

Exploring the Performance and Efficiency of Transformer Models for NLP on Mobile Devices

Ioannis Panopoulos[†], Sokratis Nikolaidis[†], Stylianos I. Venieris[‡], Iakovos S. Venieris[†]

[†]National Technical University of Athens, Athens, Greece, [‡]Samsung AI Center, Cambridge, UK

Email: ioannispanop@mail.ntua.gr, sokratisnikolaidis@mail.ntua.gr, s.venieris@samsung.com, venieris@cs.ece.ntua.gr

Abstract—Deep learning (DL) is characterised by its dynamic nature, with new deep neural network (DNN) architectures and approaches emerging every few years, driving the field’s advancement. At the same time, the ever-increasing use of mobile devices (MDs) has resulted in a surge of DNN-based mobile applications. Although traditional architectures, like CNNs and RNNs, have been successfully integrated into MDs, this is not the case for Transformers, a relatively new model family that has achieved new levels of accuracy across AI tasks, but poses significant computational challenges. In this work, we aim to make steps towards bridging this gap by examining the current state of Transformers’ on-device execution. To this end, we construct a benchmark of representative models and thoroughly evaluate their performance across MDs with different computational capabilities. Our experimental results show that Transformers are not accelerator-friendly and indicate the need for software and hardware optimisations to achieve efficient deployment.

Index Terms—Transformers, NLP, On-Device Execution, Benchmarking, Accelerators, Quantisation

I. INTRODUCTION

Being the cornerstone of many artificial intelligence (AI) tasks, DNNs are rapidly making their way into user-facing mobile applications [1], imposing an increasing demand for real-time inference with low latency and high accuracy. To achieve this, different execution options have been proposed, including running DNN inference in the cloud, at the edge, and on-device [2]. While cloud execution offers high performance, it requires a stable network connection, and edge execution involves higher latency due to data transfer. On-device execution, on the other hand, enables real-time inference without network requirements, making it an attractive option for applications that require low latency, privacy, and security.

For the past several years, convolutional neural networks (CNNs) were the dominant DNN architecture and, thus, have been extensively researched and optimised (e.g. with techniques such as the depthwise separable convolution) for various applications, leading to efficient on-device inference [3]. Since the introduction of BERT [4], however, Transformers have gained immense popularity not only in natural language processing (NLP) tasks [5], [6], but also in domains such as computer vision [7] and speech recognition [8]. With such versatility, there is an upcoming requirement for Transformer-driven applications and their deployment close to the user. Even so, the current focus of the research community is primarily on server-based training and inference [9], leaving the on-device execution of Transformers largely unexplored.

In this work, we argue that well-established practices and findings for the on-device execution of CNNs cannot translate

directly to Transformer models. This paper makes the following key contributions:

- A benchmark suite of diverse Transformer models and the associated software infrastructure, which enables on-device evaluation in a systematic and reproducible way.
- A thorough investigation of the current state of on-device Transformer inference, which includes benchmarking a wide range of models, exploring the compatibility of various mobile processors, validating on-device accuracy, and analysing the suitability of various quantisation methods.
- Suggestions for future optimisations, which can increase hardware compatibility and reduce computational needs.

II. BACKGROUND

A. Challenges of On-Device DNN Inference

Despite the benefits of on-device execution, there are several challenges that need to be addressed. MDs have limited resources, while at the same time, on-device execution requires a significant amount of memory and processing power, due to the increasing complexity of neural network models. Moreover, the heterogeneity of MDs in terms of both software and hardware often requires the deployment of a given model to be manually tuned for each target device [10].

B. Transformers

The architecture of a Transformer is typically composed of an encoder and a decoder [11]. The encoder produces a vector representation of the input sequence, while the decoder generates the output sequence one token at a time, using the encoder’s representation as context. In a typical flow, a Transformer’s sequence-to-sequence architecture is first pre-trained on language modelling or next sentence prediction tasks and, then, the encoder can be fine-tuned to a wide range of downstream tasks [4]. The architectural parameters that affect a Transformer model’s latency during the inference stage are listed in Table II.

C. Mobile Processors

Mobile processors are a core component of modern smartphones, tablets and other portable devices, as they directly impact factors such as battery life and response time. In the context of DNN inference, the central processing unit (CPU) is the baseline processor, while the rest are considered accelerators, as they can provide execution speedup and energy efficiency. The most common mobile accelerators at the moment are the graphics processing unit (GPU), the digital signal processor (DSP) and neural processing units (NPU) [12].

TABLE I
TRANSFORMER MODELS

Task	Embeddings			Name	Encoder					Overall Model		Accuracy (%)			
	Vocab	E	#Params		L	A	H	I	#Params	FLOPs	#Params	FP32	FP16	DR8	FX8
Sequence Classification on Emotions Dataset	50265	24	1.21 M	BERT Tiny	6	2	24	16	0.02 M	3.53 M	1.23 M	90.05	90.05	90.05	89.85
	30522	128	3.91 M	ELECTRA Tiny	6	2	24	16	0.03 M	4.19 M	3.94 M	90.25	90.30	90.15	90.50
	30522	256	7.81 M	XDistil-L6-H256	6	8	256	1024	4.75 M	0.49 G	12.57 M	93.20	93.25	93.00	30.95
	30522	384	11.72 M	MiniLM-L3	3	12	384	1536	5.35 M	0.54 G	17.07 M	93.25	93.25	93.00	91.95
	30522	128	3.91 M	MobileBERT	24	4	128	512	20.42 M	2.06 G	24.33 M	93.30	93.30	93.10	-
	30522	384	11.72 M	XDistil-L6-H384	6	12	384	1536	10.67 M	1.09 G	22.39 M	93.35	93.35	93.10	82.30
	30522	384	11.72 M	MiniLM-L12	12	12	384	1536	21.32 M	2.18 G	33.04 M	93.45	93.45	92.65	78.40
	28996	512	14.85 M	RoBERTa Tiny	4	8	512	2048	12.64 M	1.28 G	27.49 M	93.50	93.50	93.40	92.30
	30522	128	3.91 M	ELECTRA Small	12	4	256	1024	9.55 M	0.98 G	13.46 M	93.55	93.55	93.30	92.25
	30522	768	23.44 M	DistilBERT	6	12	768	3072	43.16 M	4.30 G	66.60 M	94.50	94.50	94.55	77.85
Text Generation	50257	768	38.60 M	DistilGPT2	6	12	768	3072	42.69 M	10.47 G	81.29 M	46.85	-	-	-
	50257	768	38.60 M	GPT2 Small	12	12	768	3072	85.28 M	15.99 G	123.88 M	51.41	51.41	-	-

TABLE II
TRANSFORMER ARCHITECTURAL PARAMETERS

Parameter	Description
Sequence Length (S)	Maximum number of tokens in one input sample.
Embedding Size (E)	Width of a token’s embedding.
Hidden Size (H)	Dimensionality of the model’s internal vector space.
Attention Heads (A)	Number of parallel heads for the attention mechanism.
Intermediate Size (I)	Width of the FFN Network.
Layers (L)	Number of layers in the model.

III. RELATED WORK

On-device execution and evaluation of Transformer models for NLP remains largely unexplored. Previous works [13], [14] have limitations, as they consider a small number of models, lack on-device accuracy evaluation and compression applicability, and do not consider all possible accelerators and execution configurations. Conversely, extensive research has been conducted to benchmark Vision Transformers and compare them to CNNs for use on MDs [15], [16], thus we do not include them in our work.

IV. EXPERIMENTAL METHODOLOGY

For our evaluation infrastructure, we chose TensorFlow (v2.11.0), because of its extensive range of quantisation methods and its support for mobile deployment through TensorFlow Lite (*nightly* builds). For the benchmarked models, we used Hugging Face’s Hub and Transformers (v4.27.4). Our experiments were conducted using Android devices and a custom-built Android application with a simple UI in order to examine how a model would perform in a real-life scenario.

A. Models and Tasks

Table I presents the benchmarked Transformer models along with their architectural parameters, workload, size, and accuracy. In order to further optimise our models, we applied three of TFLite’s post-training quantisation¹ methods: half-precision floating-point (FP16), 8-bit dynamic range (DR8), and 8-bit fixed-point (FX8). Quantisation is one of the simplest and fastest compression methods presently available, with benefits not only in model size, but also latency and memory.

For our downstream classification task, we used Emotions, a small dataset consisting of English Twitter messages with six emotions as classes. The reported accuracy corresponds

TABLE III
TARGET SMARTPHONES

Device	Samsung A71	Samsung S20 FE
Date	2020, January	2020, October
SoC	Snapdragon 730	Exynos 990
CPU	2×2.2 GHz Kryo 470 Gold	2×2.73 GHz Exynos M5
	6×1.8 GHz Kryo 470 Silver	2×2.5 GHz Cortex-A76
GPU	Adreno 618 @700 MHz	4×2.0 GHz Cortex-A55
NPU	Qualcomm Hexagon 688	Mali-G77 MP11 @800 MHz
RAM	6 GB @1866 MHz	✓
TDP	5 W	6 GB @2750 MHz
		9 W

to the top-1 accuracy on the dataset’s test set of 2000 samples when using 50 tokens as the input sequence length. We obtained the majority of models pre-trained on various large datasets, such as Reddit comments and 2ORC citation pairs, and subsequently fine-tuned them on Emotions. Selected models include optimised architectures, such as XtremeDistil [17], MiniLM [18] and MobileBERT [19], as well as light versions of the RoBERTa [6] and ELECTRA [20] original implementations. Exceptions to these are DistilBERT [21], whose Emotions-fine-tuned version was obtained directly from Hugging Face, and BERT and ELECTRA Tiny which were trained from scratch. The training/fine-tuning configuration involves a batch size of 64 and the RMSProp optimiser with initial learning rate in the order of 10^{-4} and exponential decay.

In addition to the above models, we also include GPT2 [22] for text generation in its small and distilled versions, which we obtained already converted into the `tf.lite` format from Hugging Face. We report accuracy in the context of the next token prediction task using LAMBADA’s test set of 5000 passages and 64 input sequence and output prediction tokens.

B. Mobile Devices

To cover devices with different capabilities, we target two smartphones: Samsung Galaxy A71, representing the mid-tier category, and Samsung Galaxy S20 FE, representing the high-end category of modern mobile phones (Table III).

In order to specify an accelerator to handle part or all of the computation graph in Android with TFLite, delegates can be used. In our experiments, we use the XNNPACK, GPU, NNAPI and HEX (Hexagon) delegates.² Each of the

¹https://www.tensorflow.org/lite/performance/post_training_quantization

²<https://www.tensorflow.org/lite/performance/delegates>

considered quantisation schemes performs differently on each processor. For instance, running an FP16 model on the GPU is expected to result in the highest speedup compared to other processors, whereas integer models are best suited for the Hexagon DSP. Table IV shows the average percentage of the evaluated models' nodes that are delegated through each delegate for the two target devices. *N/S* values mean that execution is not supported by default, whereas zeros mean that the given model could not be executed due to unsupported operations. Of the available accelerator delegates, only the GPU one has potential to speedup execution in both devices.

TABLE IV
DELEGATE COMPATIBILITY

Variant	Samsung A71						Samsung S20 FE		
	CPU	GPU	DSP		NPU		CPU	GPU	NPU
	XNN	GPU	NNAPI	HEX	NNAPI	NNAPI	XNN	GPU	NNAPI
FP32	74.1	99.8	72.6	N/S	N/S	N/S	74.1	99.8	0
FP16	77.6	81.8	0	N/S	N/S	N/S	77.6	81.8	0
DR8	67.6	99.8	71.3	N/S	N/S	N/S	67.6	99.8	2.6
FX8	64.0	99.8	0	24.9	0	1.4	64.0	99.8	1.1

C. Benchmarking Details

For the CPU measurements, we tested the XNNPACK library and multithreading, while for the GPU and NNAPI delegates, we used 16 bits for computation, when possible. Prior to taking any measurements, we run the model for a few iterations (1-5) to warm up the processor and decrease measurement variations. In order to keep the device's temperature as constant as possible and prevent overheating, we run fewer inferences for larger models (≈ 20 runs) than for relatively smaller models (≈ 100 runs). We also maintained a device idle period of 2-3 minutes between measurements.

V. RESULTS

A. On-Device Accuracy

The reported accuracy in Table I comes from evaluation conducted with Python's TFLite Interpreter on an Intel® Xeon® CPU. By evaluating the models on-device, we found that all of the delegate-processor combinations (columns in Table IV) deliver the accuracy presented in Table I, *except* for the GPU delegate, where only MobileBERT retains its original accuracy, while for the rest of the discriminative models the accuracy drops drastically (below 35%).

B. CPU Performance

Figures 1 and 2 show the impact of the XNNPACK and CPU multithreading on the throughput of all FP32 models for the two devices. Each model's five bars correspond to XNNPACK, 1, 2, 4 and 8 CPU threads when viewed from top to bottom. We observe the following:

- Even though XNNPACK is enabled by default in the TFLite Interpreter, it is not the optimal configuration for the majority of models in both devices.
- There is no optimal configuration across models or devices. For instance, on S20 FE, the best CPU configuration for ELECTRA Tiny is enabling the XNNPACK delegate, whereas for RoBERTa Tiny, 4 threads will lead to the optimal latency. On A71, instead of enabling the

XNNPACK delegate, the best configuration for ELECTRA Tiny is 2 threads. This observation is also valid for the quantised variants of the models.

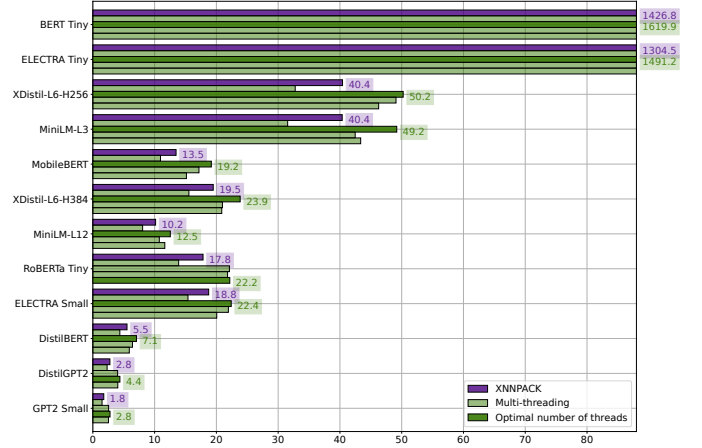


Fig. 1. CPU throughput (input samples / sec) for Samsung A71.

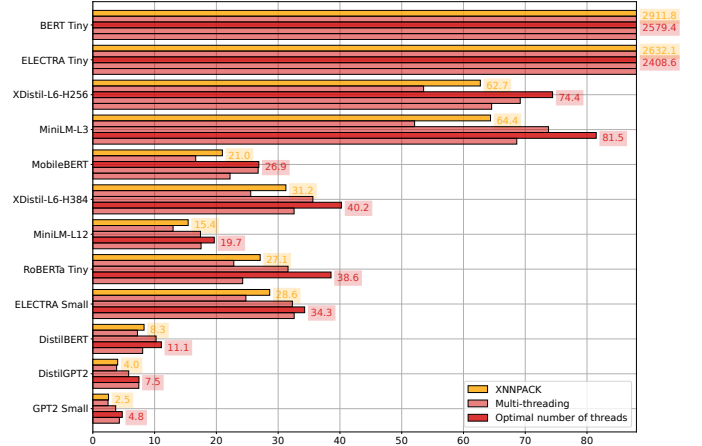


Fig. 2. CPU throughput (input samples / sec) for Samsung S20 FE.

Table V shows the effects of quantisation on latency and memory. The memory needed for each FP32 model is also shown in MB for comparison. The values are averaged across the two devices as they were very close. FP16 models achieve the same latency as FP32 models and require at least 70% more memory. In contrast, integer models accelerate the execution and at the same time reduce the memory requirements for the majority of the architectures.

TABLE V
IMPACT OF QUANTISATION ON CPU

Model	Latency Speedup			RAM	RAM Reduction		
	FP16	DR8	FX8		FP16	DR8	FX8
BERT Tiny	0.97×	0.97×	0.42×	4.7	0.41×	0.89×	1.28×
ELECTRA Tiny	0.99×	0.99×	0.39×	5.1	0.18×	0.91×	1.13×
XDistil-L6-H256	0.99×	1.62×	1.51×	26.2	0.32×	2.54×	3.03×
MiniLM-L3	1.05×	1.86×	1.64×	30.4	0.28×	2.84×	2.54×
MobileBERT	1.03×	1.72×	-	89.4	0.59×	2.93×	-
XDistil-L6-H384	0.99×	1.82×	1.75×	52.9	0.37×	3.07×	3.27×
MiniLM-L12	1.02×	1.89×	1.77×	95.2	0.47×	3.41×	3.47×
RoBERTa Tiny	0.99×	1.65×	1.81×	65.0	0.37×	3.34×	3.94×
ELECTRA Small	1.01×	1.53×	1.82×	44.7	0.51×	3.01×	3.01×
DistilBERT	1.06×	2.27×	2.32×	191.9	0.46×	3.31×	3.51×
DistilGPT2	-	-	-	585.2	-	-	-
GPT2	0.98×	-	-	673.0	0.59×	-	-

C. Accelerators

Taking into consideration Table IV, we consider execution on the NPUs purposeless, so we calculate the latency speedups provided by the GPU and DSP only, for the floating-point and fixed-point models, respectively, compared to the best CPU configuration of each model. Table VI shows the achieved speedups, as well as memory increases (underlined) for each model, except for GPT2, which was found to be accelerator-incompatible.

TABLE VI
ACCELERATOR LATENCY SPEEDUP AND MEMORY INCREASE

Model	Samsung A71			Samsung S20 FE	
	GPU(FP32/16)	DSP(FX8)		GPU(FP32/16)	
BERT Tiny	0.08×	7.31×	0.44×	0.06×	13.41×
ELECTRA Tiny	0.09×	6.12×	0.49×	0.06×	18.16×
XDistil-L6-H256	0.86×	3.52×	0.83×	0.90×	5.65×
MiniLM-L3	0.69×	3.49×	0.72×	1.06×	4.57×
MobileBERT	0.97×	3.52×	-	1.06×	4.46×
XDistil-L6-H384	1.24×	3.34×	0.86×	1.20×	4.09×
MiniLM-L12	1.22×	3.33×	0.92×	1.29×	4.12×
RoBERTa Tiny	1.19×	2.79×	0.87×	1.91×	3.43×
ELECTRA Small	1.37×	3.63×	0.85×	1.56×	4.69×
DistilBERT	1.22×	2.80×	0.71×	2.77×	3.10×

As expected, the DSP is not able to provide any acceleration due to its small compatibility ratio. The use of quantisation does not affect the performance of GPU, since all model variants are executed using `fp16` arithmetic. Hence, the GPU can only accelerate floating-point models, with the CPU providing higher speedup for integer models (compare with Table V).

VI. DISCUSSION AND FUTURE WORK

In the previous section, several conclusions were drawn which provide insights into how execution can be optimised for higher performance. These optimisations can be broadly categorised into two levels: system-level and model-level.

At the system-level, improvements are needed due to three main reasons: (a) the absence of a global optimal configuration, which stems from the heterogeneity of models and devices, (b) the dynamic environment of MDs, *e.g.* available resources at runtime, other processes' competing demands, dynamically changing factors (temperature, battery), and (c) the diverse application performance needs, in terms of accuracy, latency, memory or energy consumption. Future advancements may include software solutions which are device-dependent and take both model characteristics and performance objectives into account [23]. Additionally, due to the dynamic environment of MDs, it is important for the system to have access to as many different processors as possible. This enables switching between processors in cases when significant changes in resource availability are observed [24].

In our analysis, we found that Transformers are not accelerator-compatible; the GPU can offer marginal speedups but damages accuracy and the DSP and NPUs are mostly not Transformer-compatible. Influenced by the MobileBERT model, an interesting direction is to consider model-level substitutions, *i.e.* replace complex layers / operations with simpler ones, which have been proved to be more mobile-friendly, such

as the replacement of (a) the GELU activation function (*e.g.* with ReLU), and (b) Layer Normalisation layers (*e.g.* with elementwise linear transformations or Batch Normalisation).

VII. CONCLUSION

Our benchmarking results show that despite the increasing adoption of dedicated hardware in modern devices, general-purpose processors, like the CPU, remain highly utilised due to their flexibility in supporting new workloads. Our findings highlight the importance of (a) research into optimising the Transformer architecture, and (b) developing accelerators that are more Transformer-friendly. We also stress the need for device- and model-dependent system-level optimisations, since default configurations (*e.g.* XNNPACK) are almost never the optimal choice. We hope our observations and insights can assist future research and development efforts in the area.

VIII. ACKNOWLEDGEMENTS

This research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5578).

REFERENCES

- [1] M. Xu *et al.*, "A First Look at Deep Learning Apps on Smartphones," in *WWW*, 2019.
- [2] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proc. IEEE*, vol. 107, no. 8, 2019.
- [3] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *ICML*, 2019.
- [4] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *ACL*, 2019.
- [5] C. Raffel *et al.*, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *J. Mach. Learn. Res.*, vol. 21, 2020.
- [6] Y. Liu *et al.*, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," *arXiv*, 2019.
- [7] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *ICLR*, 2021.
- [8] L. Lu, C. Liu, J. Li, and Y. Gong, "Exploring Transformers for Large-Scale Speech Recognition," in *Interspeech*, 2020.
- [9] Y. Feng *et al.*, "Mobius: Fine Tuning Large-Scale Models on Commodity GPU Servers," in *ASPLOS*, 2023.
- [10] M. Almeida *et al.*, "Smart at what cost?: Characterising Mobile Deep Neural Networks in the wild," in *IMC*, 2021.
- [11] A. Vaswani *et al.*, "Attention is All you Need," in *NeurIPS*, 2017.
- [12] M. Dhouibi *et al.*, "Accelerating Deep Neural Networks Implementation: A Survey," *IET Comput. Digit. Tech.*, vol. 15, no. 2, 2021.
- [13] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," in *ISCA*, 2020.
- [14] Q. Cao *et al.*, "Are Mobile DNN Accelerators Accelerating DNNs?" in *EMDL*, 2021.
- [15] S. S. A. Zaidi *et al.*, "A Survey of Modern Deep Learning based Object Detection Models," *Digital Signal Processing*, vol. 126, 2022.
- [16] X. Wang *et al.*, "Towards Efficient Vision Transformer Inference: A First Study of Transformers on Mobile Devices," in *HotMobile*, 2022.
- [17] J. Niu, Y. Wang, Z. Yang, X. Liu, and J. Zhang, "XtremeDistilTransformers: Task Transfer for Task-agnostic Distillation," *arXiv*, 2021.
- [18] W. Wang *et al.*, "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers," in *NeurIPS*, 2020.
- [19] Z. Sun *et al.*, "MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices," in *ACL*, 2020.
- [20] K. Clark *et al.*, "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators," in *ICLR*, 2020.
- [21] V. Sanh *et al.*, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," in *EMC2@NeurIPS*, 2019.
- [22] A. Radford *et al.*, "Language Models are Unsupervised Multitask Learners," 2019.
- [23] B. Kütükçü *et al.*, "Contention Grading and Adaptive Model Selection for Machine Vision in Embedded Systems," *ACM TECS*, 2022.
- [24] S. I. Venieris, I. Panopoulos, and I. S. Venieris, "OODIn: An Optimised On-Device Inference Framework for Heterogeneous Mobile Devices," in *SMARTCOMP*, 2021.