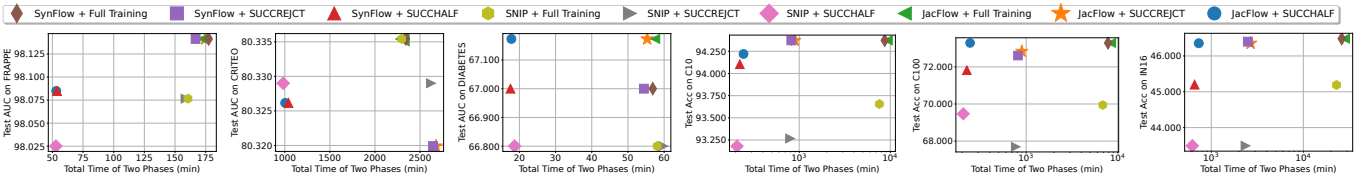
As shown in Figure 7, "JacFlow + SUCCHALF" consistently outperforms other combinations across all unstructured datasets by finding better-performing models with less time usage. For structured data, "JacFlow + SUCCHALF" is highly efficient and achieves comparable test accuracy with "SynFlow + SUCCHALF". The high effectiveness of the "JacFlow + SUCCHALF" is mainly due to the JacFlow in the filtering phase, which achieves a higher correlation value on unstructured datasets and a correlation comparable to SNIP and SynFlow on structured datasets as summarized in Table 2. Moreover, the high efficiency of 'JacFlow + SUCCHALF' can be attributed to the SUCCHALF in the refinement phase, which outperforms both SUCCREJECT and full model training as observed and discussed in Section 6.6. Therefore, we employ the JacFlow and SUCCHALF in our 2Phase-MS.

6.3.2 SLO-Aware Coordination. To assess the SLO awareness of 2Phase-MS, we vary T_{max} from seconds to hours and compare its selected model's performance against baselines. We apply 2Phase-MS to six datasets across three search spaces, running 100 trials per T_{max} and recording the 25th, 50th (median), and 75th percentiles of validation AUC or test accuracy.

Figure 8 shows that Training-Based MS requires 5 to 10 minutes to evaluate a single model, which violates the target SLO when the T_{max} is small, e.g., less than 5 minutes. In contrast, our 2Phase-MS can consistently complete model selection tasks within the allocated time, delivering high-performing models and adhering to target SLOs, which can continue to refine the search results as more time budget is available. This efficiency is mainly due to the SLO-aware coordinator, which balances the two phases according to the specified T_{max}. For example, when the budget is small, the coordinator prioritizes the filtering phase, using only training-free evaluation that estimates the performance of each model within seconds, thereby ensuring effective model selection even under such strict time constraints.

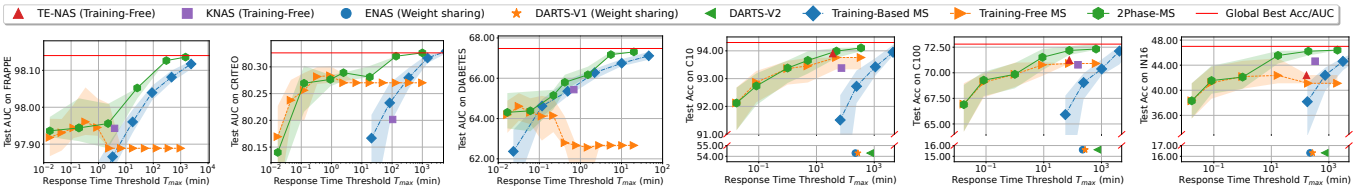
As for effectiveness, our 2Phase-MS can identify models with better performance across nearly all T_{max}. Unlike TE-NAS, which relies solely on training-free methods such as NTKCond, 2Phase-MS implements a more effective TFMEM and also incorporates training-based evaluations. As a result, 2Phase-MS can identify models with higher test accuracy under equivalent T_{max}, achieving up to a 3.83% accuracy increase of the selected model on IN16. Compared with KNAS which scores a pre-determined number of models followed by

[6]https://www.psycopg.org/docs/



(a) DNN + Frappe. (b) DNN + Criteo. (c) DNN + Diabete. (d) NB201 + C10. (e) NB201 + C100. (f) NB201 + IN16.

Figure 7: Efficiency and effectiveness analysis of various training-free and training-based combinations.



(a) DNN + Frappe. (b) DNN + Criteo. (c) DNN + Diabete. (d) NB201 + C10. (e) NB201 + C100. (f) NB201 + IN16.

Figure 8: SLO-aware of 2Phase-MS on six datasets with two search spaces.

fully training the top models, 2Phase-MS employs effective JacFlow and leverages training-based evaluation scheduled by SUCCHALF rather than full training, which is more effective and efficient. For instance, 2Phase-MS achieves an AUC of 97.96% on the Frappe dataset in only 3.6 minutes, outperforming KNAS which reaches 97.94% in 3.9 minutes. Similarly, for IN16, 2Phase-MS attains an accuracy of 46.22% in 193 minutes, a notable improvement over KNAS's 44.63% in 333 minutes.

Weight-sharing-based model selections such as ENAS and DARTS-V1/V2 perform poorly because they overfit to models with all skip-connection edges, as also reported and explained in [15, 18, 41]. Furthermore, they tend to search for similar cell structures [18]. Compared with training-based MS, 2Phase-MS is more efficient and effective due to the design of the coordinator, which jointly optimizes the two phases to maximize the performance of the selected model under the given T_{max}. Specifically, 2Phase-MS achieves 8.12x, 7.30x, and 24.38x speedups in searching for models of the same performance in the DNN search space for the three structured datasets when compared with Training-Based MS. Similarly, 2Phase-MS achieves 52.10x, 27.27x, and 26.89x speedups in NB201 on the three unstructured datasets, yielding test accuracy of 94.10% on C10, 72.32% on C100, and 46.40% on IN16, respectively.

6.3.3 Filtering Refinement Collaboration. Training-Free MS rapidly focuses on well-performing models within seconds and is able to select relatively higher-performing models at the early T_{max}, as shown in Figure 8. However, as the T_{max} increases, Training-Free MS starts to undervalue well-performing models. This is because Training-Free MS persistently tracks the model with a higher JacFlow score, which does not always precisely indicate a model's superior performance as discussed in Section 3.3. Evidently, the refinement phases play a crucial role as they can compensate for the filtering phase's uncertainty and further enhance the 2Phase-MS effectiveness. Conversely, without the filtering phase, 2Phase-MS is unable to benefit from the extremely rapid model evaluation and accomplish the model selection within seconds. In conclusion, both phases are essential for improving the efficiency and effectiveness of the model selection.

6.4 Efficiency

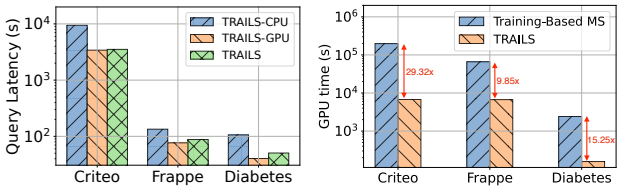
We then evaluate the efficiency of TRAILS regarding resource consumption and query latency and compare TRAILS with baseline systems. To provide a consistent workload for comparison, we set fixed workloads of exploring N models for the filtering phase and training K models for the refinement phase.

6.4.1 Resource Efficiency.

Resource-Efficient Runtime Placement. We first examine how various combinations of runtime affect the query latency of TRAILS. Specifically, we compare the query latency of TRAILS which uses a hybrid CPU/GPU runtime, against both TRAILS-GPU and TRAILS-CPU, which use GPU-only and CPU-only runtimes respectively. This comparison is consistently conducted across three datasets, with a fixed workload that involves exploring N=1500 models in the filtering phase and training as well as evaluating the top K=15 models in the refinement phase.

As shown in Figure 9a, TRAILS-CPU displays the highest query latency among the three runtime settings. TRAILS-GPU and TRAILS exhibit comparable less query latency, with TRAILS-GPU marginally outpacing TRAILS. This confirms our resource-efficient runtime placement strategy illustrated in Section 5.1 where the advantages of GPUs are not evident in the filtering phase. Specifically, our TRAILS, based on a hybrid CPU/GPU runtime, reduces GPU time by 1.37x on Frappe, 1.43x on Diabetes, and 1.37x on Criteo compared with TRAILS-GPU.

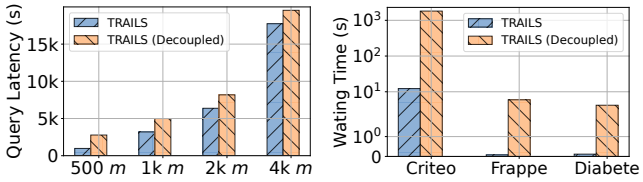
Reduced GPU time. Second, we compare TRAILS with the training-based model selection system in terms of the total GPU time for attaining a target model performance, i.e., AUC at 98.00% for Frappe, 67.10% for Diabetes, and 80.30% for Criteo.



(a) Resource-efficient runtime placement.

(b) Reduced GPU time consumption.

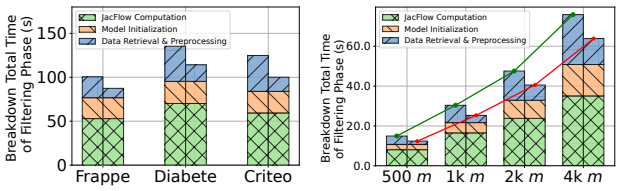
Figure 9: Resource-efficient runtime placement and reduced GPU consumption of the TRAILS in terms of GPU time.



(a) Query latency w.r.t. #models evaluated in the two-phase model selection algorithm on the Criteo.

(b) Waiting time before the training in the refinement phase on three datasets.

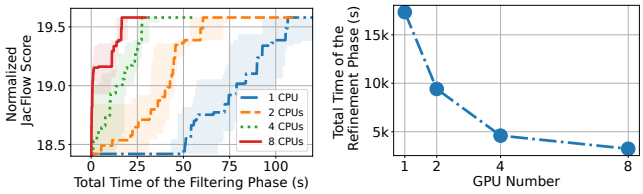
Figure 10: The efficiency of TRAILS in in terms query latency: A comparison with TRAILS (Decoupled).



(a) Total time for exploring 5K models on three datasets.

(b) Total time w.r.t. #models explored on the Criteo dataset.

Figure 11: Efficiency of in-database filtering phase. In both sub-figures, times of executing the filtering phase *outside* (left bars) and *inside* (right bars) PostgreSQL are compared.



(a) Time of the filtering phase to reach the maximum JacFlow score with different numbers of CPUs.

(b) Time of the refinement phase to train 40 models via SUCCHALF with different numbers of GPUs.

Figure 12: Scalability of the filtering and refinement phase.

Figure 9b illustrates that TRAILS is highly resource efficient by reducing GPU time by factors of 9.85x for Frappe, 15.25x for Diabetes, and 29.32x for Criteo respectively, compared to the training-based model selection system. This is because TRAILS uses the GPU runtime only during the refinement phase and trains fewer models scheduled by SUCCHALF.

6.4.2 Query Latency of In-Database Model Selection. We then compare the performance of executing the two-phase algorithm inside and outside PostgreSQL, respectively. Specifically, we compare the overall query latency of TRAILS against TRAILS (Decoupled) as introduced in Section 6.2.4. Then, we conduct breakdown experiments to measure the time usage of each phase.

Overall Query Latency. To assess overall query latency, we execute multiple model selection queries with N ranging from 500 to 4k models and K set to $N/100$. As illustrated in Figure 10a, TRAILS is consistently more efficient than TRAILS (Decoupled) for all queries since its integration of the filtering phase within PostgreSQL using UDFs and utilization of a caching service to facilitate pipeline parallelism and reduce the waiting time before model training.

Breakdown Experiments for Refinement Phase. To evaluate the efficiency of the refinement phase, we set N=1500, K=15, and cache size to ten, and measure the waiting time for data retrieval and preprocessing before training begins, comparing scenarios with and without the cache service. Figure 10b shows that the waiting time of training in the refinement phase is reduced by a factor of 76x on Frappe, 36x on Diabetes, and 148x on Criteo. This is because the cache service only retrieves and preprocesses a few batches of data rather than the entire data as in TRAILS (Decoupled).

Breakdown Experiments for Filtering Phase. To verify the efficiency of executing the filtering phase inside PostgreSQL via UDFs, we compare it against execution outside PostgreSQL and retrieving data via psycopy. This involves exploring 5k models on three datasets and further adjusting the number of models from 500 to 4k on Criteo. Figure 11a shows that executing the filtering phase inside PostgreSQL reduces the total time by a factor of 1.15x on Frappe, 1.19x on Diabetes, and 1.25x on Criteo. Figure 11b further demonstrates the consistent efficiency of our approach when exploring a varying number of models.

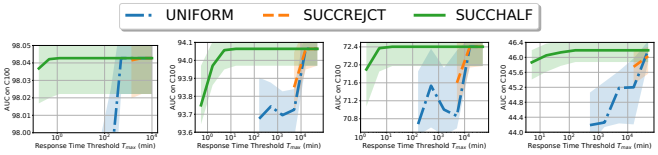
These experiments demonstrate that our TRAILS can efficiently conduct model selection with reduced overheads.

6.5 Scalability

In this subsection, we evaluate the scalability of TRAILS in terms of more computational resources and larger data sizes.

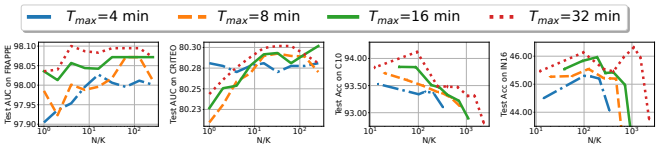
6.5.1 Scalability with Increased Resources. For the filtering phase, we adopt the centralized search strategy introduced in Section 4.1, and deploy multiple evaluation workers running on respective CPUs. We vary the number of CPUs from 1 to 2, 4, and 8 and record the time to reach the maximum JacFlow score. As for the refinement phase, we employ the budget controller with SUCCHALF as discussed in Section 4.2 and deploy multiple evaluation workers running on respective GPUs for parallel model evaluations. We vary the number of GPUs used from 1 to 2, 4, and 8, and record the total time of evaluating 40 models.

As illustrated in Figure 12a, increasing the number of CPUs allows the filtering phase to deploy more evaluation workers for



(a) Frappe. (b) C10. (c) C100. (d) IN16.

Figure 13: Benchmarking three fixed-budget algorithms.



(a) Frappe. (b) Criteo. (c) C10. (d) IN16.

Figure 14: Impact of parameter on coordinator performance.

computing the JacFlow score in parallel. This facilitates the search strategy in exploring a wider range of models and identifying those with superior JacFlow scores. Similarly, in Figure 12b, having more GPUs enables parallel model evaluation per SUCCHALF round, consequently reducing the total time needed to assess 40 models in the refinement phase. In conclusion, both phases of TRAILS are highly scalable with more CPU or GPU resources.

6.5.2 Scalability with Large Data Sizes. To validate the effectiveness and efficiency of our algorithm on larger datasets, we employ the large ImageNet dataset [46] with each image at a resolution of 224x224 pixels. We adopt the whole ImageNet and follow the setting of [18, 34], where we use the SGD optimizer with a momentum of 0.9, an initial learning rate of 0.002 with a batch size of 32, and cosine learning rate decay.

In comparison, training-based methods such as NASNET-A and AmoebaNet-C achieve Top-1 accuracy of 74% and 75.7% with 2000 GPU days and 3150 GPU days, respectively [64]. Meanwhile, TE-NAS uses training-free methods and can efficiently achieve a Top-1 accuracy of 75.5% with only 0.17 GPU days [64]. Notably, our 2Phase-MS reaches a Top-1 accuracy of 75.3% in only 0.05 GPU days, which demonstrates its effectiveness and efficiency in conducting model selection on large-scale datasets.

6.6 Component Analysis

Refinement Phase Design. We analyze three fixed-budget algorithms introduced in Section 4.2: UNIFORM, SUCCREJCT, and SUCCHALF. For each predefined T_{max}, we randomly sample 400 models and allow each algorithm to distribute time for training each model. This process is conducted 100 times, with the model's performance recorded at the 25th, 50th (median), and 75th percentiles. Figure 13 shows SUCCHALF outperforms others in either efficiency and effectiveness across four datasets, this is attributed to its efficient allocation of time to train more promising models. SUCCREJCT demands more time for evaluation since it eliminates only the worst model each round. UNIFORM is highly effective in a larger T_{max} by fully training each model, but it is less efficient since it requires consistently allocating the same time budget to evaluate each model. We, therefore, adopt SUCCHALF as the budget controller in our system.

Coordination Design. We experimentally investigate the influence of the parameters r and U_{init} on coordinator performance, where $r = N/K$. We fix U_{init} to one epoch, set T_{max} from 4 to 32 minutes, experiment with r values from 1 to 1k, and evaluate on four datasets. The results in Figure 14 demonstrate that a setting of $r \approx 100$ and $U_{init} = 1$ consistently facilitates the discovery of high-performing models across all four datasets. This indicates that the chosen values for r and U_{init} exhibit a level of generalizability, making our coordinator broadly applicable and capable of adapting to new datasets without the need for further adjustment.

7 RELATED WORK

Existing model selection algorithms for structured data, including AgEBO-Tabular [20] and TabNAS [63], share a similar goal with TRAILS, which is to efficiently identify high-performing models from a large search space for structured data. However, both approaches rely on full training and provide limited optimization for expensive model evaluation. Furthermore, they lack an integrated end-to-end system that considers data operations before model training and does not address practical system requirements such as SLO awareness and resource efficiency. In-database machine learning has been studied by [16, 23, 28, 33, 47], yet their focus is primarily on model training or inference, which neglects critical tasks such as model selection. [67] incorporates model selection into databases, while it still depends on training-based algorithms and is thus not as computationally efficient as our TRAILS.

8 CONCLUSION

In this paper, we focus on building a resource-efficient and SLO-aware in-database model selection system TRAILS, which enables average machine learning users to obtain well-performing models within their specified response-time threshold. TRAILS encompasses a novel two-phase model selection algorithm, combining the strengths of highly efficient training-free and effective training-based model evaluation algorithms. From a system perspective, we have seamlessly integrated the proposed algorithm into PostgreSQL by allocating the I/O intensive filtering phase into UDF runtime and the computational intensive refinement phase into GPU runtime to minimize high-end hardware consumption. To further reduce the latency and memory overhead caused by data retrieval and preprocessing in the refinement phase, we designed a data cache service to facilitate on-the-fly data transformations and caching. Experimental results demonstrate that TRAILS consistently selects higher-performing models compared to training-based and training-free model selection algorithms while adhering to SLOs ranging from seconds to hours. Furthermore, our system exhibits reduced MSQ execution latency and resource consumption.

ACKNOWLEDGMENTS

This research work is supported by Singapore Ministry of Education Academic Research Fund Tier 3 under MOE's official grant number MOE2017-T3-1-007.

REFERENCES

[1] 2023. Automate machine learning model selection with Azure Machine Learning. https://learn.microsoft.com/en-us/training/modules/automate-model-selection-with-azure-automl/1-introduction.

[2] 2023. AutoML: Train High-quality Custom Machine Learning Models with Minimal Effort and Machine Learning Expertise. https://cloud.google.com/automl

[3] 2023. AWS AutoML Solutions. https://aws.amazon.com/machine-learning/automl/.

[4] 2023. PostgreSQL. https://www.postgresql.org/.

[5] Mohamed S. Abdelfattah, Abhinav Mehrotra, Lukasz Dudziak, and Nicholas Donald Lane. 2021. Zero-Cost Proxies for Lightweight NAS. In *Proceedings of The International Conference on Learning Representations*.

[6] Christopher R. Aberger, Andrew Lamb, Kunle Olukotun, and Christopher Ré. 2018. LevelHeaded: A Unified Engine for Business Intelligence and Linear Algebra Querying. In *Proceedings of The International Conference on Data Engineering*. 449–460.

[7] Michael Armbrust, Kristal Curtis, Tim Kraska, Armando Fox, Michael J. Franklin, and David A. Patterson. 2011. PIQL: Success-Tolerant Query Processing in the Cloud. *Proceedings of VLDB Endowment* 5, 3 (2011), 181–192.

[8] Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. 2010. Best Arm Identification in Multi-Armed Bandits. In *Conference on Learning Theory*. 41–53.

[9] Matthias Boehm, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen, Yuanyuan Tian, Douglas Burdick, and Shivakumar Vaithyanathan. 2014. Hybrid Parallelization Strategies for Large-Scale Machine Learning in SystemML. *Proceedings of VLDB Endowment* 7, 7 (2014), 553–564.

[10] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. 2009. Pure Exploration in Multi-armed Bandits Problems. In *Algorithmic Learning Theory*, Vol. 5809. 23–37.

[11] 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*. 2787–2794.

[12] Shaofeng Cai, Gang Chen, Beng Chin Ooi, and Jinyang Gao. 2019. Model Slicing for Supporting Complex Analytics with Elastic Inference Cost and Resource Constraints. *Proceedings of VLDB Endowment* 13, 2 (2019), 86–99.

[13] Shaofeng Cai, Kaiping Zheng, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Meihui Zhang. 2021. ARM-Net: Adaptive Relation Modeling Network for Structured Data. In *Proceedings of The International Conference on Management of Data*.

[14] Wei Cao, Yingqiang Zhang, Xinjun Yang, Feifei Li, Sheng Wang, Qingda Hu, Xuntao Cheng, Zongzhi Chen, Zhenjun Liu, Jing Fang, Bo Wang, Yuhui Wang, Haiqing Sun, Ze Yang, Zhushi Cheng, Sen Chen, Jian Wu, Wei Hu, Jianwei Zhao, Yusong Gao, Songlu Cai, Yunyang Zhang, and Jiawang Tong. 2021. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *Proceedings of The International Conference on Management of Data*. 2477–2489.

[15] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. 2021. Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective. In *Proceedings of The International Conference on Learning Representations*.

[16] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, and Caleb Welton. 2009. MAD Skills: New Analysis Practices for Big Data. *Proceedings of VLDB Endowment* 2, 2 (2009), 1481–1492.

[17] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *Proceedings of The USENIX Symposium on Networked Systems Design and Implementation*. 613–627.

[18] Xuanyi Dong and Yi Yang. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *Proceedings of The International Conference on Learning Representations*.

[19] Joseph Vinish D'silva, Florestan De Moor, and Bettina Kemme. 2018. AIDA - Abstraction for Advanced In-Database Analytics. *Proceedings of VLDB Endowment* 11, 11 (2018), 1400–1413.

[20] Romain Égelé, Prasanna Balaprakash, Isabelle Guyon, Venkatram Vishwanath, Fangfang Xia, Rick Stevens, and Zhengying Liu. 2021. AgEBO-tabular: joint neural architecture and hyperparameter search with autotuned data-parallel training for tabular data. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis*.

[21] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the International Conference on Machine Learning*, Vol. 80. 1436–1445.

[22] Arash Fard, Anh Le, George Larionov, Waqas Dhillon, and Chuck Bear. 2020. Vertica-ML: Distributed Machine Learning in Vertica Database. In *Proceedings of The International Conference on Management of Data*. 755–768.

[23] Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib Analytics Library or MAD Skills, the SQL. *Proceedings of VLDB Endowment* 5, 12 (2012), 1700–1711.

[24] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. 2016. Deep Networks with Stochastic Depth. In *Proceedings of The European Conference on Computer Vision*, Vol. 9908. 646–661.

[25] Kevin G. Jamieson and Ameet Talwalkar. 2016. Non-stochastic Best Arm Identification and Hyperparameter Optimization. In *Proceedings of The International Conference on Artificial Intelligence and Statistics*, Vol. 51. 240–248.

[26] Matthias Jarke and Jürgen Koch. 1984. Query Optimization in Database Systems. *Comput. Surveys* 16, 2 (1984), 111–152.

[27] Peng Jia, Shaofeng Cai, Beng Chin Ooi, Pinghui Wang, and Yiyuan Xiong. 2023. Robust and Transferable Log-based Anomaly Detection. *Proceedings of The International Conference on Management of Data* 1, 1 (2023), 64:1–64:26.

[28] Kaan Kara, Ken Eguro, Ce Zhang, and Gustavo Alonso. 2018. ColumnML: Column-Store Machine Learning with On-The-Fly Data Transformation. *Proceedings of VLDB Endowment* 12, 4 (2018), 348–361.

[29] Arjun Krishnakumar, Colin White, Arber Zela, Renbo Tu, Mahmoud Safari, and Frank Hutter. 2022. NAS-Bench-Suite-Zero: Accelerating Research on Zero Cost Proxies. In *Proceedings of The Advances in Neural Information Processing Systems*.

[30] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. *Technical report* (2009).

[31] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2019. Snip: single-Shot Network Pruning based on Connection sensitivity. In *Proceedings of The International Conference on Learning Representations*.

[32] Rubao Lee, Minghong Zhou, Chi Li, Shenggang Hu, Jianping Teng, Dongyang Li, and Xiaodong Zhang. 2021. The Art of Balance: A RateupDB Experience of Building a CPU/GPU Hybrid Database Product. *Proceedings of VLDB Endowment* 14, 12 (2021), 2999–3013.

[33] Xupeng Li, Bin Cui, Yiru Chen, Wentao Wu, and Ce Zhang. 2017. MLog: Towards Declarative In-Database Machine Learning. *Proceedings of VLDB Endowment* 10, 12 (2017), 1933–1936.

[34] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. 2021. Zen-NAS: A Zero-Shot NAS for High-Performance Image Recognition. In *Proceedings of the IEEE International Conference on Computer Vision*. 337–346.

[35] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical Representations for Efficient Architecture Search. In *Proceedings of The International Conference on Learning Representations*.

[36] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *Proceedings of The International Conference on Learning Representations*.

[37] Zhaojing Luo, Shaofeng Cai, Can Cui, Beng Chin Ooi, and Yang Yang. 2021. Adaptive knowledge driven regularization for deep neural networks. In *Proceedings of The AAAI Conference on Artificial Intelligence*, Vol. 35. 8810–8818.

[38] Zhaojing Luo, Shaofeng Cai, Jinyang Gao, Meihui Zhang, Kee Yuan Ngiam, Gang Chen, and Wang-Chien Lee. 2018. Adaptive Lightweight Regularization Tool for Complex Analytics. In *Proceedings of The International Conference on Data Engineering*. 485–496.

[39] Zhaojing Luo, Shaofeng Cai, Yatong Wang, and Beng Chin Ooi. 2023. Regularized Pairwise Relationship based Analytics for Structured Data. *Proceedings of The International Conference on Management of Data* 1, 1 (2023), 82:1–82:27.

[40] Hanna Mazzawi, Xavi Gonzalvo, Aleks Kracun, Prashant Sridhar, Niranjan Subrahmanya, Ignacio Lopez-Moreno, Hyun-Jin Park, and Patrick Violette. 2019. Improving Keyword Spotting and Language Identification via Neural Architecture Search at Scale. In *Proceedings of The Annual Conference of the International Speech Communication Association*. 1278–1282.

[41] Joe Mellor, Jack Turner, Amos J. Storkey, and Elliot J. Crowley. 2021. Neural Architecture Search without Training. In *Proceedings of the International Conference on Machine Learning*, Vol. 139. 7588–7598.

[42] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System for Optimized Deep Learning Model Selection. *Proceedings of VLDB Endowment* 13, 11 (2020), 2159–2173.

[43] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony K. H. Tung, Yuan Wang, Zhongle Xie, Meihui Zhang, and Kaiping Zheng. 2015. SINGA: A Distributed Deep Learning Platform. In *Proceedings of the ACM International Conference on Multimedia*. 685–688.

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of The Advances in Neural Information Processing Systems*. 8024–8035.

[45] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *Proceedings of The AAAI Conference on Artificial Intelligence*. 4780–4789.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[47] Sandeep Singh Sandha, Wellington Cabrera, Mohammed Al-Kateb, Sanjay Nair, and Mani B. Srivastava. 2019. In-database Distributed Machine Learning: Demonstration using Teradata SQL Engine. *Proceedings of VLDB Endowment* 12, 12 (2019), 1854–1857.

[48] Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. 2022. NASI: Label- and Data-agnostic Neural Architecture Search at Initialization. In *Proceedings of The International Conference on Learning Representations*.

[49] Yao Shu, Zhongxiang Dai, Zhaoxuan Wu, and Bryan Kian Hsiang Low. 2022. Unifying and Boosting Gradient-Based Training-Free Neural Architecture Search. In *Proceedings of The Advances in Neural Information Processing Systems*.

[50] Michael Stonebraker, Paul Brown, Donghui Zhang, and Jacek Becla. 2013. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering* 15, 3 (2013), 54–62.

[51] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. 2020. Pruning Neural Networks without Any Data by Iteratively Conserving Synaptic Flow. In *Proceedings of The Advances in Neural Information Processing Systems*.

[52] Chaoqi Wang, Guodong Zhang, and Roger B. Grosse. 2020. Picking Winning Tickets Before Training by Preserving Gradient Flow. In *Proceedings of The International Conference on Learning Representations*.

[53] Wei Wang, Jinyang Gao, Meihui Zhang, Sheng Wang, Gang Chen, Teck Khim Ng, Beng Chin Ooi, Jie Shao, and Moaz Reyad. 2018. Rafiki: Machine Learning as an Analytics Service System. *Proceedings of VLDB Endowment* 12, 2 (2018), 128–140.

[54] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *SIGMOD Record* 45, 2 (2016), 17–22.

[55] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. 2023. Neural Architecture Search: Insights from 1000 Papers. *CoRR* abs/2301.08727 (2023).

[56] Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. 2021. How Powerful are Performance Predictors in Neural Architecture Search?. In *Proceedings of The Advances in Neural Information Processing Systems*. 28454–28469.

[57] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. 2019. A Survey on Neural Architecture Search. *CoRR* abs/1905.01392 (2019).

[58] Naili Xing, Sai Ho Yeung, Chenghao Cai, Teck Khim Ng, Wei Wang, Kaiyuan Yang, Nan Yang, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2021. SINGA-Easy: An Easy-to-Use Framework for MultiModal Analysis. In *Proceedings of The ACM International Conference on Multimedia*. 1293–1302.

[59] Jingjing Xu, Liang Zhao, Junyang Lin, Rundong Gao, Xu Sun, and Hongxia Yang. 2021. KNAS: Green Neural Architecture Search. In *Proceedings of the International Conference on Machine Learning*, Vol. 139. 11613–11625.

[60] Lijie Xu, Shuang Qiu, Binhang Yuan, Jiawei Jiang, Cédric Renggli, Shaoduo Gan, Kaan Kara, Guoliang Li, Ji Liu, Wentao Wu, Jieping Ye, and Ce Zhang. 2022. In-Database Machine Learning with CorgiPile: Stochastic Gradient Descent without Full Data Shuffle. In *Proceedings of The International Conference on Management of Data*. 1286–1300.

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

[62] Shen Yan, Colin White, Yash Savani, and Frank Hutter. 2021. NAS-Bench-x11 and the Power of Learning Curves. In *Proceedings of The Advances in Neural Information Processing Systems*. 22534–22549.

[63] Chengrun Yang, Gabriel Bender, Hanxiao Liu, Pieter-Jan Kindermans, Madeleine Udell, Yifeng Lu, Quoc V Le, and Da Huang. 2022. TabNAS: Rejection Sampling for Neural Architecture Search on Tabular Datasets. *Proceedings of The Advances in Neural Information Processing Systems* 35, 11906–11917.

[64] Taojiannan Yang, Linjie Yang, Xiaojie Jin, and Chen Chen. 2023. Revisiting Training-free NAS Metrics: An Efficient Training-based Method. In *Proceedings of The Winter Conference on Applications of Computer Vision*. 4740–4749.

[65] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *Proceedings of the International Conference on Machine Learning*, Vol. 97. 7105–7114.

[66] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *Proceedings of The USENIX Annual Technical Conference*. 1049–1062.

[67] Yuhao Zhang, Frank Mcquillan, Nandish Jayaram, Nikhil Kak, Ekta Khanna, Orhan Kislal, Domino Valdano, and Arun Kumar. 2021. Distributed Deep Learning on Data Systems: A Comparative Analysis of Approaches. *Proceedings of VLDB Endowment* 14, 10 (2021), 1769–1782.

[68] 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.