# Transfer Learning Strategies

Subjects: Computer Science, Artificial Intelligence

Contributor: Rajvardhan Patil , Venkat Gudivada

Discriminatively trained models perform well if labeled data are available in abundance, but they do not perform adequately for tasks with scarce datasets as this limits their learning abilities. To address this issue, Large language models (LLMs) were first pretrained on large unlabeled datasets using the self-supervised approach, where the learning was then transferred discriminatively on specific tasks. As a result, transfer learning helps to leverage the capabilities of pretrained models and is advantageous, especially in data-scare settings. For example, generative pretrained transformer (GPT) used the generative language model objective for pretraining, followed by discriminative finetuning. Compared to pretraining, the transfer learning process is inexpensive and converges faster than training the model from scratch. Additionally, pretraining uses an unlabeled dataset and follows a self-supervised approach, whereas transfer learning follows a supervised technique using a labeled dataset particular to the downstream task. The pretraining dataset comes from a generic domain, whereas, during transfer learning, data come from specific distributions (supervised datasets specific to the desired task).

language models PLMs large language model LLMs natural language processing NLP

## 1. Finetuning

Transfer learning started with feature-based techniques, where pretrained embeddings such as Word2Vec were used on the custom downstream models. Once learned, the embeddings are not refined to the downstream tasks, making them task-dependent. In finetuning, after copying the weights of the pretrained network, they are finetuned to adapt to the peculiarities of the target task. In finetuning, as the parameters learned during pretraining are adjusted to a specific downstream task, it outperforms the feature-based transfer technique. Such finetuning enables the model to learn task-specific features and improve the downstream task performance. As a result, the finetuned embeddings adapt not only to the context but also to the downstream task in consideration. So, unlike feature- or representation-based transfer, finetuning does not require task-specific model architecture. Although the finetuning strategy yields strong performance on many benchmarks, it has some limitations, such as the need for a large amount of downstream task-specific datasets, which can lead to poor generalization for data from out of distribution and the possibility of spurious features. During finetuning, instead of including all the parameters, adapter layers and gradual unfreezing techniques were proposed, which considered only a subset of the parameters during finetuning.

## 2. Adapter Tuning

Feature and vanilla finetuning techniques could be more parameter-efficient since they require new network weights for every downstream task. So, these techniques require an entirely new model for every downstream task. To address this issue, Ref. [1] proposed a transfer with the adapter module in which a module is added between the layers of a pretrained network. In each block of the transformer, these adapter layers, which are dense-RELU-dense blocks, are added after the feed-forward networks. Since their output dimensionality matches their input, no structural or parameter changes are required to insert adapter layers. During finetuning, most of the original model is kept fixed, and only the parameters from adapter layers get updated. In adapter tuning, task-specific layers are inserted, with only a few trainable parameters added per task. Also, a high degree of parameter sharing occurs as the original network is kept fixed.

Unlike the feature-based technique, which reads the inner layer parameters to form the embeddings, adapters write to the inner layers instead, enabling them to reconfigure network features. The main hyperparameter of this approach is the feed-forward network's inner dimensionality 'd' since it determines the number of new parameters that will be added to the model. This approach is a promising technique in the experiments conducted in [2]. Adapter tuning attains comparable performance with finetuning on NLU and NLG benchmarks by using only 2–4% task-specific parameters. Experiments from [1] demonstrated how BERT with adapters added only a few (3:6%) parameters per task to attain near SOTA on the GLUE benchmark.

## 3. Gradual Unfreezing

In gradual unfreezing, more and more of the model's parameters are finetuned over time. In this approach, at the start of finetuning, only the parameters of the final layer are updated first. Next, the parameters of the second-last layers are included in the finetuning. This process continues until the parameters of all the network layers are finetuned (updated). It is normally recommended to include an additional layer in finetuning, after each epoch of training. This approach was used in [2], where gradual unfreezing resulted in minor performance degradation across all the tasks.

## 4. Prefix Tuning

Finetuning, although it leverages the knowledge from pretrained models to perform downstream tasks, requires a separate copy of the entire model for each task as it modifies all the network parameters. To address this issue, prefix tuning [3] keeps the pretrained parameters frozen and optimizes only the task-specific vectors. These continuous task-specific vectors, called prefixes, are prepended to the input sequence so the subsequent tokens can attend to these vectors. Prefix tuning uses a small trainable module to train and optimize these small task-specific vectors associated with the prefix. The errors are backpropagated to prefix activations prepended to each layer during tuning. In prefix tuning for each task, only the prefix parameters are stored, making it a lightweight, modular, and space-efficient alternative. Despite learning 1000× fewer parameters than finetuning, prefix tuning [3] outperformed finetuning in low-data settings and maintained comparable performance in full-data settings. It also

extrapolated better to the examples with topics that were unseen during training by learning only 0.1% of the parameters.

# 5. Prompt-Tuning

Although finetuning the pretrained language models has successfully improved the results of downstream tasks, one of its shortcomings is that there can be a significant gap between the objectives used in pretraining and those required by downstream tasks. For instance, downstream tasks require objective forms such as labeling (parts of speech tagging) or classification, whereas pretraining is usually formalized as a next-token prediction task. One of the reasons behind the prompt-tuning approach was to bridge this gap between pretraining and finetuning objectives and help in better adaption of knowledge from pretrained models to downstream tasks. In prompt-tuning, prompts are used to interact with LLMs, where a prompt is a user-provided input to which the model responds. Prompting is prepending extra information for the model to condition on during the generation of output. This extra information typically includes questions, instructions, and a few examples as tokens to the task input.

## 5.1. Prompt Engineering

Prompt engineering involves the process of carefully designing optimal prompts to obtain optimal results. Prompts need to be constructed to best elicit knowledge and maximize the prediction performance of the language model. The prompt-based approach is a promising alternative to finetuning since, as the scale of LLMs grows, learning via prompts becomes efficient and cost-effective. Additionally, unlike finetuning, where a separate model is required for each downstream task, a single model serves multiple downstream tasks in prompt-tuning. They also help the model generalize better to held-out tasks and cross-tasks by using multitask prompts.

As per [4], finetuning on downstream tasks for trillion-scale models results in poor transferability. Also, these models need to be larger to memorize the samples in finetuning quickly. To overcome these issues, the prompt-tuning or P-tuning approach [5] is used, which is a parameter-efficient tuning technique. For example, GPT3 [6] (which was not designed for finetuning), heavily relied on handcraft prompts to steer the model for downstream applications. Prompt-tuning came into play to scale this (manual) prompt engineering technique. Prompt-tuning can be categorized into discrete and continuous approaches.

Unlike finetuning, where a separate model is required for each downstream task, in prompt-tuning, a single model serves multiple different downstream tasks. In discrete prompt-tuning, as human efforts are involved in crafting the prompts, the process becomes time-consuming and fallible as human efforts are involved in crafting the prompts. It sometimes can be non-intuitive for many tasks (e.g., textual entailment). Additionally, improper construction of contexts leads to low model performance. To overcome these issues, a continuous or tunable prompt-tuning technique was proposed.

## 5.2. Continuous Prompt-Tuning

In continuous prompt-tuning, additional *k* tunable tokens are used per downstream task, which are prepended to the input text. These prompts are learned through backpropagation and are tunable or adjustable to incorporate signals from any number of labeled examples. Unlike finetuning, only the parameters of these inserted prompt tokens get updated in prompt-tuning. Hence, they are also called soft prompts. Ref. [5] demonstrated how their approach outperformed GPT-3's few-shot learning based on discrete text prompts by a large margin. They also demonstrated that prompt-tuning becomes more competitive with scale, where it matches the performance of finetuned models. For example, prompt-tuning of T5 matched the model's finetuning quality as the size increased while enabling the reuse of a single frozen model for all the tasks.

P-tuning uses a small trainable model that encodes the text prompt and generates task-specific tokens. These tokens are then appended to the prompt and passed to the LLM during finetuning. When the tuning process is complete, these tokens are stored in a lookup table and used during inference, replacing the smaller model. In this approach, the time required to tune a smaller model is much less. Ref. [4] utilized a P-tuning technique to automatically search prompts in the continuous space, which enabled the GPT-style model to perform better on NLU tasks. Unlike the discrete-prompt approach, in continuous prompt, as there are trainable embedding tensors, the prompt encoder can be optimized in a differentiable way. P-tuning helped to augment the pretrained model's NLU ability by automatically searching for better prompts in the continuous space. As demonstrated in [4], the P-tuning method improves GPTs and BERTs in both few-shot and fully supervised settings.

Additionally, as only the parameters of prompt tokens are stored, which are less than 0.01% of the total model parameters, the prompt-tuning approach saves a significant amount of storage space. For example, CPM-2 [7] used only 100 prompt tokens, where only 409.6 K trainable parameters were to be updated compared to the 11B parameters of finetuning. As demonstrated in CPM-2, except for the Sogou-Log task, CPM-2 with prompt-tuning achieved comparable performance to the finetuning approach. In prompt-tuning, as the number of parameters to be optimized is much smaller, the size required for tensors (gradient and optimizer state) significantly decreased. As a result, prompt-tuning can save at most 50% GPU memory as compared to finetuning.

However, prompt engineering also has limitations, such as prompt-tuning taking many more steps to converge and hence more time. Additionally, only a small number of examples can be used, which limits the level of control. Also, as the examples are part of the prompt, it affects the token budget.

## 6. Multilingual Finetuning

Most language models are monolingual, using data in the English language only during pretraining. Such models, therefore, cannot be used to deal with tasks that are non-English-language-related. To overcome this issue, multilingual models were proposed to enable the processing of non-English languages. Such multilingual models can also be used for cross-lingual tasks like translation. However, models such as GPT-3 were potentially limited in dealing with cross-lingual tasks and generalization because most of these models had English-dominated training datasets.

XGLM [8] focused on using a multilingual dataset (comprising a diverse set of languages) for finetuning. As a result, XGLM achieved cross-lingual solid transfer, demonstrating SOTA few-shot learning performance on the FLORES-101 machine translation benchmark between many language pairs. When BloomZ [9] was finetuned with xP3, a multilingual task dataset of 46 languages, the model achieved better zero-shot task generalization (than the P3-trained baseline) on English and non-English tasks. Furthermore, when xP3mt, a machine-translated multilingual dataset of xP3, was used to finetune BloomZ on non-English prompts, the performance of held-out tasks with non-English human-written prompts significantly improved. In other words, as models generalize to tasks they had never intentionally seen, they learn the higher-level capabilities that are both task- and language-agnostic.

Typically, a cross-lingual dataset is used to make the model language-agnostic, and, to make it task-agnostic, a multitask dataset is required. Also, for large multilingual models, zero-shot performance tends to be significantly lower than finetuned performance. So, to improve the multilingual model's zero-shot task generalization, BloomZ [9] focused on cross-lingual and multitask finetuning. This enabled the model to be usable for low-resource language tasks without further finetuning.

## 7. Reinforcement Learning from Human Feedback (RLHF) Finetuning

Although the LMs can be prompted to generate responses to a range of NLP tasks, sometimes, these models might showcase unintended behavior by generating toxic responses or results that are not aligned with the user instructions. This happens because the objectives used to pretrain LLMs focus on predicting the next token, which might differ or misalign from human intention (user's query or instruction objective). To address this misalignment issue, Ref. [10] proposed reinforcement learning (RL) from human feedback to finetune GPT-3. In the RL-based approach, human labels are used to train a model of reward and then optimize that model. Using human feedback, it tries to align the model by the user's intention, which encompasses explicit and implicit (such as being truthful and not being toxic, harmful, or biased) intentions.

RLHF aims to make the model honest, helpful, and harmless. The RLHF approach uses human preferences as a reward signal to finetune the model. It was demonstrated how, despite having 100× fewer parameters, the outputs from the InstructGPT model with 1.3 B parameters were preferred over GPT-3 with 175 B parameters.

Using the RLHF approach, InstructGPT demonstrated improvement in toxicity and truthfulness over GPT-3 and generalized well to held-out instructions. Ref. [11] applied reinforcement learning (RL) to complex tasks defined only by human judgment, where only humans can tell whether a result is good or bad. In [11], the pretrained model was finetuned using reinforcement learning rather than supervised learning, where it demonstrated its results on summarizing and continuation tasks by applying reward learning to language generation. Ref. [12] recursively used the RL approach to produce novel summaries and achieve SOTA results for book-length summarizing on the BookSum dataset. Similarly, using the reinforcement learning technique, Ref. [13] trained a model to predict the human-preferred summary and used it as a reward function to finetune the summarizing policy. It could outperform

larger models finetuned using a supervised approach and human reference summaries and generalize well to new datasets.

## 8. Instruction Tuning

In instruction tuning, the model is finetuned on a collection of datasets where the NLP tasks are described using natural language instructions. Natural language instructions are added to the prompt to let the model know which task to perform for a given input. For instance, to ask the model to perform a sentiment analysis task on a given input, instructions such as ‘Classify this review either as negative, positive, or neutral’ can be provided in the prompt. Various factors determine the effectiveness of instruction tuning on LLMs, such as the prompt format used, objectives used during finetuning, diversity of tuning tasks, distribution of datasets, etc. Additionally, the zero-shot task generalization of LLMs performs poorly across tasks. To address this, multitask finetuning (MTF) has emerged and become one of the promising techniques to improve the performance of LLMs in zero-shot settings.

Creating instruction datasets for many tasks from scratch is a resource-intensive process. Instead, FLAN [14] expresses existing 62 NLP datasets in the instructional format. This transformed dataset with instructions is then used to finetune the model. For each dataset, 10 unique templates were created to describe the task in instructional format for that dataset. Based on the task type, the datasets were grouped into clusters, and then, to evaluate the performance on each task, the specific task cluster was held out while the remaining clusters were used during instruction tuning.

FLAN demonstrated how instruction tuning substantially improved the zero-shot performance on held-out tasks that were not part of the instruction tuning process and also helped the model generalize well on unseen tasks. FLAN outperformed GPT-3 (zero- and few-shot) on 20 of the 25 datasets used for evaluation. It was observed that the instruction tuning approach is more effective for tasks such as QA, NLI, and translation that can easily be verbalized as instructions. Instruction tuning is less effective for tasks where the instructions are redundant since they can be formulated simply as language modeling tasks, such as commonsense reasoning. FLAN also demonstrated how instruction tuning can hurt smaller models since their capacity is mostly exhausted in learning different instruction tasks.

Alpaca uses Meta’s LLaMA model and finetunes it with 52 K instructions following demonstrations in a supervised manner. These instructions were generated using GPT3.5 (text-davinci-003), where 175 human-written instruction–output pairs from the self-instruct were used as a seed to generate more instructions. Tk-INSTRUCT [15] proposed a benchmark with instructions for 1616 nlp tasks, so such a benchmark dataset can be beneficial in studying multitask learning and cross-task generalization. This dataset, called ‘SUPER-NATURAL-INSTRUCTIONS (SUP-NATINST)’, is publicly available. It covers instructions in 55 different languages, and the 1616 nlp tasks can be categorized under 76 broad task types. For each task, it provides instructions comprising several examples with the desired output along with the definition that maps input text to task output. When evaluated on 119 unseen tasks (English and multilingual variants), TK-INSTRUCT outperformed InstructGPT by 9.9 ROUGE-L points, and mTK-INSTRUCT outperformed InstructGPT by 13.3 points on 35 non-English tasks.

OPT-IML [16], instruction-tuned-on OPT, conducted experiments by scaling the model size and benchmark datasets to see the effect of instruction tuning on performance. It also proposed a benchmark called 'OPT-IML Bench', consisting of 2000 NLP tasks. This benchmark can be used to measure three types of generalizations to tasks from held-out categories, held-out tasks from seen categories, and held-out instances from seen tasks. OPT-IML achieved all these generalization abilities at different scales and benchmarks (PromptSource, FLAN, Super-NaturalInstructions, and UnifiedSKG), having diverse tasks and input formats. OPT-IML was also highly competitive with finetuned models on each specific benchmark. Furthermore, to improve the performance on reasoning tasks, it used 14 reasoning datasets during instruction tuning, where the output included a rationale (chain-of-thought process) before the answer. Similarly, there was experimentation by adding dialogues as auxiliary datasets to see if that could induce chatbot behavior in the model.

Ref. [17] experimented with instruction tuning regarding model size, number of tasks, and chain-of-thought datasets. It was observed that instruction finetuning scales well, and the model performance substantially improved with the increased size of models and number of finetuning tasks. Additionally, when nine CoT datasets were added to the instruction tuning dataset mixture, the model could perform better on evaluation reasoning tasks. This contradicts other work where instruction finetuning instead degraded CoT task performance. So, Ref. [17] demonstrated how CoT data improves performance reasoning tasks when jointly finetuned with an instruction dataset. After instruction tuning model classes such as T5, PaLM, and U-PaLM, Ref. [17] observed a significant boost in performance for different types of prompting setups (zero, few, and CoT) and benchmarks as compared to the original models (without instruction finetuning).

In self-instruct [18], the bootstrap technique is used to improve the model's instruction following capabilities. Here, the existing collection of instructions is leveraged to generate new and more broad-coverage instructions. Using a language model, self-instruct generates instructions along with input–output samples, filters invalid, low-quality, or repeated instructions, and uses the remaining valid ones to finetune the original model. Along with the instructions, the framework also creates input–output instances, which can be used to supervise the finetuning of instructions. When self-instruct was applied to GPT-3, it achieved a 33% performance gain on SUPER-NATURALINSTRUCTIONS over the original model, which was on par with the InstructGPT performance.

## 9. Code-Based Finetuning

Generating code is a translation task that maps a natural language problem statement to a solution or code in programming language. Recent LLMs are capable of completing programming tasks by generating code. Codex [19] uses the GPT model, which was finetuned on publicly available code from GitHub. It studied Python code-writing capabilities, focused on generating standalone Python functions from docstrings, and then evaluated the correctness of the generated code samples. It was able to solve 28.8% of the HumanEval dataset problems, while GPT-3 solved 0% and GPT-J solved 11.4%. It needs help with docstrings describing long operations chains and binding operations to variables.

To enable the model to solve complex problems and provide deeper reasoning, the AlphaCode [20] model was pretrained on a collection of open-source code from GitHub and then finetuned on a curated set called CodeContests of competitive programming problems. The pretraining dataset consisted of code from several popular programming languages. AlphaCode achieved a ranking of top 54.3% on average in simulated programming competitions with more than 5000 participants that were hosted on the Codeforces platform.

Furthermore, CodeGEN [21] introduced a multistep approach where a user can progressively communicate with the system to provide specifications. Such multiple-step specification eases the understanding of a model, leading to enhanced program synthesis. CodeGeeX [22] is a multilingual model trained on 23 programming languages. To evaluate multilingual models, it proposed a HumanEval-X benchmark where the solutions in C++, Java, JavaScript, and Go were hand-written. CodeGeeX was able to outperform multilingual code models of similar scale for translation on HumanEval-X as well as code generation tasks.

## References


1. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-efficient transfer learning for NLP. In Proceedings of the International Conference on Machine Learning, Vancouver, BC, Canada, 13 December 2019; pp. 2790–2799.
2. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. 2020, 21, 5485–5551.
3. Li, X.L.; Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. arXiv 2021, arXiv:2101.00190.
4. Liu, X.; Zheng, Y.; Du, Z.; Ding, M.; Qian, Y.; Yang, Z.; Tang, J. GPT understands, too. AI Open 2023, in press.
5. Lester, B.; Al-Rfou, R.; Constant, N. The power of scale for parameter-efficient prompt tuning. arXiv 2021, arXiv:2104.08691.
6. Brown., T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. Adv. Neural Inf. Process. Syst. 2020, 33, 1877–1901.
7. Zhang, Z.; Gu, Y.; Han, X.; Chen, S.; Xiao, C.; Sun, Z.; Yao, Y.; Qi, F.; Guan, J.; Ke, P.; et al. Cpm-2: Large-scale cost-effective pre-trained language models. AI Open 2021, 2, 216–224.
8. Lin, X.V.; Mihaylov, T.; Artetxe, M.; Wang, T.; Chen, S.; Simig, D.; Ott, M.; Goyal, N.; Bhosale, S.; Du, J.; et al. Few-shot Learning with Multilingual Generative Language Models. In Proceedings of

the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates, 7–11 December 2022; pp. 9019–9052.

9. Muennighoff, N.; Wang, T.; Sutawika, L.; Roberts, A.; Biderman, S.; Scao, T.L.; Bari, M.S.; Shen, S.; Yong, Z.X.; Schoelkopf, H.; et al. Crosslingual generalization through multitask finetuning. arXiv 2022, arXiv:2211.01786.
10. Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. Training language models to follow instructions with human feedback. Adv. Neural Inf. Process. Syst. 2022, 35, 27730–27744.
11. Ziegler, D.M.; Stiennon, N.; Wu, J.; Brown, T.B.; Radford, A.; Amodei, D.; Christiano, P.; Irving, G. Fine-tuning language models from human preferences. arXiv 2019, arXiv:1909.08593.
12. Wu, J.; Ouyang, L.; Ziegler, D.M.; Stiennon, N.; Lowe, R.; Leike, J.; Christiano, P. Recursively summarizing books with human feedback. arXiv 2021, arXiv:2109.10862.
13. Stiennon, N.; Ouyang, L.; Wu, J.; Ziegler, D.; Lowe, R.; Voss, C.; Radford, A.; Amodei, D.; Christiano, P.F. Learning to summarize with human feedback. Adv. Neural Inf. Process. Syst. 2020, 33, 3008–3021.
14. Wei, J.; Bosma, M.; Zhao, V.Y.; Guu, K.; Yu, A.W.; Lester, B.; Du, N.; Dai, A.M.; Le, Q.V. Finetuned language models are zero-shot learners. arXiv 2021, arXiv:2109.01652.
15. Wang, Y.; Mishra, S.; Alipoormolabashi, P.; Kordi, Y.; Mirzaei, A.; Arunkumar, A.; Ashok, A.; Dhanasekaran, A.S.; Naik, A.; Stap, D.; et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. arXiv 2022, arXiv:2204.07705.
16. Iyer, S.; Lin, X.V.; Pasunuru, R.; Mihaylov, T.; Simig, D.; Yu, P.; Shuster, K.; Wang, T.; Liu, Q.; Koura, P.S.; et al. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. arXiv 2022, arXiv:2212.12017.
17. Chung, H.W.; Hou, L.; Longpre, S.; Zoph, B.; Tay, Y.; Fedus, W.; Li, E.; Wang, X.; Dehghani, M.; Brahma, S.; et al. Scaling instruction-finetuned language models. arXiv 2022, arXiv:2210.11416.
18. Wang, Y.; Kordi, Y.; Mishra, S.; Liu, A.; Smith, N.A.; Khashabi, D.; Hajishirzi, H. Self-instruct: Aligning language model with self generated instructions. arXiv 2022, arXiv:2212.10560.
19. Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H.P.D.O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. Evaluating large language models trained on code. arXiv 2021, arXiv:2107.03374.
20. Li, Y.; Choi, D.; Chung, J.; Kushman, N.; Schrittwieser, J.; Leblond, R.; Eccles, T.; Keeling, J.; Gimeno, F.; Dal Lago, A.; et al. Competition-level code generation with alphacode. Science 2022, 378, 1092–1097.

21. Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. arXiv 2022, arXiv:2203.13474.
22. Zheng, Q.; Xia, X.; Zou, X.; Dong, Y.; Wang, S.; Xue, Y.; Wang, Z.; Shen, L.; Wang, A.; Li, Y.; et al. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. arXiv 2023, arXiv:2303.17568.