IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 December 2023, accepted 20 December 2023, date of publication 3 January 2024, date of current version 10 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3349497

APPLIED RESEARCH

SeqNAS: Neural Architecture Search for Event Sequence Classification

IGOR UDOVICHENKO[1], EGOR SHVETSOV[1], DENIS DIVITSKY[2], DMITRY OSIN[1], ILYA TROFIMOV[1], IVAN SUKHAREV[3], ANATOLIY GLUSHENKO[4], DMITRY BERESTNEV[3], AND EVGENY BURNAEV[1]
[1]Skolkovo Institute of Science and Technology, 121205 Moscow, Russia
[2]Tinkoff, 127006 Moscow, Russia
[3]Zvuk, 107045 Moscow, Russia
[4]VTB, 190000 Moscow, Russia

Corresponding author: Igor Udovichenko (i.udovichenko@skoltech.ru)

The work was supported by the Analytical center under the RF Government (subsidy agreement 000000D730321P5Q0002, Grant No. 70-2021-00145 02.11.2021).

ABSTRACT Neural Architecture Search (NAS) methods are widely used in various industries to obtain high-quality, task-specific solutions with minimal human intervention. Event Sequences (**EvS**) find widespread use in various industrial applications, including churn prediction, customer segmentation, fraud detection, and fault diagnosis, among others. Such data consist of categorical and real-valued components with irregular timestamps. Despite the usefulness of NAS methods, previous approaches only have been applied to other domains: images, texts or time series. Our work addresses this limitation by introducing a novel NAS algorithm — **SeqNAS**, specifically designed for event sequence classification. We develop a simple yet expressive search space that leverages commonly used building blocks for event sequence classification, including multi-head self attention, convolutions, and recurrent cells. To perform the search, we adopt sequential Bayesian Optimization and utilize previously trained models as an ensemble of teachers to augment knowledge distillation. As a result of our work, we demonstrate that our method surpasses state-of-the-art NAS methods and popular architectures suitable for sequence classification and holds great potential for various industrial applications.

INDEX TERMS NAS, temporal point processes, event sequences, RNN, transformers, knowledge distillation, surrogate models.

I. INTRODUCTION

Motivation. Event Sequences (**EvS**) with marker and timing information are very common in real-world applications such as medicine [1], biology [2], social medial analysis [3], fault diagnosis [4], [5], churn prediction [6], [7], customer segmentation [8], fraud detection [9] and more. Consequently, there is a demand to model such data.

In [5], the authors focus on predicting failures using telemetry data as a **EvS** classification task. They emphasize the significant benefits of even predicting a small fraction of these incidents, including improved availability, cost reduction, and avoidance of reactive maintenance.

The associate editor coordinating the review of this manuscript and approving it for publication was Alberto Cano.

Based on the results presented in [1], applications of Automated Machine Learning (AutoML) in healthcare require additional research and development. The utilization of automated methods in medicine holds great potential for substantially enhancing accuracy, with even minor improvements carrying significant weight in the healthcare sector.

In the field of biology, representing biological datasets as sequences is common. In [2], the authors develop an end-to-end automated machine learning tool specifically designed for explaining and designing biological sequences.

In [10], the authors explore a wide range of machine learning applications for enhancing efficiency in the financial sector. These technologies have the potential to automate processes, improve risk assessment and management, and enable more accurate and efficient decision-making. It is important to note that the financial sector, particularly banking, accumulates a significant amount of sequential data based on customer behavior and market events.

EvS classification methods are used in various fields, however, the successful utilization of machine learning still requires substantial effort from human experts, as no algorithm can achieve optimal performance on all possible problems.

Most of machine learning research and applications have been centered around *core domains: text, images, time-series, and speech*. However, **EvS** differ from these well-studied domains in several ways:

(1) events are usually described by both categotical and numerical features,
(2) events in an arbitrary sequence are not uniformly spaced in time (an example can be seen in Figure 1), whereas data in core domains is usually uniformly distributed spatially (image pixels) or temporally (speech signals), and
(3) elements that are close together in space or time have a shared context, and valuable information can be inferred from their neighboring elements. However, this principle does not necessarily apply to **EvS** since the events may not occur in close proximity to each other in time.

The properties mentioned above can vary significantly across different datasets. As a result, effectively modeling **EvS** requires the development of task-specific deep learning architectures. This process involves leveraging domain knowledge and can be labor-intensive due to the iterative nature of trial and error.

Our work aims to develop a NAS procedure specifically designed to effectively handle diverse event sequences. We refer to this approach as **SeqNAS** (Sequence NAS).

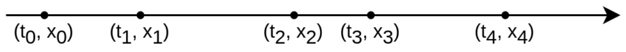

FIGURE 1. An example of the marked temporal event process. Event *i* occurs at time t_i and is characterized (marked) by the feature vector x_i.

Our contributions and results:

- **Performance**. Our simple yet efficient method **SeqNAS** shows the superior performance when compared to existing NAS methods and popular architectures that are used for **EvS** classification.
- **Search Space Design**. We *design a novel search space* of size $\sim 5 \times 10^6$ possible architectures. The search space contains multi-head self attention, convolutions, and recurrent cells and is tailored to handle event sequence data. To the best of our knowledge, we are the first to develop and analyze such a search space for event sequence datasets.
- **Ensemble Of Teachers**. Typically, intermediate (suboptimal) architectures obtained during architecture search are thrown away. We propose to utilize them as an ensemble of teachers for subsequent models via knowledge distillation [11].
- **Benchmark Datasets**. To advance the development of event sequence classification methods, we have initiated a benchmark for **EvS** classification by comparing various models and methods. Our study employs six event sequence datasets that were sourced from online competitions held over time or used by other authors. Our work is the first one to carry out a comparison of **EvS** classification methods for a diverse list of datasets. We make these pre-processed datasets openly available.
- **NAS-Bench Event Sequences**. We present a novel neural architecture dataset comprising 3200 trained architectures, each accompanied by its corresponding scores. This dataset can further facilitate development of predictor-based NAS methods.

We provide the source code for the experiments conducted on publicly available datasets, as described in this paper, along with the datasets themselves.[1]

II. RELATED WORK

A. SEARCH SPACE

The performance of NAS algorithms heavily relies on the search space design, which should exhibit a reasonable degree of flexibility and accommodate established high-performing solutions. A well-designed search space can deliver a satisfactory outcome even with a random search strategy [12]. Therefore, search space design is a primary focus of our work.

Although event sequence data are very common in various applications, the majority of NAS algorithms are designed to solve image classification problems and only rarely extend their search procedures to other core domains.

Closest to our domain are works exploring text classification and multivariate time-series classification. To evaluate the performance of our procedure for **EvS** classification, we included methods from both domains in our benchmark and studied their transferability to **EvS**. These methods are discussed in Sections II-B and II-C.

B. NAS FOR TEXT CLASSIFICATION

The most frequently used building **blocks** in modern deep neural network (DNN) architectures are: 1) Convolutional; 2) Recurrent; 3) Multi-Head Self-Attention; 4) Pooling Layers; and 5) Identity layers. **AutoAttend** [13] and **TextNas** [14] have both explored different methods of combining these building blocks as graph nodes to construct a search space.

In **TextNas** [14], each node is selected from a pool of blocks described above, and each incoming feature (edge) is selected from a pool of nodes from previous layers. This work is the closest one to ours.

AutoAttend [13] introduces a NAS procedure to search for attention representations. The main idea is that **K**-keys, **V**-values, and **Q**-queries may originate as features from

[1] Our code: https://github.com/On-Point-RND/SeqNAS

distinct layers and that different searchable blocks are used to project each incoming feature. Then, incoming features are aggregated in an attention or an addition layer.

In **NAS-Bench-NLP** [15], authors search for an Recurrent Neural Network (RNN) structure. The RNN cell is represented as a directed graph in which nodes correspond to specific operations and edges encode their inputs. The authors use four different types of operations in their study, namely: linear layers, element-wise weighted summation, element-wise product, and various activation functions. The operations associated with the nodes and edges are selected during the search process.

C. DNN FOR MULTIVARIATE TIME SERIES CLASSIFICATION

Gated-Transformer-"GTN" [16] uses two attention blocks instead of one, one to model step-wise and another channel-wise correlation between components of Multivariate Time Series. The performance of this approach for **EvS** is evaluated in Section IV-C.

In **ROCKET** [17], the authors focus on Univariate Time Series (UTS) classification. **ROCKET** generates feature representations using several randomly initialized convolutional kernels and then trains a linear layer a top of these features. A comparison of **SeqNAS** and ROCKET can be found in Table 4.

D. SEARCH METHODS

The architecture of a deep neural network (DNN) can be modeled as a directed acyclic graph (DAG), where the nodes represent operations, and the edges denote incoming or outgoing features. In this way, neural architecture search (NAS) is viewed as an algorithms for discovering a task-specific DAG.

There are various search strategies available, including Bayesian Optimization (BO), Evolutionary Methods, Reinforcement Learning, and Differentiable NAS (DNAS). These methods aim to find the best suitable architecture in the vast space of neural architectures with significantly fewer resources than an exhaustive search requires.

In ENAS [18], reinforcement learning is utilized to train a super-net, which is an over-parameterized architecture allowing for efficient weight sharing among sub-models. This eliminates the need to train each candidate sub-model from scratch, resulting in a significant reduction in search time. Additionally, DNAS [19], [20], [21], [22], [23], [24] builds upon this idea by representing the over-parameterized super-net as DAG and assigning differentiable importance weights to each edge. The highest edge values determine the selected sub-graph or path. This approach minimizes the need to evaluate multiple models, thereby accelerating the search process. Knowledge distillation without an ensemble of teachers was applied for NAS in [25] for image classification.

However, [26] and [23] demonstrated that the optimal architecture is not always selected using DNAS and that the procedure requires various modifications to perform well. On the other hand, [15], [27] have demonstrated that various BO-based methods perform effectively across varied search spaces, including text classification. One such method, highlighted by the authors, is **BANANAS** [28], which relies on BO and the neural *Predictor-model* — the model designed to predict architecture performance bypassing full train and validation cycle. The neural *Predictor-model* is trained on previously queried architectures to score new potential candidates before training them. Unlike DNAS, **BANANAS** requires training multiple models. However, we show that previously trained models can be used as a practical advantage. This is discussed in Section III-D.

To train a *Predictor-model* on a set of architecture-score tuples, it is necessary to have a procedure for architecture encoding. The study [29] propose eight architechure encoding schemes categorized into two groups: adjacency matrix-based and path-based. The authors evaluate the performance of each encoding scheme for different NAS subroutines: *Predictor-model* training, architecture perturbation and random architecture sampling. They show that no encoding scheme performs well across all subroutines, but the path-based encoding outperforms the adjacency matrix-based one on the task of training the *Predictor-model*.

E. NAS BENCHMARKS

The NAS-Bench series of benchmarks [15], [30], [31], [32], [33], [34] have made significant contributions to the advancement of scientific research in neural architecture search (NAS). These benchmarks aim to establish a standardized measurement procedure and provide datasets for easy comparison and reproducibility in NAS research. They include datasets of trained architectures and their corresponding scores, along with detailed discussions on the characteristics and performance of various NAS algorithms. However, these benchmarks have not yet explored the domain of event sequence. In our work, we extend the NAS-Bench series with NAS-Bench Event Sequences, a dataset of architectures specifically designed to model **EvS**.

F. TEMPORAL POINT PROCESSES MODELING

Recently, different neural architectures and approaches were used to model **EvS** as Temporal Point Processes (TPP) [3]. These data exhibit complex short-term and long-term temporal dependencies. Existing methods heavily rely on Recurrent Neural Networks (RNNs) due to the sequential nature of event sequences [35]. However, RNN units are not effective in capturing long-term dependencies. On the other hand, transformer and convolutional-based models are capable of handling long-term dependencies, but they assume a uniform temporal distribution. To address these challenges, authors in [36] propose a transformer-based architecture that models the dynamics of temporal point processes using a continuous conditional events intensity function. Additionally, in [37], a long convolutional kernel with weights, conditioned on event samples intensity, is prpposed. This parameterization enables the handling of non-uniformly sampled and irregularly-sampled datasets.

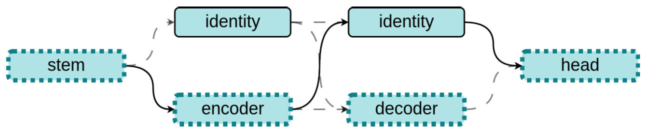

FIGURE 2. The general layout of our search space. Dotted borders indicate that blocks contain searchable operations. Dashed lines indicate that connections between nodes are searchable. The solid line is an example of selected architecture.

III. METHODOLOGY

A. SEARCH SPACE DESIGN

The general layout of all the blocks in our search space is illustrated in Figure 2. There are four main blocks in the search space:

- **Stem** *(always present)* has a searchable structure depicted in Figure 3.
- **Encoder** *(optional)* is searchable with multiple layers, whose structure is depicted in Figure 5.
- **Decoder** *(optional)* has a searchable number of layers.
- **Head** *(always present)* has a searchable structure depicted in Figure 4.

Here, the term *optional* denotes that the presence of a particular block is determined during the search procedure. For instance, the minimal architecture would consist only of the **Stem** and **Head** blocks. Now we describe earch architecture block in more details.

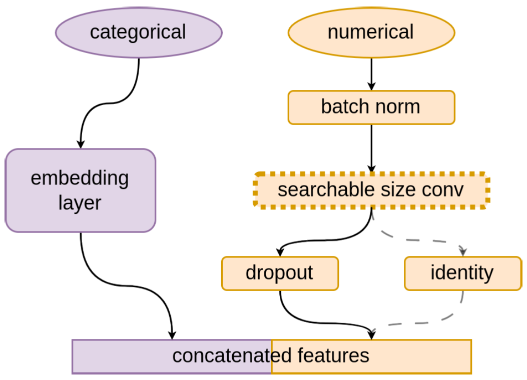

FIGURE 3. Searchable part of Stem block is depicted with dashed and dotted lines. Convolutional layers with different kernels and the presence of dropout are selected at each search step. A solid line is an example of a selected path.

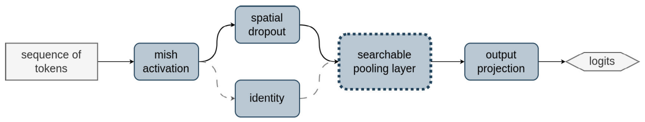

FIGURE 4. There are two searchable pooling layers in Head Block: Max pooling and Average pooling. The type of a pooling layer and the presence or absence of spatial dropout are determined by the search procedure. A solid line is an example of a selected path.

1) STEM

The Stem fuses categorical and numerical features from input data into one vector as depicted in Figure 3.

The Stem pipeline is threefold:

- Categorical features are encoded using an embedding layer, and the size of the embedding is automatically determined by the formula: $\min(600, \mathrm{round}(1.6 \times N^{0.56}))$. Where N is a sequence length.
- Numerical features are processed using BatchNorm [38]; afterwards, convolution with a searchable kernel size is applied to each numerical input along the temporal dimension, and optionally, dropout may be applied after convolution.
- Finally, all embeddings of all types are concatenated along the feature dimension to obtain the input sequence.

2) ENCODER

Encoder brings the most variability into the search space. Common operations are available in the encoder: Multi-Head Self-Attention (MHA) [39], Gated Reccurent Unit (GRU) [40], and Convolution. Each of these operations entails different assumptions about the nature of the data: RNN units assume a sequential nature of the data, convolutional layers are effective at capturing temporally local correlations, and transformers excel at capturing long-term dependencies throughout the entire sequence.

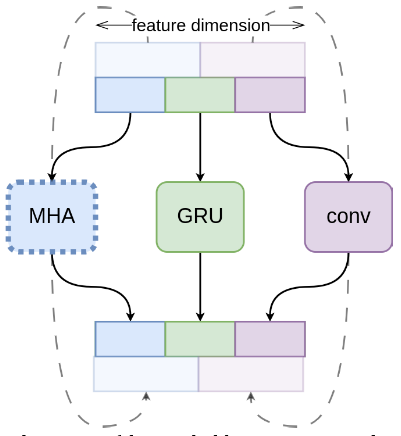

FIGURE 5. Encoder Layer with searchable MHA, GRU and conv operations. A combination of one, two, or three operations can be selected during each search step. Different combinations are selected on different layers. Incoming features are divided into several selected operations. An example combination with MHA, GRU and conv operations is depicted with solid lines, and an example combination with MHA and conv operations is depicted with dashed lines. Dotted border around MHA indicates that it has a searchable number of heads.

Encoder has both a searchable number of layers and the operations within each layer. Input of each Encoder layer is divided into one to three blocks along the feature dimension, which can be processed using one of six potential operations, such as MHA, GRU, or Convolution. The number of heads in MHA is searchable and is chosen from the set $\{1, 2, 4, 8\}$. In total it provides up to 19 variations for a single Encoder layer. It is worth noting that each layer has a distinct set of operations. The outputs from each block are concatenated and sent to the next layer within the Encoder. This structure is illustrated in Figure 5. The encoder may have three possible values for the number of layers (1, 2, or 4), resulting in approximately 130×10^3 possible Encoder variations.

3) TEMPORAL ENCODING

The traditional positional encoding uses the token position in the sequence to obtain the embedding [39]. However, in the case of non-uniformly spaced event sequences, this traditional positional embedding is inadequate for capturing the relative arrangement of events. Therefore, in line with the approach proposed in Transformer Hawkes process [36], we utilize temporal encoding to address this limitation.

$$TE(t, 2i) = \sin\left(t/10000^{2i/d}\right),$$
$$TE(t, 2i+1) = \cos\left(t/10000^{2i/d}\right),$$

where t is an event time normalized to [0, 1], i — dimension, and d is the embedding size.

4) DECODER

In the decoder block, we utilize a standard transformer architecture with a searchable number of heads and layers without recurrent or convolutional layers used in the encoder. We adopt the transformer variant proposed in [41] with Sub-LayerNorm and same weights initialization.

5) HEAD

Head aggregates the sequence of feature vectors to produce the final classification as depicted in Figure 4. First, optional spatial dropout operation is performed [42]. Next, the sequence of tokens is aggregated into a single feature vector, aggregation is also a searchable operation consisting from either the maximum operation, averaging, or a combination of both. Finally, the feature vector is projected to obtain the final logits.

B. THE ARCHITECTURE VECTORIZER

To extract the architecture features for a *Predictor-model* - the model designed to predict architecture performance, we focus on a group of path-based encoders described in [29]. According to [29] Path-based encoders outperform the adjacency matrix-based ones for the *Predictor-model* setting. Our encoding is done as follows. A binary variable is assigned to each block, layer and the particular operation in each layer. If the block or layer is not involved in the architecture, the corresponding binary variable, and all variables responsible for the operations inside the block or layer are set to zero. Finally, all variables are concatenated to obtain the feature vector. For simplicity we call our architecture encoding scheme — **AVec**.

C. THE SEARCH PROCEDURE

With the reasoning presented in Section II-D, we have opted to use a bayesian optimization similar to [28]. However, instead of an ensemble of DNNs we employed **CatBoost** [43] to obtain predictions and corresponding uncertainty estimates. This alteration from [28] has enabled us to leverage the benefits that **CatBoost** offers over the ensemble of DNNs, a more precise uncertainty estimation following a theoretically justified approach [43]. We analyze this choice in Section IV-D2. The search procedure is outlined in Algorithm 1. There are **three main components of the search process**: 1) Architecture vectorizer — **AVec**, 2) Score prediction and uncertainty estimation — *Predictor-model* as CatBoost, 3) Candidate selection — *Thompson sampling*.

Initially, a set of N_{init} architectures is randomly sampled from the search space A. After training all of them, actual performance scores are obtained for each architecture. Next, the *Spredictor* is trained using the architecture features and actual scores. Architecture features are obtained with our architecture vectorizer — **AVec**. Then, the trained $Predictor-model$ is used to estimate scores and associated uncertainties for new randomly sampled N_{iter} architectures. It is crucial to balance the exploration-exploitation trade-off during the search process. To achieve the balance we use Thompson sampling with estimated scores and uncertainties. $L_{candidates}$ architectures are sampled for further trainig. These steps are repeated until the allocated budget is met as described in Algorithm 1.

For further technical details on each parameter, Section A provides a detailed explanation.

D. ENSEMBLE OF TEACHERS

Our search procedure involves training many models. By combining these models as an ensemble of teachers, we are able to leverage the benefits of this approach. In [44], authors demonstrated that a weak teacher ensemble could lead to improved student performance. This observation allows us to construct an ensemble of best-performing models at a current search step. We use an average of different models predictions as a teacher model. Before enabling distillation loss, we train a total of 30 architectures. On each search iteration all models predictions for all training examples are being cached to avoid computational complexity. At each new iteration, we update members of the ensemble by selecting *topK* best performing models from our cached predictions. We use ensembles of three models in most of our experiments.

To compute the model-to-model loss, we opted for Mean Squared Error (MSE) instead of Kullback–Leibler divergence. This choice allows us to avoid using additional temperature hyper-parameter [45].

IV. EXPERIMENTS AND RESULTS

A. DATASETS

We utilize six publicly available datasets consisting of event sequences sourced from different data science competitions and prior studies. These sequential datasets were carefully

[2]https://www.kaggle.com/competitions/amex-default-prediction/overview/evaluation

TABLE 1. Statistics of sequential datasets used for our analysis.

	Amex	ABank	VBank	AGE	RBchurn	Taobao
Number of observations / Transactions	5531451	270450066	7636113	26450577	490513	7893060
An average number of observations per user	12	280	897	881	98	806
STD for the number of observations per user	2.68	270	371	124	78	1042
Number of users	458913	963812	8509	30000	5000	9791
Number of classes	2	2	2	4	2	2
Number of categorical features	11	16	2	1	1	2
Number of real-valued features	177	3	1	1	5	0
Class balance	0.26	0.01	0.75	0.25 (equal)	0.65	0.6
Target	Default	Default	Education	Age group	Churn	Payment

TABLE 2. Comparison of our method with two NAS procedures 1) AutoAttend [13], 2) TextNAS [14] and four fixed architectures 3) Gated Transformer Networks [16], and baseline models such as 4) Fixed Transformer, 5) GRU, 6) LSTM. We report MEAN and STD of the 3 best models found, for both HPO and NAS procedures. We mark the First and the Second best performing models as highlighted in this text.

Dataset	Model Search Space	SeqNAS	AutoAttend	TextNAS	GTN	Fixed TF	GRU	LSTM
	Metric / Search Method	Our	Context-Aware Weight Sharing	ENAS	HPO	HPO	HPO	HPO
AmEx	Custom[2]	$\mathbf{0.7911 \pm 0.0004}$	0.6170 ± 0.0033	0.7818 ± 0.002	0.7717 ± 0.006	0.7850 ± 0.0002	0.7718 ± 0.0005	0.7709 ± 0.0005
ABank	ROC-AUC	$\mathbf{0.7963 \pm 0.0014}$	0.6827 ± 0.0160	0.7653 ± 0.002	0.7462 ± 0.001	0.7747 ± 0.0011	0.7699 ± 0.0002	0.7451 ± 0.0032
VBank	ROC-AUC	$\mathbf{0.8032 \pm 0.0022}$	0.6533 ± 0.0408	0.7951 ± 0.001	0.7362 ± 0.001	0.7883 ± 0.0013	0.7980 ± 0.0008	0.7704 ± 0.0012
RBchurn	ROC-AUC	$\mathbf{0.8525 \pm 0.0033}$	0.7345 ± 0.0028	0.7936 ± 0.002	0.7701 ± 0.003	0.8170 ± 0.0012	0.8300 ± 0.002	0.8090 ± 0.0027
AGE	Accuracy	$\mathbf{0.6445 \pm 0.0018}$	0.6251 ± 0.0013	0.6016 ± 0.003	0.5363 ± 0.019	0.6170 ± 0.001	0.6300 ± 0.001	0.5920 ± 0.0010
TaoBao	ROC-AUC	$\mathbf{0.7138 \pm 0.0007}$	0.6352 ± 0.0023	0.7079 ± 0.002	0.6713 ± 0.001	0.7107 ± 0.0011	0.7100 ± 0.0004	0.6680 ± 0.0008

selected to include both categorical and real-valued features. In each dataset, a sequence of events is provided as an input to predict a categorical target, making it a classification task. Detailed statistics and target regarding each dataset can be found in Table 1.

1) BANK TRANSACTION DATA

VBank, **AmEx**, **AGE**, **RBchurn** and **ABank** datasets consist of card transactions, financial records, and other user-related data. Leveraging these datasets, we utilize event sequences to predict specific targets such as default events, churn, user higher education, age and etc. Mainly each transaction is characterized by its date, type, amount, and Merchant Category Code.

2) TAOBAO

Taobao dataset is a subset of the Taobao APP user behavior data, comprising millions of items recorded over one month. The dataset is organized in a user-item interaction format, consisting of user ID, item ID, category ID, behavior type, and timestamp.

To suit the context of our task, we preprocess the dataset by excluding the item ID for simplicity. Additionally, we merge all categories that appear less than 500 times in the dataset into a single category. This preprocessing step allows us to reduce the number of unique categories from 8,900 to 1,900. We focus on the client's behavior within a 7-day window to predict whether they will make a payment in the following 7 days.

No manual feature generation or preprocessing was conducted on most of the datasets, except the Taobao and AmEx datasets. In the case of the AmEx dataset, we utilized a cleaner version obtained from the Kaggle competition platform.

To create train, test, and validation sets, we performed a random split for each dataset. The split ratios were set to 0.6 for the training set, 0.2 for the test set, and 0.2 for the validation set, based on the total sample size. Sequences shorter than specified in Table 1 were padded with zeros; for sequences longer than specified, we took N last events, where N is the sequence length specified.

B. METHODS

We compared our results with two NAS approaches, namely **AutoAttend** [13] and **TextNAS** [14]. However, we had to make some modifications to adapt these methods for the **EvS** domain. These modifications are described in Appendix A. Furthermore, we also compared our approach with fixed architectures such as GRU, LSTM, and Transformer. To ensure optimal performance, we performed Hyper-Parameter Optimization (HPO) for all fixed architectures. The details of the HPO can be found in Appendix A.

C. RESULTS

Our main results are presented in Table 2. **SeqNAS** outperforms all other methods. The second place is shared by fixed architectures: **Transformer** and **GRU**. We further analyze importance of **MHA** and **RNN** as building blocks of our search space in Section IV-D3 and show that all blocks are complementary to each other.

Algorithm 1 *Predictor − model* — Model Score Predictor With Parameters θ, N_{init} — Initial Number of Architectures to Train, N_{iter} — Number of Architectures to Sample During Each Iteration, $L_{candidates}$ — Number of Architectures to Train During Each Iteration, *Ensemble* — Ensmbling Function, M — Number of Iterations, $\hat{X}, S$ — Predicted Scores and Corresponding Uncertainties

```
1:  $K_1 \leftarrow Sample(N_{init})$, sample random architectures from the search space.
2:  $TrainedArches \leftarrow Train(K_1)$. Train all architectures in $K_1$ and obtain their scores, $X$ is a set of scores from all trained models, $X_i$ are scores for a current iteration.
3:  $ArchFeatures \leftarrow AVec(TrainedArches)$, encode architectures into features.
4:  $\min_\theta(MAE(Predictor - model(ArchFeatures; \theta), X))$, train a score predictor model.
5:  for $i = 1, 2, \ldots, M$ do
6:      $K_{1+i} \leftarrow Sample(N_{iter})$, sample random architectures from the search space such that $K_{i+1} \cap TrainedArches = \emptyset$.
7:      $\hat{X}, S = Predictor - model(K_{i+1}; \theta)$, predict scores and score uncertainties.
8:      Select $L_{candidates}$ architectures from $K_{1+i}$ with Thomson sampling using obtained uncertainties $S$.
9:      $T \leftarrow topK(TrainedArches)$, select the best performing teacher models from already trained models and obtain an ensemble of teachers $Ensemble(T)$.
10:     Train all models in $L_{candidates}$ with distillation loss and $Ensemble(T)$ and obtain actual scores $X_i$.
11:     $TrainedArches \leftarrow TrainedArches \cup L_{candidates}$.
12:     $X \leftarrow X \cup X_i$.
13:     $ArchFeatures = AVec(TrainedArches)$, encode architectures into features.
14:     Update a score predictor model $\theta \leftarrow \arg\min_\theta L(\theta)$, where $L(\theta) = MAE(Predictor - model(ArchFeatures; \theta), X)$.
15: end for
16: Select the best architecture from $TrainedArches$ according to some performance metric.
```

Our experiments show that search spaces from related domains, such as text, do not always transfer well to **EvS** and sometimes underperform even simple models such as RNN.

A potential disadvantage of **SeqNAS** is its longer training time compared to other approaches, as shown in Table 5. However, as discussed in Section I, it is still a reasonable time complexity given significant gains for various applications.

D. ABLATION STUDIES

1) ENSEMBLE OF TEACHERS

In Figures 6 and 7 we demonstrate the results of the search procedure with and without Knowledge Distillation (KD) on two datasets, AmEx and RBchurn correspondingly. We show the average scores of the top 3 models over the search steps. It can be seen that KD significantly improves performance metrics for both datasets. The same observations can be seen for other datasets in Table 3, where we demonstrate the final metrics for the single best architecture using a random search procedure.

We experimented with different approaches to (1) select diverse teachers and (2) combine their predictions into an

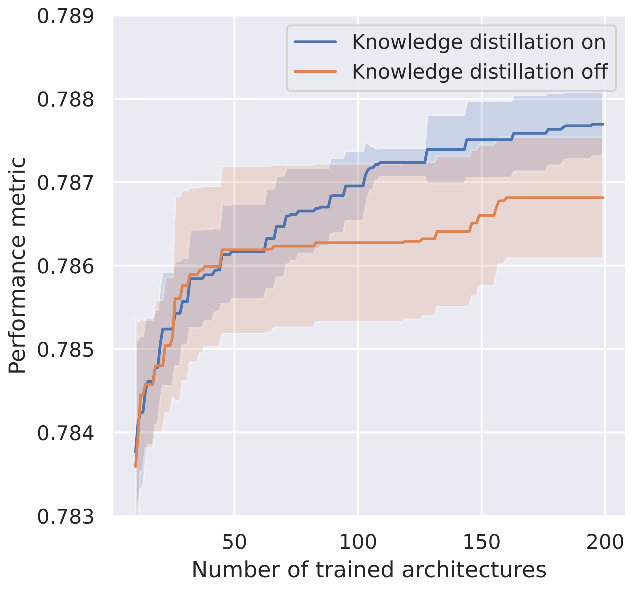

FIGURE 6. Search performance of SeqNAS with and without KD on AmEx dataset over a number of trained architectures, for 200 architectures in total. Results are averaged over 3 best performing models as a sliding window. Performance is measured with a metric specified in Table 2 for AmEx dataset. KD is employed after 30 models were trained. It can be seen that lines start to diverge approximately after 60th iteration.

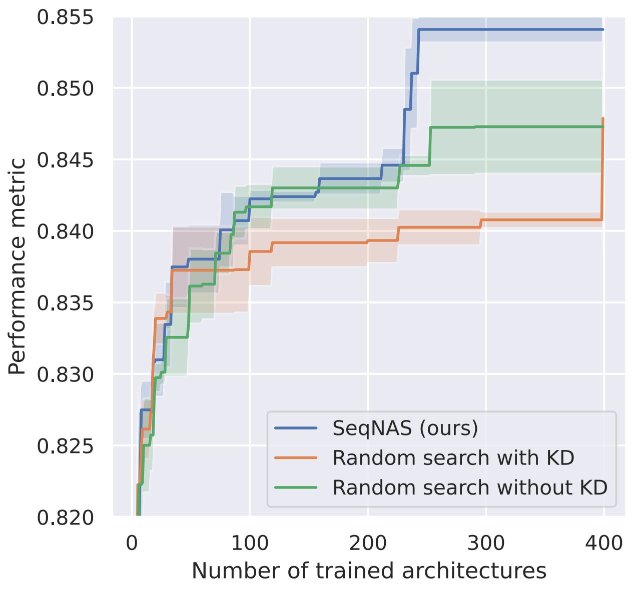

FIGURE 7. Comparison of SeqNAS, Random search with KD and Random search without KD on RBchurn dataset over a number of trained architectures, for 400 architectures in total. Results are averaged over 3 best performing models as a sliding window. Performance is measured with a metric specified in Table 2 for RBchurn dataset. KD is employed after 30 models were trained.

TABLE 3. Effect of knowledge distillation on search score using random search.

Dataset	Metric	KD	Score
VBank	ROC-AUC	on	**0.7997**
		off	0.7986
AmEx	Custom	on	**0.7889**
		off	0.7876
Insect	Accuracy	on	**0.6164**
		off	0.6122

ensemble. We found that averaging the predictions of 3 best performing models resulted in the best performance. During training, we had access to hundreds of trained models, and we believe there are further opportunities for improvement and potential applications in this direction.

2) CatBoost VS. DNN PREDICTIOR

We evaluate two types of models for architecture scoring and uncertainty estimation (*Predictor* − *model*) in Figure 8. The results demonstrate that the model based on CatBoost outperforms the one based on an ensemble of DNNs with slightly superior performance. We used an ensemble of eight models.

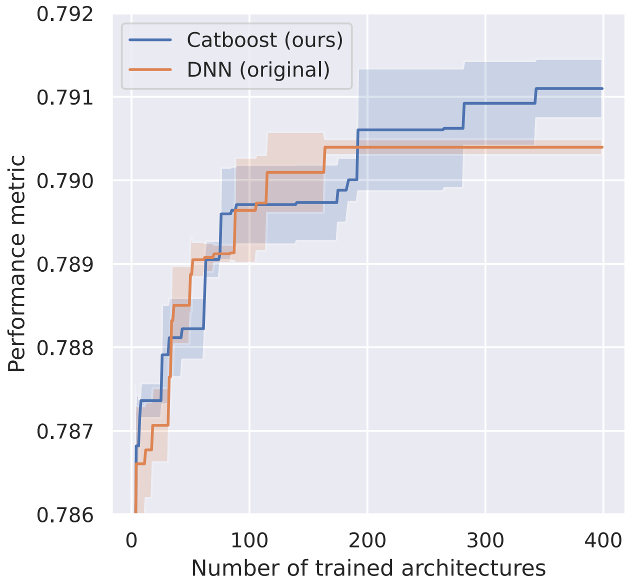

FIGURE 8. Comparison of different predictors on AmEx dataset: CatBoost [46] and an ensemble of DNNs originally proposed in BANANAS [28]. Results are averaged over 3 best performing models as a sliding window. Performance is measured with a metric specified in Table 2 for AmEx dataset.

3) ENCODER BLOCKS IMPORTANCE: MHA, GRU, OR CONVOLUTIONS

To better understand the roles of MHA, GRU, and Convolutions in the encoder layers, we conducted a search procedure where we removed one of these blocks at a time. As shown in Figure 9, models without GRU block exhibited a significant drop in performance compared to those with all three blocks

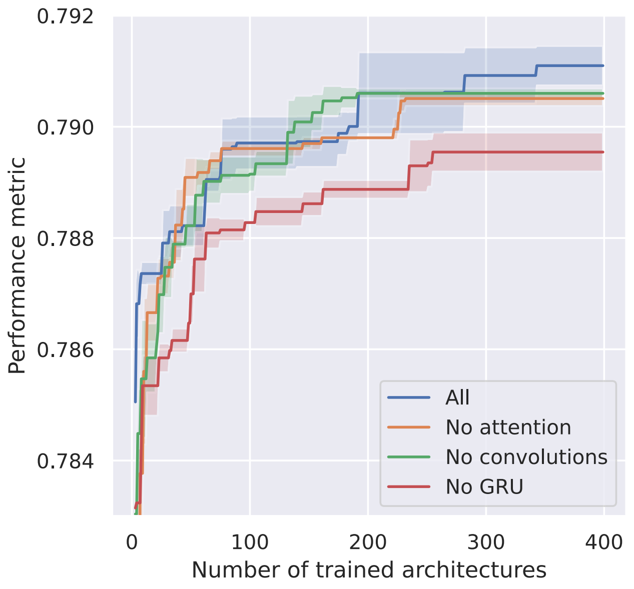

FIGURE 9. Search performance of SeqNAS without different blocks in Encoder layer and with all blocks included — *ALL* on AmEx dataset. Results are averaged over 3 best performing models as a sliding window. Performance is measured with a metric specified in Table 2 for AmEx dataset. We see that different types of operations complement each other.

present. Unsurprisingly, these results can be explained by the good performance of GRU model alone presented in Table 2. Nonetheless, it is worth noting that the relative importance of each block varies depending on the dataset used.

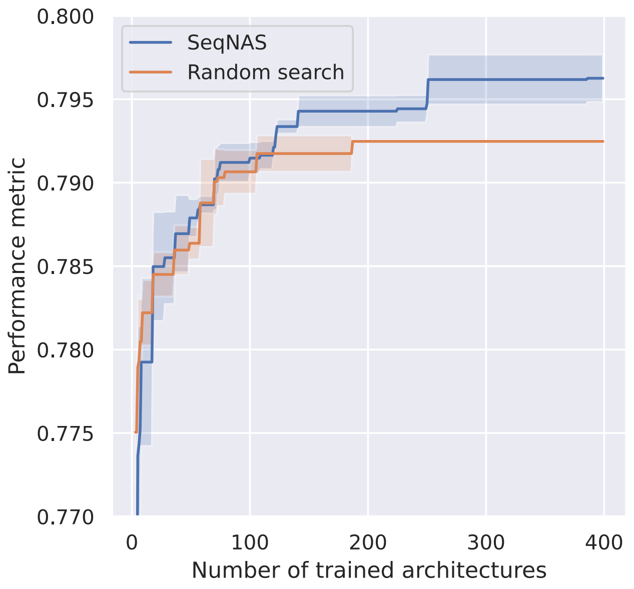

FIGURE 10. Search performance of SeqNAS and Random search with KD. Results are averaged over 3 best performing models as a sliding window. Performance is measured with a metric specified in Table 2 for ABank dataset. Both SeqNAS and Random search use the ensemble of teachers.

4) RANDOM SEARCH VS SEQNAS

In Figures 10 and 7, we compare **SeqNAS** and Random search procedure for two datasets, AmEx and RBchurn correspondingly. In our settings, we first train 100 randomly sampled architectures and then fit *Predictor* − *model* to score new candidates. It can be seen that **SeqNAS** starts to outperform Random search approximately after 100 steps for both datasets.

E. NAS-BENCH EVENT SEQUENCES

Analogous to the precomputed NAS benchmarks [15], [31], [32], we release a dataset with trained and evaluated architectures. The details can be found in Appendix C.

V. DISCUSSION

- The search space of **TextNAS** contains same operations as **SeqNAS**. However, **TextNAS** differs from **SeqNAS** in terms of graph topology and search procedure. **TextNAS** performs worse than both **SeqNAS** and fixed architectures with HPO, raising questions about the impact of search space design or search method, such as ENAS.
- It's important to note that **SeqNAS** does not outperform **ROCKET** for univariate time series classification. Many recent methods that perform well on UCR datasets utilize specific convolutional operations or fixed feature generation. Thus, our search space could benefit from incorporating new searchable operations proposed in various works for UTS classification.
- There are potential improvements that can be made based on TPP modeling works, such as better estimation of event densities [36] or the use of temporally parameterized long convolutions [37].
- While our results highlight the importance of the GRU unit in **EvS** modeling, this observation may be specific to our datasets. It's possible that with larger datasets, Transformer-based blocks may offer better performance. Different training strategies, like applying causal masks, may also help improve Transformer performance.
- Currently, our method may produce over-parameterized models without considering hardware constraints. Further improvement can be done to develop a more computationally efficient approach.
- The literature currently lacks a NAS procedure that features a broad search space suitable for a wide variety of tasks, referred to as the **Universal Search Space.** Moreover, we see that the generalization ability of existing search methods is limited even across similar domains.

VI. CONCLUSION

In this paper, we introduce **SeqNAS**, a novel method for automatically searching neural architectures specifically designed for **EvS** data. Our approach outperforms other NAS methods and standard architectures with hyper-parameter optimization in the **EvS** domain. We demonstrate the versatility of our method by applying it to various datasets. To the best of our knowledge, our work represents the first extensive exploration of NAS for **EvS**.

We show that in our search space different types of operations complement each other, leading to the discovery of improved architectures. There is no architecture which performs better without one of the operations: MHA, RNN unit, or convolution.

Our approach combines knowledge distillation with sequential Bayesian Optimization to achieve significant performance improvements in a computationally efficient way.

Additionally, we establish a benchmark for **EvS** classification by comparing different models and techniques. This benchmark can serve as a valuable resource for researchers looking to advance the field of **EvS** classification.

We release the **NAS-BENCH Event Sequences** dataset, which includes architectures and corresponding scores, to support research on predictor-based NAS methods.

APPENDIX A
TECHICAL DETAILS

In **SeqNAS**, for all of our datasets, we used the following hyper-parameters parameters: $N_{init} = 100$, $N_{iter} = 100$, $M = 40$ and $L_{candidates} = 15$. For more details regarding each dataset, please refer to our repository.

To evaluate various NAS methods, including **TextNas**, **AutoAttend**, and **GTN**, we used the hyper-parameters and search procedures outlined in their original papers.

For **TextNas** and **AutoAttend**, we incorporated ur Stem blocks to combine real-valued and categorical features. Similarly, for **GTN**, we added embedding layers for categorical features but chose not to utilize convolutional layers for real-valued features, as they were not utilized in the original paper.

For fixed transformer architecture, we used a simple model with two MHA layers in both Encoder and Decoder, with 8 heads in each. LSTM and GRU models consisted of only one RNN layer, with Stem and Head blocks. Fixed models were optimized using hyper-parameters optimization.

For all models, we used the identical fixed structure for Stem and Head blocks described in our architecture. In the Stem block, we did not use dropout and set fixed convolutional layer kernel size to 3×3. For Head, we fixed the spatial dropout rate at 0.3, and for pooling we used Max pooling.

For HPO, we employed Optuna [47] and optimized the following hyper-parameters for 30 iterations: 1) batch size, 2) optimizer type, 3) learning rate, 4) weight decay, 5) embedding size, 6) linear layer size for Transformer or hidden size for RNN, and 7) dropout rate.

APPENDIX B
ADDITIONAL EXPERIMENTS WITH TIME-SERIES CLASSIFICATION

Additionally, we assessed the performance of our classification method on **UTS** (Univariate Time Series) using datasets obtained from the UCR archive [48], specifically the

TABLE 4. Comparison of UTS classification for datasets from USR archive with ROCKET, ROC-AUC is computed for both datasets.

Dataset	SeqNAS	ROCKET [3]	Train size	Test size
InsectSound	**0.797**	0.7823	25000	25000
ElectricDevices	0.73	**0.7305**	8926	7711

TABLE 5. Approximate search cost in GPU hours for SeqNAS, TextNAS and GRU with HPO. SeqNAS used 400 iterations. TextNAS uses ENAS for architecture search, which is significantly faster. For HPO details, we refer to Section A.

Dataset	SeqNas	TextNAS	GRU - HPO
RBchurn	60	1.2	2.1
Taobao	60	7.55	2.25
AGE	80	24.2	2.88

TABLE 6. In this table, we demonstrate the distribution of architectures among various datasets and methods. The architectures are presented based on our final search procedure as well as randomly queried ones.

Method / Dataset	RBchurn	ABank	AmEx	AGE	VBank	TaoBao
Ours	400	400	400	400	400	400
Random	400	400	-	-	-	-

InsectSound and ElectricDevices datasets. Comprehensive descriptions of these datasets are publicly available through the UCR archive. We evaluate the performance of **SeqNAS** on Univariate Time Series (UTS) classification against two datasets from the UCR archive [48]. Our results, as displayed in Table 4, demonstrate that SeqNAS produces reasonable results for UTS classification due to its flexible search space. However, it should be noted that **SeqNAS** performance suffers when used with small-size datasets from the UCR archive.

APPENDIX C
NAS-BENCH EVENT SEQUENCES

Our dataset consists of 3200 architectures obtained on six different datasets. Out of the total architecture pool, 800 architectures were randomly queried from our search space, while 2400 architectures were queried using our search procedure. The distribution of these architectures across datasets and methods can be found in Table 6. For each architecture, we provide the best score achieved across all epochs, along with its feature vector encoded using **AVec**. The corresponding metrics for each score are listed in Table 2. All architectures within a dataset were trained for an equal number of epochs.

We specifically present architectures obtained through random search and our search procedure, as the architectures found through our procedure tend to have better performance metrics. This bias towards better performance can be observed in Figure 11, which shows the distribution of

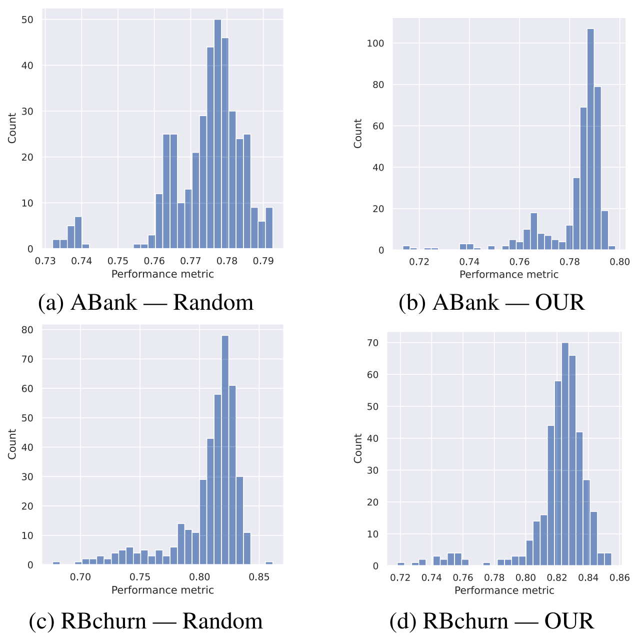

FIGURE 11. We present the distributions of queried architectures on two datasets using both our search procedure and random search. The distribution labeled with OUR represents our search procedure. It can be seen that architectures queried with OUR procedure are slightly shifted towards higher performance metrics, this shift is more significant for ABank dataset.

queried architectures on two datasets using both our search procedure and random search.

[3]Results are obtained from [49].

REFERENCES

[1] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artif. Intell. Med.*, vol. 104, Apr. 2020, Art. no. 101822.
[2] J. A. Valeri, L. R. Soenksen, K. M. Collins, P. Ramesh, G. Cai, R. Powers, N. M. Angenent-Mari, D. M. Camacho, F. Wong, T. K. Lu, and J. J. Collins, "BioAutoMATED: An end-to-end automated machine learning tool for explanation and design of biological sequences," *Cell Syst.*, vol. 14, no. 6, pp. 525–542.e9, Jun. 2023.
[3] A. Liguori, L. Caroprese, M. Minici, B. Veloso, F. Spinnato, M. Nanni, G. Manco, and J. Gama, "Modeling events and interactions through temporal processes—A survey," 2023, *arXiv:2303.06067*.
[4] S. Shao, R. Yan, Y. Lu, P. Wang, and R. X. Gao, "DCNN-based multi-signal induction motor fault diagnosis," *IEEE Trans. Instrum. Meas.*, vol. 69, no. 6, pp. 2658–2669, Jun. 2020.
[5] I. Giurgiu and A. Schumann, "Explainable failure predictions with RNN classifiers based on time series data," 2019, *arXiv:1901.08554*.
[6] H. Jain, A. Khunteta, and S. Srivastava, "Telecom churn prediction and used techniques, datasets and performance measures: A review," *Telecommun. Syst.*, vol. 76, no. 4, pp. 613–630, Apr. 2021.
[7] R. Sudharsan and E. N. Ganesh, "A swish RNN based customer churn prediction for the telecom industry with a novel feature selection strategy," *Connection Sci.*, vol. 34, no. 1, pp. 1855–1876, Dec. 2022.
[8] M. Carnein and H. Trautmann, "Customer segmentation based on transactional data using stream clustering," in *Advances in Knowledge Discovery and Data Mining*. Cham, Switzerland: Springer, 2019, pp. 280–292.
[9] Y. Xie, G. Liu, C. Yan, C. Jiang, and M. Zhou, "Time-aware attention-based gated network for credit card fraud detection by extracting transactional behaviors," *IEEE Trans. Computat. Social Syst.*, vol. 10, no. 3, pp. 1004–1016, Jun. 2022.
[10] E. B. Boukherouaa, M. G. Shabsigh, K. AlAjmi, J. Deodoro, A. Farias, E. S. Iskender, M. A. T. Mirestean, and R. Ravikumar, *Powering the Digital Economy: Opportunities and Risks of Artificial Intelligence in Finance*. Washington, DC, USA: International Monetary Fund, 2021.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. Neural Inf. Process. Syst.*, 2014, pp. 1–12.
[12] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in Artificial Intelligence*, vol. 115, 2020, pp. 367–377.
[13] C. Guan, X. Wang, and W. Zhu, "AutoAttend: Automated attention representation search," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 3864–3874.
[14] Y. Wang, Y. Yang, Y. Chen, J. Bai, C. Zhang, G. Su, X. Kou, Y. Tong, M. Yang, and L. Zhou, "TextNAS: A neural architecture search space tailored for text representation," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 5, pp. 9242–9249.
[15] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, A. Filippov, and E. Burnaev, "NAS-Bench-NLP: Neural architecture search benchmark for natural language processing," *IEEE Access*, vol. 10, pp. 45736–45747, 2022.
[16] M. Liu, S. Ren, S. Ma, J. Jiao, Y. Chen, Z. Wang, and W. Song, "Gated transformer networks for multivariate time series classification," 2021, *arXiv:2103.14438*.
[17] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining Knowl. Discovery*, vol. 34, no. 5, pp. 1454–1495, Sep. 2020.
[18] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.
[19] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10726–10734.
[20] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019.
[21] Y. Fu, W. Chen, H. Wang, H. Li, Y. Lin, and Z. Wang, "AutoGAN-distiller: Searching to compress generative adversarial networks," in *Proc. ICML*, 2020, pp. 3292–3303.
[22] Y. Wu, Z. Huang, S. Kumar, R. S. Sukthanker, R. Timofte, and L. Van Gool, "Trilevel neural architecture search for efficient single image super-resolution," in *Proc. Comput. Vis. Pattern Recognit.*, 2021.
[23] E. Shvetsov, D. Osin, A. Zaytsev, I. Koryakovskiy, V. Buchnev, I. Trofimov, and E. Burnaev, "QuantNAS for super resolution: Searching for efficient quantization-friendly architectures against quantization noise," 2022, *arXiv:2208.14839*.
[24] N. Mazyavkina, S. Moustafa, I. Trofimov, and E. Burnaev, "Optimizing the neural architecture of reinforcement learning agents," in *Intelligent Computing*, vol. 2. Cham, Switzerland: Springer, 2021, pp. 591–606.
[25] I. Trofimov, N. Klyuchnikov, M. Salnikov, A. Filippov, and E. Burnaev, "Multi-fidelity neural architecture search with knowledge distillation," *IEEE Access*, vol. 11, pp. 59217–59225, 2023.
[26] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, "Rethinking architecture selection in differentiable NAS," in *Proc. Int. Conf. Learn. Represent.*, 2021.
[27] A. Zela, J. N. Siems, L. Zimmer, J. Lukasik, M. Keuper, and F. Hutter, "Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks," in *Proc. 10th Int. Conf. Learn. Represent.*, 2022, pp. 1–36.
[28] C. White, W. Neiswanger, and Y. Savani, "BANANAS: Bayesian optimization with neural architectures for neural architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 12, pp. 10293–10301.
[29] C. White, W. Neiswanger, S. Nolen, and Y. Savani, "A study on encodings for neural architecture search," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 20309–20319.
[30] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, and Y. Lin, "HW-NAS-Bench: Hardware-aware neural architecture search benchmark," in *Proc. Int. Conf. Learn. Represent.*, 2021.
[31] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
[32] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2020.
[33] A. Zela, J. Siems, L. Zimmer, J. Lukasik, M. Keuper, and F. Hutter, "Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks," 2020, *arXiv:2008.09777*.
[34] A. Mehrotra, A. G. C. P. Ramos, S. Bhattacharya, Ł. Dudziak, R. Vipperla, T. Chau, M. S. Abdelfattah, S. Ishtiaq, and N. D. Lane, "NAS-Bench-ASR: Reproducible neural architecture search for speech recognition," in *Proc. Int. Conf. Learn. Represent.*, 2020.
[35] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1555–1564.
[36] S. Zuo, H. Jiang, Z. Li, T. Zhao, and H. Zha, "Transformer Hawkes process," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 11692–11702.
[37] D. W. Romero, A. Kuzina, E. J. Bekkers, J. M. Tomczak, and M. Hoogendoorn, "CKConv: Continuous kernel convolution for sequential data," in *Proc. Int. Conf. Learn. Represent.*, 2022.
[38] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 2488–2498.
[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6000–6010.
[40] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.
[41] H. Wang, S. Ma, S. Huang, L. Dong, W. Wang, Z. Peng, Y. Wu, P. Bajaj, S. Singhal, A. Benhaim, B. Patra, Z. Liu, V. Chaudhary, X. Song, and F. Wei, "Foundation transformers," 2022, *arXiv:2210.06423*.
[42] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 648–656.
[43] A. Malinin, L. Prokhorenkova, and A. Ustimenko, "Uncertainty in gradient boosting via ensembles," in *Proc. Int. Conf. Learn. Represent.*, 2021.
[44] G. Kaplun, E. Malach, P. Nakkiran, and S. Shalev-Shwartz, "Knowledge distillation: Bad models can be good role models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 28683–28694.
[45] T. Kim, J. Oh, N. Kim, S. Cho, and S.-Y. Yun, "Comparing Kullback–Leibler divergence and mean squared error loss in knowledge distillation," 2021, *arXiv:2105.08919*.
[46] A. V. Dorogush, V. Ershov, and A. Gulin, "CatBoost: Gradient boosting with categorical features support," in *Proc. Neural Inf. Process. Syst. Conf.*, 2018.
[47] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2623–2631.
[48] H. A. Dau, A. Bagnall, K. Kamgar, C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The UCR time series archive," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 6, pp. 1293–1305, Nov. 2019.
[49] A. Dempster, D. F. Schmidt, and G. I. Webb, "Hydra: Competing convolutional kernels for fast and accurate time series classification," *Data Mining Knowl. Discovery*, vol. 37, no. 5, pp. 1779–1805, Sep. 2023.

IGOR UDOVICHENKO received the B.S. degree in applied mathematics and informatics from Lomonosov Moscow State University, Moscow, Russia, in 2021. Since 2020, he has been involved in various researches in the field of deep learning and quantitative finance. He is currently a Research Engineer with the Applied AI Center, Skolkovo Institute of Science and Technology. His current research has primarily focused on neural architecture search and optimal transport.

EGOR SHVETSOV received the B.S. and M.S. degrees in applied physics and material science from Tomsk Polytechnic University, in 2008 and 2010, respectively. Since 2020, he has been a Research Engineer with the Applied AI Center, Skolkovo Institute of Science and Technology. His current research has primarily focused on neural architecture search and computationally efficient deep learning.

DENIS DIVITSKY received the B.S. degree in information security from the National Research Nuclear University MEPhI, Moscow, Russia, in 2023. From 2020 to 2022, he was an Intern with the Huawei Noah's Ark Laboratory. From 2022 to 2023, he was a Research Engineer with the Skolkovo Institute of Science and Technology. Since 2023, he has been a Research Engineer with Tinkoff, Moscow. His research interests include neural architecture search, reinforcement learning, and automatic speech recognition. He was a Winner (second place) of a Data Science Hackathon by Tinkoff and McKinsey, in 2019.

DMITRY OSIN received the B.S. degree in applied mathematics and informatics from Lomonosov Moscow State University, Moscow, Russia, in 2021. Since 2020, he has been a Research Engineer with the Skolkovo Institute of Science and Technology. His main research interests include neural architecture search, quantization, model compression, representation learning, super resolution, and sequential data.

ILYA TROFIMOV received the M.Sc. degree in theoretical physics from Moscow State University, in 2006, and the Ph.D. degree in computer science from the Federal Research Center "Computer Science and Control" of RAS, in 2018. He is currently a Research Scientist with the Skolkovo Institute of Science and Technology. His main research interests include large-scale machine learning, AutoML, neural architecture search, and generative adversarial networks.

IVAN SUKHAREV received the B.S. degree in mathematics from the National Research University Higher School of Economics, Moscow, Russia, in 2018. From 2018 to 2020, he was a Data Scientist with the Risk Assessment Unit, Sberbank's Research Center. He has developed an innovative approach to credit risk assessment based on graph transactional data, which was presented in the article Sukharev et al., EWS-GCN: Edge Weight-Shared Graph Convolutional Network for Transactional Banking Data. From 2020 to 2023, he was the CV Team Leader with VTB Bank. Since 2023, he has been the Head of the ML Research Team, Zvuk, Moscow. His research interests include graph neural networks, neural architecture search, and computer vision fields.

ANATOLIY GLUSHENKO received the B.S. degree in applied mathematics and informatics and the M.S. degree in discrete mathematics from the Moscow Institute of Physics and Technology, in 2020. From 2017 to 2020, he was an Analyst with the Scoring, Big Data, and Innovations Department, Home Credit and Finance Bank, Moscow. Since 2020, he has been a Lead Data Scientist with the Embeddings and Depersonalized Data Team, VTB. His research interests include the development of unsupervised neural network models for sequential data and NLP models, including LLMs.

DMITRY BERESTNEV received the M.Sc. degree in applied physics and mathematics from the Moscow Institute of Physics and Technology, in 2010. He has more than 15 years of industrial experience in machine learning and deep learning in banking and IT areas. He is currently a Chief Data Scientist with Zvuk.com. His main research interests include AutoML, neural architecture search, graph neural networks, and adversarial attacks.

EVGENY BURNAEV received the M.Sc. degree in applied physics and mathematics from the Moscow Institute of Physics and Technology, in 2006, and the Ph.D. degree in foundations of computer science from the Institute for Information Transmission Problem RAS, in 2008. He is currently a Full Professor with the Skolkovo Institute of Science and Technology and the Director of the Skoltech Applied AI Center. His current research interests include regression based on Gaussian processes and kernel methods for multi-fidelity surrogate modeling and optimization, deep learning for 3-D data analysis and manifold learning, online sequence learning for prediction, and non-parametric anomaly detection.