A Taste of Generative Pre-Training on Code Generation

Jiheng Wei and **Madina Babazhanova** and **Henrik Abgaryan**
Master IASD

Abstract

Generative Pre-trained Transformer (GPT) has emerged as a highly successful approach in natural language processing (NLP), demonstrating impressive results on a range of tasks. In this article, we explore the potential of GPT-2 on code generation tasks, specifically on generating Python code. We present our experiments and results on the widely-used CodeParrot dataset, and discuss the potential of GPT in tackling a variety of natural language generation (NLG) tasks in the future.

1 Introduction

Generative Pre-trained Transformer is a powerful approach for unsupervised learning, which has been successfully applied to various natural language processing tasks, such as language modeling, text classification, and machine translation. The initial idea of GPT is to pre-train a deep neural network on a large corpus of unlabeled data, and then fine-tune the model on a downstream task with a small amount of labeled data (Radford et al., 2018). This approach has been shown to achieve state-of-the-art performance on many natural language processing benchmarks.

In recent years, there has been growing interest in applying GPT to code generation tasks. Code generation is the process of automatically generating computer programs from high-level specifications, such as natural language descriptions, graphical models, or structured data. It has many practical applications, such as software engineering, data analysis, and scientific computing. However, code generation is a challenging task due to the complexity and diversity of programming languages, as well as the high level of abstraction required for generating code from non-code inputs.

In this report, we first provide a brief review of the Transformer (Vaswani et al., 2017), which is the underlying building block of GPT. We also summarise the key innovations of the first two open-source generations of GPT (Radford et al., 2018, 2019). Furthermore, we present the results of our experiments, where we used GPT-2 as the generative model and the CodeParrot dataset, which focuses on Python codes. Finally, we discuss the future potential of GPT for code generation and other natural language generation tasks.

2 Related Work

Code generation has been a topic of interest in the natural language processing and machine learning (ML) communities for many years (Iyer et al., 2018). Recently, deep learning methods have shown promising results in code generation tasks, including generating method names (Alon et al., 2019), and generating entire functions from natural language descriptions (Yin and Neubig, 2017). Among deep learning methods, the transformer model has been widely used in NLP tasks and has also shown potential in code generation (Thapa et al., 2022). Researchers have also explored different modifications to the transformer architecture to improve its performance in various NLP tasks (Lu et al., 2019). In this paper, we focus on using the GPT-2 model for code generation tasks, which has shown promising results in natural language generation tasks (Radford et al., 2019).

3 Transformer

3.1 Attention

Recurrent Neural Networks (RNNs) have been widely used for machine translation tasks. However, they have some limitations, including the representation of the input sentence by a single embedding and the propagation of errors during decoding.

The *attention* mechanism (Figure 1) addresses these limitations by allowing the decoder to focus selectively on relevant parts of the encoded information, rather than relying solely on the context

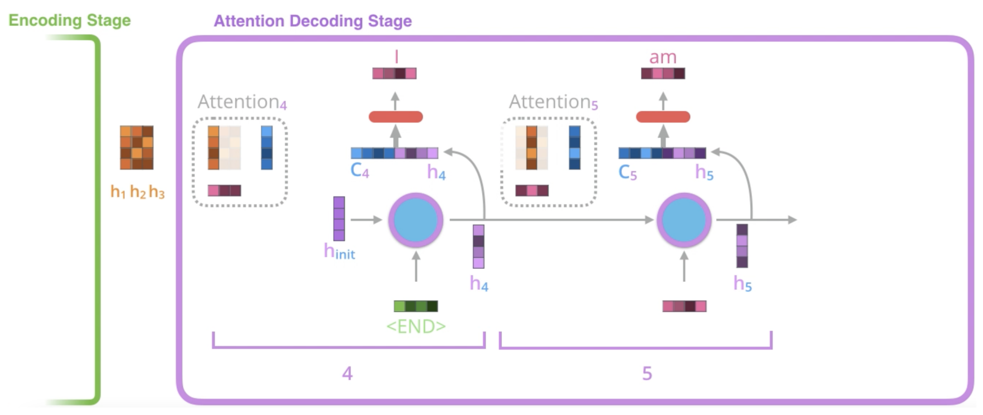


Figure 1: Attention mechanism illustrated (Alammar, 2018)

vector. Attention is essentially a way for the decoder to score the relevance of each encoded hidden state to the current decoding step, and then use this information to compute a weighted average of the hidden states. At each decoding step, it allows the decoder to selectively attend to different parts of the input sequence, based on the relevance of each part to the current decoding step. In a nutshell, attention helps the model to capture long-range dependencies.

In addition to machine translation, attention has also been used in many other NLP tasks, such as sentiment analysis, question answering, and language modelling. Later, *Transformer* architecture, based on attention, became one of the most widely used and effective architectures in language understanding and generation.

3.2 Transformer

The attention mechanism is a powerful tool for processing sequential data, as it allows the model to selectively attend to relevant parts of the input sequence at each decoding step. However, attention mechanism still processes the input sequence sequentially, which can be computationally expensive for long sequences. Transformer (Vaswani et al., 2017) is a model architecture that was specifically designed to address this issue by enabling parallel processing of the input sequence.

Transformer architecture (Figure 2) consists of a stack of encoder and decoder layers, with positional encoding used to represent the position of each token in the sequence. The output of the last encoder layer is passed to all decoder layers, and the encoding of the output stage is processed to ensure that the decoder can only access historical items when making predictions (known as *causal language modelling*).

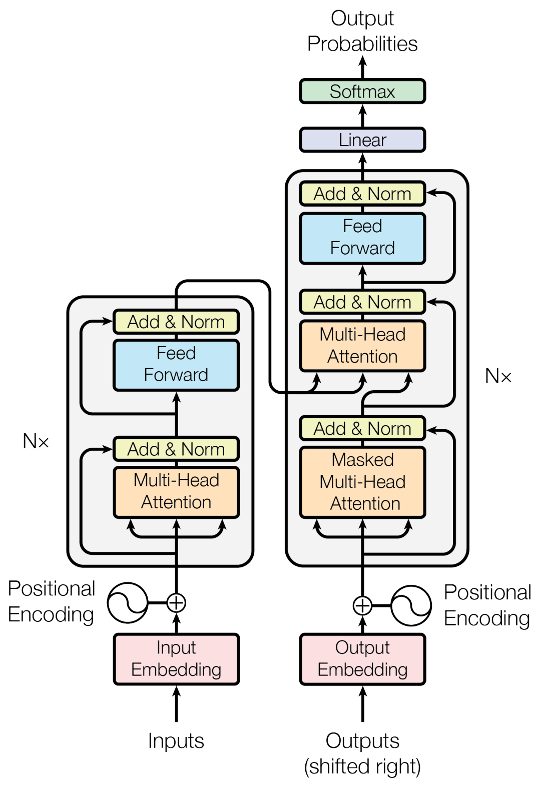
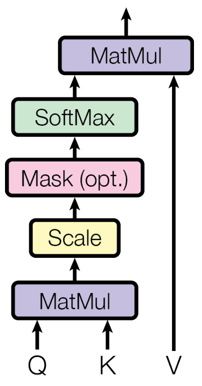


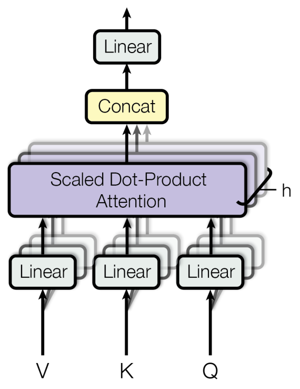
Figure 2: The Transformer model architecture (Vaswani et al., 2017)

The key innovation of the Transformer is multi-head attention (Figure 3a), which allows the model to compute attention in parallel across multiple heads. For each token in the input sequence, the model calculates query and key vectors of dimension d_k and a value vector of dimension d_v. These vectors are multiplied with weight matrices W^Q, W^K, W^V to obtain matrices Q, K, V. Then,

$$\text{Attention}(Q,K,V) = V \cdot \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right).$$



(a) Scaled Dot-Product Attention



(b) Multi-Head Attention consists of several attention layers running in parallel

Figure 3: Attention in Transformers (Vaswani et al., 2017)

Multi-head attention trains a set of weight matrices in each head in parallel and outputs respective attention. The outputs of all heads are concatenated and then multiplied with the matrix W^O for the final attention. This approach enables the model to process the input sequence in parallel, which makes it highly efficient and scalable.

4 Generative Pre-trained Transformers

OpenAI's GPT models have been incredibly successful and have become almost ubiquitous in NLP. These models have significantly advanced decoder-only models (Figure 4), and have pushed the boundaries of natural language understanding. This section introduces the first two generations of GPT that are *open-source* and freely available.

4.1 GPT-1

Supervised learning approaches had demonstrated impressive results on various NLP tasks. However, obtaining the large amounts of manually-labelled data required for supervised training can be prohibitively expensive. Moreover, these models are often task-specific, which limits their scalability and flexibility. As a result, there is a growing interest in leveraging the vast amounts of unannotated text corpora available.

OpenAI's GPT model, first introduced by Radford et al. (2018), aims to improve the state-of-the-art in natural language processing by leveraging unsupervised pre-training to learn better text representations. This approach allows the model to understand language comprehensively from large amounts of unlabelled text data, which can then be fine-tuned for specific downstream tasks.

The model architecture used in GPT (Figure 5) is a multi-layer Transformer decoder (Liu et al., 2018). The model training process involves two stages: *generative (unsupervised) pre-training* and *discriminative (supervised) fine-tuning*.

4.1.1 Unsupervised pre-training

The unsupervised pre-training stage involves training a large neural language model to predict the next token in an unannotated corpus $\mathcal{U} = \{u_1, \dots, u_n\}$. This approach, known as *autoregressive next-word-prediction*, is commonly used in NLP to learn high-quality text representations. With stochastic gradient descent, it learns the model parameters Θ that maximises the sequence's likelihood expressed the autoregressive formulation with a context window of size k:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i \mid u_{i-k}, \dots, u_{i-1}; \Theta). \quad (1)$$

4.1.2 Supervised fine-tuning

After the unsupervised pre-training stage, GPT model is fine-tuned on the specific downstream task in a supervised manner. The fine-tuning process involves updating the parameters of the pre-trained model on a task-specific labelled dataset $\mathcal{C}$ consisting of sequence-label pairs $(x^1, \dots, x^m; y)$, and then obtaining the final transformer block's activation h_l^m. This activation is then passed through an extra linear output layer with parameters W_y and softmax activation function to estimate the likelihood of a label conditioned on the input sequence:

$$P(y \mid x^1, \dots, x^m) = \text{softmax}(h_l^m W_y).$$

Hence, we maximise during fine-tuning the following objective:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y \mid x^1, \dots, x^m). \quad (2)$$

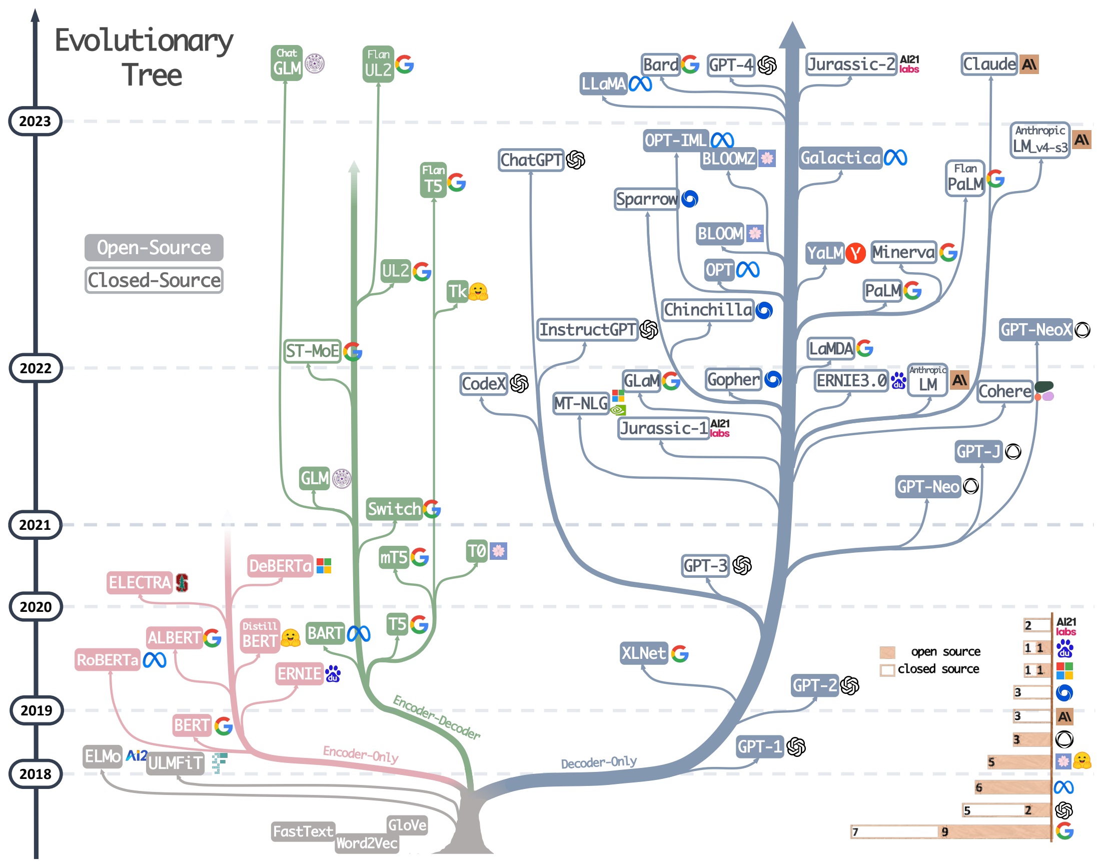


Figure 4: The evolutionary tree of modern large language models (LLMs), where OpenAI models occupy important positions (Yang et al., 2023)

To further refine the GPT model, Radford et al. (2018) proposed an alternative fine-tuning objective combining task-specific inference objective (Equation 2) and an auxiliary language modelling objective (Equation 1) weighted by hyperparameter λ:

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda \cdot L_1(\mathcal{C}). \qquad (3)$$

This new objective helps accelerate the convergence of model parameters. More importantly, the auxiliary objective contributes noticeably to the generalisability of the model: it helps the model retain its ability to generate high-quality text and capture the long-term dependencies between the tokens. Thus, the model can transfer its pre-existing knowledge of language to the specific task while still maintaining its high-level language understanding ability. Also, this auxiliary term is a regulariser that helps prevent overfitting on small labelled datasets, enabling the model to understand language more robustly.

4.2 GPT-2

LLMs such as GPT, have shown great potential to learn and perform various language tasks in an unsupervised manner, given a large enough training dataset. This success is in line with the work we presented in the course, where Pires et al. (2019) demonstrated that BERT, another Transformer-based language model, can perform zero-shot cross-lingual tasks effectively without supervised training. Furthermore, the discovery of Sutskever et al. (2015) that the supervised training objective is essentially the unsupervised one evaluated on a subset of data suggests that unsupervised pre-training is a direction worth investing in. The supervised minimum coincides with the unsupervised one; thus, we only need to train a model to optimise the unsupervised loss.

Based on the finding of Sutskever et al. (2015), Radford et al. (2019) proposed GPT-2, an extension of its predecessor, GPT-1. This new model further exploits LLM's unsupervised learning ability, and can get rid of supervised fine-tuning. It represents a significant advancement in natural language pro-

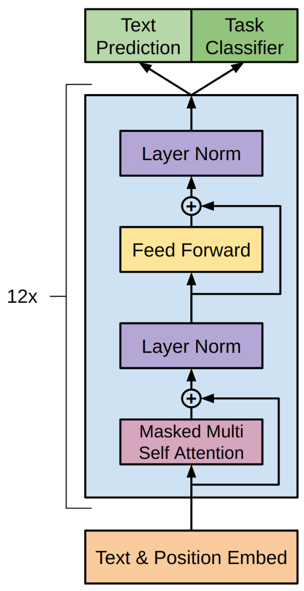


Figure 5: Transformer architecture used in the initial work of GPT (Radford et al., 2018)

cessing, and has demonstrated impressive zero-shot results on various tasks.

4.2.1 Model architecture

Table 1: Architecture hyperparameters for GPT-2's four model sizes

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

GPT-2's model architecture is based on that of GPT-1, with just several slight changes that help it capture more nuanced and sophisticated patterns in the data:

1. **Number of parameters.** GPT-2 has four model sizes (Table 1). The largest one has 1.5 billion parameters, which is ten times more than GPT-1's 117 million. This increase allows the model to capture more complex patterns and structures in the inputs, and learn more accurate language representations.

2. **Vocabulary and context sizes.** The vocabulary size of GPT-2 is enlarged from 40 478 to 50 257. Besides, the model can handle contexts of up to 1024 tokens, i.e., twice the length of GPT-1's 512-token limit. These increases in token limit enable the model to process longer and more complex sequences of text, which is especially useful for tasks like translation and summarisation.

3. **Layers.** As for initialisation, since signal on the residual path will accumulate as model depth grows, the weights of residual layers are scaled down by a factor of $1/\sqrt{N}$, where N is the number of residual layers, to compensate for this accumulation. Layer normalisation is moved to each sub-block's input, making it resemble a pre-activation residual network (He et al., 2016). Then, the output of the final self-attention block will pass through an extra normalisation layer.

4.2.2 Dataset

A crucial factor in the success of unsupervised training for language models is the availability of a single large and diverse dataset. Only with such a dataset can the model learn accurate and robust representations of language. The larger the dataset and the more diverse its contents, the better the model's ability to capture the nuances and complexities of language.

To address this need, Radford et al. (2019) scraped approximately 8 million web pages, providing over 40 GB of text for model training. To ensure data quality, they only include web pages that had received a certain level of engagement from users, as measured by their Reddit karma score. Specifically, only pages that received more than two karma were scraped, as this score indicates that the content has attracted other users, and is likely to be of high quality. Adopting this heuristic indicator requires no manual selection, which would have been prohibitively time-consuming. The resulting *WebText* dataset provides a rich and varied source of training data that has been instrumental in the success of GPT-2.

5 Experiments

5.1 Experimental design

In our experiments, we finetuned GPT-2 model for code generation task using codeparrot dataset. We used HuggingFace's `datasets` and `transformers` libraries to load the dataset and GPT-2 models and conducted the training in the Google Colab's GPU. For comparison, we considered CodeParrot model from HuggingFace, which has fine-tuned GPT-2 model on codeparrot dataset as well.

5.2 Dataset

`CodeParrot` dataset was created by accessing the Github dump from Google's BigQuery and was filtered for only Python files. As a result, initial codeparrot is 180 GB dataset with 20 million files. However, due to the presence of duplicates, the performance was very low. After removing duplicates using some heuristics described in Chen et al. (2021), the cleaned dataset (`codeparrot-clean`) is still 50 GB big and available on the Hugging Face Hub.

5.3 Results

We evaluated the models on OpenAI's HumanEval benchmark which was proposed by Chen et al. (2021). It measures the performance of code generation models on almost 164 coding challenges. In the pass@k metric, k code samples are generated per problem, and a problem is considered solved if any sample passes the unit tests and the total fraction of problems solved is reported. In most papers, 200 candidate program completions are sampled, and pass@1, pass@10, and pass@100 are computed using an unbiased sampling estimator. For CodeParrot, the pass@100 metric is slightly above 10 % for small model and close to 20 % for large model. In our experiments, we successfully achieved pass@1 with score 0.5 and pass@2 with with score 1.0.

6 Discussion

Our study aimed to determine if it's feasible to teach large language models to generate accurate code bodies based on code pieces passed to them. By fine-tuning GPT-2 on code from GitHub, we found that our models displayed strong performance on a dataset of human-written problems with difficulty level comparable to easy interview problems.

The recent advancements in large language models, including GPT-3 (Brown et al., 2020), ChatGPT (Ouyang et al., 2022), and GPT-4 (OpenAI, 2023), have revolutionised natural language understanding and pushed NLG research to new heights. These powerful language models enable us to bypass fine-tuning and generate satisfactory outputs through zero-shot or few-shot prompting.

However, these models often suffer from hallucinations and repeated patterns in the single-round output. A current research direction to address this is guiding the model to revise its output through rounds of interactions, such as correcting generated code through rubber duck debugging and unit tests (Chen et al., 2023). Also, we can let the model perform human-like logical reasoning with the help of tools like search engines and Wikipedia (Yao et al., 2023). Additionally, assigning roles (e.g., researcher and decider) to multiple language models and having them interact can lead to higher-quality outputs (Nair et al., 2023).

Another research direction is developing more effective decoding algorithms. In our previous expriments (Josifoski et al., 2023), we found that, since the likelihood is the surrogate function of the task-specific utility during training, the two cannot align very well in GPT-2 and previous models. In other words, the output the model thinks is more likely is not necessarily what we want in the task. As the number of model parameters and dataset size increases, we expect this issue to be less prominent. On the other hand, current models commonly use sampling and beam search, but these algorithms cannot guarantee finding the most likely output sequence. Value-guided beam search (He et al., 2017; Krishna et al., 2022) and Monte-Carlo tree search, which incorporate external knowledge, have shown promising results but require higher computing power. Therefore, defining and effectively utilising additional knowledge is also worth investing in.

Nonetheless, the lack of transparency in closed-source models poses challenges for research and may lead to misleading results. For example, GPT-4 does not disclose the training dataset used. Thus, we may coincidentally use some of those data in our experiments and obtain overly optimistic results; publicly accessible datasets are more susceptible to this risk (Nair et al., 2023).

Overall, large language models have enormous potential for further advancing natural language understanding, and continued research into their capabilities and limitations is essential.

References

Jay Alammar. 2018. Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention).

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2019. code2seq: Generating sequences from structured representations of code.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind

Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. ArXiv:2005.14165 [cs].

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching Large Language Models to Self-Debug. ArXiv:2304.05128 [cs].

Di He, Hanqing Lu, Yingce Xia, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2017. Decoding with value networks for neural machine translation. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. In *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 630–645, Cham. Springer International Publishing.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context.

Martin Josifoski, Maxime Peyrard, Frano Rajič, Jiheng Wei, Debjit Paul, Valentin Hartmann, Barun Patra, Vishrav Chaudhary, Emre Kiciman, and Boi Faltings. 2023. Language model decoding as likelihood–utility alignment. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1455–1470, Dubrovnik, Croatia. Association for Computational Linguistics.

Kalpesh Krishna, Yapei Chang, John Wieting, and Mohit Iyyer. 2022. RankGen: Improving text generation with large ranking models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 199–232, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by Summarizing Long Sequences. ArXiv:1801.10198 [cs].

Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Understanding and improving transformer from a multi-particle dynamic system point of view.

Varun Nair, Elliot Schumacher, Geoffrey Tso, and Anitha Kannan. 2023. DERA: Enhancing Large Language Model Completions with Dialog-Enabled Resolving Agents. ArXiv:2303.17071 [cs].

OpenAI. 2023. GPT-4 Technical Report. ArXiv:2303.08774 [cs].

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. ArXiv:2203.02155 [cs].

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. *OpenAI blog*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Ilya Sutskever, Rafal Jozefowicz, Karol Gregor, Danilo Rezende, Tim Lillicrap, and Oriol Vinyals. 2015. Towards Principled Unsupervised Learning. ArXiv:1511.06440 [cs].

Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. 2022. Transformer-based language models for software vulnerability detection.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. ArXiv:2304.13712 [cs] version: 2.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation.