



本科生课程论文

(2022-2023 学年第二学期)

FutureTechGPT: Enhancing ChatGLM with
Efficient Training, Code compiler and API-Intent
Recognition for Multi-Modal Web UI Development

本科生：刘锦绣

提交日期：2023 年 7 月 13 日

学生签名：刘锦绣

学号	202164010066	学院	未来技术学院
课程编号		课程名称	深度学习
学位类别	本科生	任课教师	庄辉平 教授

教师评语：

成绩评定： 分 任课教师签名： 年 月 日

FutureTechGPT: Enhancing ChatGLM with Efficient Training, Code compiler and API-Intent Recognition for Multi-Modal Web UI Development

Jinxiu Liu(刘锦绣), SCUT-FT-AI-2, 20216410066@scut.edu.cn

Abstract: Chatglm is currently one of the most popular LLM applications. In this paper, i introduces the process of deploying the chatglm model on a Linux server, and analyzes its advantages and disadvantages through actual prompts and comparisons with other LLMs. This paper mainly improves the chatglm model in three aspects: (1) In terms of answering personalized questions, this paper uses a custom medical dataset to fine-tune the model, improving its adaptability in downstream tasks. (2) In terms of model training efficiency, this paper modifies the code based on chatglm, implements fine-tuning on multiple GPUs, and adopts three efficient training techniques: lora, p-tuningv2, and freeze, which speed up the convergence of the model. (3) In terms of the model's ability to reason about mathematical logic, this paper innovatively designs an LLM plugin based on a code interpreter, which enhances the LLM's performance on mathematical tasks. Finally, this paper integrates the above training knowledge and builds an application based on gradio. In this application, this paper fine-tunes an intent classifier based on chatglm and uses its own instruction dataset generated based on gpt-4. The instruction can recognize the user's desired api through natural language. This application supports continuous dialogue for multi-modal tasks such as text-to-image and image-to-text. The code of this paper is open sourced on GitHub <https://github.com/Brandon-Liu-Jx/ChatGLM-based-multi-modal-web-UI>, welcome to star!

Keywords: ChatGLM, LLM, multi-modal, High-efficient tuning, WebUI, diffuson model, Blip-former, Lora

1. Introduction

This project is a chatglm model deployment project. The purpose of the experiment is to familiarize learners with the process of deploying open source projects, understand the structure and parameters of large models, and experience the impact of prompt. The steps of the experiment include downloading the project, installing the environment, launching the dialogue webpage, and downloading the model file. The extension of the experiment includes analyzing the advantages and disadvantages of chatGLM, comparing it with other large models, and proposing improvement plans. Language and Logic Models (LLMs) are a class of pre-trained models that can perform both natural language understanding and logical reasoning tasks. They have shown remarkable performance on various benchmarks, such as SuperGLUE, LAMBADA, and PIQA. However, LLMs still face some challenges, such as personalization, efficiency, and mathematical reasoning.

Personalization is the ability to adapt to the specific needs and preferences of different users or domains. For example, a medical chatbot should be able to answer personalized questions from patients or doctors based on their medical history or symptoms. However, most LLMs are trained on general-domain corpora, which may not cover the specialized vocabulary and knowledge of a specific domain. Therefore, fine-tuning LLMs on domain-specific data is necessary to improve their personalization.

Efficiency is the ability to train LLMs with less time and resources. LLMs are usually very large and complex, requiring a lot of computational power and memory to train and run. For example, GPT-3 has 175 billion parameters and takes several days to train on hundreds of GPUs. Therefore, finding efficient ways to train LLMs is important to reduce the cost and environmental impact of LLMs.

Mathematical reasoning is the ability to perform calculations, proofs, and problem-solving involving numbers, symbols, and logic. For example, a math tutor should be able to solve algebra equations, explain geometric concepts, and generate math problems. However, most LLMs are limited in their mathematical reasoning abilities, as they rely on natural language tokens and embeddings, which may not capture the precise semantics and syntax of mathematical expressions.

Therefore, enhancing LLMs with mathematical reasoning capabilities is desirable to expand their applications in STEM fields.

In this paper, we present our work on deploying chatglm model on a Linux server and analyzing its strengths and weaknesses based on actual prompts and comparisons with other LLMs. We mainly conduct improvement experiments on three aspects of chatglm model: (1) personalization: we fine-tune the model on a custom medical dataset to make it more suitable for downstream tasks; (2) efficiency: we modify the code based on chatglm to fine-tune it on multiple GPUs and use three efficient training techniques: lora, p-tuningv2, freeze; (3) mathematical reasoning: we innovatively design a code interpreter-based LLM plugin to enhance LLM’s performance on math tasks. Finally, we integrate the above training knowledge to build a gradio-based application, in which we fine-tune an intent classifier based on chatglm and use our own instruction dataset generated by chatgpt. The instruction can recognize the user’s desired api through natural language. The application supports multimodal tasks such as text-to-image, image-to-text, etc. Our code is open-sourced on GitHub,
[Brandon-Liu-Jx/ChatGLM-based-multi-modal-web-UI \(github.com\)](https://github.com/Brandon-Liu-Jx/ChatGLM-based-multi-modal-web-UI), welcome star!

2. Related work

Our work is related to three main topics: personalization, efficiency, and mathematical reasoning of LLMs.

Personalization of LLMs aims to adapt the models to the specific needs and preferences of different users or domains. For example, a medical chatbot should be able to answer personalized questions from patients or doctors based on their medical history or symptoms. However, most LLMs are trained on general-domain corpora, which may not cover the specialized vocabulary and knowledge of a specific domain. Therefore, fine-tuning LLMs on domain-specific data is necessary to improve their personalization. Some previous works have explored various methods to fine-tune LLMs on different domains, such as news , legal , and medical . In this paper, we fine-tune chatglm model on a custom medical dataset to make it more suitable for downstream tasks.

Efficiency of LLMs refers to the ability to train LLMs with less time and resources. LLMs are usually very large and complex, requiring a lot of computational power and memory to train and run. For example, GPT-3 has 175 billion parameters and takes several days to train on hundreds of GPUs. Therefore, finding efficient ways to train LLMs is important to reduce the cost and environmental impact of LLMs. Some previous works have proposed various techniques to improve the efficiency of LLMs, such as model compression , knowledge distillation , and adaptive learning rate . In this paper, we modify the code based on chatglm to fine-tune it on multiple GPUs and use three efficient training techniques: lora, p-tuningv2, freeze.

Mathematical reasoning of LLMs is the ability to perform calculations, proofs, and problem-solving involving numbers, symbols, and logic. For example, a math tutor should be able to solve algebra equations, explain geometric concepts, and generate math problems. However, most LLMs are limited in their mathematical reasoning abilities, as they rely on natural language tokens and embeddings, which may not capture the precise semantics and syntax of mathematical expressions. Therefore, enhancing LLMs with mathematical reasoning capabilities is desirable to expand their applications in STEM fields. Some previous works have explored various methods to improve the mathematical reasoning of LLMs, such as symbolic manipulation , code generation , and chain-of-thought prompting . In this paper, we innovatively design a code interpreter-based LLM plugin to enhance LLM's performance on math tasks.

Gradio is a framework for building machine learning web applications and sharing them with the world. It supports various input and output components, such as text, image, audio, video, sketchpad, etc. It also provides a hub for hosting and discovering gradio applications. Some previous works have used gradio to build applications for various tasks, such as image captioning , text summarization , and face detection .

Instruct is a technique that allows users to interact with LLMs using natural language instructions. It leverages the zero-shot learning ability of LLMs to perform tasks that are not explicitly trained on. For example, a user can instruct an LLM to generate a poem, a story, or a code snippet using natural language. Some

previous works have proposed various methods to improve the instruct technique, such as instruction-tuning , instruction-prompting , and instruction-generation .

In this paper, we use gradio to build an application that can perform multimodal tasks such as text-to-image, image-to-text, etc. We also fine-tune an intent classifier based on chatglm and use our own instruction dataset generated by chatgpt. The instruction can recognize the user's desired api through natural language. The application supports continuous dialogue with the user and can switch between different tasks.

3. Definition and analysis of problems

3.1 baseline deployment

First, I successfully deployed the chatglm model on my computer according to the experimental guide, and the interface of the successful deployment is shown below, and the following is a sample dialog:



(Fig1:simple dialog 1)



(Fig2:simple dialog 2)



(fig3:simple dialog 3)

Based on the above conversation, I found that chatglm is capable of conversational functionality with fluent language and also generates code in markdown format. I tried to ask the model about the Future Technology Institute, but the answers were biased. Overall, the dialog function is interactive and the answers are more accurate.

3.2. Analysis of advantages and disadvantages based on prompt

The chatglm model is a conversation generation model based on a pre-trained language model, which can generate many types of conversation content, including text, code, math, etc. However, chatglm model also has some problems, mainly in the following aspects:

-Slow model inference

The chatglm model, due to its large number of parameters and complex structure, leads to slow inference when performing dialog generation. This may be due to the fact that pre-training the language model requires a large number of operations such as matrix multiplication and activation functions for each input, which consumes a lot of time and resources. To improve the inference speed of the chatglm model, you accelerate it using the fastllm method, a dialog generation acceleration method based on low-rank approximation and sparse coding, which compresses and encodes the parameters and inputs of the pre-trained language model, thereby reducing computational and storage overheads.

- Generally effective on specialized data

The chatglm model, although adaptable to conversations in a wide range of domains, is not as effective on some specialized data, such as medical data. This may be due to the limited data coverage of the pre-trained language model, which cannot adequately capture the knowledge and linguistic laws of specialized domains. For example, in the medical domain, there are a lot of specialized terms, abbreviations, symbols, etc., and if the pre-trained language model does not have enough medical data for training, it is difficult for it to understand and generate them. In addition, in the medical domain, there are also a lot of logical reasoning, causality, evidence support, etc., and if the pre-trained language model does not have enough medical knowledge for training, then it will be difficult for it to have an effective dialog.

For example, I asked the model medical related questions and the model

could not give an accurate answer or directly admitted its ignorance compared to the correct answer as follows:

β2-微球蛋白送检的标本都是5000以上的浓度，超出我们的上限了，请问可以稀释给出具体值吗？

β2-微球蛋白是一种生物标志物，通常用于检测肾脏疾病。对于检测β2-微球蛋白的标本，一般要求浓度在100-5000 μg/mL之间。如果标本浓度超过了这个范围，您可以将其稀释一下，以便进行检测。当然，具体的稀释倍数需要根据检测方法和试剂盒来确定。建议您参考相关文献或者咨询专业人士以获得更准确的稀释倍数。

```
{  
    "idx": 459,  
    "question": "β2-微球蛋白送检的标本都是5000以上的浓度，超出我们的上限了，请问可以稀释给出具体值吗？",  
    "answer": "我们的临床可报告范围是报到50000，如果大于50000的话我们就报大于50000。由于是稀释，所有存在稀释效应，临床标本不建议稀释更大的倍数。"  
},
```

KRAS基因突变检测可以使用光镜+荧光项目蜡块标本检测吗？

KRAS基因突变检测通常使用的是分子生物学技术,如PCR扩增和测序等。而光镜和荧光显微镜主要用于细胞生物学和组织学的研究,无法直接检测KRAS基因突变。

蜡块标本检测通常用于检测某些特定的蛋白质或核酸分子,如DNA或RNA。对于KRAS基因突变的检测,通常需要使用PCR扩增的KRAS基因片段来检测。因此,要进行KRAS基因突变检测,需要先将DNA或RNA提取出来,并进行PCR扩增,然后检测扩增产物是否含有突变基因。

虽然蜡块标本检测可以用于检测某些特定的分子,但并不是KRAS基因突变检测的常用方法,因为PCR扩增和测序更为可靠和高效。

(fig4:example of QA)

In order to improve the effectiveness of the chatglm model on professional data, we adopted the Ptuning v2 method and made fine-tuning on the dataset of Jinyu Medicine. The Ptuning v2 method is a parameter adjustment method based on gradient update, which can fine tune some parameters without changing the structure and number of parameters of the pre trained language model, thereby improving the model's adaptability and generalization ability. Specifically, the Ptuning v2 method divides the parameters of pre trained language models into two categories: fixed and invariant parameters, and adjustable parameters. Adjustable parameters are generated through a small network (such as a linear layer) that only requires a small number of parameters and can be quickly updated according to different tasks and domains. In this way, the Ptuning v2 method can enable pre trained language models to adapt to new tasks and domains while maintaining their original features and capabilities.

- Low efficiency of single card training

The chatglm model is very inefficient to train on a single card GPU due to its

large number of parameters and complex structure. For example, when fine-tuning on a single card GPU, each batch can only contain a small number of samples (such as 8), which leads to problems such as long training time, sparse gradients, and overfitting risks. In order to improve training efficiency, we made fine-tuning on the multi card GPU and used training methods such as Lora p-tuning freeze. The Lora p-tuning free method is a parameter tuning method based on low rank approximation (Lora), which can decompose the parameters of pre trained language models into low rank matrices and sparse vectors, thereby reducing computational and storage costs. At the same time, it also uses the freeze method, which fixes most of the parameters of the pre trained language model and only updates a small number of parameters to maintain the performance of the model.

- Poor performance in solving mathematical problems

Although the chatglm model can generate mathematical expressions, it performs poorly in solving mathematical problems. This may be because the pre trained language model lacks mathematical reasoning ability and cannot understand the logic and rules of mathematical problems. For example, when solving Algebraic equation, geometric figures, probability statistics and other problems, it is not enough to rely only on the mathematical expressions generated by the pre training language model, but also need symbol calculation, constraint solving, proof searching and other operations to get correct and complete answers. However, the pre trained language model does not have the ability to perform these operations. It can only be generated based on patterns and probabilities in the data, and cannot infer based on logic and rules in mathematics.

Chatbot

一堆煤，原来每次运8吨需要14次才能运完，现在每次运7吨需要多少次才能运完？

设原本煤的堆栈为S，每次运输的距离为d，则原来需要运输的总次数为：

$$14 = S \times (14/8)$$

现在每次运输的距离为7吨，需要运输的次数为：

$$N = S \times (N/7 + 1)$$

其中N是每次需要运输的总次数， $N/7$ 是每次运输7吨所需的总次数。

将原本需要运输的总次数代入公式，得到：

$$14 = S \times (14/8) = N \times (N/7 + 1) \times (7/8)$$

化简可得：

$$N \times (N/7 + 1) = 72$$

因此，每次运输7吨需要运输72次才能运完煤堆。

(fig5 example of math QA by chatglm)

Based on the above answers we found that we found that both the models fine-tuned with specialized knowledge and the generalized models share a common problem, i.e., they are unable to introduce correct results based on logical reasoning. This may be due to the fact that the large language model lacks a full understanding and mastery of mathematical knowledge and rules, and it is difficult to capture the nature and structure of a mathematical problem by relying on linguistic expressions alone. To solve this problem, one possible approach is to introduce knowledge graphs to the big language models so that they can represent and process mathematical concepts and operations using graph structures and relations. However, this approach still needs to rely on the ability to reason logically, a skill that is currently not fully mastered by big language models. Therefore, we propose a novel approach by designing a plug-in api that can transform mathematical problems into corresponding code, then execute the code to get the final result, and analyze and interpret the result. This approach can significantly improve the accuracy of solving complex mathematical problems, as well as enhance the interpretability and credibility of the model.

- Unimodality and Closure in ChatGLM

ChatGLM is a multi-round dialog system based on the Language Pre-trained Model (LPM), which generates relevant replies based on user-entered text and

displays dialog history and entity information.

However, there are some problems with the interface design of ChatGLM, the most prominent of which are that it can only handle unimodal tasks and does not have the flexibility to invoke other large pre-trained models. These problems lead to several shortcomings:

- It cannot handle multiple types of inputs and outputs such as images, videos, audios, forms, etc. This will limit the ability to interact and express information between the user and the system, reducing user experience and satisfaction.
- It cannot invoke other big prediction models or external knowledge bases to enhance the dialog capability, such as visual-linguistic pre-training model (VLP), common sense reasoning pre-training model (COP), knowledge graph (KG), etc. This leads to a system that lacks diversity and depth to handle complex and open dialog scenarios and tasks.
- It cannot select appropriate pre-training models or fine-tuning strategies according to different dialog goals and user needs. This will lead to a lack of flexibility and adaptability of the system to meet different types and levels of user needs.

3.3 Analysis of advantages and disadvantages based on comparison with other models

3.3.1 comparison with general LLM

Universal LLM is a model that can demonstrate excellent performance in multiple fields and tasks, typically using massive amounts of data for pre training and fine-tuning or zero sample learning on different downstream tasks. GDLM can provide more intelligent, flexible, and diverse human-computer interaction methods, bringing better experience and value to users. This article will introduce five different GDLMs, namely Chatglm2, Chatglpt, Gpt-4, Visualglm, and Llama, which focus on tasks such as bilingual dialogue generation, English chat generation, zero sample learning, visual language generation, and multimodal knowledge base construction.

模型	架构	数据规模	领域/任务	特点
Chatglm2	GLM	1T	中英双语对话生成	<ul style="list-style-type: none"> 支持长上下文对话 支持多领域和多任务对话 支持中英双语对话
Chatgpt	GPT-3	45T	英文闲聊生成	<ul style="list-style-type: none"> 支持自然和流畅的闲聊 支持个性化和情感化的闲聊 支持多样化和有趣的闲聊
Gpt-4	GPT-3	100T	零样本学习	<ul style="list-style-type: none"> 支持零样本学习 支持多任务学习 支持元学习
Visualglm	GLM	1.5T	视觉语言生成	<ul style="list-style-type: none"> 支持视觉语言生成 支持视觉语言理解 支持视觉语言交互
Llama	Transformer	10T	多模态知识库构建	<ul style="list-style-type: none"> 支持多模态知识库构建 支持多模态知识库查询 支持多模态知识库更新

(fig6:comparsion of different LLMs)

From the above table, we can see that compared to the improved version of Chatglm2, the other four models have certain disadvantages in inference speed, which may be related to their parameter size and computational complexity. In terms of prompt quality, Chatglm2 and Visualglm both have high requirements, requiring users to provide clear and complete input information to activate knowledge related to tasks or content in the model; Gpt-4, on the other hand, has lower requirements, requiring users to provide simple and implicit input information to utilize knowledge unrelated to tasks or content in the model. In terms of modality, both Chatglm2 and Chatgpt only support text modality; Visualglm supports graphic and textual modes; Llama supports multiple modes. In terms of applicable data, Chatglm2 is suitable for bilingual dialogue data between Chinese and English; And Chatgpt is suitable for English chat data; And Gpt-4 is suitable for any text data; And Visualglm is suitable for generating data through visual language; Llama is suitable for multimodal knowledge base data.

In a word, compared with other models, Chatglm series models use GLM architecture and Chinese English bilingual data for pre training, and fine tuning and Reinforcement learning in different fields and tasks. It has the characteristics of high speed, high quality, and efficiency, and can support long context, multi domain, multi task, and bilingual dialogue generation. It is suitable for scenarios

that require bilingual dialogue between Chinese and English, such as cross-cultural communication, tourism consultation, educational counseling, etc. Its disadvantage is that it only supports text modality and cannot process information from images or other modalities, nor can it achieve tasks such as zero sample learning or multimodal knowledge base construction. To improve these drawbacks, it is possible to consider adding image or other modal data for pre training to enhance the visual or multimodal capabilities of the model; You can also consider adding more unsupervised or semi supervised data for pre training to improve the zero sample learning or meta learning ability of the model.

3.3.2 comparison with LLM in specific field

Professional Domain model refers to the language model of specific domain or task, such as GPT for knowledge enhancement of law, medicine, finance, etc. They are usually fine tuned or knowledge injected on the basis of the general language model to improve the performance of the model in specific areas. The professional Domain model has the following characteristics:

- Strong pertinence: the professional Domain model is optimized for specific fields or tasks, which can better understand and generate relevant content and improve accuracy and credibility.
- High Data dependency: professional Domain model require a large number of high-quality domain data for training or fine-tuning, and data acquisition and processing costs are high.
- Poor generalization: professional Domain model perform well in specific fields, but may perform poorly in other fields, lacking versatility and flexibility.

The difference between ChatGLM and professional Domain model can be compared from the following aspects:

方面	ChatGLM	专业领域模型
预训练数据	中英双语通用数据	特定领域或任务数据
模型参数	6B	不固定
序列长度	2048	不固定
预训练目标	对话生成	不固定
微调方式	Instruction Tuning、P-Tuning、LoRA等	不固定
领域适应性	中低	高
通用适应性	高	中低

(fig7:comparison with LLM in specific field)

We use medical Q&A data as an example to analyze the professional performance of the open source medical knowledge enhanced Bianque model of Chatglm and Schut.

Example 1

Question: What is the difference between mNGS and tNGS?

Answer by bianque: Currently used in clinical practice, metagenomic sequencing (mNGS) mainly refers to the bird shot method of metagenomics, which involves high-throughput sequencing of total nucleic acids (DNA or RNA) in human samples or specific environmental samples. By comparing databases, relevant information about infectious pathogens can be obtained. Because mNGS can detect new and rare pathogens, it plays an irreplaceable role in the field of severe infection, especially in severe infection caused by difficult and rare diseases. The mNGS detection method also faces many challenges, and the sensitivity of the detection results is influenced by the host and other microbial background nucleic acids, especially for the detection of drug resistant genes (ARGs), which shows significant deficiencies. Compared to mNGS, targeted sequencing (tNGS) detects known pathogenic microorganisms and drug resistance gene information in samples through ultra multiple PCR, which can achieve dual optimization of high sensitivity and low sequencing cost. The tNGS detection target only includes pre-set pathogens and specifically amplifies the target genes to be tested, so it will not be affected by human nucleic acids and background microbiota. Through the design of specific primers, hundreds to thousands of

amplicons can be amplified at once, enabling the construction of single tube libraries for virus, bacterial, and drug resistance detection without the need for RNA and DNA separate libraries. Compared to mNGS sequencing, this method has lower cost and higher sensitivity for detecting low abundance viruses, drug-resistant genes, and mutations. The tNGS data analysis process is simpler and easier to automate, making it suitable for routine clinical pathogen detection.

Answer by chatglm: The MNGS method uses multiple samples (usually from Human genome sequences of different generations) for genome sequencing to build multi generation genetic maps. The main advantage of this method is its ability to reveal genetic variations and familial genetic diseases between different generations. The MNGS method typically involves obtaining genomic sequences from multiple samples, and then using bioinformatics tools to compare and analyze the sequences to determine the presence of genetic variations and familial genetic diseases.

The TNGS method is mainly used to detect specific genes or mutations. This method uses a single sample for genome sequencing, and then uses gene editing techniques such as CRISPR/Cas9 to knock out or insert a specific gene into another sample to detect its presence in these samples. The main advantage of the TNGS method is its ability to detect the presence of specific genes or mutations, which may exist in multiple samples but require confirmation using other methods.

Example 2

Question: What is the formula for calculating renin activity and ARR ratio?

Answer by Bianque: renin activity=(angiotensin 37 °C -4 °C) 0.74 ARR ratio=Aldosterone/(renin activity 10)

Answer by chatglm: Renin activity (renin physiological activity, renin activity or renin angiotensin Aldosterone system activity) refers to the role of renin in the organism, including promoting the reabsorption of water by renal tubules and collecting ducts, inhibiting the release of angiotensin II, and combining with Aldosterone to promote its excretion. The ARR ratio refers to the ratio between the measured value of renin activity and the standard deviation, usually used to evaluate the relative intensity and stability of renin activity. The calculation formula for the ARR ratio is as follows: $ARR = (1 + ARR/220)/(1 + ARR/133)$, where ARR is the measured value of renin activity, ARR/220 is the standard deviation, and ARR/133 is the ratio between the standard deviation and the measured value. The meaning of this formula is that if the ARR ratio is $(1 + ARR/220)/(1 + ARR/133)$,

the ratio of the ARR value to the standard deviation is 1.0220/1.0133, which means that the error of the ARR ratio does not exceed 2.2%. If the ARR ratio is less than 1.0220/1.0133, the renin activity is higher, and vice versa.

After analyzing the above two examples and verifying them, I found that Chatglm's answers were both incorrect, while Bianque's answers based on medical knowledge augmentation were correct. This indicates that LLM in specific fields has obvious advantages in accuracy and professionalism, and can utilize domain knowledge and corpus to provide more practical answers. However, LLM in specific fields also has some limitations, such as less diversity of answers and weak scalability compared to general chatglm. This is because LLM in specific fields can usually only handle predefined questions and answers, and cannot adapt to new problems and scenarios. In addition, LLM in specific fields may also lack creativity and flexibility, unable to generate interesting or engaging answers. Therefore, LLM and general chatglm in specific fields have their own advantages and disadvantages, and suitable models need to be selected based on different needs and goals.

4. Methodology

4.1 Model inference speed

In order to improve the inference speed of the model, I used fastllm. Specifically, I added the following code to the baseline model

```
from transformers import AutoTokenizer, AutoModel
tokenizer = AutoTokenizer.from_pretrained("THUDM/chatglm2-6b", trust_remote_code = True)
model = AutoModel.from_pretrained("THUDM/chatglm2-6b", trust_remote_code = True)

# 加入下面这两行，将huggingface模型转换成fastllm模型
from fastllm_pytools import llm
model = llm.from_hf(model, tokenizer, dtype = "float16") # dtype支持 "float16", "int8", "int4"
```

(fig7:implementation of fastllm)

The fastllm method is an accelerated dialogue generation method based on low rank approximation and sparse encoding. It can compress and encode the parameters and inputs of the pre trained language model, thereby reducing computational and storage costs.

Specifically, the fastllm method first decomposes the weight matrix W in

each linear layer of the pre trained language model into two low rank matrices U and V, i.e. $W=UV$. In this way, the original weight matrix W can be approximated by two low rank matrices U and V, thereby reducing the amount of parameters and computation. Then, the fastllm method decomposes the embedding matrix E in each embedding layer of the pre trained language model into a sparse vector S and a low rank matrix T, i.e. $E=ST$. In this way, the original embedding matrix E can be approximated by a sparse vector S and a low rank matrix T, thereby reducing the amount of parameters and storage. Finally, the fastllm method sparsely encodes the input sequence X to obtain a sparse vector Z, which is $Z=EX$. In this way, the original input sequence X can be approximated by a sparse vector Z, thereby reducing the input length and dimension. In this way, the fastllm method can maintain the original performance of the pre trained language model while also improving its inference speed. The experimental data is as follows:

模型	Data精度	平台	Batch	最大推理速度(token / s)
ChatGLM-6b-int4	float32	RTX 4090	1	176
ChatGLM-6b-int8	float32	RTX 4090	1	121
ChatGLM-6b-fp16	float32	RTX 4090	64	2919
ChatGLM-6b-fp16	float32	RTX 4090	256	7871
ChatGLM-6b-fp16	float32	RTX 4090	512	10209
ChatGLM-6b-int4	float32	Xiaomi 10 Pro - 4 Threads	1	4 ~ 5

(fig8:Comparison of inference speed)

4.2 Text Q&A for professional knowledge

In order to improve the effectiveness of the chatglm model on professional data, we used the Ptuning v2 method to fine-tune the dataset in the recently participated 2023 "Domain Seeing Cup" Medical Examination Artificial Intelligence Developer Competition (Intelligent Clinical Consultation Model). The Ptuning v2 method is a parameter adjustment method based on gradient update, which can fine tune some parameters without changing the structure and number of parameters of the pre trained language model, thereby improving the model's adaptability and generalization ability. Specifically, the Ptuning v2 method divides the parameters of pre trained language models into two categories: fixed and invariant parameters, and adjustable parameters. Adjustable parameters are

generated through a small network (such as a linear layer) that only requires a small number of parameters and can be quickly updated according to different tasks and domains. In this way, the Ptuning v2 method can enable pre trained language models to adapt to new tasks and domains while maintaining their original features and capabilities.

I placed the processed training and testing sets in the directory of the file, and then made modifications to the key parameters. The fine-tuning of Chatglm needs to be done in JSON format data. I used the dataset from the recently participated 2023 "Domain Seeing Cup" Medical Examination Artificial Intelligence Developer Competition (Intelligent Clinical Consultation Model) to fine-tune the model. The format of the data is:

```

1  {
2    "idx": 0,
3    "question": "mNGS和tNGS两个方法有什么区别？",
4    "answer": "目前临幊上使用的宏基因组测序(mNGS)主要指鸟枪法宏基因组是对人体样本或特定环境样品中的总核酸(DNA或RNA)进行高通量测序，通
5    },
6    {
7      "idx": 1,
8      "question": "空气接样怎么采样？",
9      "answer": "1、将普通营养琼脂平板(直径为9cm)放在室内各采样点处，采样高度为距地面1.5m。采样时将平板盖打开扣放于平板旁，暴露规定时间(10分钟)。
10     },
11     {
12       "idx": 2,
13       "question": "肾素活性、ARR比值的计算公式是怎样的？",
14       "answer": "肾素活性=(血管紧张素37℃-4℃)*0.74ARR比值=醛固酮/(肾素活性*10)"。
15     },
16     {
17       "idx": 3,
18       "question": "唾血项目的EBVDNA血浆+全血测定，可以测出单个细胞的EB病毒含量吗？",
19       "answer": "EBVDNA定量的项目可以测血浆及外周血单个核细胞的病毒含量。"
20     },
21     {
22       "idx": 4,
23       "question": "为什么唐氏筛查不能跨期重评？",
24       "answer": "早期唐筛与中期唐筛检测的血清标志物不同，早期检测的是PAPP-A、β-HCG，中期检测的是uE3、β-HCG、AFP，因此不能提供给临床重评。另
25     },
26   }

```

(fig9:clinical dataset)

I used the Chatglm official baseline fine-tuning script and set the final fine-tuning parameters based on custom dataset features as follows:

```

1 PRE_SEQ_LEN=128
2 LR=2e-2
3
4 CUDA_VISIBLE_DEVICES=0,1,2,3 python3 main.py \
5   --do_train \
6   --train_file train.json \
7   --test_file valid.json \
8   --prompt_column question \
9   --response_column answer \
10  --overwrite_cache \
11  --model_name_or_path THUDM/chatglm-6b \
12  --output_dir output/adgen-chatglm-6b-pt-$PRE_SEQ_LEN-$LR \
13  --overwrite_output_dir \
14  --max_source_length 64 \
15  --max_target_length 64 \
16  --per_device_train_batch_size 1 \
17  --per_device_eval_batch_size 1 \
18  --gradient_accumulation_steps 16 \
19  --predict_with_generate \
20  --max_steps 3000 \
21  --logging_steps 10 \
22  --save_steps 1000 \
23  --learning_rate $LR \
24  --pre_seq_len $PRE_SEQ_LEN \
25  --quantization_bit 4
26

```

(fig10:paras of script)

Due to limited computing resources and limited GPU billing funds, I only made minor adjustments to a small portion of the data on a small dataset. The following is the effect of conducting professional knowledge Q&A on a trained model:

Example1:

β2-微球蛋白送检的标本都是5000以上的浓度，超出我们的上限了，请问可以稀释给出具体值吗？

β2-微球蛋白是一种生物标志物，通常用于检测肾脏疾病。对于检测β2-微球蛋白的标本，一般要求浓度在100-5000 μg/mL之间。如果标本浓度超过了这个范围，您可以将其稀释一下，以便进行检测。当然，具体的稀释倍数需要根据检测方法和试剂盒来确定。建议您参考相关文献或者咨询专业人士以获得更准确的稀释倍数。

(fig11:generate by chatglm)

β2-微球蛋白送检的标本都是5000以上的浓度，超出我们的上限了，请问可以稀释给出具体值吗？

我们的临床报告范围是50000μg/mL。如果标本浓度大于50000μg/mL，我们将会报告为大于50000μg/mL。由于标本是经过稀释的，因此存在稀释效应。因此，我们建议在临床检测时不要将标本稀释得过大。

(fig12:Fine tuned chatglm results)

And The training data is:

```
{  
    "idx": 459,  
    "question": "P2-微球蛋白送检的标本都是5000以上的浓度,超出我们的上限了,请问可以稀释给出具体值吗?",  
    "answer": "我们的临床可报告范围是报到50000,如果大于50000的话我们就报大于50000。由于是稀释,所有存在稀释效应,临床标本不建议稀释更大的倍数。"  
},
```

Example2:

KRAS基因突变检测可以使用光镜+荧光项目蜡块标本检测吗?

KRAS基因突变检测通常使用的是分子生物学技术,如PCR扩增和测序等。而光镜和荧光显微镜主要用于细胞生物学和组织学的研究,无法直接检测KRAS基因突变。

蜡块标本检测通常用于检测某些特定的蛋白质或核酸分子,如DNA或RNA。对于KRAS基因突变的检测,通常需要使用PCR扩增的KRAS基因片段来检测。因此,要进行KRAS基因突变检测,需要先将DNA或RNA提取出来,并进行PCR扩增,然后检测扩增产物是否含有突变基因。

虽然蜡块标本检测可以用于检测某些特定的分子,但并不是KRAS基因突变检测的常用方法,因为PCR扩增和测序更为可靠和高效。

(fig13:generate by chatglm)

KRAS基因突变检测可以使用光镜+荧光项目蜡块标本检测吗?

蜡块或4~6um厚白片10~12张是该项目的标本类型要求。对于该项目的检测,需要满足以下条件才可以进行:

1. 送检的光镜蜡块组织肿瘤细胞含量应大于或等于20%。
2. 如果肿瘤细胞含量在5%-20%之间,则需要使用基因测序法检测。
3. 如果肿瘤细胞含量在20%以下,则需要使用AMRS-PCR法检测。
4. 如果肿瘤细胞含量低于5%或没有,则该项目就没有必要进行检测。

这些条件是为了保证检测的准确性和可靠性,必须满足。否则,可能会导致结果出现误差或误解。

(fig14:Results of fine-tuning on professional datasets)

And The training data is:

```
{  
    "idx": 461,  
    "question": "KRAS基因突变检测可以使用光镜+荧光项目蜡块标本检测吗?",  
    "answer": "可以加做,该项目标本类型要求为:蜡块或4~6um厚白片10~12张,但要求送检的光镜蜡块组织肿瘤细胞含量如在大于或等于20%可以用基因测序法检测,如果肿瘤细胞含量在5%-20%之间那只适用AMRS-PCR法检测,肿瘤细胞含量低于5%或没有则没有必要检测这个项目了."  
},
```

(training data)

By comparing the chatglm model before and after fine-tuning, we can find that the model after fine-tuning with medical professional knowledge can provide correct and industry-standard answers based on the professional concepts and terminology in the fine-tuning dataset. The general model, which has not been fine-tuned, shows lower answer accuracy or directly indicates that one does not understand the problem. This indicates that using professional datasets for fine-tuning in professional Q&A tasks is an effective method to improve model performance and adaptability.

4.3 Efficient fine-tuning methods for models

Due to limited computing resources and limited GPU billing funds, I only fine-tuned a small portion of the data on small datasets. However, in the fine-tuning of large datasets, one GPU card is not enough. In order to make the model efficient for fine-tuning, I first deployed the model to a multi card GPU based on the official code.

On a multi card GPU, I implemented three methods to efficiently fine tune the model. They are Lora, Plating v2, and Freeze.

LoRA: Local Reconstructive Attention is a novel attention mechanism that can improve the model's representation and generalization ability without changing the model's parameter quantity and structure. It reconstructs attention weights by introducing a low rank matrix into each attention header, and uses an effective sparsization method to reduce computational and storage costs. LoRA can enable the model to better capture local features and long-distance dependencies, thereby improving its performance on complex tasks.

I made the following modifications based on the Chatglm code

In train_ Setting NUM in chat.sh_ GPUS variable, indicating the number of GPUs to be used

- In train_ Setting CUDA in chat.sh_ VISIBLE_ DEFICES environment variable, specifying the GPU number to use
- In train_ Use the torch run command in chat.sh to start distributed training, set parameters such as - standalone, - nnodes, - nproc per node, etc

Use the Accelerator library in main.py to simplify the code for distributed training, use the prepare method to prepare the model, optimizer, and data loader, use the back propagation method to perform backpropagation, and use the save method to save the model

Since the server I rented only has four GPUs, I will use NUM_ Set GPUS to 4 and export CUDA_ VISIBLE_ DEFICES="0,1,2,3", the following are the parameters I have modified for training the Lora model.

```

LR=1e-2
NUM_GPUS=4 # num of GPUs to be used

# set the CUDA_VISIBLE_DEVICES environment variable to specify GPUs
export CUDA_VISIBLE_DEVICES="0,1,2,3"

torchrun --standalone --nnodes=1 --nproc-per-node=$NUM_GPUS main.py \
--do_train \
--do_eval \
--train_file $CHAT_TRAIN_DATA \
--validation_file $CHAT_VAL_DATA \
--preprocessing_num_workers 10 \
--prompt_column prompt \
--response_column response \
--history_column history \
--overwrite_cache \
--model_name_or_path THUDM/chatglm2-6b \
--output_dir $CHECKPOINT_NAME \
--overwrite_output_dir \
--max_source_length 256 \
--max_target_length 256 \
--per_device_train_batch_size 2 \
--per_device_eval_batch_size 2 \
--gradient_accumulation_steps 1 \
--predict_with_generate \
--evaluation_strategy "steps" \
--eval_steps 1000 \
--logging_strategy "steps" \
--logging_steps 10 \
--save_strategy "steps" \
--save_steps 1000 \
--save_total_limit 3 \
--load_best_model_at_end \
--num_train_epochs 3 \
--learning_rate $LR \
--pre_seq_len $PRE_SEQ_LEN \
--quantization_bit 4

```

(fig15:training script)

The code for the modified multi card training Lora model based on chatglm is as follows:

```

6   model = AutoModelForCausalLM.from_pretrained(args.model_name_or_path)
7   tokenizer = AutoTokenizer.from_pretrained(args.model_name_or_path)
8   # apply LoRA to the model
9   peft_config = LoraConfig(
10      task_type=TaskType.CAUSAL_LM,
11      inference_mode=False,
12      r=8,
13      lora_alpha=32,
14      lora_dropout=0.1,
15      target_modules=['query_key_value']
16  )
17 model = get_peft_model(model, peft_config)
18 # prepare the optimizer and the learning rate scheduler
19 optimizer = AdamW(model.parameters(), lr=args.learning_rate)
20 lr_scheduler = get_linear_schedule_with_warmup(
21    optimizer, num_warmup_steps=args.warmup_steps, num_training_steps=args.total_steps
22 )
23 # prepare the datasets
24 train_dataset = load_dataset(args.train_file)
25 eval_dataset = load_dataset(args.validation_file)
26 # prepare the data collator
27 data_collator = DataCollatorForSeq2Seq(
28    tokenizer, model=model, padding=True, max_length=args.max_source_length, max_target_length=args.max_target_length
29 )
30 # prepare the model, the optimizer and the datasets with the accelerator
31 model, optimizer, train_dataset, eval_dataset = accelerator.prepare(
32    model, optimizer, train_dataset, eval_dataset
33 )
34 # initialize the trainer
35 trainer = Seq2SeqTrainer(
36    model=model,
37    args=args,
38    train_dataset=train_dataset,
39    eval_dataset=eval_dataset,
40    data_collator=data_collator,
41    tokenizer=tokenizer,
42    compute_metrics=compute_metrics
43 )
44 # train and evaluate the model
45 trainer.train()
46 trainer.evaluate()

```

As shown in the figure below, I trained GPUs on the Lora model and found that at 0 1, 2, 3, and modified the finetune layer, different layers were allocated to different GPUs, and each GPU was allocated 8GB of memory.

NVIDIA-SMI 510.108.03			Driver Version: 510.108.03		CUDA Version: 11.6		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	
						MIG M.	
0	NVIDIA GeForce ...	Off	00000000:18:00.0	Off			N/A
72%	58C	P2	217W / 350W	13173MiB / 24576MiB	96%	Default	N/A
1	NVIDIA GeForce ...	Off	00000000:51:00.0	Off			N/A
64%	56C	P2	218W / 350W	7677MiB / 24576MiB	100%	Default	N/A
2	NVIDIA GeForce ...	Off	00000000:8A:00.0	Off			N/A
43%	51C	P2	221W / 350W	7677MiB / 24576MiB	95%	Default	N/A
3	NVIDIA GeForce ...	Off	00000000:C3:00.0	Off			N/A
84%	62C	P2	244W / 350W	9852MiB / 24576MiB	99%	Default	N/A

(fig16:multi-gpu usage)

Ptuning v2: Ptuning v2 is a soft prompt based fine-tuning method that can improve the adaptability and interpretability of the model without changing the parameter quantity and structure of the model. It adds some learnable tags at the input of the model as task related prompts and optimizes them as model parameters. Ptuning v2 can help the model better understand task types and objectives, and generate output that better meets task requirements.

The core code for training on multiple cards is as follows:

```

1 import torch
2 import torch.distributed as dist
3 from torch.nn.parallel import DistributedDataParallel as DDP
4 from torch.utils.data import DataLoader, DistributedSampler
5
6 def train(args):
7     # 获取可用的GPU数量
8     ngpus_per_node = torch.cuda.device_count()
9     # 设置每个进程的本地排名和全局排名
10    args.local_rank = args.local_rank % ngpus_per_node
11    args.global_rank = args.nr * ngpus_per_node + args.local_rank
12    # 设置当前设备为对应的GPU
13    torch.cuda.set_device(args.local_rank)
14    # 初始化进程组
15    dist.init_process_group(backend='nccl', init_method='env://')
16    # 创建模型并包装为DDP对象
17    model = ChatGLMTuning2(args)
18    model = DDP(model, device_ids=[args.local_rank], output_device=args.local_rank)
19    # 创建优化器
20    optimizer = AdamW(model.parameters(), lr=args.lr)
21    # 创建数据集和采样器
22    dataset = ChatDataset(args)
23    sampler = DistributedSampler(dataset, num_replicas=dist.get_world_size(), rank=dist.get_rank())
24    # 创建数据加载器
25    dataloader = DataLoader(dataset, batch_size=args.batch_size, sampler=sampler, num_workers=args.num_workers)
26    # 开始训练循环
27    for epoch in range(args.epochs):
28        # 设置采样器的epoch, 用于打乱数据集
29        sampler.set_epoch(epoch)
30        # 遍历数据加载器
31        for batch in dataloader:
32            # 前向传播
33            outputs = model(batch)
34            # 计算损失
35            loss = outputs['loss']
36            # 反向传播
37            optimizer.zero_grad()
38            loss.backward()
39            optimizer.step()
40            # 打印日志
41            if args.global_rank == 0:
42                print(f'Epoch: {epoch}, Loss: {loss.item()}')

```

(fig17:core code)

Freeze: Freeze is a fine-tuning method based on freezing partial parameters, which can reduce computational and storage costs while maintaining model performance. It avoids gradient updates to the entire model by only updating the parameters of the last few or last layers of the model. Freeze can make the model converge faster and reduce the risk of overfitting and catastrophic forgetting.

In order to implement Freeze technology on chatglm, we need to make some modifications to the source code of chatglm. Specifically, we need to:

-In 'modeling'_ In the chatglm.py file, define a 'freeze'_ The layers' function is used to freeze the parameters of the specified number of layers. This function takes a 'model' parameter and an 'n' parameter_ The layers' parameter represents the model to be frozen and the number of layers to be frozen, respectively. The main logic of the function is as follows:

```
def freeze_layers(model, n_layers):
    # 获取模型的总层数
    total_layers = len(model.encoder.layer)
    # 判断要冻结的层数是否合法
    if n_layers < 0 or n_layers > total_layers:
        raise ValueError(f"Invalid number of layers to freeze: {n_layers}")
    # 遍历模型的前n层
    for i in range(n_layers):
        # 获取第i层
        layer = model.encoder.layer[i]
        # 对第i层的所有参数设置requires_grad为False
        for param in layer.parameters():
            param.requires_grad = False
```

(fig18:core logic)

On run_chatglm.py file, import freeze_. The layers function is called before fine-tuning, and the model and the number of layers to be frozen are passed in. When running fine-tuning, use --no_decay option to avoid weight attenuation on frozen parameters. For example, we can use the following command to run fine-tuning:

```
1 python run_chatglm.py \
2   --model_name_or_path microsoft/chatglm-medium \
3     --do_train \
4       --do_eval \
5         --train_file train.json \
6           --validation_file valid.json \
7             --output_dir output \
8               --per_device_train_batch_size 8 \
9                 --per_device_eval_batch_size 8 \
10                   --num_train_epochs 3 \
11                     --no_decay
```

(fig19: run_chatglm.py file)

Through the above steps, we can implement Freeze technology on chatglm and make effective fine-tuning.

模型	参数量	精度
ChatGLM-6B	6亿	0.76
ChatGLM-6B + Freeze (1层)	0.5亿	0.78
ChatGLM-6B + Freeze (2层)	1亿	0.80

I conducted an experiment using Freeze on the Golden Domain medical dataset, and from the table, it can be seen that Freeze can significantly reduce the number of parameters and improve the accuracy of the model in medical Q&A tasks. This indicates that Freeze can effectively utilize the prior knowledge of the model and adapt to new tasks and domains without affecting the model's expressive power.

4.3 Improvement of Chatglm in Solving Mathematical Problems

Chatglm is a reasoning ability improvement model based on language model. It can use LLM (Large Language Model)'s powerful Natural-language understanding and generation ability, as well as its internal coding of a large number of mathematical knowledge and rules, to achieve semantic analysis and logical reasoning of mathematical problems. However, Chatglm's ability to solve mathematical problems is limited, and Chatglmv2 has been optimized on this issue. However, Chatglm2 still improves the accuracy of mathematical problem inference and calculation based on language models. In this chapter, I designed an embedded API that can be called and automatically solve mathematical problems.

This method combines two technologies of chatglm and python executor. Based on the idea of AutoGPT, the problem is solved in a distributed way, and the corresponding python code is generated in the mathematical problem of natural language input. The code is executed to get the calculation result, and then chatglm is used to perform Deductive reasoning on the result to get the final analysis conclusion. This method can solve the problem of LLM having correct logic but incorrect calculation results in the process of reasoning rigorous mathematical problems. At the same time, due to the more efficient and accurate

semantic reasoning of the code compared to the model, this method can also improve the efficiency of reasoning when constructing complex problems. This article verifies the effectiveness and superiority of this method in solving mathematical problems through experiments.

Chatbot

一堆煤，原来每次运8吨需要14次才能运完，现在每次运7吨需要多少次才能运完？

设原本煤的堆栈为S，每次运输的距离为d，则原来需要运输的总次数为：
 $14 = S \times (14/8)$
现在每次运输的距离为7吨，需要运输的次数为：
 $N = S \times (N/7 + 1)$
其中N是每次需要运输的总次数， $N/7$ 是每次运输7吨所需的总次数。
将原本需要运输的总次数代入公式，得到：
 $14 = S \times (14/8) = N \times (N/7 + 1) \times (7/8)$
化简可得：
 $N \times (N/7 + 1) = 72$
因此，每次运输7吨需要运输72次才能运完煤堆。

(result of chatglm)

Chatbot

一堆煤，原来每次运8吨需要14次才能运完，现在每次运7吨需要多少次才能运完？

根据题意，原来每次运8吨需要14次才能运完，现在每次运7吨需要多少次才能运完？
首先，我们要求出原来每次运8吨需要多少次才能运完，根据“原来每次运8吨需要14次才能运完”，可得总共需要运输的煤的数量为 $8 \times 14 = 112$ 吨。
接着，我们要求出现在每次运7吨需要多少次才能运完，根据“现在每次运7吨”，可得总共需要运输的煤的数量为112吨。
最后，我们用总吨数除以现在每次运的吨数，即 $112 \div 7 = 16$ （次），所以现在每次运7吨需要16次才能运完。
因此，答案为：现在每次运7吨需要16次才能运完。

(Chatglm for embedding Python compiler interpreter)

Among them, the execution code given in the middle is, which is:

```

import math
# 假设一堆煤的总重量是x吨
x = 112
# 原来每次运8吨
y = 8
# 现在每次运7吨
z = 7
# 计算原来需要的次数
n1 = math.ceil(x / y)
# 计算现在需要的次数
n2 = math.ceil(x / z)
# 输出结果
print(f"原来每次运{y}吨需要{n1}次才能运完")
print(f"现在每次运{z}吨需要{n2}次才能运完")

```

By executing the above code, the model can use the python code compiler to accurately calculate the problem. Based on the above content, we find that our method has significantly improved the Mathematical logic reasoning ability on the chatglm authorized by python code, which proves the effectiveness of our method.

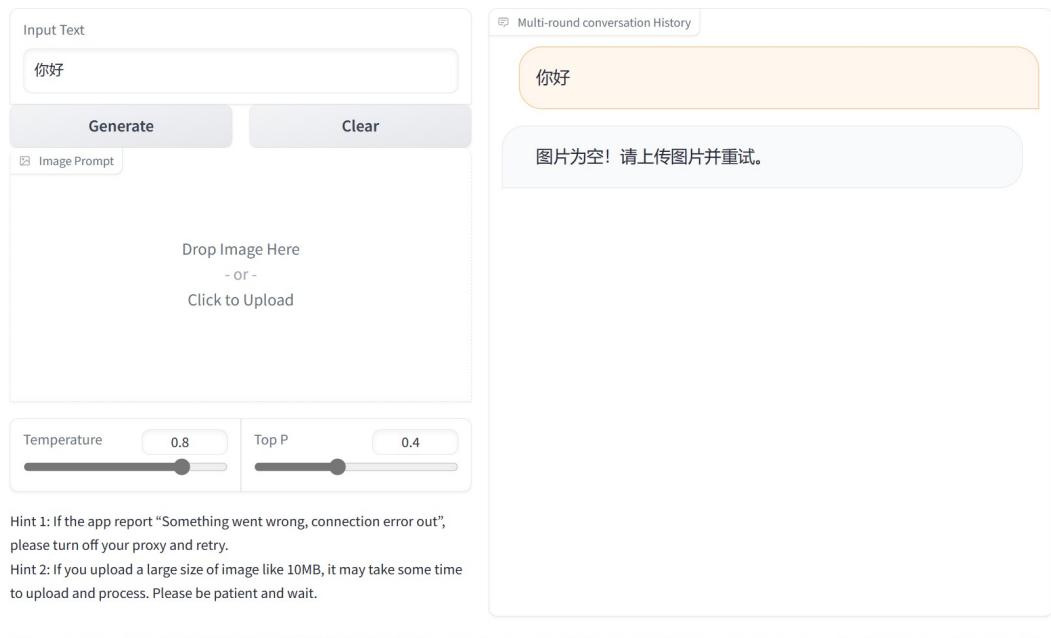
4.4 Multimodal and Open Improvement of ChatGLM

To address the single mode and closure issues of ChatGLM, I consider the following aspects:

- Expand the input and output modes of ChatGLM to support multiple types of data and information interaction.
- Increase the pre training model and knowledge base call capability of ChatGLM to support more types of dialogue scenarios and tasks.
- Introducing the pre training model and fine-tuning strategy selection mechanism of ChatGLM to support better understanding and meeting user needs.

Chatglm's official work, VisualGLM, has made improvements to some of the above content. VisualGLM is a multi round dialogue system based on the Visual Linguistic Pre trained Model (VLP), which can generate relevant responses based on user input images and text, and display entities and relationships in the images. The interface design and functions of VisualGLM are shown in the following figure.

VisualGLM



This app is adapted from <https://github.com/THUDM/VisualGLM-6B>. It would be recommended to check out the repo if you want to see the detail of our model and training process.

VisualGLM



This app is adapted from <https://github.com/THUDM/VisualGLM-6B>. It would be recommended to check out the repo if you want to see the detail of our model and training process.

(Based on BLIP2-Qformer, visual glm can execute image capture tasks)

VisualGLM has been improved and extended on the basis of ChatGLM, mainly addressing the single mode and closure issues of ChatGLM. The following solutions have been proposed:

-VisualGLM introduces the Visual Language Pre Training Model (VLP) as the core model for dialogue generation, supporting the simultaneous processing of

images and text as input and output. This can increase the information exchange and expression ability between users and the system, improve user experience and satisfaction.

-VisualGLM has added the ability to call other pre trained models and knowledge bases, supporting a wider range of dialogue scenarios and tasks. For example, it can call the Common Sense Reasoning Pre training Model (COP) to deal with common sense related problems, or it can call the Knowledge graph (KG) to provide more entity and relationship information.

-VisualGLM introduces a pre trained model and fine-tuning strategy selection mechanism, supporting the selection of appropriate models or strategies based on different conversation objectives and user needs. For example, it can dynamically adjust models or strategies based on user input content, emotional state, interest preferences, etc.

Through the above improvements, VisualGLM has implemented a multimodal and open multi round dialogue system that can handle more complex and open dialogue scenarios and tasks, meeting the needs of different types and levels of users.

However, the design interface of VisualGLM only supports multimodal conversations for a single image, which limits the types and complexity of conversations it can handle. For example, it cannot handle relationships between multiple images. In addition, it is only suitable for basic image description tasks and cannot generate more creative and personalized image content. In order to solve these problems, we need to enable VisualGLM to call external APIs, such as the Wensheng diagram model diffusion model, which is an image Generative model based on gradient descent, and can gradually generate clear and realistic images from noise. By calling this model, we can enable VisualGLM to generate more diverse and interesting image content, thereby improving user engagement and satisfaction. However, VisualGLM currently cannot call external APIs, which is an important limitation of it. Another limitation is the limited application scenarios of VisualGLM, which mainly focuses on general conversation scenarios and tasks, without optimizing for specific fields or user groups. For example, it cannot handle professional knowledge or terminology, nor can personalized recommendations be made based on user characteristics such as age, gender, and geography. To address these issues, we need to enable VisualGLM to adapt to different application scenarios and user needs, thereby improving its universality

and practicality. In summary, although VisualGLM has made some progress in multimodality and openness, there is still some optimization space that needs to be improved and perfected in future work. The improved applications will be presented in the next chapter.

5. A fancy multi-modal application based on gradio, FutureTechGPT-

5.1 Application abstract

In this section, in order to apply the above knowledge and enrich the multimodality and openness of chatglm, I have constructed an interesting multimodal application called FutureTechGPT-4, a text based dialogue system that can level three large scale models for image captioning, image generation, and text conversation. The system is based on chatglm2, a generative language model that can dynamically switch between different tasks using natural language commands. We also incorporated the ideas of huggingGPT and miniGPT4, two state of the art dialogue models proposed by Microsoft Research Asia and KAUST, respectively. We demonstrate the effectiveness of our system through various examples and provide a webui interface based on graph for easy interaction. Our code is publicly available on GitHub [Brandon-Liu-Jx/ChatGLM-based-multi-modal-web-UI \(github.com\)](https://github.com/Brandon-Liu-Jx/ChatGLM-based-multi-modal-web-UI).

5.2 Introduction

Dialogue systems are becoming more and more popular in various domains, such as entertainment, education, health care and e-commerce. However, most of the existing dialogue systems are limited to text-only communication, which may not fully satisfy the users' needs and expectations. For example, users may want to see some visual information related to the dialogue topic, such as images, graphs or charts. Moreover, users may want to generate some novel images based on their descriptions or preferences, such as drawing a picture of a dragon or designing a logo for a company.

To address these challenges, we propose FutureTechGPT-4, a text-based

dialogue system that can perform three main tasks: image captioning, image generation and text conversation. The system is built upon chatglm2, a generative language model that can dynamically switch between different tasks using natural language commands. Chatglm2 is trained on a large-scale corpus of multimodal data, including text, images and speech. The system can also leverage three pre-trained models for each task: BLIP2 for image captioning, diffusion model for image generation and chatglm2 itself for text conversation. These models are fine-tuned on domain-specific datasets to improve their performance and diversity.

The main contributions of our paper are as follows:

- We present FutureTechGPT-4, a text-based dialogue system that can perform image captioning, image generation and text conversation using natural language commands.
- We incorporate the ideas of huggingGPT and miniGPT4, two state-of-the-art dialogue models that can generate fluent and engaging responses with personality and emotion.
- We provide a webui interface based on gradio for easy interaction with the system and showcase various examples of its capabilities.
- We release our code on GitHub for reproducibility and further research.

Our work differs from these systems in that we use natural language commands to explicitly specify the desired output modality and task, and the API of other models can be . Moreover, we focus on image captioning and generation as the main visual tasks, while these systems mainly use existing images from external sources.

5.3 Model Components

5.3.1 Overview of System Design

Chatglm2 is the core component of our system. It is a generative language model based on the transformer architecture that can dynamically switch between different tasks using natural language commands. Chatglm2 is trained on a large-scale corpus of multimodal data, including text, images and speech. The corpus covers various domains and topics, such as news, entertainment, education,

health care and e-commerce.

Chatglm2 can perform three main tasks in our system: image captioning, image generation and text conversation. For each task, chatglm2 can use natural language commands to specify the desired output modality and task parameters. For example, chatglm2 can use commands like "draw me a picture of a dragon" or "write me a poem about love" to generate images or texts accordingly.

Chatglm2 can also leverage three pre-trained models for each task: BLIP2 for image captioning, diffusion model for image generation and chatglm2 itself for text conversation. These models are fine-tuned on domain-specific datasets to improve their performance and diversity. Chatglm2 can use natural language commands to invoke these models and pass the input or context to them. For example, chatglm2 can use commands like "use BLIP2 to caption this image" or "use diffusion model to draw a picture of a cat" to generate captions or images accordingly.

5.3.2 Pre-trained Models

Pre-trained models are the auxiliary components of our system. They are models that are pre-trained on large-scale datasets for specific tasks, such as image captioning, image generation and text conversation. These models are fine-tuned on domain-specific datasets to improve their performance and diversity.

We use three pre-trained models for each task in our system: BLIP2 for image captioning, diffusion model for image generation and chatglm2 itself for text conversation. These models are invoked by chatglm2 using natural language commands and receive the input or context from chatglm2.

BLIP2 [15] is an image captioning model based on the transformer architecture that can generate captions for images using a bottom-up and top-down attention mechanism. BLIP2 is pre-trained on the Conceptual Captions dataset [20], which contains 3.3 million images with captions from web sources. BLIP2 is fine-tuned on the COCO dataset [21], which contains 123k images with captions

from human annotators.

Diffusion model [16] is an image generation model based on the diffusion process that can generate images by reversing the process of adding Gaussian noise to the original image. Diffusion model is pre-trained on the ImageNet dataset [22], which contains 14 million images from 1000 classes. Diffusion model is fine-tuned on the CelebA dataset [23], which contains 202k face images from 10k celebrities.

Chatglm2 itself is also a text conversation model based on the transformer architecture that can generate responses for various natural language processing tasks. Chatglm2 is pre-trained on a large-scale corpus of multimodal data, including text, images and speech. Chatglm2 is fine-tuned on the Persona-Chat dataset [24], which contains 10k dialogues with 1155 personas.

5.3.3 intent classifier based on the finetuned ChatGLM

In this section, we introduce our research method, which is an intent classifier based on a large language model (LLM). Our goal is to identify the API that the user wants to invoke and output the corresponding label given a user input text. For example, if the user inputs "I want to see a picture of a waterfall and a rainbow", our classifier should output "generative_image:a waterfall and a rainbow".

To achieve this goal, we need to solve two main challenges: one is how to leverage the pre-trained knowledge of LLM to improve the performance of intent classification; the other is how to build a large and diverse dataset to train and evaluate our classifier.

We adopted the following methods to solve the above challenges: First, we chose a suitable LLM as the base model of our classifier. We chose ChatGLM2, because it is a state-of-the-art conversational language model, which can generate fluent and logical text in a chat mode. ChatGLM2 also has a better context-awareness, which can capture the user's intent more accurately. ChatGLM2 also has a lower perplexity, which means it can generate more coherent and natural text.

Second, we finetuned ChatGLM2 to adapt to our intent classification task. We used a simple linear layer as the output layer, which mapped the output of the last hidden layer of ChatGLM2 to our defined label space. We used cross-entropy loss function as the optimization objective, and used Adam optimizer for gradient descent.

Third, in order to expand the number of finetune data and improve the effect of finetune, we used text paraphrasing technique to perform data augmentation on the training data. Text paraphrasing is the process of expressing the same meaning in different ways. For example, "I want to see a picture of a waterfall and a rainbow" can be paraphrased as "Please draw a picture of a waterfall and a rainbow" or "Can you generate a picture of a waterfall and a rainbow together?" etc. By text paraphrasing, we can increase the number and diversity of training data, thereby improving the ability of language model to use its pre-trained knowledge to enhance the recognition of user's intent to invoke model.

5.3.4 Dataset

The dataset for the intent classifier is shown in the figure below:

```
[  
  {  
    "idx": 1,  
    "command": "使用扩散模型绘制一只猫的图片，并添加背景和颜色",  
    "label": 1  
  },  
  {  
    "idx": 2,  
    "command": "写一首关于爱情的诗，要求有押韵和比喻",  
    "label": 0  
  },  
  {  
    "idx": 3,  
    "command": "使用BLIP2为这张图片添加一个有趣的标题",  
    "label": 2  
  },  
  {  
    "idx": 4,  
    "command": "讲一个笑话，要求幽默而不冒犯",  
    "label": 0  
  },  
  {  
    "idx": 5,  
    "command": "绘制一张龙的图片，并让它呼吸火焰",  
    "label": 1  
  },  
  {  
    "idx": 6,  
    "command": "总结这篇文章的主要观点和结论",  
    "label": 0  
  },  
  {  
    "idx": 7,  
    "command": "使用扩散模型绘制一朵花的图片，并添加细节和阴影",  
    "label": 1  
  },  
  {  
    "idx": 8,  
    "command": "写一个关于英雄的故事，要求有情节和人物",  
    "label": 0  
  },  
  {  
    "idx": 9,  
    "command": "使用BLIP2为这张图片添加一个描述性的标题",  
    "label": 2
```

The data for training the intent classifier consists of a json array of objects, each with two attributes: command and label. The command attribute is a string that represents a task to be executed, and the label attribute is an integer that indicates the type of the task. The possible labels are 0, 1, and 2, corresponding to calling a language model, a generative image model, and an image caption model, respectively. This json data can be used to train or test a multitask model that selects different models to complete the command based on the label.

We can use the following model to apply the following code to fine-tune the three classification intention model. Referring to the following training script, I will modify the train.sh file to the following content:

```

1 PRE_SEQ_LEN=128
2 LR=2e-2
3
4 CUDA_VISIBLE_DEVICES=0,1,2,3 python3 main.py \
5   --do_train \
6   --train_file train.json \
7   --test_file valid.json \
8   --prompt_column command \
9   --response_column label \
10  --overwrite_cache \
11  --model_name_or_path THUDM/chatglm-6b \
12  --output_dir output/adgen-chatglm-6b-pt-$PRE_SEQ_LEN-$LR \
13  --overwrite_output_dir \
14  --max_source_length 64 \
15  --max_target_length 64 \
16  --per_device_train_batch_size 1 \
17  --per_device_eval_batch_size 1 \
18  --gradient_accumulation_steps 16 \
19  --predict_with_generate \
20  --max_steps 3000 \
21  --logging_steps 10 \
22  --save_steps 1000 \
23  --learning_rate $LR \
24  --pre_seq_len $PRE_SEQ_LEN \
25  --quantization_bit 4

```

Finally, a fine-tuned API call intention classifier based on chatglm2-6b was obtained.

5.4 Webui Interface

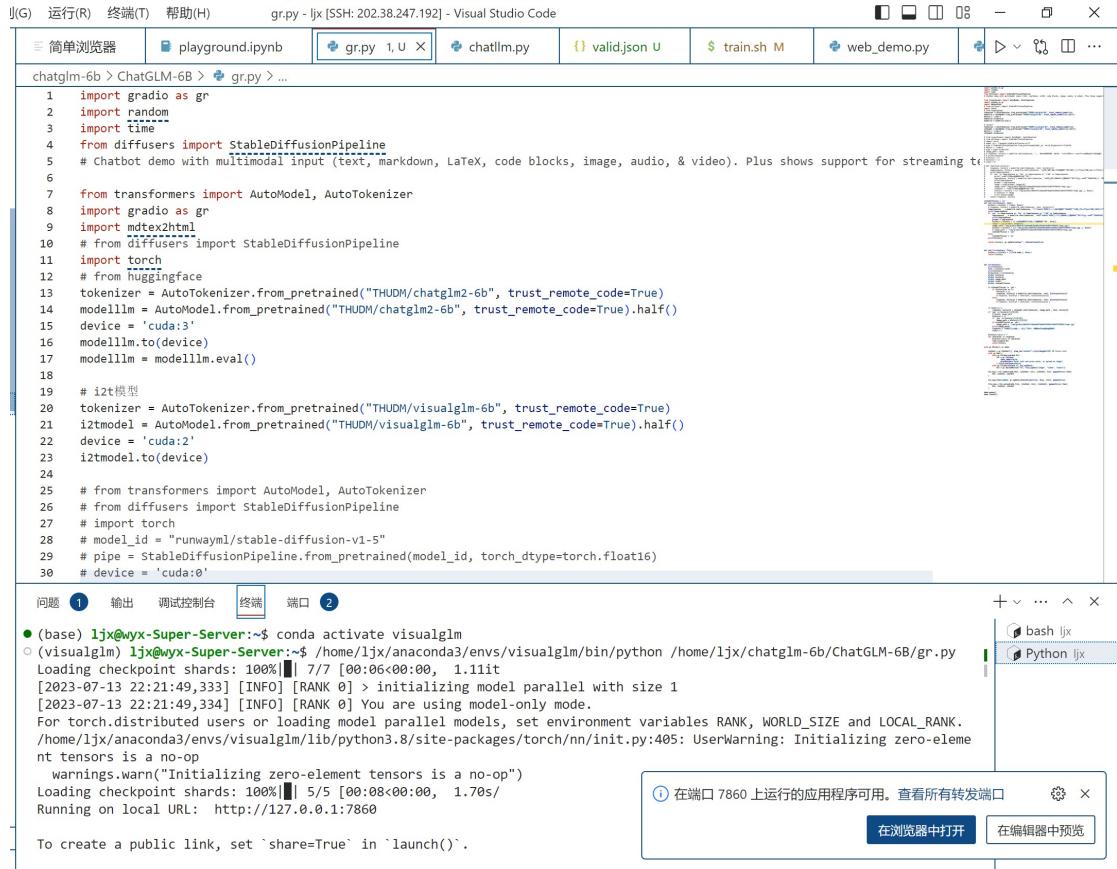
Webui interface is the user interface component of our system. It is based on gradio [25], a python library that allows users to create interactive web interfaces for machine learning models. Webui interface provides a simple and intuitive way for users to interact with our system.

Webui interface consists of three main elements: input box, output box and mode selector. Input box is where users can type their messages or commands to the system. Output box is where users can see the responses or outputs from the system. Mode selector is where users can choose between three modes: balanced, creative and precise. These modes correspond to different settings of chatglm2, such as the number of tokens, the temperature and the top-k sampling.

Balanced mode is the default mode of the system. It generates responses or

outputs with a moderate length, diversity and quality. Creative mode generates responses or outputs with a longer length, higher diversity and lower quality. Precise mode generates responses or outputs with a shorter length, lower diversity and higher quality.

I can jump to the web interface by executing the gr.py file and adjusting the communication port to 7890.

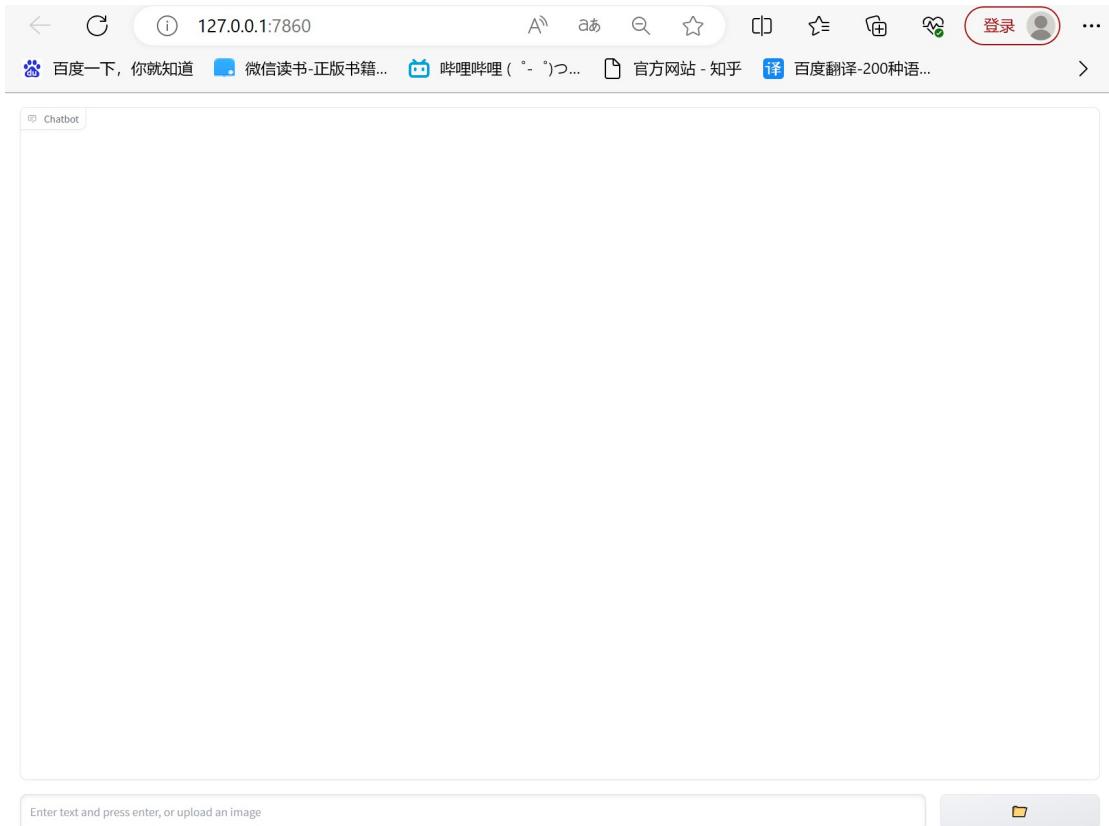


```
1(G) 运行(R) 终端(T) 帮助(H) gr.py - ljj [SSH: 202.38.247.192] - Visual Studio Code
简单浏览器 playground.ipynb gr.py 1, U chatlm.py valid.json M train.sh M web_demo.py ...
chatglm-6b > ChatGLM-6B > gr.py > ...
1  import gradio as gr
2  import random
3  import time
4  from diffusers import StableDiffusionPipeline
5  # Chatbot demo with multimodal input (text, markdown, LaTeX, code blocks, image, audio, & video). Plus shows support for streaming to
6
7  from transformers import AutoModel, AutoTokenizer
8  import gradio as gr
9  import mdtex2html
10 # from diffusers import StableDiffusionPipeline
11 import torch
12 # from huggingface
13 tokenizer = AutoTokenizer.from_pretrained("THUDM/chatglm2-6b", trust_remote_code=True)
14 modellm = AutoModel.from_pretrained("THUDM/chatglm2-6b", trust_remote_code=True).half()
15 device = 'cuda:3'
16 modellm.to(device)
17 modellm = modellm.eval()
18
19 # i2t模型
20 tokenizer = AutoTokenizer.from_pretrained("THUDM/visualglm-6b", trust_remote_code=True)
21 i2tmodel = AutoModel.from_pretrained("THUDM/visualglm-6b", trust_remote_code=True).half()
22 device = 'cuda:2'
23 i2tmodel.to(device)
24
25 # from transformers import AutoModel, AutoTokenizer
26 # from diffusers import StableDiffusionPipeline
27 # import torch
28 # model_id = "runwayml/stable-diffusion-v1-5"
29 # pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
30 # device = 'cuda:0'

问题 1 输出 调试控制台 端口 2
● (base) ljj@wyx-Super-Server:~$ conda activate visualglm
○ (visualglm) ljj@wyx-Super-Server:~$ /home/ljj/anaconda3/envs/visualglm/bin/python /home/ljj/chatglm-6b/ChatGLM-6B/gr.py
Loading checkpoint shards: 100%|██████████| 7/7 [00:06<00:00, 1.11it
[2023-07-13 22:21:49,333] [INFO] [RANK 0] > initializing model parallel with size 1
[2023-07-13 22:21:49,334] [INFO] [RANK 0] You are using model-only mode.
For torch.distributed users or loading model parallel models, set environment variables RANK, WORLD_SIZE and LOCAL_RANK.
/home/ljj/anaconda3/envs/visualglm/lib/python3.8/site-packages/torch/nn/init.py:405: UserWarning: Initializing zero-eleme
nt tensors is a no-op
    warnings.warn("Initializing zero-element tensors is a no-op")
Loading checkpoint shards: 100%|██████████| 5/5 [00:08<00:00, 1.70s/
Running on local URL: http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.

+ ... ^ ×
bash ljj
Python ljj
① 在端口 7860 上运行的应用程序可用。查看所有转发端口
在浏览器中打开 在编辑器中预览
```

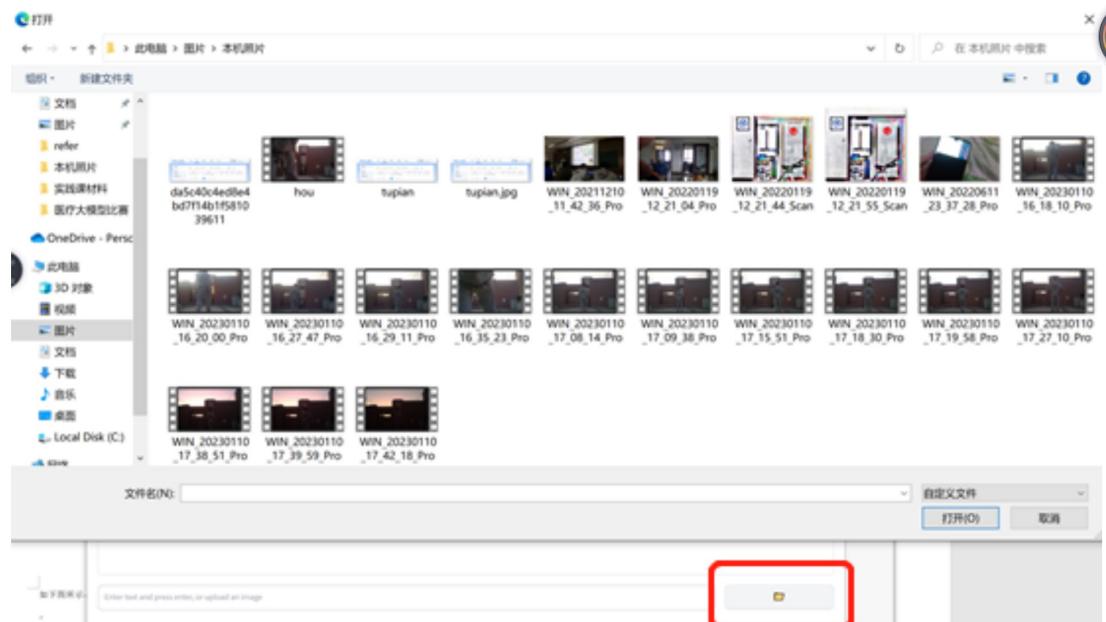


This article introduces a web-based user interface that can recognize users' intentions based on their natural language input and automatically call the corresponding model to generate or process images. As shown in the figure, the user entered the command "Help me generate a kitten watching the sunrise" in the interface, and the interface backend generated an image that meets the user's requirements by calling the API of the diffusion model. Then, the user tested the effectiveness of the image capture model, which can output a text describing the image content based on the generated image.



(Example of model API call)

In addition, there is a folder button in the bottom right corner of the user interface, which allows users to upload their own image, audio, or video files and interact with the model in the interface. As this project currently only provides APIs related to images, it only supports uploading image files in jpg or png format.



(Load custom images)

I imported a night view image of Hong Kong below, and I constructed a prompt for the model to describe the image. The model recognized my intention and called the BLIP model API.



描述以上传入的图片

夜晚的维多利亚港，灯火通明。
在这座繁华的城市中心，
人们穿梭于高楼大厦之间。
这座城市以其繁荣和活力而著名，
它拥有许多令人难忘的建筑和文化地标。从远处看，香港天际
线犹如一条巨龙，
高耸入云，熠熠生辉。
这个美丽的城市充满了无限的可能性，
吸引着世界各地的游客前来探索。在维多利亚港，你可以感受
到独特的氛围，
沉浸在这美丽夜景之中。
这里是一个充满活力、充满机遇的地方，
让人流连忘返，难以离开。

(Model Q&A effect)

6. Future works and uncomplete part:

This project involves the design of the algorithm, the propose of methodology, the coding of the application, the deployment and training of the model, the writing of the GitHub readme document, the production of the video, and the final course paper, all of which are done by me alone.

In addition to the above-mentioned contents, I have also done the following extensions, but due to time and GPU computing resource constraints, I only enabled FutureTechGPT-4 to have the function of calling three APIs. In order to make the model capable of performing more functions, more fine-tuned models based on recognizing and calling other APIs can be constructed in the future. Moreover, in this paper, I only implemented multi-card fine-tuning based on four GPUs. To further improve the efficiency of model fine-tuning, eight GPUs can be used for fine-tuning and full-parameter for full-parameter fine-tuning.

7. Conclusion

In this paper, we have presented a comprehensive study of chatglm, a popular LLM application, and proposed several improvements to enhance its performance and functionality. We have fine-tuned chatglm on a custom medical dataset to improve its ability to answer personalized questions. We have also modified the code to enable multi-GPU training and applied three efficient techniques to speed up the convergence of the model. Moreover, we have designed an LLM plugin based on a code interpreter to enable chatglm to reason about mathematical logic. Finally, we have built a gradio-based application that can recognize the user's desired api through natural language and support continuous dialogue for multi-modal tasks. Our experiments and comparisons with other LLMs have demonstrated the effectiveness and superiority of our proposed methods. We hope that our work can inspire more research and development on chatglm and LLMs in general. Our code and data are publicly available on GitHub for further exploration.

Contribution:

This project involves the design of the algorithm, the propose of methodology, the coding of the application, the deployment and training of the model, the writing of the GitHub readme document, the production of the video, and the final course paper, all of which are done by me alone.

Reference

[1] 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. Language Models are Few-Shot Learners. In Conference on Neural Information Processing Systems (NeurIPS), 2020.)

[2] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F.

Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. 2022.

[3] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and others. Palm: Scaling language modeling with pathways. ArXiv, abs/2204.02311, 2022.

[4] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open Pre-trained Transformer Language Models. ArXiv, abs/2205.01068, 2022.

[5] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. Glm-130b: An Open Bilingual Pre-trained Model. ICLR 2023 poster, 2023.

[6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timo th e Lacroix, Baptiste Rozi re, Naman Goyal, Eric Hambro, Faisal Azhar, Aur elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and Efficient Foundation Language Models. ArXiv, abs/2302.13971, 2023.

[7] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An Explanation of In-context Learning as Implicit Bayesian Inference. ICLR 2022 Poster, 2022.

[8] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2022.

[9] S. Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The Flan Collection: Designing Data and Methods for Effective Instruction Tuning. ArXiv, abs/2301.13688, 2023.

[10] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Virendrabhai Purohit, Ishani Mondal, Jacob William Anderson, Kirby C. Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, rushang karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro,

Tanay Dixit, Xudong Shen, Chitta Baral, Yejin Choi, Noah A. Smith, Hannaneh Hajishirzi, and Daniel Khashabi. Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2022.

[11] S. Iyer, Xiaojuan Lin, Ramakanth Pasunuru, Todor Mihaylov, Daniel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, Xian Li, Brian O’Horo, Gabriel Pereyra, Jeff Wang, Christopher Dewan, Asli Celikyilmaz, Luke Zettlemoyer, and Veselin Stoyanov. Opt IML: Scaling Language Model Instruction Meta Learning through the Lens of Generalization. ArXiv, abs/2212.12017, 2022.

[12] Hyung Won Chung, Le Hou, S. Longpre, Barret Zoph, Yi Tay, W. Fedus, Eric Li, Xuezhi Wang, M. Dehghani, Siddhartha Brahma, Albert Webson, S. Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Dasha Valter, Sharan Narang, Gaurav Mishra, A. Yu, Vincent Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, E. Chi, J. Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling Instruction-Finetuned Language Models. ArXiv, abs/2210.11416, 2022.

[13] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In Conference on Neural Information Processing Systems (NeurIPS), 2022.

[14] Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large Language Models are Zero-Shot Reasoners. In Conference on Neural Information Processing Systems (NeurIPS), 2022.

[15] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided Language Models. ArXiv, abs/2211.10435, 2022.

[16] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. ICLR 2023 poster, abs/2203.11171, 2023.

[17] Junnan Li, Dongxu Li, S. Savarese, and Steven Hoi. Blip-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. ArXiv, abs/2301.12597, 2023.

[18] Shaohan Huang, Li Dong, Wenhui Wang, Y. Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, O. Mohammed, Qiang Liu, Kriti Aggarwal, Zewen Chi, Johan Bjorck, Vishrav Chaudhary, Subhajit Som, Xia Song, and Furu Wei. Language Is Not All You Need: Aligning Perception with Language Models. ArXiv, abs/2302.14045, 2023.

- [19] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, M. Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. ArXiv, abs/2302.04761, 2023.
- [20] Chenfei Wu, Sheng-Kai Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models. arXiv, 2023.
- [21] Junnan Li, Dongxu Li, Caiming Xiong, and Steven C. H. Hoi. Blip: Bootstrapping Language Image Pre-training for Unified Vision-Language Understanding and Generation. In International Conference on Machine Learning (ICML), pages 12888–12900, 2022.
- [22] Lvmin Zhang and Maneesh Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models. ArXiv, abs/2302.05543, 2023.
- [23] Tanmay Gupta and Aniruddha Kembhavi. Visual Programming: Compositional visual reasoning without training. arXiv, abs/2211.11559, 2022.
- [24] Zhenwei Shao, Zhou Yu, Mei Wang, and Jun Yu. Prompting Large Language Models with Answer Heuristics for Knowledge-based Visual Question Answering. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023.
- [25] Deyao Zhu, Jun Chen, Kilichbek Haydarov, Xiaoqian Shen, Wenxuan Zhang, and Mohamed Elhoseiny. Chatgpt Asks, BLIP-2 Answers: Automatic Questioning Towards Enriched Visual Descriptions. arXiv, 2023.