CPM-2: Large-scale Cost-efficient Pre-trained Language Models

Zhengyan Zhang*, Yuxian Gu*, Xu Han*, Shengqi Chen*, Chaojun Xiao*, Zhenbo Sun
Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, Yanzheng Cai, Guoyang Zeng, Zhixing Tan,
Zhiyuan Liu†, Minlie Huang†, Wentao Han†, Yang Liu, Xiaoyan Zhu, Maosong Sun
Department of Computer Science and Technology, Tsinghua University & BAAI

Abstract

In recent years, the size of pre-trained language models (PLMs) has grown by leaps and bounds. However, the efficiency issue of these large-scale PLMs limits their utilization in real-world scenarios. In this work, we improve the efficiency of large-scale PLMs in a full pipeline, including pre-training, fine-tuning, and inference: (1) We introduce knowledge inheritance to accelerate the pre-training process based on existing PLMs instead of training models from scratch; (2) We explore the best practice of prompt tuning with large-scale PLMs. Compared with full-model fine-tuning, prompt tuning significantly reduces the number of task-specific parameters; (3) We implement a new inference toolkit, namely INFMOE, for large-scale PLMs with limited computational resources. Based on our optimized pipeline for PLMs, we pre-train two models: an encoder-decoder language model with 11 billion parameters (CPM-2) pre-trained on both Chinese and English corpora, and its corresponding MoE version with 198 billion parameters. We compare CPM-2 with mT5-XXL on downstream tasks and the results show that CPM-2 has excellent general language intelligence. Moreover, we validate the efficiency of INFMOE when conducting inference of MoE models with tens of billions of parameters on single GPU. All source code and model parameters are available at https://github.com/TsinghuaAI/CPM-2.

1 Introduction

Training much larger models is an important research direction in deep learning (Bengio, 2013). Recently, pre-training has become the mainstream technique to develop large-scale neural networks and achieved great success in both computer vision (CV) and natural language processing (NLP) (He et al., 2016; Dosovitskiy et al., 2020; Devlin et al., 2019; Raffel et al., 2020). Recently, some much larger pre-trained language models (PLMs) with hundreds of billions of parameters have been proposed, such as GPT-3 (Brown et al., 2020), PANGU-α (Zeng et al., 2021), and Switch-Transformer (Fedus et al., 2021).

However, the cost of PLMs is increasing rapidly with the growth of model sizes and becomes unaffordable for most users and researchers. The cost consists of three parts: (1) **Large computation cost** for pre-training: a super large model requires pre-training with thousands of GPUs for several weeks. (2) **Large storage cost** for fine-tuned models: a super large model usually takes hundreds of gigabytes (GBs) to store and we need to store as many models as downstream tasks. (3) **Strict equipment requirement**: it is a common practice to apply model parallelism to a super large model, which uses multiple GPUs to compute one model, so these models are hard to be run with limited computation resources.

To reduce the cost of large-scale PLMs, from its pre-training to fine-tuning, we try to improve the full pipeline of developing PLMs as follows:

(1) We adopt knowledge inheritance to accelerate the pre-training process. Current PLMs are usually trained from scratch on pre-training data via self-supervised methods, while there are many existing PLMs that can also provide much knowledge. Knowledge inheritance aims to use the knowledge of existing PLMs to help the pre-training of new large-scale models.

(2) We use prompt tuning instead of full-model fine-tuning to reduce the storage of task-specific parameters. With prompt tuning, we only need to save the parameters of prompt tokens, which is usually less than 0.01% of the whole model parameters.

(3) We design a high-performance and memory-efficient inference framework INFMOE with a

* Equal contribution
† Corresponding authors: Z. Liu (liuzy@tsinghua.edu.cn), M. Huang (aihuang@tsinghua.edu.cn), W. Han (hanwentao@tsinghua.edu.cn)

dynamically-scheduled offloading strategy, to support the inference of MoE models on single GPU.

Based on our optimized pipeline for PLMs, we develop two large-scale **C**ost-efficient **P**re-trained language **M**odels (CPM-2), an Chinese-English bilingual models with 11 billion parameters and its Mixture of Experts (MoE) version with 198 billion parameters. Specifically, we accelerate the pre-training process by dividing the pre-training process to three stages with knowledge inheritance: Chinese pre-training, bilingual pre-training, and MoE pre-training. Then, we compare CPM-2 with mT5-XXL (Xue et al., 2020). The experimental results show that CPM-2 has excellent general language intelligence including seven specific capabilities. Finally, we introduce INFMOE for users to conduct inference of MoE models with tens of billions of parameters on single GPU.

2 Pre-Training

In this section, we present the details of CPM-2.

2.1 Model

CPM-2 is a standard Transformer-based model combined with a bidirectional encoder and a unidirectional decoder (Vaswani et al., 2017). The comparison between our model and CPM (Zhang et al., 2020) is presented in Table 1. To store the model on GPUs, we use the model parallelism which splits Transformer's self-attention layers and feed-forward layers along the width dimension and finally distributes the partitions of one model on 4 GPUs.

To reduce the memory requirement and accelerate pre-training, we employed mixed-precision training (Micikevicius et al., 2018), gradient checkpointing (Chen et al., 2016) and ZERO-stage-1 (Rajbhandari et al., 2020) optimization implemented by DeepSpeed (Rasley et al., 2020).

For CPM-2-MoE, we expand the feed-forward layer of each Transformer block to multiple experts. During each forward pass, for each token, we select one expert according to its previous hidden state and a gate function. We balance the expert selection using the same planning approach as the base layers in Lewis et al. (2021).

2.2 Data

In order to construct a better Chinese BPE vocabulary, we improve the vocabulary of CPM with the following methods.

We find that the original sentencepiece (Kudo and Richardson, 2018) tokenizer will insert many redundant white space token "_" to the tokenized sequences. This makes the sequences become much longer. The implementation of sentencepiece has a weak encapsulation of interfaces, and it is thus unfriendly towards programmers. Inspired by WoBERT (Su, 2020), we replace sentencepiece tokenizer with a tokenizer combined with Jieba word segmentation and remove the white space insertion. Compared with sentencepiece, our newly-implemented tokenizer is more effective and easier to use.

Since in the writing system of Chinese, it is not important whether a token in the vocabulary appears at the beginning of a word or not, we combine the tokens like "快乐" (happy) and "_快乐" (_happy) to a single token "快乐" to simplify the vocabulary.

We pre-train our model on WuDaoCorpus which contains 2.3TB cleaned Chinese data as well as additional 300GB cleaned English data. Data in both languages are collected from multiple domains including encyclopedia, novels, Q&A, scientific literature, e-book, news and reviews.

2.3 Pre-Training with Knowledge Inheritance

The pre-training process of CPM-2 can be devided into three stages: Chinese pre-training, bilingual pre-training, and MoE pre-training. Compared to training models from scratch, multi-stage training with knowledge inheritance can significantly reduce the computation cost.

Chinese Stage. In this stage, we only use Chinese corpora as the training data. We suppose the model can focus on learning Chinese information and have a good basis to generalize to other languages.

Bilingual Stage. In this stage, we further pre-train the model from Chinese stage on both Chinese corpora and English corpora. There are two main challenges, how to initialize the input embeddings of English tokens and how to prevent the model from catastrophic forgetting. (1) To make the English tokens more familiar to the model, we use the embeddings of their prefixes to initialize their embeddings. If all prefixes of a English token are not in the original vocabulary, we randomly select a existing token embedding for initialization. (2) We carefully design the ratio between English data

	n_{param}	L	n_{head}	d_{head}	d_{ff}	d_{model}	Encoder	Decoder	MoE
CPM-Small	109M	12	12	64	3,072	768	✗	✓	✗
CPM-Medium	334M	24	16	64	4,096	1,024	✗	✓	✗
CPM-Large	2.6B	32	32	80	10,240	2,560	✗	✓	✗
CPM-2	11B	24	64	64	10,240	4,096	✓	✓	✗
CPM-2-MoE	198B	24	64	64	10,240	4,096	✓	✓	✓

Table 1: Comparison between CPM and CPM-2. n_{param} is the amount of model parameters. L is the number of model layers. n_{head} is the number of attention heads. d_{head} is the dimension of each attention head. d_{ff} is the dimension of the hidden layer in the feed-forward layer. d_{model} is the dimension of the model hidden states.

and Chinese data and find 1:2 can well maintain the language knowledge of Chinese and capture new knowledge of English.

MoE Stage. In this stage, we duplicate the model from the bilingual stage several times to initialize a MoE model. For the gating network, we adopt a random projection as a local sensitive hashing function (Har-Peled et al., 2012) and will not update the gating network in this stage. We suppose that the representation space of the model of the second stage is well organized and similar tokens should use the same expert.

3 Experiment Setups

To validate the effectiveness of our model, we evaluate CPM-2 on a general language intelligence benchmark, CUGE (Yao et al., 2021). CUGE consists of 40 mainstream Chinese NLP datasets and each dataset is categorized into one of the important types of language capabilities. Due to the limitation of computation, we select a representative dataset for each language capability to speed up the experiments. We describe each language capability and dataset as follows. The detailed statistics of these datasets are shown in Table 2.

Recall Capability. Recall capability aims to evaluate the models' ability to memorize and apply the general literature knowledge, such as the famous quotes, classical poems, and idioms. We adopt CCPR to test the models' recall ability. Given a modern Chinese translation of a classic poem, the model is required to select the corresponding poem from four candidates.

Comprehension Capability. Comprehension capability aims to evaluate the models' ability to understand the given text and perform reasoning for specific tasks. For this capability, we select the C^3 dataset (Sun et al., 2020) to evaluate our model. C^3 is a free-form multiple-choice reading comprehension dataset, which requires the model to understand the given documents or dialogues and answer several related questions.

Classification Capability. Text classification is a classic task in natural language processing. We evaluate the classification capability with a large scale natural language inference dataset, LCQMC (Liu et al., 2018). Given two questions, LCQMC requires the model to answer whether the two questions express similar intent.

Calculation Capability. Calculation capability aims to test the models' ability to perform numerical reasoning. For this capability, we select Math23K (Wang et al., 2017) as the benchmark. Math23K consists of tens of thousands of real math word problems for elementary school students.

Cross-lingual Capability. Cross-lingual capability aims to evaluate the models' performance in understanding multi-lingual text. We adopt the machine translation task to evaluate the ability of CPM-2 in understanding English and Chinese sentences. The dataset we used in this task are provided by WMT20.

Generation Capability. Text generation is one of the important tasks in natural language processing, which aims to generate fluent and diverse text. We adopt the AdGen (Shao et al., 2019) as our benchmark, which requires the model to generate long advertising text given the several keywords.

Summarization Capability. Summarization requires the model to read a long document, and produce a concise summary while keeping the key information. We utilize LCSTS (Hu et al., 2015) to evaluate the summarization capability. LCSTS consists of tweets and their corresponding abstracts from the Chinese largest microblogging website (Sina Weibo).

We compare our model with mT5-XXL (Xue et al., 2020) on the downstream tasks mentioned

	CCPR	C^3	LCQMC	WMT20-enzh	Math23K	AdGen	LCSTS
Train	21k	8k	238k	21,000k	21k	114k	2,400k
Valid	2.7k	2.7k	8.8k	2k	1k	1k	8.6k
Test	2.7k	2.7k	12.5k	2k	1k	3k	0.7k

Table 2: Statistics of the datasets.

above. Notably, mT5-XXL also adopts an encoder-decoder architecture with 11 billion parameters, which is comparable to CPM-2. To the best of our knowledge, Pangu-α (Zeng et al., 2021) with 200 billion parameters is the largest Chinese pre-trained language model, which performs well in many downstream tasks. However, the parameters of Pangu-α are not publicly available, and thus we leave the comparison between CPM-2 and Pangu-α for future work.

4 Fine-Tuning

In this section, we fine-tune CPM-2 and mT5 on downstream tasks to evaluate their general language intelligence.

4.1 Experimental Setups

We adjust maximum lengths, batch sizes, learning rates for different models and datasets. Considering that the tokenizers of CPM-2 and mT5 are different, we first tokenize the whole dataset and then set the maximum length of samples as the maximum length instead of a pre-defined length. For the batch size, we search from 128 to 512 to ensure the number of input tokens is around 2^{16} following Raffel et al. (2020). For learning rates, we search from 1e-6 to 5e-6 and find that larger learning rates will make the training unstable.

4.2 Results

The results of fine-tuning are shown in Table 3. We observe that CPM-2 is better than mT5 in most language capabilities including Chinese language understanding tasks, generation tasks and English to Chinese Translation tasks. Especially, CPM-2 outperforms mT5 by over 10% in Math23K, which is for calculation capability. On average, CPM-2 outperforms mT5 by 2.22%. This demonstrates that CPM-2 is an omnipotent large-scale multi-lingual PLM.

5 Prompt Tuning

In this section, we study **prompt tuning** based on the proposed CPM-2. To do prompt tuning, we insert several random initialized prompt tokens into the tokenized input of the encoder. When fine-tuning the model, we only tune the inserted prompt tokens and fix the rest of parameters. To make it clear, we refer to the conventional full-parameter fine-tuning as **full-model tuning**. Throughout our experiments, we keep the number of prompt tokens as 100 to control the number of trainable parameters. Thus in the prompt tuning setting, the amount of the parameters needed to tune is only 409.6K. Compared to the 11B parameters of full-model tuning, prompt tuning only needs to modify 0.0037% parameters. We present the main results of prompt tuning in Section 5.1. We also explore how the positions of inserted prompt tokens effect the model performance (Section 5.2), how the prompt tokens work (Section 5.3) and propose a two-stage fine-tuning strategy to improve model performance on downstream tasks (Section 5.4).

5.1 Main Results

We present the model performance and GPU memory usage of prompt tuning in Table 4. From the results, we can observe that with prompt tuning, CPM-2 can achieve a comparable performance as full-model tuning, at most 2% behind, while prompt tuning is much more memory-efficient. The results of the GPU memory usage show that prompt tuning can save at most 50% GPU memory compared with full-model tuning. This is because when the model is trained with the Adam optimizer, gradients and optimizer states account for a large proportion of the overall GPU memory. Since the number of parameters needed to be optimized is much small in prompt tuning, the total sizes of gradient tensors and optimizer state tensors decrease. Note that little sizes of gradient tensors and optimizer state tensors also lead to small communication overhead during synchronization of distributed training. This makes the optimization of a single step in prompt tuning faster than full-model tuning. However, we also observe in our experiments that it takes much more steps for prompt tuning to converge than full-model tuning, which makes

	CCPR	C^3	LCQMC	WMT20-enzh	Math23K	AdGen	LCSTS	Avg.
	Acc	Acc	Acc	BLEU	Acc	BLEU/Distinct	Rouge-L	
mT5	90.62	**86.36**	88.32	23.98	59.27	9.82/68.67	34.80	60.37
CPM-2	**91.63**	86.05	**89.16**	**26.21**	**69.37**	**10.60/70.22**	**35.88**	**62.59**

Table 3: Performance of mT5-XXL and CPM-2 with fine-tuning.

the whole time of prompt tuning longer. We leave the question "How to accelerate the convergence of prompt tuning?" to future work.

5.2 Position of Prompt

We study the effect of the positions where we insert the prompt tokens into the input sequence. For single-sentence tasks, like Math23k, there exist 3 positions to insert the prompt: front, back, and front + back. For multi-sentence tasks, like LCQMC, prompt tokens can also be inserted between the input sentences (middle). Taking two-sentence input as an example, there are 7 ways to insert the prompt tokens. The illustration of all possible prompt insertions of the two-sentence input task is shown in Table 5.

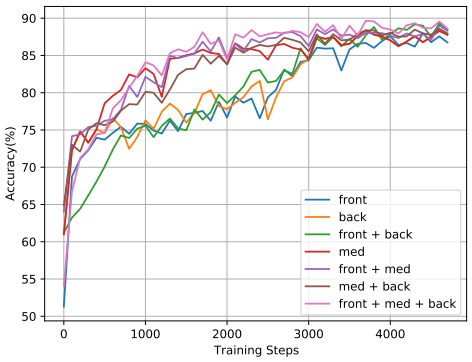
We conduct experiments on the positions of prompt tokens on Math23k and LCQMC, respectively. We keep the number of prompt tokens as 100 throughout the experiments. When there are 2 positions to insert tokens, we insert 50 tokens at each position. When there are 3 positions, we insert 33, 34, 33 tokens at each position. The results are shown in Table 6.

From Table 6, we can see that for single sentence tasks (Math23k), the positions of the prompt tokens have no significant influence on the model performance. But for multi-sentence tasks (LCQMC), whether to insert the prompt between sentences significantly matters. Compared with inserting prompts between the 2 input sentences, only considering the front and the back positions drops about 2% accuracy.

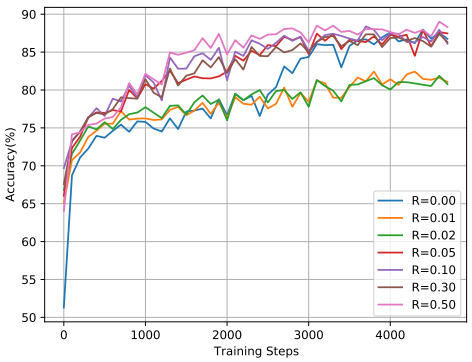
We also plot the accuracy curves on LCQMC dev set of different prompt positions. Furthermore, we also take F+M as an example and change the proportion of the number of prompt tokens at different positions. The results are shown in Figure 1. In Figure 1(b), R denotes the ratio of the prompt token number between the two sentences and the total number of prompt tokens.

From the figure we can conclude that:

(1) Figure 1(a) shows, for "Front", "Back" and "Front + Back" settings, the convergence is much slower than the settings with prompt tokens inserted between the two input sentences, which means it is necessary to insert the prompt between sentences to improve convergence speed.



(a) Accuracy curve of different prompt positions.



(b) Accuracy curve of different ratio of the prompt token inserted between the two sentences.

Figure 1: Accuracy curves on the LCQMC dev set with different prompt insertion strategies.

(2) As Figure 1(b) shows, when $R = 0.00$ (front), $R = 0.01$ and $R = 0.02$ (insert 1 or 2 tokens between sentences), the model converges slowly.But when we insert 5 or more tokens between the two sentences, the convergence speed is significantly improved. This means only a few middle-inserted tokens can help the model converge and when we add more tokens afterward, the impact of the token number is much less.

We think that the influence of the prompt token

	CCPR	C^3	LCQMC	WMT20-enzh	Math23K	AdGen	LCSTS
Performance on test set							
	Acc	Acc	Acc	BLEU	Acc	BLEU/Distinct	Rouge-L
CPM-2-F	91.63	86.05	89.16	26.21	69.37	10.60/70.22	35.88
CPM-2-P	90.85	85.33	88.36	24.13	67.48	8.63/72.02	34.17
Δ(P − F)	-0.78	-0.72	-0.80	-2.08	-1.89	-1.97/+1.80	-1.71
GPU memory usage(%)							
CPM-2-F	98	96	98	98	93	98	98
CPM-2-P	50	46	54	75	68	53	76
Δ(P − F)	-48	-50	-44	-23	-25	-45	-22

Table 4: Comparisons between fine-tuning and prompt tuning. CPM-2-F represents fine-tuning. CPM-2-P represents prompt tuning. Δ(P − F) means the difference between fine-tuning and prompt tuning.

Position	Prompt Inserted Sequence
F	$P_1 \| \| \text{sentence}_1 \| \| \text{sentence}_2$
B	$\text{sentence}_1 \| \| \text{sentence}_2 \| \| P_1$
M	$\text{sentence}_1 \| \| P_1 \| \| \text{sentence}_2$
F+B	$P_1 \| \| \text{sentence}_1 \| \| \text{sentence}_2 \| \| P_2$
F+M	$P_1 \| \| \text{sentence}_1 \| \| P_2 \| \| \text{sentence}_2$
M+B	$\text{sentence}_1 \| \| P_1 \| \| \text{sentence}_2 \| \| P_2$
F+M+B	$P_1 \| \| \text{sentence}_1 \| \| P_2 \| \| \text{sentence}_2 \| \| P_3$

Table 5: Different designs to insert prompts for the task with two input sentences. P_1, P_2, P_3 represent different input prompts. F, B, M represent Front, Back, and Middle. For different dataset, the optimal ways to concatenate prompts with sentences are different. We denote the concatenation by $\cdot||\cdot$ here.

	Math23k	LCQMC
F	71.74	88.38
B	72.40	88.50
F+B	72.66	88.48
M	-	89.20
F+M	-	90.21
M+B	-	90.38
F+M+B	-	90.65

Table 6: Effects of prompt positions on Math23k and LCQMC. For both datasets, we report the accuracy on dev set.

positions is related to the relative position embedding we use in CPM-2. When there are multiple input sentences, CPM-2 needs to model the tokens with a long distance. For relative position embedding, the long-range tokens will be assigned the same position embeddings, which may harm long-distance modeling. The prompt tokens inserted between the sentences can bridge the gap between long-range tokens, which makes it easier for the model to learn the relationships between two input sentences.

5.3 How Prompt Works

Although prompt tuning can reach comparable performance with full-model tuning by only modifying a small number of parameters, how the prompt works is still unclear. We assume that the prompt can play two kinds of roles in model tuning:

1. Working as a “Provider”. Provide an additional context for the model input.

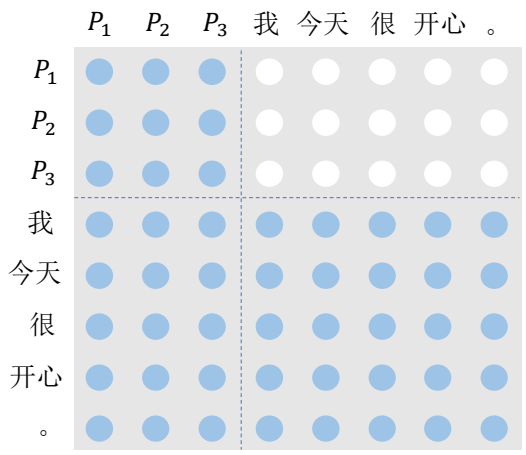
2. Working as a “Gatherer”. Gather the information from the input text.

To verify our hypothesis, we use attention masks to control the attentions between the prompt tokens and the text tokens. Specifically, for “Provider”, we mask the attention of the prompt to text tokens such that the representations of prompt tokens can not be computed by attending to text tokens, disabling their ability to gather information. But they can still work as contexts and provide information to text tokens. For “Gatherer”, on the contrary, we mask the attentions from text tokens to prompt tokens. In this way, prompt tokens can not work as contexts but can gather information by attending to text tokens. The illustration of our attention mask is shown in Figure 2.

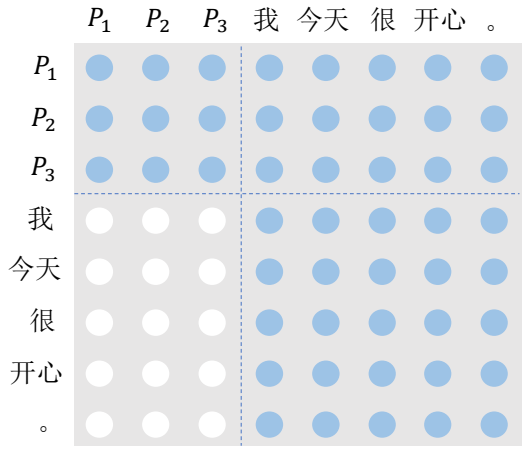
We add the above mentioned attention masks to the model when doing prompt tuning. We conduct experiments on Math23k and LCQMC datasets. The results are shown in Table 7.

From Table 7 we can conclude that:

1. Both attention masks hurt the model performance on the two datasets. This means that prompt should work as "Provider" and "Gatherer" at the same time for the model to reach a high performance.



(a) The attention mask for "Provider". Attentions from prompt to text tokens are masked.



(b) The attention mask for "Gatherer". Attention from text tokens to prompt are masked.

Figure 2: Attention masks for "Provider" and "Gatherer". P_1, P_2, P_3 are prompt tokens.

2. On both two datasets, the impact of masking text to prompt is larger than that of masking prompt to text. This means prompt tokens are more likely to work as "Provider" than as "Gatherer" in prompt tuning.

5.4 Two Stage Fine-tuning

Inspired by (Schick and Schütze, 2020a) and (Schick and Schütze, 2020b), the good prompt can help stimulate the model's ability in full-model tuning. However, early works usually manually design prompt or search prompt in a discrete space, which is hard to find the suitable prompt for model fine-tuning. As a result, we attempt to search for good prompts in a continuous space that can benefit full-model tuning afterward. Specifically, we

	Math23K	LCQMC
Full Attention	71.35	90.88
Mask Prompt to Text	69.92	81.50
Mask Text to Prompt	35.29	79.45

Table 7: Results of masking the attentions between prompts and texts.

	C^3	Math23k	LCQMC
CPM-2-F	85.66	73.85	**90.88**
CPM-2-P	85.75	71.74	90.21
CPM-2-P+F	**86.77**	75.26	90.45
+fix prompt	86.27	**76.17**	89.64
-stage 1	85.04	72.40	88.76

Table 8: Results of two-stage fine-tuning on three tasks using the dev sets. CPM-2-F stands for full-model tuning, CPM-2-P stands for prompt tuning. CPM-2-P+F is our two-stage fine-tuning. "+fix prompt" means we fix the parameters of the prompt we have found in stage 1 when we do full-model tuning in stage 2. "-stage 1" means we randomly initialize the prompt tokens and do full-model tuning directly without stage 1.

propose two-stage fine-tuning. In the first stage, we insert the prompt into the model input and perform prompt tuning with other model parameters fixed. In this stage, the model can search for a prompt suitable for the downstream task. Then, in the second stage, we unfix the model parameters and fine-tune the whole model together with the prompt token embeddings. We hope that the model can take the advantage of the prompt that we have found in the first stage and have a better performance than the vanilla full-model tuning. We do experiments on C^3, Math23k, and LCQMC dataset. We try several checkpoints in the process of stage 1 (prompt tuning) and select the one with the best results after stage 2 (full-model tuning). For each dataset, we use the same hyper-parameters as in Section 4 and 5.1. Our results on dev set are shown in Table 8.

From Table 8, we can see that on C^3 and Math23k dataset, two-stage fine-tuning can significantly improve the model performance by 2.16% and 1.41%, respectively. On the LCQMC dataset, two-stage fine-tuning has a similar performance as vanilla full-model tuning. We think this is because the LCQMC dataset is relatively easier than the other two datasets and vanilla fine-tuning can perform well enough without a better prompt. If we fix the prompt parameters during the stage 2,

the model performance does not change much. We think this is because as fine-tuning goes, the gradients become small when backward to the input prompt. Therefore, the prompt tokens do not change much even when they are not fixed. We can also conclude that without stage 1, even if we add additional parameters, the model can not reach a good performance which proves the necessity of our two-stage fine-tuning.

6 INFMOE: Memory-Efficient Inference Framework for MoE Layers

Although MoE linear layers could outperform dense linear layers with almost the same computational cost (Fedus et al., 2021), they greatly enlarge the number of model parameters and require more memory to store these parameters. When increasing the number of experts, the parameter size of the model can easily reach the order of tens or even hundreds of GBs. Such storage requirements greatly exceed the capacity of commodity GPUs, bringing difficulty not only to model training but also to model inference.

To make well-trained MoE layers more accessible to downstream tasks (e.g., to researchers using the aforementioned prompt tuning for downstream tasks), we introduce INFMOE[1], a high-performance and memory-efficient inference framework that can offload parameters of experts of MoE layers to CPU memory.

INFMOE enables the inference of MoE layers with hundreds of billions of parameters using one single GPU. To preserve the efficiency of computation, we design a dynamic scheduling strategy that can overlap data movement of parameters with inference computation to the greatest extent.

6.1 Existing Inference Frameworks

PyTorch and TensorFlow are widely-used deep learning frameworks in industry and academia for both training and inference. There are also many other frameworks like TensorRT and ONNX Runtime that are specially designed for efficient model inference on different devices. However, they are currently not fit for the efficient inference of MoE layers for various reasons.

One category of these frameworks, like TensorFlow Serving, uses static computational graphs for training and inference. Typically, graphs can only be moved between CPUs and GPUs as a whole, so it is difficult to offload selected parameters in the inference process. Also currently, no existing static-graph-based framework can provide full support for all required operators of MoE layers.

Another category, including PyTorch, uses dynamic computational graphs and provides simple interfaces to control data storage location (such as `layer.cuda()` and `layer.cpu()`. However, these frameworks usually take full control of the scheduling of computation and data movement. When handling MoE layers, they do not provide enough flexibility to implement the aforementioned overlapping mechanism. FastMoE (He et al., 2021) is a novel high-performance MoE implementation on top of PyTorch. Yet it focuses on the large-scale distributed training and also lacks delicate control on scheduling.

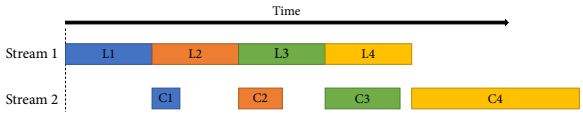
TensorRT is a high-performance (yet relatively low-level) inference SDK developed by NVIDIA. It employs several optimization techniques like tensor fusion, kernel auto-tuning, and memory reusing. Our toolkit INFMOE is developed based on TensorRT. The reason why we choose TensorRT is that it supports custom plugins. Therefore, we can implement our own plugin only for MoE layers with a specially designed scheduling strategy, handing over the remaining layers to TensorRT to get optimal performance.

6.2 Scheduling Strategy for Offloading

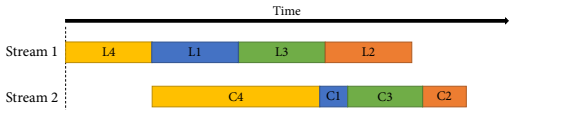
The main challenge of the offloaded MoE layer inference lies in workload imbalance, as the amount of computation performed on different experts may be unbalanced. Tokens are routed and batched to different experts before computation. The workload distribution of experts may vary with different gating mechanisms (Lewis et al., 2021; Lepikhin et al., 2020; Fedus et al., 2021). Experts having more tokens to process will spend more time in computation, while the overhead of data movement (which must be done prior to its computation) of each expert remains the same, for they all have the same amount of parameters.

In INFMOE, by using different CUDA streams, parameter-loading and computation of different experts can be easily overlapped (i.e. executed at the same time). However, as shown in Figure 3(a), naïvely running experts in order easily leads to a waste of time on waiting for parameter loading due to the imbalanced computation time.

[1] INFMOE is an open-source toolkit with MIT License at https://github.com/TsinghuaAI/InfMoE.



(a) In naïve order: C2, C3, C4 must wait for loading.



(b) INFMOE scheduling: all computation run without gaps.

Figure 3: Different scheduling strategies of load-imbalanced experts (L: parameter loading, C: computation).

In order to maximize the overlap between the communication and computation, we design a dynamic schedule strategy in INFMOE to reorder the loading and computation sequence of these experts:

Assume there are T experts in an MoE layer, we can estimate the computation time of the i-th expert (denoted as α_i) and its communication time (denoted as β). α_i is obtained by dividing the amount of floating operations by the peak computation performance of the GPU. With common expert workload (such as feed-forward layers in Transformers), it is proportional to the number of tokens. β can be calculated as the size of parameters to load from the CPU divided by the peak bandwidth of the GPU. It remains the same for all experts. In addition, due to the limit of GPU memory capacity and the existence of parameters belonging to non-MoE layers, only the parameters of a certain number (denoted as K and can be either configured or automatically inferred) of experts can reside in GPU memory simultaneously.

In order to obtain optimal overlapping with negligible cost, INFMOE use a greedy algorithm to generate a computation order of experts that satisfies the following two constraints:

- $\forall 1 \leq t \leq T, \sum_{i=1}^{t-1} \alpha_i \geq (t-1)\beta$. This means the parameter loading of each expert can be fully covered by the computation of previously loaded experts.
- $\forall 1 \leq t \leq T, \sum_{i=1}^{t-1} \alpha_i \leq (t+K-1)\beta$. This means no more than K experts will be loaded to GPU memory simultaneously during the whole process.

This computation order can guarantee that no expert would have to wait for the loading of its parameters except the first one, thus fully hiding the overhead of data movement caused by offloading and leveraging full GPU computing performance (as shown in Figure 3(b). It is possible that these constraints cannot be satisfied at the same time. Such unsatisfiability indicates either the total computation amount is too small, or the workload is extremely imbalanced. The former cause can be mitigated by increasing the batch size, while the latter is out of the scope for inference. As for the MoE gating mechanism described in Section 2.3, it shows relatively good balance between experts in our evaluation, thus fits well for INFMOE.

7 More Promising Directions for Effective and Efficient Pre-trained Language Models

In this section, we will briefly introduce our four novel explorations in tokenization, architecture, pre-training, and fine-tuning to achieve a more efficient pipeline of PLMs.

7.1 Tokenization Based on Pronunciation and Glyph

For Chinese PLMs, input tokenization is quite important. The conventional tokenization methods applied by existing PLMs may treat each character as an indivisible token. However, there are more linguistic information beyond characters. To explore a better tokenization method for Chinese PLMs, we consider pronunciation, glyph, and word segmentation to tokenize the input for PLMs. More specifically, we build pronunciation-based tokenizers, glyph-based tokenizers, and segmentation-based tokenizers respectively, and then systematically evaluate their performance based on BERT. Sufficient experimental results on various downstream NLU tasks have shown that applying pronunciation-based and glyph-based tokenizers can outperform existing used character-based tokenizers, and meanwhile more robust on the text noise. For more details, we refer to our paper (Si et al., 2021).

7.2 Architecture Based on Non-Euclidean Geometry

Some recent efforts have shown that models learned in non-Euclidean geometry could better model complex data , especially those hyperbolic neural networks. However, existing hyperbolic networks are not completely hyperbolic, and training a deep hyperbolic networks is also not trivial. To this end, we introduce a fully hyperbolic framework

to build hyperbolic networks based on the Lorentz model and the Lorentz transformations. Based on the fully hyperbolic framework, we successfully train a hyperbolic Transformer and outperform existing Euclidean baselines. The experimental results show that hyperbolic Transformer can achieve comparable performance to Euclidean Transformers with half the size of model parameters, which may lead to more efficient PLMs in the future. In our paper (Chen et al., 2021), we introduce more details of building hyperbolic neural networks.

7.3 Pre-training Based on Knowledge Inheritance

As we mentioned before, large-scale PLMs have achieved the success on various NLP tasks. However, training a large-scale PLM requires huge amounts of computational resources, which is time-consuming and expensive. Hence, taking the availability of existing well-trained PLMs into consideration is of importance. To this end, we propose knowledge inheritance to make previously trained PLMs benefit later larger PLMs. In fact, CPM-2 is built based on knowledge inheritance. In (Qin et al., 2021), we introduce the overall framework of knowledge inheritance, indicating the effect of teacher PLMs' settings, including pre-training methods, model architectures, training data, etc. For more details, we refer to our orignial paper.

7.4 Fine-tuning Based on Rich Knowledge

In our experiments, we have shown that CPM-2 can perform well with prompt tuning, as additional prompts can stimulate the rich knowledge of PLMs to better serve downstream task. Besides model knowledge distributed in PLMs, we explore to utilize the human prior knowledge to make fine-tuning PLMs more efficient and effective. To this end, we propose prompt tuning with rules, which can apply logic rules to construct prompts with several sub-prompts. By encoding prior knowledge of each class into prompt tuning, PLMs can converge faster and achieve better results on downstream tasks. More details of this part are included in our paper (Han et al., 2021).

Acknowledgments

Thanks to the Beijing Academy of Artificial Intelligence (BAAI) for providing the computing resources. In addition, we would like to thank BAAI, NetEase Inc., zhihu.com, and aminer.cn for the support in collecting the Chinese corpus.

References

Yoshua Bengio. 2013. Deep learning of representations: Looking forward. In *Proceedings of SLSP*, volume 7978, pages 1–37. Springer.

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. In *Proceedings of NeurIPS*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *Computing Research Repository*, arXiv:1604.06174.

Weize Chen, Xu Han, Yankai Lin, Hexu Zhao, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2021. Fully hyperbolic neural networks. Technical report.

J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *Computing Research Repository*, arXiv:2010.11929.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Computing Research Repository*, arXiv:2101.03961.

Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. 2021. PTR: Prompt tuning with rules for text classification. *Computing Research Repository*, arXiv:2105.11259.

Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput.*, 8(1):321–350.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. 2021. FastMoE: A Fast Mixture-of-Expert Training System. *Computing Research Repository*, arXiv:2103.13262.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of CVPR*.

Baotian Hu, Qingcai Chen, and Fangze Zhu. 2015. Lcsts: A large scale chinese short text summarization dataset. In *Proceedings of EMNLP*, pages 1967–1972.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of EMNLP*.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *Computing Research Repository*, arXiv:2006.16668.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. BASE layers: Simplifying training of large, sparse models. *Computing Research Repository*, arXiv:2103.16716.

Xin Liu, Qingcai Chen, Chong Deng, Huajun Zeng, Jing Chen, Dongfang Li, and Buzhou Tang. 2018. LCQMC:a large-scale Chinese question matching corpus. In *Proceedings of COLING*, pages 1952–1962.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. In *ICLR*.

Yujia Qin, Yankai Lin, Jing Yi, Jiajie Zhang, Xu Han, Zhengyan Zhang, Yusheng Su, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2021. Knowledge inheritance for pre-trained language models. *Computing Research Repository*, arXiv:2105.13880.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of SC*, pages 1–16.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of KDD*, pages 3505–3506.

Timo Schick and Hinrich Schütze. 2020a. Exploiting cloze questions for few-shot text classification and natural language inference. *Computing Research Repository*, arXiv:2001.07676.

Timo Schick and Hinrich Schütze. 2020b. It's not just size that matters: Small language models are also few-shot learners. *Computing Research Repository*, arXiv:2009.07118.

Zhihong Shao, Minlie Huang, Jiangtao Wen, Wenfei Xu, et al. 2019. Long and diverse text generation with planning-based hierarchical variational model. In *Proceedings of EMNLP-IJCNLP*, pages 3248–3259.

Chenglei Si, Zhengyan Zhang, Yingfa Chen, Fanchao Qi, Xiaozhi Wang, Zhiyuan Liu, and Maosong Sun. 2021. ShuoWenJieZi: Linguistically informed tokenizers for chinese language model pretraining. Technical report.

Jianlin Su. 2020. Wobert: Word-based chinese bert model - zhuiyiai. Technical report.

Kai Sun, Dian Yu, Dong Yu, and Claire Cardie. 2020. Investigating prior knowledge for challenging chinese machine reading comprehension. *TACL*, 8:141–155.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NeurIPS*, pages 5998–6008.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of EMNLP*, pages 845–854.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2020. mt5: A massively multilingual pre-trained text-to-text transformer. *Computing Research Repository*, arXiv:2010.11934.

Yuan Yao, Qingxiu Dong, Jian Guan, Boxi Cao, Fanchao Qi, Jinliang Lu, Jinran Nie, Junwei Bao, Kun Zhou, Shuhuai Ren, Xiaozhi Wang, Xuancheng Huang, Zheni Zeng, Zile Zhou, Zhiyuan Liu, Erhong Yang, Zhifang Sui, Maosong Sun, Jiajun Zhang, Juanzi Li, Minlie Huang, Rui Yan, Xianpei Han, Xiaodong He, Xiaojun Wan, Xin Zhao, Xu Sun, and Yang Liu. 2021. CUGE: A chinese language understanding and generation evaluation benchmark. Technical report.

Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, et al. 2021. Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation. *arXiv preprint arXiv:2104.12369*.

Zhengyan Zhang, Xu Han, Hao Zhou, Pei Ke, Yuxian Gu, Deming Ye, Yujia Qin, Yusheng Su, Haozhe Ji, Jian Guan, et al. 2020. CPM: A large-scale generative chinese pre-trained language model. *arXiv preprint arXiv:2012.00413*.

A Contributions

Yuxian Gu and Zhengyan Zhang implemented the basic pre-training framework.

Xu Han implemented pipeline parallel strategy for better efficiency.

Zhengyan Zhang implemented the MoE pre-training.

Yuxian Gu, Zhengyan Zhang, Chaojun Xiao, and Xu Han implemented the downstream tasks.

Shengqi Chen, Zhenbo Sun, Xu Han, and Yanzheng Cai implemented the toolkit of INF-MOE.

Jian Guan, Pei Ke, Guoyang Zeng, and Zhixing Tan prepared the pre-training data.

Yuan Yao and Fanchao Qi prepared the fine-tuning data.

Zhengyan Zhang, Yuxian Gu, Xu Han, Chaojun Xiao, Zhenbo Sun, and Shengqi Chen wrote the paper.

Zhiyuan Liu, Minlie Huang, and Wentao Han designed and led the research.

Xiaoyan Zhu and Maosong Sun provided valuable advice to the research.