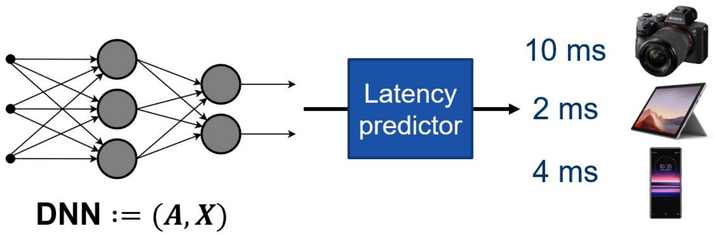
COBRA: Enhancing **DNN Latency Prediction** with Language Models trained on Source Code

Robin Zbinden, robin.zbinden@epfl.ch
Lukas Mauch, lukas.mauch@sony.com
Fabien Cardinaux, fabien.cardinaux@sony.com

ICLR DL4C

Why do we need DNN latency prediction?

Deep learning methods have had tremendous success for a variety of applications. A major challenge of this field is the development of efficient **Deep Neural Network (DNN) architectures that can also be deployed on mobile devices** or on embedded platforms with restricted memory or computational power. When tailoring DNN architectures to a specific target device, deep learning engineers typically require knowledge of the DNN latency. Simply deploying a DNN and measuring its latency requires a lot of manual effort. Many methods for **DNN latency prediction** have been proposed, but often struggle to generalize well to different types of DNN architectures. We therefore introduce COBRA to alleviate this issue.



DNN latency predictors aim at predicting the DNN inference time, namely the **latency**.

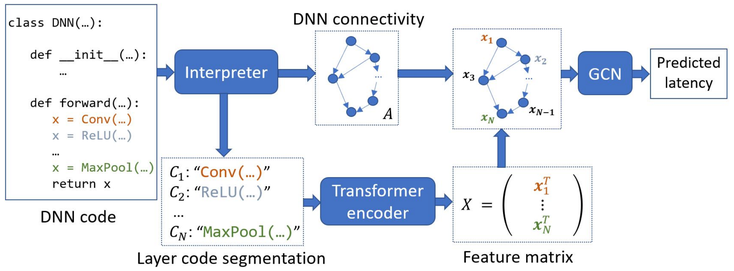
Code **B**ased **R**untime **A**pproximation (**COBRA**)

We propose a novel and flexible method to predict the latency of DNNs using **layer representations extracted from source code**. Our method, COBRA, interprets the source code to extract the corresponding network graph and layer implementations. The layer implementations consists of small code snippets with function calls to a deep learning framework. Second, it uses a transformer encoder to embed each layer implementation into a representation that is well suited for latency estimation. **We propose a special training trick for the transformer encoder** that enables us to learn source code embeddings that are especially convenient for DNN latency prediction. Finally, these extracted layer representations are aggregated by a Graph Convolutional Network (GCN) that captures the data dependencies between the function calls and estimates the latency of the DNN.

Dataset

We collect a **dataset of 2000 random ResNet-like models**, allowing skip-connections between neighbouring, but also between any two arbitrary network layers. The number of network layers is random, but is never larger than 226. Also the individual layer types and their configuration are chosen randomly.

Overview of COBRA

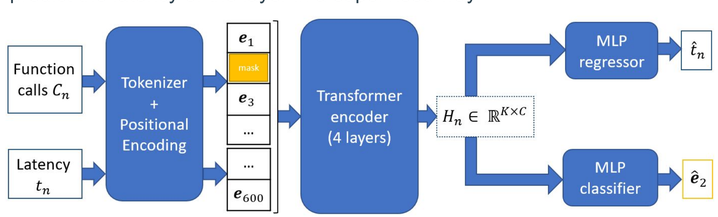


Cobra estimates the latency of a DNN from its source code representation.

- **The Interpreter:** is a rule-based module. It identifies and extracts all layer implementations, that are single function calls to a deep learning toolbox. It provides two outputs:
  1) The set of layer source code implementations.
  2) An adjacency matrix that encodes the dependencies between the layer implementations.
- **The Transformer encoder:** processes each of the layer implementations individually and outputs a corresponding embedding and latency. The architecture is similar to BERT. We use a specific training scheme, described in the next section.
- **GCN:** combines the output of the transformer encoder with the adjacency matrix and calculates a latency estimate for the full DNN.

Pre-training the Transformer encoder

Our pre-training setup uses a **combination of self-supervised and supervised training** to learn embeddings that are suited for latency prediction. For the self-supervised part the encoder is pre-trained to predict masked input tokens. This way, the transformer encoder learns to capture the properties of dependence in single layer implementations and **learns to act as a language model for source code**. We append the tokenized measured latency of the layer implementations to make the layer parameters interdependent. Simultaneously, the transformer also learns to predict the latency of the layer in a supervised way.



We train the transformer with a procedure combining self-supervised and supervised learning, reducing the average perplexity of the transformer by a factor of 2.

Results

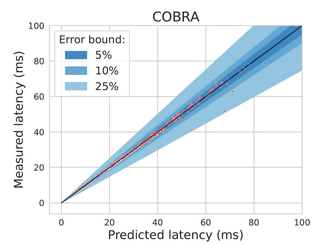
We first show how accurate the transformer encoder alone can predict the latency of single layers from their source code implementation. We observe that the **transformer encoder can accurately predict the latency of unseen layer implementations**, and can interpolate and extrapolate between layers of very different parameterizations, benefiting from the pre-training setup.

Type	MAPE	RMSE (ms)	error bound (%) ±1%	±5%	±10%	±25%
Convolution	0.28	1.01	2.41	13.17	26.72	57.61
BatchNormalization	0.06	0.21	9.50	54.96	86.36	98.45
ReLU	0.05	0.23	11.46	56.35	90.56	100.0
Add	0.06	0.27	10.0	49.0	81.0	100.0
Linear	0.022	0.001	31.31	94.44	98.48	100.0
AveragePooling	0.09	0.52	5.26	47.37	68.42	97.37
MaxPooling	0.09	0.23	10.0	35.0	60.0	100.0
GlobalAveragePooling	0.06	5.19	11.11	51.39	84.72	99.31
All	0.15	0.88	8.62	40.26	61.07	81.44

We compare then the performance of COBRA to the following two other methods for DNN latency prediction using the dataset described previously:

1) The **Layer-wise sum predictor** that estimates the latency of a DNN by summing up the measured latencies of the individual network layers. It is also calibrated by a scaling factor to fit the latencies in the training set.
2) **BRP-NAS** [1], a GCN based predictor which is state-of-the-art in latency prediction. To adapt this predictor to our more general dataset, we use hand-crafted layer representations, consisting of the one-hot encoded layer type and all the layer parameters. This representation is proposed by Zhang et al. in [2].

For each method we compute the mean absolute percentage error (MAPE), the root mean square error (RMSE), as well as different error bounds. **COBRA clearly outperforms the other baselines,** achieving state-of-the-art results.



method	MAPE	RMSE (ms)	error bound (%) ±1%	±5%	±10%
Layer-wise sum	0.0807	**3.54**	15.2	51.8	78.0
BRP-NAS	0.0358±7e-3	13.3±3.4	26.1±5.8	81.9±7.6	93.4±3.1
COBRA (Ours)	**0.0165±1e-3**	6.89±1.9	**45.3±2.7**	**96.2±1.6**	**99.0±0.4**

Our code based method beats BRP-NAS, the state-of-the-art in DNN latency prediction.

References:

[1] Dudziak et al. "Brp-nas: Prediction-based nas using gcns", 2020

[2] Zhang et al. "nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices.", 2021