CS499 Independent Study Spring 2023

Adit Goyal – AGI8759

June 7, 2023

Faculty Advisor: Prof. Zach Wood-Doughty

1 Introduction

Our aim for last quarter's independent study was to understand the progress in the Large Language Models (LLMs) over the last couple of years that led to the creation of the underlying architecture in ChatGPT. ChatGPT is a chat bot that was deployed and launched in November, 2022 for public use. The chat bot used a transformer-based Large Language Model called GPT-3, which was developed by the researchers at OpenAI. OpenAI had released two previous versions, namely GPT and GPT-2, which were much smaller in size compared to GPT-3. Over the course of the study, we went through a hand-picked set of research papers that we felt contributed to the research of GPT-3 in a major way. Furthermore, we covered a research that discussed the various ways in which LLMs have been affecting our surroundings (environmentally and socially) [1].

During the independent study in the Spring 2023 quarter, we decided to put the theoretical knowledge of the previous quarter to a practical use. We thought about getting insights as to how to train a Large Language Model (transformer model) without the resources of a multi-million dollar organization. We picked the task of Machine Language Translation, because of the ease of verification of translation results. After careful consideration, we picked the SumGenToBT model, proposed by the authors of [2].

2 Preliminary Knowledge

In this section, we will discuss some of the common terms that will help the reader to easily understand the topics we cover in Section 3.

2.1 Large language models in machine translation

This was one of the early papers on Large Language Models, published by Google in 2007 [3].

Backoff smoothing

Backoff smoothing is a technique used in NLP to smooth the probabilities estimated by language models. The idea behind backoff smoothing is to estimate the probability of an n-gram based on the probabilities of lower-order n-grams if the higher-order n-gram has not been seen in the training data. For example, if the probability of a bigram (e.g., “cat dog”) cannot be estimated from the training data, the probability of the unigram “cat” may be used instead. If the unigram “cat” has not been seen in the training data either, the language model can fall back to a default probability based on some other smoothing method. Backoff smoothing can help to address the problem of zero probabilities in language models and prevent overfitting to the training data.

Information loss in two-pass decoding

In two-pass decoding, the first pass generates an initial hypothesis, and the second pass refines the hypothesis based on additional information. Information loss in two-pass decoding occurs because some information from the first pass is discarded during the second pass, which can lead to a reduction in the quality of the final output. For example, the first pass may generate a hypothesis that is grammatically correct but semantically incorrect, and the second pass may not be able to correct this error because the information that was available in the first pass is lost.

Kneser-Ney smoothing

Kneser-Ney smoothing works by estimating the probability of an n-gram based on the number of times it has been seen in the training data and the number of times its lower-order n-grams have been seen. This allows the model to more accurately capture the context in which the n-gram occurs and to avoid overfitting to the training data.

BLEU score

The BLEU (Bilingual Evaluation Understudy) score is a common evaluation metric for machine translation. It measures the overlapping n-grams (sequence of words) between the predicted translation and the reference translation and assigns a score based on the number of matches. Higher BLEU scores indicate a higher degree of similarity between the predicted and reference translations, and BLEU scores closer to 1.0 are considered better. It is a commonly used metric, but has limitations and has been criticized for not always accurately reflecting the quality of a machine translation.

2.2 Attention is all you need

This landmark paper by Google Brain introduced transformers in 2017, which formed the basis of entire development in the field of NLP since then [4].

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of deep learning model that are used for processing sequential data such as speech, text, and time series. Unlike traditional neural networks, RNNs have a "memory" mechanism in the form of hidden states that allow them to retain information from previous time steps, making them well-suited for tasks such as language modeling, machine translation, and speech recognition. The hidden states are computed based on the current input and the previous hidden state, and this process is repeated for every time step in the sequence. There are various types of RNNs, including Vanilla RNNs, Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs), which vary in terms of the type of hidden state used and the mechanism for controlling information flow through the hidden states.

Self-attention mechanisms

Self-attention mechanisms are a type of attention mechanism used in deep learning models to dynamically weight the importance of different elements in a sequence of data. They allow the model to focus on relevant parts of the input and produce more context-aware representations. In self-attention, each element in the sequence is first transformed into a query, key, and value representation. The model then computes attention scores between each query and key and uses these scores to weigh the values to compute a weighted sum, which becomes the new representation of the input sequence. This process is repeated multiple times to produce the final representation. Self-attention has been successfully used in various tasks such as machine translation, text classification, and image captioning. It has become a popular building block for Transformer-based models, which have achieved state-of-the-art results in a variety of NLP tasks.

Multi-headed self-attention mechanisms

It allows the model to attend to multiple parts of the input sequence in parallel, thus capturing more fine-grained information from the input. In multi-head self-attention, the input sequence is transformed into multiple sets of queries, keys, and values, each representing a different aspect of the input. Attention scores are computed between each query and key, and the values are then weighed and summed to produce multiple attention outputs. These outputs are concatenated and passed through a feedforward layer to produce the final representation.

Residual connection

The idea behind residual connections is to make it easier for the network to learn complex functions by allowing the gradients to flow more easily through the network during backpropagation. The residual connection enables the network to learn the residual or the difference between the desired output and the current output, rather than learning the entire function from scratch.

Key-value pairs in self-attention

The key-value pairs in a self-attention mechanism provide a way to weigh the importance of each element in the input sequence, which allows the model to attend to relevant parts of the input and produce context-aware representations. Key is the similarity score between input elements, and value is the information about each element.

2.3 Improving language understanding by generative pre-training

This was the paper that introduced Generative Pre-Training (GPT), published by OpenAI in 2018 [5].

Token embedding matrix

A token embedding matrix is a matrix in a natural language processing (NLP) model that represents words or subwords (tokens) as numerical vectors (embeddings). Each row in the matrix corresponds to a unique token, and each column represents a dimension of the token's representation. The values in the token embedding matrix are learned during training, typically using an optimization algorithm such as stochastic gradient descent. The goal is to learn meaningful representations that capture the relationships between words in the language.

Zero-shot learning

Zero-shot learning is a machine learning setting in which the model is expected to classify or predict instances of classes that it has never seen during training. In other words, the model must generalize its knowledge to new, unseen classes based on its prior experience with other classes.

2.4 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Published in 2018 by Google Ai [6].

Cloze task

The Cloze task is a type of natural language processing (NLP) task that involves predicting missing words in a sentence or a text. In the Cloze task, a portion of the sentence or text is removed and replaced with a placeholder, such as [MASK]. The goal is to predict what word or phrase should be inserted in place of the [MASK]. The Cloze task is typically used as a way to evaluate the language understanding capabilities of NLP models. For example, a model trained on a large corpus of text can be evaluated on its ability to predict the missing words in a Cloze task. The performance of the model can be evaluated based on its accuracy in predicting the correct words. Also known as MLM (masked language model).

Downstream task

Downstream tasks refer to specific NLP tasks that are applied to a pre-trained model to fine-tune it for a specific task. These tasks are called ”downstream” because they are applied after the model has been pre-trained on a large corpus of data. For example, a pre-trained language model such as BERT can be fine-tuned for downstream tasks such as sentiment analysis, question answering, or text classification. In this process, the model is fine-tuned on a smaller, task-specific dataset to adapt it to the specific task at hand. This fine-tuning process can improve the performance of the model on the downstream task compared to using the pre-trained model directly.

2.5 Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context

Research on transformers without a fixed-length constraint, published by Google and CMU in 2019 [7].

Vanilla transformers

A vanilla Transformer is a standard Transformer model with no additional bells and whistles. It has the standard structure, consisting of an encoder and decoder, multi-head attention mechanism, feed-forward network, and residual connections, as described in the original paper “Attention is All You Need” by Vaswani et al [4]. However, researchers and practitioners often modify the Transformer architecture to improve performance, and these modified versions are not considered ”vanilla.” Some examples include the BERT (Bidirectional Encoder Representations from Transformers) model.

Perplexity

Perplexity is a common evaluation metric used in Natural Language Processing (NLP) to measure the quality of a language model. It provides an estimate of how well a language model predicts a sample of text by comparing its predictions to the actual text. In mathematical terms, perplexity is defined as the exponentiation of the cross-entropy loss between the predicted probabilities and the true probabilities of the words in the text. The cross-entropy loss is used to measure the difference between the predicted probabilities and the actual probabilities, and the exponentiation makes the result a more intuitive measure of the model’s quality.

2.6 Language Models are Unsupervised Multitask Learners

GPT-2 published by OpenAI in 2019 [8].

ROUGE

It calculates the overlap between the generated summary and the reference summary in terms of n-grams, where n can be 1 (unigrams), 2 (bigrams), or 3 (trigrams). The most common ROUGE metric is ROUGE-N, which calculates the recall of the n-grams in the generated summary with respect to the reference summary. There are several variants of ROUGE, including ROUGE-1, ROUGE-2, and ROUGE-L, which differ in the way they calculate the overlap between the generated and reference summaries. For example, ROUGE-L measures the longest contiguous sequence that appears in both the generated and reference summaries.

2.7 Language Models are Few-Shot Learners

GPT-3 published by OpenAI and JHU in 2020 [9].

Few-shot learning

Few-shot learning is a type of machine learning that focuses on learning from a small number of examples. It is a crucial problem in artificial intelligence and machine learning, as real-world applications often involve learning from very few examples, or even from just one example. In few-shot learning, the goal is to learn a model that can generalize to new classes based on only a few examples of each class.

Beam search

Beam search is a search algorithm that is commonly used in Natural Language Processing (NLP) and other domains. It is a heuristic search algorithm that explores a search space by maintaining a set of k-best candidates at each step, where k is a user-defined parameter known as the beam width. In a beam search, the algorithm starts with an initial state, and at each step, it generates a set of k-best candidates by applying a set of possible actions. The set of k-best candidates is then used to generate the next set of k-best candidates, and this process is repeated until a stopping criterion is met.

Length penalty in beam search

The idea behind length penalty is to reward shorter sequences with higher likelihood, while penalizing longer sequences with lower likelihood. This helps to balance the trade-off between generating short, high-quality sequences and generating longer, lower-quality sequences.

2.8 DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

Distilled version of BERT, published in 2020 [10].

Distillation

It is a powerful technique for compressing large pre-trained transformer models into smaller, more efficient models, while maintaining their accuracy and generalization ability. It allows for the deployment of these models on resource-constrained devices and can make them more adaptable to new tasks and domains.

Soft targets

Soft targets in multi-class classification refer to the use of class probabilities as targets instead of hard labels (i.e., one-hot encoded vectors). Instead of using a one-hot encoded vector to represent the true label for each instance, soft targets use a probability distribution over the classes, where the probability of the true label is higher than the probabilities of the other classes.

3 Observations and Notes

3.1 Large language models in machine translation

Encoder decoder structure, trying to reduce computation time by Map Reduce techniques. The encoder creates n-grams (vocabulary) and the decoder uses a distributed language model to try and get the ideal sequence. Tries to guess a word based on the last n words. Problems in calculating probabilities due to sparse data. Can make n bigger in n-grams but the problem persists. Introduces a new smoothing algorithm called stupid backoff that uses relative frequencies. Average sentence length is 22. N-grams max n is 5 for experiments.

3.2 Attention is all you need

Limitation of RNNs is not being able to parallelize due to being sequential, high complexity of each layer and can't handle long sequence lengths. Attention mechanisms help form dependencies irrespective of their distance in input/output sequences. Self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d, and have a constant number of sequential operations. Transformers are unique since they rely only on self-attention, without any use of RNNs or CNNs. Scaled dot-product attention is used because of better performance in matrix multiplication. To prevent very small gradients, scaling is used (factor of square root (dk)). Transformers use stacked self-attention sub-layers within the encoder and decoder, performing different roles for encoder-decoder, encoder and decoder layers.

3.3 Improving language understanding by generative pre-training

Rather than training a model individually for each NLP task, pre-train a model on a large corpus of data and fine-tune for each use-case (transfer learning). Semi-supervised learning setting is used, wherein pre-training is done on an unlabelled corpus and fine-tuning is done for labeled training sets depending on the use-case. The pre-training part uses a decoder-only transformer with multi-headed self-attention creating a generative model that produces an output distribution over target tokens. The aim is to have the pre-trained model generate outputs, similar to the text in the initial corpus, depending on the task we fine-tune it for during the training stage.

3.4 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

A pre-trained NLP model that can be fine-tuned for any task and by adding an additional layer at the end, we can perform different kinds of tasks. GPT-1 was unidirectional but BERT is bidirectional. Pre-training is for two tasks: masked language model (predicting the missing word in a sentence) and next sentence prediction (guessing whether two sentences are consecutive or not). Bi-directional attention mechanism helps it to learn contextual meaning and helps in tasks like question-answering.

3.5 Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context

Authors introduce recurrence into the self-attention models and use relative positional encodings rather than absolute ones. Recurrence is introduced by sending information through these connections. One hidden state sends the information to the next segment's hidden state. Every two consecutive segments in the corpus share information like this, essentially creating recurrence in the hidden states. Relative positional encodings help avoid temporal confusions. All this helps the transformer to remember long range dependencies and maintain contextual information.

3.6 Language Models are Unsupervised Multitask Learners

Very similar to GPT-1. Key differences include addition of layer normalization in sub-layers, bigger training set and a bigger model to train. The authors focus on providing SOTA results in zero-shot task-transfer, rather than fine-tuning the model for each task (like GPT-1). The model is supposed to guess the input and the task it has to perform based on the query.

3.7 Language Models are Few-Shot Learners

GPT-2 mistakenly calls few-shot learning as zero-shot since they might have some examples of the task they intend to perform in the training set. They train the model on a huge corpus initially in an unsupervised manner. Then for making sure that the model performs on different kinds of tasks, considering few-shot learning, they give k examples (where k is based on the context window and example is the context and the output) for a task. At $(k+1)^{th}$ example, we just give the context and expect it to produce the output.

Table 1: Comparison between GPT, GPT-2 and GPT-3.

GPT-1	GPT-2	GPT-3
117M parameters	117M, 345M, 762M, 1.5B parameters	125M - 13B, 175B parameters
Unsupervised pre-training and supervised fine-tuning	Unsupervised pre-training and zero-shot task transfer	Unsupervised pre-training and in-context learning for results in few-shot setting
Learning objective: P(output—input)	Learning objective: P(output—input, task)	Similar to GPT-2, but now focused on few-shot, one-shot and zero-shot learning

3.8 On the Dangers of Stochastic Parrots: Can Language Models be Too Big

The authors of the paper claim that while large language models in NLP like GPT-2, GPT-3, T-NLG, BERT, etc. have shown impressive results over a variety of tasks, they also have a negative impact on the society in terms of environmental concerns, increasing marginalization of under-represented communities and producing results that might be deemed inappropriate by certain people, based on their thinking. The authors do not propose a new framework, which they can compare with previous researches. However, they do suggest a few steps towards mitigating the negative effects of LM, like curating the dataset with careful consideration, reporting environmental costs to train and deploy the model, etc. The authors do not give theoretical insights into how the existing methods work, the central focus of the paper was towards the impacts of the LM on the environment and the people. These claims matter since it has been a trend to increase the number of model parameters and increasing the training set size to improve the performance of transformers, without taking into consideration the amount of resources each research would take and how some sections of people might respond to the results produced by deploying the model in the real world.

3.9 Distilling the Knowledge in a Neural Network

Distilling is the concept analogous to the physics concept where you raise the temperature and entropy increases. With increased entropy, there is an equal probability of any possible event happening. In classification models, with increased value of T, there is an equal probability of all the classes. We want the model to be able to relate to two classes, and say how different they are. We don't want absolute probabilities, we want to know how close two classes are.

4 Summarize and Generate to Back-Translate: Unsupervised Translation of Programming Languages

We started the quarter off with the aim of getting hands-on with LLMs, and have a project that could give me an overview of the entire life-cycle of a Deep Learning project. After careful consideration, we landed on programming language translation because of two reasons: primarily because we could easily find a dataset for this task and not a lot of research had been done for this use-case in particular, and secondarily due to the fact that the results can be validated easily by us. Picking any other language translation would always have the language barrier, and could restrict our understanding of the performance of the model. However, with programming language translation, validation becomes easier to us since the codes can be checked for errors and as a CS major, I would understand the results easily with decent reliability.

For the next step, we looked over the past work done by researchers for this task. In particular, the key research was presented in:

- Unsupervised Translation of Programming Languages (TransCoder): Lachaux *et al.* [11] use Google's monolingual BigQuery dataset for training.

- CodeXGLUE: The authors of [12] performed code-to-code translation using the Code-Trans dataset. The authors use bilingual data, and provide a pipeline to finetune on CodeBERT.

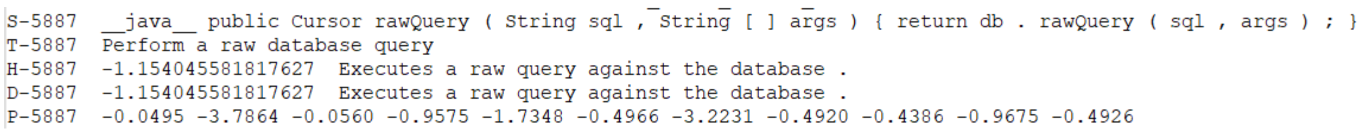
- Facebook CodeGen: The user must give the source code written in the original programming language and define the target language in order to use CodeGen for programming language translation. Then, CodeGen employs a series of modifications and enhancements to automatically create code that upholds the target language's syntax and rules while maintaining the original functionality. It uses monolingual data from GCP for training.

- SumGenToBT (Summarise and generate to back translate): The authors of [2] keep CodeGen as baseline to evaluate their performance. The caveat in programming language translation is lack of parallel data available. The authors first train a summarize and generate model which translates from Java Code to Natural Language and from Summary to Python Code, and vice versa. For the next step, the authors perform unsupervised program translation training using Back Translation. The authors use the pre-trained PLBART Program and Language, a Bidirectional and Autoregressive Transformer, and finetune the SumGenToBT model further. The PLBART model was pre-trained on unlabeled data across PL&NL to learn multilingual representatives, using the Java, Python and English dataset.

We decided to pick SumGenToBT because, as compared to others, it is much smaller and gave comparable results. SumGenToBT uses 140 Million trainable parameters, which is less than half the size of CodeGen, having 312 Million parameters. However, CodeGen does not suffer from the language generation issue like SumGenToBT. We were hoping that if we have time left over at the end of the quarter, we could work on helping SumGenToBT to overcome this problem.

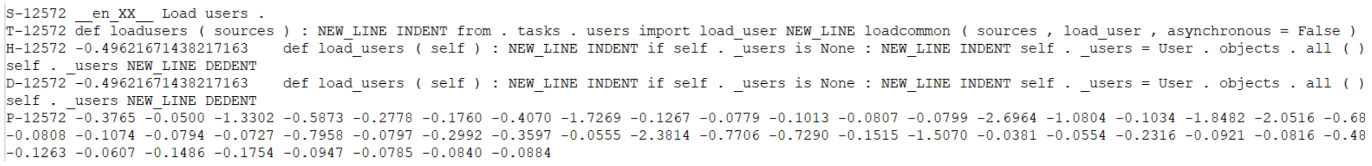
5 Results

We trained the summarize and generation part of the SumGenToBT model from scratch. The results for the same can be observed in Figures 1 and 2.



```
S-5887   __java__ public Cursor rawQuery ( String sql , String [ ] args ) { return db . rawQuery ( sql , args ) ; }
T-5887   Perform a raw database query
H-5887   -1.154045581817627   Executes a raw query against the database .
D-5887   -1.154045581817627   Executes a raw query against the database .
P-5887   -0.0495 -3.7864 -0.0560 -0.9575 -1.7348 -0.4966 -3.2231 -0.4920 -0.4386 -0.9675 -0.4926
```

Figure 1: Sample of the results generated by our model so far when translating from Java to English.



```
S-12572 __en_XX__ Load users .
T-12572 def loadusers ( sources ) : NEW_LINE INDENT from . tasks . users import load_user NEW_LINE loadcommon ( sources , load_user , asynchronous = False )
H-12572 -0.49621671438217163    def load_users ( self ) : NEW_LINE INDENT if self . _users is None : NEW_LINE INDENT self . _users = User . objects . all ( )
self . _users NEW_LINE DEDENT
D-12572 -0.49621671438217163    def load_users ( self ) : NEW_LINE INDENT if self . _users is None : NEW_LINE INDENT self . _users = User . objects . all ( )
self . _users NEW_LINE DEDENT
P-12572 -0.3765 -0.0500 -1.3302 -0.5873 -0.2778 -0.1760 -0.4070 -1.7269 -0.1267 -0.0779 -0.1013 -0.0807 -0.0799 -2.6964 -1.0804 -0.1034 -1.8482 -2.0516 -0.68
-0.0808 -0.1074 -0.0794 -0.0727 -0.7958 -0.0797 -0.2992 -0.3597 -0.0555 -2.3814 -0.7706 -0.7290 -0.1515 -1.5070 -0.0381 -0.0554 -0.2316 -0.0921 -0.0816 -0.48
-0.1263 -0.0607 -0.1486 -0.1754 -0.0947 -0.0785 -0.0840 -0.0884
```

Figure 2: Sample of the results generated by our model so far when translating from English to Python.

The code provided by authors had some integration error, which made the second part of the model not usable. Eventually we decided to pivot early on, rather than waiting for the back translation to train completely. We decided to try making a single pipeline to translate from one programming language to the second, without going from Java to English and then from English to python.

Some of the significant hurdles we faced while trying this model out were:

- We relied on the Google Colab's free GPU for a significant portion of the quarter, which made things really difficult since we had both memory and computational constraints. Large language models are both memory and computationally intensive. We spent a significant amount of time trying to figure out ways to train small batches, or split the training time into multiple short intervals. Once we tried the model out and established that it is learning something, the way we want it to, we migrated the whole setup on the CS department's servers and trained the SumGenToBT's first part, which took around 3 days to run.

- The paper we picked was quite recent, and apparently we were the first ones to try their code out. The repository was not updated the final code authors must've used, and it was hard to figure out what was going wrong. To make things harder, they used the fairseq pipeline for setting up their training and testing, which made some things kind of implicit and we had to go down to the library level to make some changes, so that the code works. Furthermore, we couldn't train the model for back translation because the code-base had a lot of missing information, rendering it useless.

Further Steps

We had to conclude the study earlier because of lack of time, but in this section, we list out the things we would've liked to take if time wasn't a constraint:

- Create a single pipeline to observe the results of java-python, java-cpp, etc. rather than having to manually compare the results in the files with English language intervention.
- Fix the code for back translation, and observe whether the results are similar to the expectations or not.
- Try to overcome the language generation issue that authors faced with SumGenToBT. Moreover, also try to see if we can reduce the computational and memory needs of the model during training time.
- Deploy the model in a chatbot-like environment and playing around with it a little!!

References

[1] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, 2021, pp. 610–623.

[2] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Summarize and generate to back-translate: Unsupervised translation of programming languages," *arXiv preprint arXiv:2205.11116*, 2022.

[3] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean, "Large language models in machine translation," 2007.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[5] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[8] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[9] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[10] 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.

[11] M.-A. Lachaux, B. Roziere, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," *arXiv preprint arXiv:2006.03511*, 2020.

[12] Microsoft, "Codexglue." [Online]. Available: https://github.com/microsoft/CodeXGLUE/tree/main/Code-Code/code-to-code-trans