Proceedings of Machine Learning Research 1–16

TAYSIR Competition: Transformer+RNN: Algorithms to Yield Simple and Interpretable Representations*

Rémi Eyraud REMI.EYRAUD@UNIV-ST-ETIENNE.FR
Dakotah Lambert DAKOTAHLAMBERT@ACM.ORG
Badr Tahri Joutei TAHRIBADR@GMAIL.COM
Aidar Gaffarov AIDAR.GAFFAROFF@GMAIL.COM
Université Jean Monnet Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, 42023 Saint-Etienne, France

Mathias Cabanne MATHIAS.CABANNE@EURANOVA.EU
Euranova France

Jeffrey Heinz JEFFREY.HEINZ@STONYBROOK.EDU
Department of Linguistics & Institute of Advanced Computational Science, Stony Brook University

Chihiro Shibata CHIHIRO@HOSEI.AC.JP
Department of Advanced Sciences, Faculty of Science and Engineering, Hosei University

Editor: François Coste, Faissal Ouardi, Guillaume Rabusseau

Abstract

This article presents the content of the competition Transformers+RNN: Algorithms to Yield Simple and Interpretable Representations (TAYSIR, the Arabic word for 'simple'), which was an on-line challenge on extracting simpler models from already trained neural networks held in Spring 2023. These neural nets were trained on sequential categorial/symbolic data. Some of these data were artificial, some came from real world problems (such as Natural Language Processing, Bioinformatics, and Software Engineering). The trained models covered a large spectrum of architectures, from Simple Recurrent Neural Network (SRN) to Transformers, including Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM). No constraint was given on the surrogate models submitted by the participants: any model working on sequential data was accepted. Two tracks were proposed: neural networks trained on Binary Classification tasks, and on Language Modeling tasks. The evaluation of the surrogate models took into account both the simplicity of the extracted model and the quality of the approximation of the original model.

Keywords: Recurrent Neural Networks, Transformers, Knowledge Distillation, Surrogate Model, Benchmark

1. Introduction

The main shadow over the rise of deep learning is the lack of understanding of the models it produces. Indeed, most trained models are black-boxes that escape human understanding, which causes important scientific (Räz and Beisbart, 2022) and legal (GDPR, 2016) problems. To tackle the issue, a whole field is quickly developing, commonly referred as eXplainable AI (XAI). A wide spectrum of approaches are being investigated, with one of the most studied ones being local interpretability, where the decision of a trained model on a given datum is analyzed (Guidotti et al., 2018).

* This work is supported in part by the ANR TAUDoS (ANR-20-CE23-0020).

Another promising, though more challenging, line of research focuses on providing a general explanation of the trained Neural Network by providing a simpler model that mimics the trained one: this approach is usually referred as Knowledge Distillation (Hinton et al., 2015). Actually, the surrogate model does not have to be completely explainable to be of interest: a simple model requiring less computation power than a deep neural network, and whose decisions are similar or close to the original one, is already making a difference (Harmon and Auseklis, 2009).

It is in this context that the TAYSIR competition was held during the months of March and April 2023 as a satellite event of the 16th International Conference on Grammatical Inference. The aim of this challenge was to regroup researchers interested in extracting surrogate models from Neural Nets trained on finite sequences of symbols (Giles et al., 1992; Deoras et al., 2011; Wang et al., 2017; Eyraud and Ayache, 2021; Dong et al., 2021; Lacroce et al., 2021; Barbot et al., 2021; Panchapagesan et al., 2021; Muškardin et al., 2022; Hong et al., 2022; Mayr et al., 2022). Two tracks were proposed. For the first one, the models were trained on binary classification tasks (language membership in a formal language theory sense). For the second one, the models were trained on language modeling tasks. A classical use case of this latter task is in Natural Language Processing where the model calculates the probability of the next symbol given a prefix. Each track provided 11 and 10 models, respectively, carefully selected among the $10\,000^{+}$ models we trained specifically for the competition. The competitors then had to submit a simpler model extracted from the already-trained ones.

The training datasets covered a wide spectrum of tasks, from artificially generated ones to ones from Natural Language Processing, Bio-informatics and Software Engineering. For each task, the participant only received the part of the data used as a validation set during the training phase together with the trained Neural Network model.

More than 40 teams registered to the challenge website to access the trained Neural Nets. Among them, only 7 teams submitted at least one surrogate model. However, the total number of surrogate models submitted exceed 700 (each participant was allowed to submit 20 models per day at most).

This article is organized as follows. First, it describes the data used for the training phase in Section 2. It then details the training approach and the chosen Neural Nets in Section 3. Section 4 depicts the baselines that were made available in order to show the participants how to use the trained models and in order to help the organizers select the models used in the challenge. The competition framework, including the metrics used to evaluate the quality and the simplicity of the surrogate models, is detailed in Section 5. The results of the competition are briefly described in Section 6 while Section 7 concludes.

2. Training Data

This section describes the data used for training the Neural Nets (NN) in TAYSIR. Two separate tracks were created. The first one corresponds to NN trained on a binary classification task, that is, a language membership task in Formal Language Theory. The second track provided a NN trained on a Language Modeling task, a framework where the models are required to provide a probability distribution over the set of potential next symbols given any prefix.

It is important to note that the participants of the competition had only access to the part of the data used as a validation set in training the NNs: in most cases, we divided each dataset in three, using 70% for training, 10% for validation and 20% for testing.

In some cases, the same dataset was used to generate two different TAYSIR tasks. This can happen because each TAYSIR task corresponds to a trained Neural Net. Since we trained different architectures on all datasets, we sometimes picked two different models trained on the same dataset if we thought they were interesting enough to deserve their own distillation task.

Notice that more datasets were initially considered but were not used *in fine*, mainly for two reasons: either we did not manage to train a NN with good performances, or the distillation problem was solved by our simple baselines (see Section 4 for details on the latter).

2.1. Track 1: Binary Classification Tasks

The data for this Track came from 3 different sources. The first one is the new ML-RegTest (van der Poel et al., 2023) benchmark. This benchmark provides 1 800 artificially generated datasets covering a large hierarchy of subregular formal languages. We chose 6 languages in this set corresponding to different classes of complexity and used the large datasets for training, validation and testing.

The second source of data is the Enzyme/Not-Enzyme (ENE) dataset developed by Nicolas Buton. It is generated using the UniProtKB/Swiss-Prot protein knowledgebase release 2022_01 (The UniProt Consortium, 2022). Firstly, mmseq2 (Steinegger and Söding, 2017) is employed to form clusters of sequences having 40% or higher identity, and these clusters are then distributed randomly across three sets. The training set contains 90% of the clusters, while the validation and test sets each contain 5%. In the validation and test sets, only the representative sequence for each cluster is included, whereas all sequences are retained in the training set. This strategy avoids overemphasizing highly represented sequences. The enzyme/non-enzyme labels are obtained from the "recommendedName_ecNumber" field of the UniProtKB/Swiss-Prot XML dump. Sequences possessing this field are labeled as enzymes, while those without it are classified as non-enzymes.

The last source of data is the previously organized Omphalos Competition (Starkie et al., 2005). The Omphalos data was generated from artificially created Context-Free Grammars.

Table 1 summarize the different datasets used to trained the provided NNs.

2.2. Track 2: Language Modeling Tasks

Two sources were used to train NNs for this Track: the SPiCe benchmark (Balle et al., 2017) and the PAutomaC one (Verwer et al., 2014).

The PAutomaC competition provides artificial problems generated by randomly generated finite state machines: Deterministic Probabilistic Finite Automata (DPFA), non-deterministic Probabilistic Finite Automata (PFA), and Hidden Markov's Models (HMM). A large range of machine sizes, alphabet sizes, and sparsity indicators values are covered by the different instances and the competition results proved it is covering a wide range of difficulty levels. For each of these datasets, we have access to a training set (20 000 or 100 000 sequences).

Table 1: Track 1 training data. $|\Sigma|$ is the number of distinct symbols.

TAYSIR Number	Dataset	$\|\Sigma\|$	|validation set|
1.0	MLRegTest - 16.04.TLT.2.1.4	16	10 000
1.1	MLRegTest - 04.04.Reg.0.0.9	4	10 000
1.2	MLRegTest - 16.16.LT.4.1.5	16	10 000
1.3	MLRegTest - 16.04.TLT.2.1.4	16	10 000
1.4	MLRegTest - 16.16.LT.4.1.5	16	10 000
1.5	MLRegTest - 16.16.SP.2.1.0	16	10 000
1.6	MLRegTest - 64.64.SF.0.0.0	64	10 000
1.7	MLRegTest - 64.64.SL.4.1.0	64	10 000
1.8	ENE	25	4 775
1.9	Omphalos 1	5	968
1.10	Omphalos 1	5	968
1.11	ENE	25	4 775

The SPiCe benchmark is made of 15 problems covering a large variety of contexts, from various natural languages processing data to software engineering data, including bioinformatics and well-chosen synthetic ones. The original test sets from SPiCe consisting only of prefixes, they do not correspond to what we need for our task, since they do not provide complete information on whole sequences. We thus split randomly the available learning samples into a training, a validation, and a test sample, with proportion 70%, 10%, and 20%, respectively.

Table 2 summarizes the datasets used for training the models for this Track.

Table 2: Track 2 training data

TAYSIR Number	Dataset	$\|\Sigma\|$	|validation set|
2.0	Spice 4 (NLP)	33	489
2.1	Pautomac 18 (DPFA)	20	9 090
2.2	Pautomac 23 (HMM)	7	9 090
2.3	PAutomac 47 (DPFA)	15	9 090
2.4	Spice 1 (HMM)	20	1 643
2.5	Spice 4 (NLP)	33	489
2.6	Pautomac 18 (DPFA)	20	9 090
2.7	Spice 10 (Biological)	20	4 491
2.8	Spice 6 (Software)	66	411
2.9	Spice 10 (Biological)	20	4 491
2.10	Spice 4 (NLP)	33	489

3. Trained Models

3.1. Recurrent Neural Networks training

For each task, we trained several neural networks of different architectures using Torch library version 1.11.0. These included simple recurrent neural networks (SRN) (Elman, 1990), gated recurrent units (GRU) (Cho et al., 2014), and long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) networks. Models were created as a stack of one or two recurrent layers of the specified type fed into a dense region of one or two hidden linear layers followed by a sigmoidal activation. For consistency and ease of use, we created a custom class that

arranges the PyTorch implementations of these layers and manages the transfer of data between and among them.

In order to accommodate distillation methods that require inspection of the model's internal state, we made sure to expose this state in forward calls. For SRN and GRU this is trivial, as the hidden state for each input step is already provided. However, for LSTM the internal state consists of two parts: a hidden state h and a cell (or carry) state c. While the PyTorch implementation emits h for each step, only the final c is available. We built a compatible replacement module over the LSTMCell class in order to make both parts available at each step.

In addition to cell type, we varied several other parameters. As previously stated, the number of recurrent layers (`Nb layers`) could be either one or two, as could the number of dense layers (`Nb Dense`). The number of neurons per layer (`Neurons/Layer`) covered powers of two from 32 through 512. Patience, the number of consecutive iterations allowed without improvement, was set to either five or twenty. Another parameter (`Bidirectional`) determined whether the recurrent layer was bidirectional or unidirectional. And finally, the batch size was varied between 32 and 128. For small datasets, we performed an exhaustive grid search over this space. In the interest of time, only a few combinations were tested for the larger datasets.

For each dataset, the best model was selected per cell type. In some cases, no trained model could achieve satisfactory performance against the test data. No models were included in the competition for these datasets. On the other extreme, our baseline extraction method performed at ceiling on some models. Those were also excluded.

3.2. Transformers training

As a transformer model, we employed Hugging Face's DistillBERT (Sanh et al., 2019). DistillBERT is one of the standard implementations of BERT-type models, maintaining a network structure nearly identical to the original BERT (Devlin et al., 2019), with the exception of halving the number of layers. For binary classification, the model was trained to predict the binary label from the output vector corresponding to the classification token, which was a special token inserted at the beginning of the sentence. To develop a language model, the loss function was designed to predict subsequent characters from output vectors associated with characters in the input word. In this context, future masking was essential. Since it is lacking in BERT, we modified that specific part.

Competition datasets had different characteristics compared to natural language, the usual target. For example, the maximum number of symbols was about 64, which is much smaller than the vocabulary of a natural language. Also, the relatively small sample size suggests that a more compact model may be more appropriate. We searched for the hyper-parameters: number of transformer blocks, the vector size, the number of heads, and the size of the middle layer, which is the union of all transformer blocks.

Table 3 details the parameter sets explored for each track. We trained models on each dataset by selecting 100 uniformly randomized combinations of hyperparameters from the search space.

In Track 1, the percentage of correct answers was close to 1.0 for the majority of problems. Therefore, we selected the models with the lowest computational cost among those with

Table 3: Search spaces for hyperparameters.

Track	Type	Search Space
1	Num. of T. Blocks	$\{1, 2, 3, 4, 5, 6\}$
	Dimension	$\{4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256, 384, 512, 768\}$
	Num. of Heads	$\{2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192\}$
	Ratio of Interm. Dim.	$\{1, 2, 4\}$
2	Num. of T. Blocks	$\{2, 4, 6, 8, 10, 12\}$
	Dimension	$\{16, 32, 64, 128, 256\}$
	Num. of Heads	$\{2, 4, 8, 16\}$
	Ratio of Interm. Dim.	$\{1, 2, 4\}$
	Positional Embedding	{ trainable, sinusoidal }

error rates less than ϵ. For all problem sets, ϵ was set to 0.02. The computational cost is an approximation of the total number of sums and products in the Transformer block: $m(3Ld^2 + 2L^2d + 2rd^2)$, where m, L, d, and r are the number of transformer blocks, the maximum length of input words, the embedding dimension, and the scale factor for the intermediate layer in the fully connected layer (FCN) each transformer block has, respectively. Figure 1 shows the relationship between the computational cost and accuracy, illustrated only for problems among all MLRegTest ones (1.0-1.7 in Table 1) where accuracy reaches close to 1.0. For those problems, we see that the accuracy quickly reaches around 1 while the computational cost is very low.

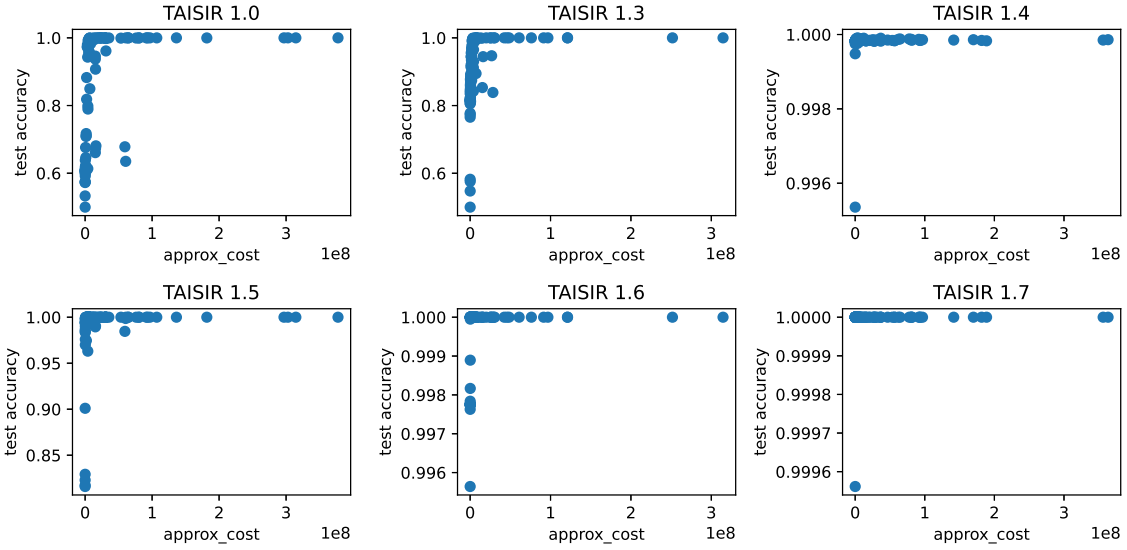


Figure 1: Accuracy (correct answer rate) as a function of approximate computational cost. Results for other TAYSIR tasks are not reported because their test accuracy never statistically outperformed the random classifier.

In Track 2, since the upper bound on quality was usually not reached, i.e., Word Error Rate (WER) was not zero, the one with the lowest WER was selected.

3.3. Provided Models

Table 4 and Table 5 describes the architectures of the trained models provided for Track 1 and Track 2, respectively. The last column of these two tables gives an evaluation of the quality of the Neural Nets on the test set: its accuracy for the first track and 1 minus the Word Error Rate (WER) for the second one (for both metrics, the higher the better).

Table 4: Track 1 model architectures

Number	Type	Nb layers	Neurons/Layer	Bidirectional	Patience	Nb Dense	Accuracy
1.0.	SRN	1	32	false	20	1	1
1.1	SRN	1	32	true	5	1	0.998
1.2	SRN	1	32	true	5	2	0.998
1.3	GRU	2	512	false	5	1	1.0
1.4	GRU	1	512	true	5	1	0.999
1.5	GRU	1	512	false	5	2	1.0
1.6	LSTM	1	64	true	5	2	0.997
1.7	Transformer	1*	12*				1.0
1.8	GRU	1	64	false	5	1	0.730
1.9	SRN	2	32	true	20	1	0.639
1.10.	GRU	2	32	false	20	2	0.647
1.11	LSTM	1	64	false	5	1	0.598

The transformer (TAYSIR 1.7) for Track 1 has 1 transformer block, 6 heads, a dimension of 12, and a scaling factor for FCN of 1.

Table 5: Track 2 model architectures

Number	Type	Nb layers	Neurons/Layer	Bidirectional	Nb Dense	1-WER
2.0	GRU	2	512	false	1	0.237
2.1	GRU	1	32	true	2	0.348
2.2	LSTM	2	256	false	2	0.364
2.3	LSTM	2	64	false	2	0.393
2.4	GRU	1	32	false	2	0.154
2.5	LSTM	1	512	false	2	0.252
2.6	SRN	2	64	false	2	0.372
2.7	SRN	2	64	false	2	0.339
2.8	SRN	2	128	false	1	0.201
2.9	GRU	1	32	true	2	0.361
2.10	Transformer	8*	256*			0.452

The transformer (TAYSIR 2.10) for Track 2 has 8 transformer blocks, 8 heads, a dimension of 256, and a scaling factor for FCN of 2.

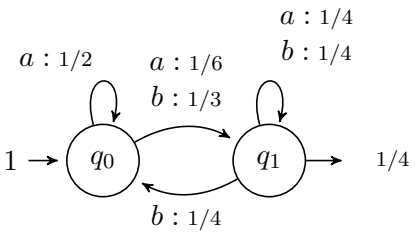
Finally, note that models 1.0 and 2.0 were used for beta-testing our framework and the one of the participants, and were thus not part of the competition *per se*.

4. Baselines

In order to evaluate the complexity of the task of extracting a model from a trained NN, and to provide examples to the participants, we developed and provided two baseline algorithms, one per task.

Both the baselines produced surrogate models which were finite state machines: the one for Track 1 outputted a Deterministic Finite State Automaton (DFA) while the one for

Track 2 extracted a Weighted Automaton (WA) from an already-trained NN. To provide diversity in the type of approaches exemplified, the core of the two baselines were different. However they both return an automaton in linear form (Arrivault et al., 2017): A *linear representation* of an automaton A is a triplet $\langle I, (E_\sigma)_{\sigma\in\Sigma}, F\rangle$ where: the vector I provides the initial weights; the vector F is the terminal weights; each matrix E_σ corresponds to the σ-labeled transition weights, with σ a symbol and Σ the set of all symbols. The dimension of each element is the number of states of the automaton. Figure 2 shows the same automaton using the usual state diagram as well as its linear form.



$$I = \begin{bmatrix}1\\0\end{bmatrix} \quad F = \begin{bmatrix}0\\1/4\end{bmatrix}$$

$$E_a = \begin{bmatrix}1/2 & 1/6\\0 & 1/4\end{bmatrix} \quad E_b = \begin{bmatrix}0 & 1/3\\1/4 & 1/4\end{bmatrix}$$

Figure 2: An automaton in its graphical form and its equivalent linear representation.

The first baseline relied on the ability to open the black-box (Giles et al., 1992): it parsed the dataset inside the trained model, kept trace of the hidden states encountered, and then used a clustering algorithm on this set of latent vectors to define the states of the DFA (one per cluster). Finally, the transitions were constructed by initializing the hidden state with the centroid of each cluster and feeding the network with each symbols of the alphabet Σ. The pseudo-code is given in Algorithm 1.

The baseline for Track 2 treated the NN as a pure black-box and used it as an oracle. This baseline was based on the spectral distillation algorithm (Eyraud and Ayache, 2021): the oracle was first used to generate prefixes and suffixes in order to define a finite block of an object called the Hankel matrix, then the trained model was used to fill this matrix, and finally the equations for spectral learning (Balle et al., 2014) were used to obtain a WA in linear form. Its pseudo-code is given in Algorithm 2.

5. Competition framework

The manual of the competition, available on its website[1], provided a large amount of technical information. We detail some of them here.

5.1. Technical implementation

The Neural Nets were trained using python 3.9.15, cuda 11.6, cudnn 8.63.8, and torch 1.13.1. They were provided to participants in the form of a MLflow Model (Chen et al., 2020) version 1.25.1 which offers the ability to run the models on any system while at the same time allowing the participants to have access to their inner functions.

The participants had to submit their surrogate models as a MLflow pyfunc, which is the basic generic interface for MLflow Python models. This forbade us to open their models

1. https://remieyraud.github.io/TAYSIR/

Algorithm 1: TAYSIR baseline for Track 1 – Extraction of a DFA from a RNN

Inputs : $R(\sigma, h)$: trained RNN , X: set of sequences, Σ: set of symbols, n: number of states

Output: An extracted DFA (I, F, E_Σ) in linear form

$H = []$ `// To store the latent states`

for $w \in X$ **do**

  $h = h_0$

  for $i = 0 : |w| - 1$ **do** `// loop to get all hidden states`

    $h \leftarrow RNN(w[i], h).latent$

    $H.add(h)$

$k\_means = k\_means\_algorithm(H, n)$

$Q = k\_means.centroids$ `// get centroids as the DFA states`

$I, F \leftarrow zeros(|Q|)$ `// initialize final and initial state matrices`

$E_\Sigma \leftarrow [E_\sigma = zeros(|Q|, |Q|), \forall \sigma \in \Sigma]$ `// initialize the transition matrices`

for $q \in Q$ **do** `// Constructing the transitions`

  for $\sigma \in \Sigma$ **do**

    $out, h \leftarrow RNN(\sigma, q)$

    $q' \leftarrow k\_means.classify(h)$

    $E_\Sigma[\sigma][q, q'] \leftarrow 1$

  if $out \geq 0.5$ **then** `// Checking if current state is final`

    $F[q] \leftarrow 1$

$q_0 \leftarrow k\_means.classify(h_0)$ `// Finding the initial state`

$I[q_0] \leftarrow 1$

return (I, F, E_Σ)

Algorithm 2: TAYSIR baseline for Track 2 – Extraction of a WA from any LM-NN

Input : LM-NN model $\mathcal{M}$, numbers of prefixes p and of suffixes s, number of states r

Output: A, a Weighted Automaton

$(\mathcal{P}, \mathcal{S}) \leftarrow Generate\_Basis(\mathcal{M}, p, s)$

$H_\mathcal{B}, (H_\sigma)_{\sigma \in \Sigma} \leftarrow Fill\_Hankels(\mathcal{M}, \mathcal{P}, \mathcal{S})$

$A \leftarrow Spectral\_Extraction(H_\mathcal{B}, (H_\sigma)_{\sigma \in \Sigma}, r)$

return A

but provided an easy to use 'pickle' that we could run on a test set. A toolbox was made available to help the participants to create their pyfunc archive.

We used the Codalab platform (Pavao et al., 2022) for the front-end: participants had to create an account there to have access to the trained Neural Nets and to submit their surrogate models.

The submitted models were tested on a dedicated back-end server hosted in our lab. Only a CPU (i9-12900k) was allowed for running the surrogate models and we controlled its use to be sure that only one process ran at any given time.

5.2. Evaluation Metrics

The submissions were evaluated on 3 criteria:

1. **Approximation quality**: We used the error rate for the binary classification and the Mean Square Error (MSE) for language modeling. Even when true targets were available, these metrics were computed with the output of the trained neural net output as ground truth.

2. **Memory usage**: The memory footprint of the surrogate model during a prediction on the test set, in mebibytes.

3. **CPU time**: The CPU run time spent by the surrogate model during a prediction on the test set, in milliseconds.

The goal of the participants was to reach the smallest score possible on all these criteria. To evaluate the overall quality of a submission, we computed a global score:

$$\text{Score} = \frac{1}{2}\text{Quality} \times C + \frac{1}{4}\text{Memory\_ratio} + \frac{1}{4}\text{Time\_ratio}$$

where

$$\text{Memory\_ratio} = \frac{\text{surrogate memory usage}}{\text{Neural Net memory usage}}$$
$$\text{Time\_ratio} = \frac{\text{surrogate CPU time}}{\text{Neural Net CPU time}}$$

The value of C was always 1 for Track 1, but was the magnitude order of the mean square error of the baseline on the test set for Track 2. This was done because MSE values are extremely small compare to the other metrics, which implies that without this constant C MSE would have had small impact in the global score.

The memory usage was tracked using the psutil (process and system utilities) package (version 5.9.4). This had the drawback to come with a fixed offset which was due to the overhead of using MLflow to run the surrogate models: even an empty model that just always returned a fixed value, for instance *True* for Track 1, had a memory usage of 120mb in our local setting (python 3.8 and MLflow 1.25.1). It also forced the participants to be very cautious with their variable usage since any declared variable was serialized by MLflow. Unfortunately, it appeared during the competition that the offset size changes with the

version of the libraries used, minimizing the interest of this metric – though finding the versions that requires the smallest memory usage is of interest of its own, it was not the aim of TAYSIR.

To deal with the CPU time computation, we created a virtual environment for each participant that was sourced for all their submissions. This means that if their code required the installation of specific packages to run, the first submission CPU time would be biased by the installation of the packages. However, this would happen only once and the participants were invited to resubmit the same model to have a fair evaluation.

Finally, each run of a surrogate model was limited to 300 seconds.

6. Results

At the time we are writing this article, we do not know about the approaches followed by the participants, which makes it hard to give relevant insights about the results.

Table 6 provides the detailed results for Track 1 of the top 3 participants (and of the baseline, for comparison) for each task.

The first element to notice is that tasks 1.2 to 1.7 were solved from a knowledge distillation perspective: the error rate of the submitted surrogate models is 0 or very close to. On these tasks, the memory usage is the metric that gave the final ranking of participants.

Secondly, on all the last 3 problems, it is worth noticing that the leader of the task almost always have an error rate an order of magnitude lower than their opponents.

Finally, on problem 1 and 8, the competition finished with extremely tight results on almost all metrics, making these problems the most disputed ones.

Table 7 details the results of Track 2. One team (named EdiMuskardin) managed to win every tasks which made it the clear winner of the Track. As the same team did also great on Track 1, it is the obvious winner of the competition.

7. Conclusion

Despite the lower than expected participation on the second track, the success of the first one validated the interest of this event for the community.

Given the diversity of the architectures and of the tasks on which the Neural Nets were trained, it is likely that TAYSIR will become a reference benchmark for knowledge distillation of models trained on sequences of symbols. All the elements, including the test sets and different scripts used for evaluation, are available on the competition website. In this benchmark archive, we also included as a bonus the transformers trained on TAYSIR datasets but not used during the competition.

Acknowledgments

We are deeply grateful to Nicolas Buton for the ENE dataset. We also want to thank the members of our scientific committee for the amazing discussions and the work realized for this competition.

Table 6: Track 1 results. Detailed score of the Top 3 on each task and of the baseline.

Task Number	Team Name	Error Rate	Memory Usage	CPU time	Score
1.1	EdiMuskardin	0.0750300 (1)	122.00 (3)	0.045 (1)	0.1286291 (1)
	neuralchecker	0.0844100 (2)	139.00 (4)	0.068 (2)	0.1478028 (2)
	kawa_yo	0.2030100 (3)	99.00 (1)	0.090 (3)	0.1812771 (3)
	TAYSIR-Baseline	0.4478500 (4)	150.00 (5)	0.167 (4)	0.3482493 (4)
1.2	kawa_yo	0.0000100 (2)	89.00 (1)	0.032 (1)	0.0664160 (1)
	EdiMuskardin	0.0000000 (1)	120.00 (2)	0.039 (2)	0.0891704 (2)
	neuralchecker	0.0000000 (1)	121.00 (3)	0.072 (3)	0.0934730 (3)
	TAYSIR-Baseline	0.5001800 (3)	150.00 (4)	0.166 (4)	0.3740798 (4)
1.3	kawa_yo	0.0000000 (1)	93.00 (1)	0.085 (3)	0.0639508 (1)
	EdiMuskardin	0.0000000 (1)	120.00 (2)	0.038 (1)	0.0824521 (2)
	neuralchecker	0.0000000 (1)	121.00 (3)	0.074 (2)	0.0831711 (3)
	TAYSIR-Baseline	0.5000000 (2)	150.00 (5)	0.168 (5)	0.3531740 (5)
1.4	kawa_yo	0.0000000 (1)	90.00 (1)	0.032 (1)	0.0627416 (1)
	neuralchecker	0.0000000 (1)	96.00 (2)	0.057 (3)	0.0669726 (2)
	EdiMuskardin	0.0000000 (1)	119.00 (3)	0.038 (2)	0.0829482 (3)
	TAYSIR-Baseline	0.5001700 (2)	150.00 (4)	0.164 (4)	0.3548888 (4)
1.5	kawa_yo	0.0000000 (1)	93.00 (1)	0.086 (3)	0.0654267 (1)
	neuralchecker	0.0000000 (1)	96.00 (2)	0.056 (2)	0.0674238 (2)
	EdiMuskardin	0.0000000 (1)	120.00 (3)	0.038 (1)	0.0841690 (3)
	TAYSIR-Baseline	0.5207700 (2)	150.00 (5)	0.168 (5)	0.3660161 (5)
1.6	kawa_yo	0.0000100 (1)	93.00 (1)	0.086 (3)	0.0663661 (1)
	neuralchecker	0.0000100 (1)	96.00 (2)	0.054 (2)	0.0683040 (2)
	EdiMuskardin	0.0000200 (2)	120.00 (3)	0.037 (1)	0.0852084 (3)
	TAYSIR-Baseline	0.4977400 (3)	149.00 (4)	0.166 (4)	0.3553508 (4)
1.7	kawa_yo	0.0000000 (1)	92.00 (1)	0.086 (3)	0.0621229 (1)
	neuralchecker	0.0000000 (1)	96.00 (2)	0.055 (2)	0.0646378 (2)
	EdiMuskardin	0.0000000 (1)	120.00 (3)	0.037 (1)	0.0806272 (3)
1.8	kawa_yo	0.3269795 (2)	105.00 (1)	0.025 (1)	0.2256924 (1)
	neuralchecker	0.3269795 (2)	108.00 (2)	0.030 (2)	0.2275360 (2)
	EdiMuskardin	0.3267700 (1)	132.00 (3)	0.038 (3)	0.2416978 (3)
	TAYSIR-Baseline	0.3269795 (2)	163.00 (4)	0.166 (4)	0.2622803 (4)
1.9	EdiMuskardin	0.0074658 (1)	118.00 (2)	0.040 (2)	0.0917099 (1)
	kawa_yo	0.0725840 (3)	90.00 (1)	0.024 (1)	0.1027667 (2)
	neuralchecker	0.0306927 (2)	121.00 (3)	0.057 (3)	0.1072263 (3)
	TAYSIR-Baseline	0.9274160 (4)	149.00 (4)	0.154 (4)	0.5855869 (4)
1.10	EdiMuskardin	0.0149316 (1)	119.00 (2)	0.042 (2)	0.0944037 (1)
	kawa_yo	0.0680216 (3)	91.00 (1)	0.023 (1)	0.0999327 (2)
	neuralchecker	0.0522605 (2)	126.00 (3)	0.059 (3)	0.1191474 (3)
	TAYSIR-Baseline	0.0680216 (3)	149.00 (4)	0.172 (5)	0.1505605 (4)
1.11	EdiMuskardin	0.0083787 (1)	133.00 (2)	0.059 (3)	0.0854677 (1)
	neuralchecker	0.0219941 (2)	186.00 (6)	0.086 (4)	0.1246985 (2)
	kawa_yo	0.2899036 (3)	104.00 (1)	0.025 (1)	0.2083020 (3)
	TAYSIR-Baseline	0.7100964 (4)	164.00 (5)	0.164 (6)	0.4561645 (6)

Table 7: Track 2 results. Detailed score of the Top 2 on each task and of the baseline.

Task Number	Team Name	Score	MSE×C	Memory Usage	CPU Time
2.1	EdiMuskardin	0.1750089 (1)	0.1562607 (1)	118.0 (1)	0.04500 (1)
	neuralchecker	0.2858219 (2)	0.3767927 (2)	118.0 (1)	0.09700 (2)
	TAYSIR-Baseline	0.4090530 (3)	0.5728363 (3)	148.0 (2)	0.16350 (4)
2.2	EdiMuskardin	0.0097656 (1)	0.0012267 (1)	118.0 (1)	0.04350 (1)
	neuralchecker	0.0134414 (2)	0.0085731 (2)	118.0 (1)	0.09700 (2)
	TAYSIR-Baseline	0.1665942 (3)	0.3102189 (3)	148.0 (2)	0.17100 (3)
2.3	EdiMuskardin	0.0959499 (1)	0.0000366 (1)	117.0 (1)	0.04450 (1)
	neuralchecker	0.0970022 (2)	0.0004323 (2)	118.0 (2)	0.09600 (2)
	TAYSIR-Baseline	0.1897990 (3)	0.1367553 (3)	148.0 (3)	0.16300 (3)
2.4	EdiMuskardin	0.0966079 (1)	0.0000064 (2)	118.0 (1)	0.04300 (1)
	neuralchecker	0.0972557 (2)	0.0000001 (1)	118.0 (1)	0.09750 (2)
	TAYSIR-Baseline	0.3531379 (3)	0.4612565 (3)	148.0 (2)	0.16650 (3)
2.5	EdiMuskardin	0.0943649 (1)	0.0000001 (1)	117.0 (1)	0.04350 (1)
	neuralchecker	0.0951833 (2)	0.0000001 (2)	118.0 (2)	0.09550 (2)
	TAYSIR-Baseline	0.3684454 (3)	0.4981066 (3)	148.0 (3)	0.16200 (3)
2.6	EdiMuskardin	0.1956771 (1)	0.1971876 (1)	118.0 (1)	0.04550 (1)
	neuralchecker	0.3246898 (2)	0.4544016 (2)	118.0 (1)	0.09650 (2)
	TAYSIR-Baseline	0.4473630 (3)	0.6495335 (3)	148.0 (2)	0.16150 (3)
2.7	EdiMuskardin	0.0959622 (1)	0.0000000 (1)	118.0 (1)	0.04400 (1)
	neuralchecker	0.0973649 (2)	0.0000000 (1)	119.0 (2)	0.09700 (2)
	TAYSIR-Baseline	0.1942414 (3)	0.1435849 (2)	149.0 (3)	0.16950 (3)
2.8	neuralchecker	0.0569728 (1)	0.0053492 (1)	118.0 (1)	0.09700 (2)
	EdiMuskardin	0.0601653 (2)	0.0118116 (2)	118.0 (1)	0.04350 (1)
	TAYSIR-Baseline	0.1949540 (3)	0.2536402 (3)	148.0 (2)	0.16450 (3)
2.9	EdiMuskardin	0.0975634 (1)	0.0000000 (1)	119.0 (1)	0.04350 (1)
	neuralchecker	0.0977556 (2)	0.0000000 (1)	119.0 (1)	0.06800 (2)
	TAYSIR-Baseline	0.2016953 (3)	0.1572833 (2)	149.0 (2)	0.16850 (3)
2.10	EdiMuskardin	0.1554995 (1)	0.1442694 (1)	119.0 (1)	0.04250 (1)
	neuralchecker	0.1757673 (2)	0.1832887 (2)	120.0 (2)	0.12050 (2)
	TAYSIR-Baseline	0.2038741 (3)	0.2002162 (3)	148.0 (3)	0.16800 (3)

References

D. Arrivault, D. Benielli, F. Denis, and R. Eyraud. Scikit-SpLearn: a toolbox for the spectral learning of weighted automata compatible with scikit-learn. In *Conférence francophone en Apprentissage*, 2017.

B. Balle, X. Carreras, F. Luque, and A. Quattoni. Spectral learning of weighted automata. *Machine Learning*, 96(1-2):33–63, 2014.

B. Balle, R. Eyraud, F. M. Luque, A. Quattoni, and S. Verwer. Results of the sequence prediction challenge (SPiCe): a competition on learning the next symbol in a sequence. In *Proc. of the International Conference on Grammatical Inference*, volume 57 of *PMLR*, pages 132–136, 2017.

B. Barbot, B. Bollig, A. Finkel, S. Haddad, I. Khmelnitsky, M. Leucker, D. Neider, R. Roy, and L. Ye. Extracting context-free grammars from recurrent neural networks using tree-automata learning and a* search. In *Proc. of ICGI*, volume 153, pages 113–129. PMLR, 2021.

A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, Sue Ann Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar. Developments in mlflow: A system to accelerate the machine learning lifecycle. In *DEEM'20*, 2020.

K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

A. Deoras, T. Mikolov, S. Kombrink, M. Karafiát, and S. Khudanpur. Variational approximation of long-span language models for lvcsr. *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, pages 5532–5535, 2011.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

G. Dong, J. Wang, J. Sun, Y. Zhang, X. Wang, T. Dai, J. S. Dong, and X. Wang. Towards interpreting recurrent neural networks through probabilistic abstraction. In *Proc. of ASE*, page 499–510. Association for Computing Machinery, 2021.

J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

R. Eyraud and S. Ayache. Distillation of Weighted Automata from Recurrent Neural Networks using a Spectral Approach. *Machine Learning*, 2021.

GDPR. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal

data and on the free movement of such data (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

R. Guidotti, A. Monreale, F. Turini, D. Pedreschi, and F. Giannotti. A survey of methods for explaining black box models. *CoRR*, abs/1802.01933, 2018.

R. R. Harmon and N. Auseklis. Sustainable it services: Assessing the impact of green computing practices. In *PICMET '09 - 2009 Portland International Conference on Management of Engineering Technology*, pages 1707–1717, 2009.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

D. Hong, A. M. Segre, and T. Wang. Adaax: Explaining recurrent neural networks by learning automata with adaptive states. In *Proc. of KDD*, page 574–584. Association for Computing Machinery, 2022.

C. Lacroce, P. Panangaden, and G. Rabusseau. Extracting weighted automata for approximate minimization in language modelling. In *Proc. of ICGI*, volume 153 of *Proceedings of Machine Learning Research*, pages 92–112. PMLR, 2021.

F. Mayr, S. Yovine, F. Pan, N. Basset, and T. Dang. TOWARDS EFFICIENT ACTIVE LEARNING OF PDFA. In *LearnAut 2022*, Paris, France, 2022.

E. Muškardin, B. K. Aichernig, I. Pill, and M. Tappler. Learning finite state models from-recurrent neural networks. In *Proc. of IFM*, page 229–248. Springer-Verlag, 2022.

S. Panchapagesan, D. S. Park, C.-C. Chiu, Y. Shangguan, Q. Liang, and A. Gruenstein. Efficient knowledge distillation for rnn-transducer models. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5639–5643, 2021.

A. Pavao, I. Guyon, A.-C. Letournel, X. Baró, H. Escalante, S. Escalera, T. Thomas, and Z. Xu. Codalab competitions: An open source platform to organize scientific challenges. *Technical report*, 2022.

Tim Räz and Claus Beisbart. The importance of understanding deep learning. *Erkenntnis*, 2022.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

B. Starkie, F. Coste, and M. van Zaanen. Progressing the state-of-the-art in grammatical inference by competition: The omphalos context-free language learning competition. *AI Commun.*, 18(2):93–115, 2005.

M. Steinegger and J. Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nat Biotechnol*, 35:1026–1028, 2017.

The UniProt Consortium. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*, 51(D1):D523–D531, 11 2022.

S. van der Poel, D. Lambert, K. Kostyszyn, T. Gao, R. Verma, D. Andersen, J. Chau, E. Peterson, C. St. Clair, P. Fodor, C. Shibata, and J. Heinz. Mlregtest: A benchmark for the machine learning of regular languages, 2023.

S. Verwer, R. Eyraud, and C. de la Higuera. PAutomaC: a probabilistic automata and hidden markov models learning competition. *Machine Learning*, 96(1-2):129–154, 2014.

Q. Wang, K. Zhang, A. Ororbia, X. Xing, X. Liu, and C. Lee Giles. An empirical evaluation of recurrent neural network rule extraction. *Neural Computation*, 30, 2017.