CodeGPT on XTC

Compressing a CodeGPT Model Using Hybrid Layer Reduction and Extreme Quantisation through Knowledge Distillation

Aral de Moor

Supervisors: Arie van Deursen, Maliheh Izadi, Ali Al-Kaswan

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Aral de Moor
Final project course: CSE3000 Research Project
Thesis committee: prof. dr. Arie van Deursen, dr. Maliheh Izadi, ir. Ali Al-Kaswan

Abstract

Large language models are powerful because of their state-of-the-art language processing abilities. But, they come at the cost of being extremely resource-intensive, and are steadily growing in size. As a result, compressing such models for resource-constrained devices is an active and promising research area. In spite of their current popularity, many novel compression techniques lack implementation for GPT models. We apply the XTC pipeline, consisting of layer-reduction and quantisation through knowledge distillation, to a CodeGPT generative model. The resulting models are evaluated on the CodeXGLUE line-level code-completion benchmark. Based on this, we demonstrate that (1) XTC can be adapted to GPT-like models, translating many of the findings of the original study; (2) a 6-layer reduction with 1-bit weight and 8-bit activation quantisation is able to reduce model size $15\times$, in addition to almost doubling inference speed, with minimal performance degradation. The resulting compressed models show promise for use in local code generation. By showing that a novel compression technique can be adapted to GPT-like models, we hope to further inspire research in this field.

1 Introduction

Large Language Models are used increasingly often in language processing tasks due to their superior understanding [3], and their ability to be fine-tuned to a wide array of downstream applications. However, with their high language understanding comes the cost of high resource requirements. Furthermore, as language models have been rapidly growing in size [16], they become prohibitively expensive to train and run.

Auto-completion in IDEs can be compared to having a reliable personal assistant by your side as you code, and as such is among the most prominent features in IDEs [1]. Tools such as Github Copilot and Kite take this pair-programmer concept further with the use of generative language models, which are able to provide more than merely rule-based suggestions. However, such online-hosted subscriptions raise concerns regarding property infringement, and post a paywall on a model that is trained on public data.

Compressing large language models (LLMs) is thus an active area of research. A few notable examples include TinyBERT, a 4-layer reduced model that is $7.5\times$ smaller, $9.5\times$ faster on inference, and maintains 96.8% of its original accuracy [6]; and a 1-bit weight quantised XTC-BERT with $32\times$ smaller size maintaining 99.4% accuracy [15]. However, many novel compression techniques lack implementation for generative models. Such a compressed model could be used for inference locally, addressing the issues concerns raised by online subscriptions. This leads to our research question:

How effective are hybrid in-training knowledge distillation, layer reduction, and quantisation techniques for compressing a CodeGPT generative model?

This work applies the state-of-the-art XTC [15] compression pipeline for general-language BERT models to a programming-language CodeGPT model [7]. This entails two steps of knowledge distillation: a layer-reduction, followed by extreme quantisation. To investigate the effect of each step, we compress three models: using 1-bit weight and 8-bit activation quantisation; using 6-layer-reduction; and a hybrid combination of both.

We evaluate the compressed models' accuracy on the CodeXGLUE *Code-Completion* task [7]. To determine their suitability for resource-limited devices, we study their disk usage, size in memory, and inference speed. Furthermore, we place our results in context with other novel compression techniques adapted to GPT in our research group [8, 12, 13].

Our evaluation reveals that the hybrid XTC pipeline can compress a CodeGPT model by $15\times$, while maintaining 83.7% of its original accuracy. Additionally, we observe that 6-layer-reduction results in a considerable inference speedup, as half of the computationally-expensive self-attention layers are removed. Moreover, we note that extreme 1-bit quantisation can reduce model size by more than ten-fold.

Our contributions are as follows. Firstly, we demonstrate that XTC can be adapted to GPT-like models, yielding impressive compression results, and translating many of the findings of the original study. Furthermore, we find that while XTC results in the largest CodeGPT size reduction, a hybrid of layer-reduction through knowledge distillation, combined with post-training quantisation of [13] may significantly reduce training time for an equivalent model.

2 Background and Related Works

This section provides background knowledge on Transformer models, the GPT architecture in particular. We further mention the CodeXGLUE benchmark and its CodeGPT baseline model. Lastly, we detail typical compression techniques for such Large Language Models (LLMs), and the XTC compression pipeline adapted in this paper.

2.1 Transformer Models

Transformer models [14] are a type of deep learning model designed for natural language processing (NLP) tasks. Its main innovation, the self-attention mechanism, efficiently captures long-range dependencies between tokens of its input sequence, which can be difficult with traditional recurrent or convolutional neural networks [14]. This allows them to reach state-of-the-art results on NLP tasks such as generating text and translating language [4, 9]. Over the last few years, the parameter count of these models has been growing exponentially [16], to improve their language understanding ability.

GPT (Generative Pre-trained Transformer) [9] refers to a family of LLMs based on the transformer architecture. Their causal language modeling objective teaches them to predict the next word in a sequence. They are pre-trained on a large dataset of text, at which point they are able to generate novel human-like text. This pre-trained transformer excels on general tasks, and can be fine-tuned to a wide array of specific generative inference tasks, such as code completion.

BERT (Bidirectional Encoder Representations from Transformers) [4] models are similarly pre-trained on a large corpus of text. But, unlike GPT models, which take into account only previous tokens to generate the next, their bidirectional approach allows them to pay attention to both the left and right context of a token. This makes them suited for NLP tasks such text classification or question answering.

2.2 CodeXGLUE Benchmark

CodeXGLUE (General Language Understanding Evaluation benchmark for Code) consists of a collection of code intelligence tasks and a platform for model evaluation and comparison [7]. It aims to be a programming-language equivalent of the widespread GLUE natural-language benchmark. In particular, we use the line-level *Code-Completion* task as our accuracy metric, scoring predictions on the *Exact Match* and *Edit Similarity* metrics.

Lu et al. [7] further provide CodeGPT[1] as a baseline model for their *Code Completion* task. This model has the same 12-layer architecture and training objective of GPT-2, but is pre-trained on the Python corpora from the CodeSearchNet dataset [5]. This gives it a general understanding of the Python programming language.

2.3 Compressing LLMs

As the parameter count of LLMs has been growing steadily [16], reducing their size through means of compression is an active research area. This study combines the following compression techniques in a hybrid setting:

- **Knowledge Distillation**: Training a smaller student model from the outputs of a larger teacher model. This can be applied during the pre-training [8, 10] or fine-tuning [15] stages, in conjunction with another compression method applied to the student.
- **Layer Reduction**: Removing whole hidden layers from the neural network, while maintaining the network's dimension. This falls under the *pruning* category of removing unnecessary parameters of a LLM. Layer reduction in this context, however, directly targets only the decoder layers containing the self-attention mechanism, which is the most expensive operation [2] in processing input sequences. Existing layer-reduced models include the 6-layer-reduced TinyBERT [6], and the 6-layer XTC-BERT [15], both maintaining the accuracy of the original 12-layer model.
- **Quantisation**: Mapping full-precision weights to low-bit ones, e.g. FP32 to INT8. When moving from 32 to 8 bits, disk usage decreases by a factor of 4, while the computational cost for matrix multiplication decreases by a factor of 16 [2]. This technique can be broadly organised into two classes. In-training approaches such as [15] refer to performing quantisation operations during training, and manage up to a $32\times$ reduction in size. Post-training approaches, such as [2], directly convert a pre-trained floating-point network into a fixed-point one, yielding up to a $8\times$ size reduction. The latter of these two requires only a few calibration passes through the model, however, allowing for significant compression without taking time to re-train.

[1]https://huggingface.co/microsoft/CodeGPT-small-py

2.4 XTC Pipeline

The eXTreme Compression (XTC) pipeline proposed by Wu et al. [15] combines the aforementioned techniques to create a 'simple yet effective' compression method. They find that 1-bit quantisation with a 5-layer reduction is able to reduce disk usage by $50\times$, resulting in state-of-the-art results on GLUE tasks for a BERT-like model [15]. As a point for future research, they pose how their findings may translate to GPT-like models, which is addressed in this study.

3 Methodology

We adapt the XTC pipeline published under the DeepSpeed package [2]. Our baseline model is compressed through a 6-layer reduction, followed by 1-bit weight and 8-bit activation quantisation. This consists of the following two knowledge distillation steps.

1. **Lightweight Layer Reduction.** Select a subset of the fine-tuned teacher weights by initialising the student model with every other layer of the teacher model. We select every other layer as this is shown to perform better than students initialised with the bottom or top half layers in [15]. So, the six-layer model is initialised from the l-layer of the baseline model with $l \in \{1, 3, 5, 7, 9, 11\}$
2. **Low-Bit Quantisation.** On the layer-reduced model, apply a quantisation-aware knowledge distillation with the full model as teacher. Following the XTC specification, we use a one-/eight-bit quantiser to compress model weights/activations on a forward pass, and use STE for passing gradients in the backward pass.

We note our implementation differs than that for BERT models. For DeepSpeed's compression library to accommodate GPT models, we convert the `Conv1D` layers to functionally equivalent `Linear` layers by transposing their weights.

We further make modifications to the original XTC training pipeline. Due to limited computational resources, we limit the number of epochs in each knowledge distillation from 18 to 1. To accommodate this, we increase the learning rate from $5e^{-5}$ to $5e^{-4}$, and reduce the number of warm-up epochs from 1 to 0.2.

4 Experimental Setup

To assess the effectiveness of CodeGPT on XTC, we first evaluate the model on the *Code-Completion* task it is trained on. Then, we study its size, memory usage, and inference time, on both CPU and GPU. Lastly, we compare our compressed model to other compression techniques adapted to CodeGPT in our research group [8, 12, 13], specifically those discussed for BERT models in [2, 10, 11].

[2]https://pypi.org/project/deepspeed/

4.1 Research Questions

In the context of this study, we formulate three research questions (RQ):

RQ 1 *How does CodeGPT on XTC affect its accuracy on the CodeXGLUE line-level Code Completion task?* We first set a baseline model fine-tuned on the *Code Completion* task. We then adapt XTC for GPT-like models and create three compressed models: a 6-layer-reduced; a 1-bit weight and 8-bit activation quantised; and, a hybrid model combining both methods. The accuracy of the baseline and compressed models is evaluated on the *Code Completion* task.

RQ 2 *How CodeGPT on XTC affect its disk size, CPU/GPU memory usage, and CPU/GPU inference times?* Besides evaluating the accuracy of the compressed models, we are also interested in their efficiency. We study how layer-reduction and quantisation affect the aforementioned memory metrics, by running CPU/GPU inference on the same *Code Completion* task.

RQ 3 *How does CodeGPT on XTC compare to other compression techniques, in terms of training time, accuracy, and efficiency?* Given the models at different levels of compression, we place CodeGPT on XTC in context with other techniques studied in the research group [8, 12, 13].

4.2 Datasets

Given the CodeGPT baseline provided by CodeXGLUE [7], we further fine-tune the model on the entire PY150[3] *Code Completion* dataset provided by CodeXGLUE. To test the model, we augment the token-level *Code Completion* dataset for line-level completion by simply stripping the last line from each input sequence. For our evaluation, we use a 1000-sample subset of the test set.

4.3 Evaluation Metrics

We evaluate the resulting models on the line-level *Code-Completion* task of the CodeXGLUE benchmark, where the quality of the generated code is measured through Exact Match accuracy (EM) and Levenshtein Edit Similarity (ES). EM accuracy is given as the percentage of perfect responses generated by the model. ES is a measure of how many single character edits are required to transform one string into another, which should correlate with the effort it takes for a developer to correct the generated code.

We further study the models' disk size, CPU/GPU memory usage, and CPU/GPU inference speed during generation. This should demonstrate how well the compression-related findings of [15] translate to a GPT-like model.

4.4 Baseline

We, and our research group, create a baseline model for compression as follows. Given the CodeXGLUE [7] CodeGPT[4] model pre-trained on the Python corpora from the CodeSearchNet dataset [5], we fine-tune it to the PY150 *Code Completion* task[5]. This baseline model takes 510 MB of disk space and scores 39.1 on the Edit Similarity metric and 14.5% on Exact Match.

[3]https://huggingface.co/datasets/0n1xus/codexglue

[4]https://huggingface.co/microsoft/CodeGPT-small-py

Table 1: Summary of baseline (base), 1-bit weight and 8-bit activation quantised (1W8A), 6-layer reduced (6L), and hybrid (1W8A6L) compressed models. Given are their Size (MB), Compression Factor, Parameter Count, and CodeXGLUE line-level *Code Completion* Scores: Edit Similarity (ES) and Exact Match Accuracy (EM).

	Size	Factor	Params	ES	EM (%)
base	510.0	1.0×	124M	39.0	14.5
1W8A	42.6	12.0×	124M	37.2	14.1
6L	445.6	1.1×	82M	36.9	13.5
1W8A6L	32.3	15.8×	82M	33.9	10.9

4.5 Configuration

Our experimental setup runs the compression training and CodeXGLUE evaluation on a single NVIDIA A100 GPU (40GB VRAM). However, to maintain consistency with the research group, we evaluate inference-related metrics from RQ 2 on a NVIDIA V100 (16GB VRAM) for GPU metrics, and an Intel Xeon 2.20GHz (12.7GB RAM) for CPU metrics.

5 Results

Table 1 presents the resulting disk usage and *Code Completion* scores, as stipulated in RQ 1. The 1-bit weight and 8-bit activation quantised (1W8A), 6-layer reduced (6L), and hybrid (1W8A6L) model scores on the *Code Completion* task metrics, as well as the baseline (base) are given. Most notably, it can be seen that quantisation yields the largest size reduction: the 1W8A quantised model has a considerable compression factor of $12\times$, with only -1.8 ES and -0.4% EM reduction.

Table 2 summarises GPU size, GPU/CPU memory usage, and CPU/GPU inference speed, addressing RQ 2. The GPU size is essentially the size of the unpacked model in memory, and was found to be similar to the memory size on the CPU. In particular, it can be seen that the 6L layer-reduced model takes only 340 MB in memory, compared to the baseline 510 MB, and has a significant inference speedup on both CPU and GPU.

On our configuration specified in Section 4.5, training the 1W8A quantised model required 3h 45m, and the 6L-reduced model took 2h 30m. The hybrid model is essentially a sum of these two, taking around 6h 20m.

6 Discussion

Our results strongly show promise for adapting XTC to GPT-like models. Despite the computational limitations in our training setup, we are able to compress the baseline model up to $15\times$ its size while maintaining considerable accuracy. Within the related literature of our research group [8, 12, 13], this is the largest size reduction with acceptable accuracy.

[5]https://github.com/microsoft/CodeXGLUE/tree/main/Code-Code/CodeCompletion-line

Table 2: Summary of Model Memory Size on GPU (MB), GPU/CPU Memory Usage during Inference (MB), and CPU/GPU Inference Speeds (samples/sec.).

	GPU Size	Memory Usage		Inference	
	(MB)	GPU	CPU	GPU	CPU
base	509.6	4000	4870	16.6	0.36
1W8A	663.2	4000	4850	11.4	0.39
6L	341.8	4420	4770	26.9	0.68
1W8A6L	341.8	4380	4354	26.2	0.70

This implies that many of the findings of the original XTC paper for BERT-like models [15] translate to GPT-like models.

6.1 Accuracy of Compressed Models

Table 1 shows that XTC is effective at compressing CodeGPT models. The accuracy degrades only slightly, despite the significantly shorter training epochs, which is discussed as a limitation in Section 7. Quantising a model results in the largest disk size reduction. This is because FP32 to 1-bit results in a $32\times$ size reduction, and FP32 to 8-bit is still a considerable $4\times$ reduction.

The 6L-reduction has a smaller compression factor, but by targeting the decoder layers, allows for another considerable size reduction for the 1W8A6L-hybrid model. Furthermore, while the 6L model removes every other layer, its parameter count does not halve as one would expect. This is due to the token embedding layers remaining as-is, which are around 40M parameters. As the layer reduction is only applied to the 12 GPT decoder layers, this results in less of a size decrease than [15].

6.2 Memory Usage and Efficiency of Compressed Models

Table 2 shows that XTC can compress CodeGPT for resource-constrained devices. Most notably, layer reduction almost halves the space required by the model in memory, and roughly doubles the inference speed. This is likely because it removes half the decoder layers containing the self-attention mechanism, which is the most expensive operation crucial for efficient processing of long input sequences [2].

On the other hand, the unpacked 1W8A-quantised model takes more space in memory than the baseline. Similarly, quantising does not seem to have an inference speedup despite fixed-point arithmetic being generally faster. We assume that this is due to the compression library wrappers that expand the quantised weights into floats, in order to work with our evaluation pipeline. This limitation relating to the compression library itself is further discussed under Section 7.

6.3 Comparison with Concurrent Work

Lastly, we comment on the training time of the models, and place their accuracy and efficiency in context with other compression methods investigated by our research group. In this context, XTC yields the largest size reduction with minimal accuracy loss. Given its drastic size reduction, CodeGPT on XTC shows potential for compression training on a capable GPU, after which it can be used for inference on resource-constrained devices.

However, it is evident that XTC requires a considerable training time, especially for quantisation. Yet, post-training quantisation techniques studied by Storti [13] manage to convert weights to INT4 with higher accuracy in only a few calibration data passes, taking no more than a few minutes. Given that in-training knowledge distillation is relatively slow, for faster compression of LLMs, future research could investigate the hybrid effectiveness of XTC layer reduction through knowledge distillation, combined with post-training quantisation techniques in [12, 13].

7 Threats to Validity

Internal validity questions if other factors could have affected the outcome. External validity refers to the generalisability of our results. Construct validity relates to the adequacy of the theoretical constructs and the use of appropriate evaluation metrics.

7.1 Internal Validity

A key finding in [15] is that quantised models are often under-trained, and thus longer training iterations with learning rate decay are highly preferred for closing the accuracy gap of extreme quantisation. The XTC pipeline calls for 18-epoch knowledge distillation for both the layer reduction, and the quantisation step. However, as mentioned under Section 3, this is unfeasible in our setup due to computational limitations. We maintain a linear learning rate decay like the original pipeline, but opt for a 1-epoch knowledge distillation for each step, and further modify the learning rate from $5e^{-5}$ to $5e^{-4}$ to accommodate this limitation. To investigate its effect, we consider the loss curves of each compressed model in Figure 1.

Figure 1 shows a relatively stable loss curve for both the 1W8A-quantised and 6L-reduced model. However, the 1W8A6L-hybrid model displays more volatile training. Despite the learning rate having decayed to almost 0 at the last 1000 steps, it makes a considerable loss improvement. This indicates that the model is potentially under-trained. We regard this sub-optimal compression result as a serious limitation of our study. While our results show XTC applies well to a CodeGPT model, future research should investigate the potential benefit of more training epochs.

A further internal limitation of our study is that we have only attempted the layer-reduction and quantisation configurations that worked best for BERT in [15]. Different optimal configurations, that outperform our compressed models may exist as the GPT architecture is not identical to BERT. For instance, we select every odd when initialising our layer-reduced model, but one could also select the even layers, or the first six, etc. Furthermore, XTC uses symmetric activation quantisation, while related work such as [13] indicates that asymmetric quantisation also shows promise. We suggest that future research investigates a wider range of such configurations.

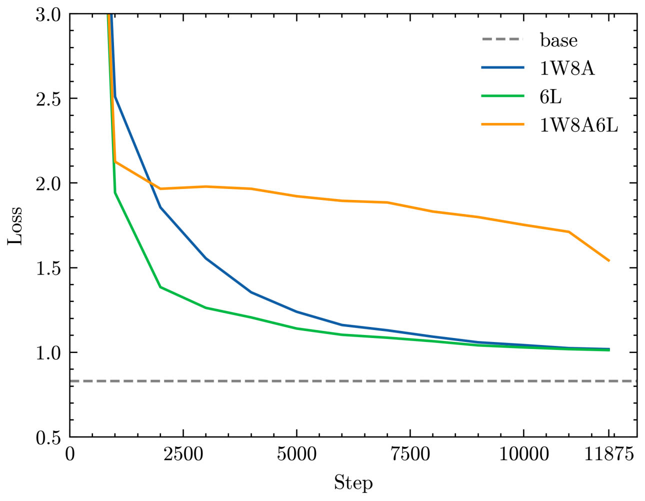

Figure 1: Compressed Model Loss over Training Steps (1 Epoch).

Lastly, we faced issues with adapting the XTC compression library[6] to GPT-like models. Namely, extreme quantisation ($<$ 8-bit) is not supported by the underlying PyTorch[7] model, and requires architecture modifications, converting `Conv1D` to `Linear` layers. The compression library remedies this with wrapper layers, which we believe to be the reason why there is no memory reduction or inference speedup for the quantised models in Table 2. We hope that our findings inspire the continued development of such compression libraries and the eventual support of PyTorch for extreme quantisation.

7.2 External Validity

The CodeGPT baseline provided to our research group does not perform as well as in the CodeXGLUE paper where it is introduced, scoring 39.0 ES and 14.5% ES, as opposed to 69.7 ES and 39.1% EM [7]. We resorted to this sub-optimal setup due to time limitations in our study. Given that the starting accuracy is sub-optimal, it could be that a fully-trained initial model suffers a larger accuracy loss through XTC. Hence, for realistic generalisability to real-world use-cases, we advise future research to use an optimal baseline for compression.

7.3 Construct Validity

Given that the main application of code-generation models is to assist developers, there is a threat that the CodeXGLUE *Code Completion* metrics do not accurately represent the true performance of the model in this use-case. There are several equivalent ways to write programming statements, e.g. ternary operators and if-else blocks. This could mean that functionally identical code results in a lower ES or EM score. While we are unaware of a benchmark task that directly tests the functionality of generated code, it could be an avenue for the future study of code-generation models. Alternatively, subsequent studies could investigate the effects of XTC on several GPT models, fine-tuned to different tasks.

8 Conclusion

Our study highlights the remarkable potential of GPT-like models on XTC, demonstrating its ability to compress code-trained generative models while maintaining accuracy. Layer reduction and quantization through knowledge distillation can reduce the size of CodeGPT by $15\times$, in addition to inference speedups. By affirming the effectiveness of XTC at compressing CodeGPT models, we hope to have inspired future advancements in this field.

We briefly summarise the points raised for future research. Firstly, as CodeGPT on XTC shows significant promise, it is worthwhile for subsequent studies to use the intended 18-epoch knowledge distillation. Such studies can also investigate the effect of different compression configurations, such as asymmetric activation quantisation. Secondly, we note the potential of post-training quantisation techniques such as [13], and suggest a hybrid approach of in-training layer-reduction and post-training quantisation. Lastly, we suggest the generalisability of our findings is studied by applying XTC to a variety of GPT models fine-tuned to different downstream tasks.

9 Responsible Research

We promote the reproducibility of our research by providing comprehensive details on our methodology under Section 3 and our computational configuration under Section 4.5. We further publish our implementation code on the TU Delft AI-enabled Software Engineering Research Group's GitHub repository[8], and our compressed models are available on their HuggingFace Hub[9].

Further, it is essential to acknowledge the potential ethical implications of large language models. While autocompletion tools powered by generative models can greatly enhance developers' productivity, they also raise concerns about intellectual property infringement, code ownership, and potential biases in the generated code. Throughout this study, we use a modified dataset that is stripped of any identifiable literals and strings.

10 Acknowledgements

This work was performed as the final thesis of the Bsc. Computer Science and Engineering at TU Delft. We appreciate the support and guidance of the project supervisors, as well as the help from our research project group. We further thank the – particularly computational – support of the University.

References

[1] Sven Amann, Sebastian Proksch, Sarah Nadi, and Mira Mezini. A study of visual studio usage in practice. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 124–134, 2016.

[6]https://pypi.org/project/deepspeed/
[7]https://pypi.org/project/torch/
[8]https://github.com/AISE-TUDelft/LLM4CodeCompression
[9]https://huggingface.co/AISE-TUDelft

[2] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and Overcoming the Challenges of Efficient Transformer Quantization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7947–7969, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[5] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search, 2020.

[6] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for Natural Language Understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online, November 2020. Association for Computational Linguistics.

[7] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation, 2021.

[8] Emil Malmsten. Distilling code-generation models for local use, 2023.

[9] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[10] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. October 2019.

[11] Haihao Shen, Ofir Zafrir, Bo Dong, Hengyu Meng, Xinyu Ye, Zhe Wang, Yi Ding, Hanwen Chang, Guy Boudoukh, and Moshe Wasserblat. Fast DistilBERT on CPUs, December 2022. arXiv:2211.07715 [cs].

[12] Dan Sochirca. Compressing code generation language models on cpus, 2023.

[13] Mauro Storti. Leveraging efficient transformer quantization for codegpt: A post-training analysis, 2023.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[15] Xiaoxia Wu, Zhewei Yao, Minjia Zhang, Conglong Li, and Yuxiong He. Extreme Compression for Pre-trained Transformers Made Simple and Efficient, June 2022. arXiv:2206.01859 [cs].

[16] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent J. Hellendoorn. A systematic evaluation of large language models of code, 2022.