Enhancing InstructPix2Pix Multilingual Reasoning Capabilities by Instruction Finetuning and In-Context-Learning

Shizhuo Li Haoxiang Sun Yiyan Zhai
{shizhuol, haoxians, yiyanz}@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, 15213

1 Introduction

The ability to edit and manipulate 2D content has become crucial in many fields, including computer graphics, virtual reality, and digital content creation. Emerging techniques, such as InstructPix2Pix (Brooks et al., 2023), have revolutionized the way users interact with image editing software with natural language instructions. However, InstructPix2Pix still faces significant challenges.

Currently, the model completely fails on Chinese instructions, despite an increasing number of Chinese-speaking populations who could benefit from this technology. Moreover, the model only works well when the instruction is very detailed, and it cannot make any inferences about what the user wants.

In this project, we present novel work to solve two major limitations of InstructPix2Pix: **(1)** add the ability for InstructPix2Pix to understand Chinese editing instructions, and **(2)** add the ability for InstructPix2Pix to understand vague editing instructions that require careful reasoning.

Our key contributions include:

- Leverage advanced LLM in-context learning and chain-of-thought techniques to inject instruction reasoning capabilities to InstructPix2Pix
- Extend InstructPix2Pix's capability to understand Chinese editing instructions at inference time using LLM-based translation
- Propose new reasoning dataset for image instruction editing benchmarking and training
- Fine-tune InstructPix2Pix on our new reasoning dataset and demonstrate its improvement

2 Dataset and Tasks

2.1 Terminology Definition

In this report, we refer to several kinds of texts in the image editing process.

Original caption: Image caption for the unedited image. In Appendix Figure 7 left example from (Jin et al., 2024), it would be "Picture of the white house".

Modified caption: Image caption for the edited image. In Appendix Figure 7 left example, it would be "Picture of the white house with fireworks in the sky".

Edit/Original prompt: Detailed image editing text instructions (in English). In Appendix Figure 7 left example, it would be "put fireworks in the sky". This is what InstructPix2Pix is trained on.

Vague/Reasoning prompt: Vague or unclear instructions that require careful reasoning to figure out what the user truly wants. In Appendix Figure 7 left example, it would be "celebrate the independence day". It is up to the model to infer what edits should be made given the vague reasoning instruction. In this case, the model made the connection that celebration should mean the addition of fireworks.

Chinese prompt: Detailed image editing instructions in Chinese. It is produced by translating the English edit/original prompt into Chinese, so they have the same meaning.

2.2 Task Definition

The task is natural language instruction-guided zero-shot image editing. Given an input image and a natural language instruction describing the desired edit, the model generates the edited image directly in a forward pass.

Specifically, in this task, we focus on the model's capability from two perspectives: **(1)** the degree to which it understands text instruction and performs edits given Chinese prompt, and **(2)** the degree to which it understands text instruction and performs edits given vague prompts.

2.3 Baseline

We use the original InstructPix2Pix model as our baseline and evaluate it on two benchmarks: a Chinese dataset benchmark and a reasoning dataset benchmark. In the Chinese benchmark, we input Chinese prompts as edit instructions to assess the model's ability to process Chinese language inputs. In the reasoning benchmark, we provide vague prompts as edit instructions to test the model's capability to infer user intent from less detailed instructions.

2.4 Dataset Generation

The original InstructPix2Pix dataset framework utilizes 700 human-created text prompt examples [1] to generate around 450,000 image editing samples. They then apply a CLIP similarity score to these samples to create a subset of 313,010 high-quality images for training the model.

[1]https://github.com/timothybrooks/InstructPix2Pix

There is no existing public dataset for multilingual image editing or image editing dataset that includes reasoning. Thus, we take the CLIP-filtered portion from the original dataset and augment it with the Qwen LLM model (Yang et al., 2024). Specifically, we start with around 20% of the 436GB CLIP-filtered dataset, which is 62,000 samples and 84GB.

Then, we use custom few-shot in-context-learning techniques to take in the original caption, original edit prompt, and the modified caption, to generate the Chinese prompt and reasoning prompt. This process draws inspiration from ReasonPix2Pix (Jin et al., 2024). We include the LLM prompts in the Appendix Figure 8, 9.

For the translation task, we considered using Google Translate [2]. However, we found the quality is better with LLM-based translation for many situations. This is because with LLMs, we are able to give it the the complete context, and tell it the relationships between the captions and the edit prompts. By having better context awareness and understanding of the task, the LLM is able to generate more accurate translations. For example, in one edit prompt, Google Translate converted the text “make it at night” into the Chinese equivalent of “go at night”, while Qwen produced the Chinese equivalent of “change it to be at night”, which is more accurate.

We used Qwen2.5-32B-Instruct-GPTQ-Int8, which uses around 40GB of GPU memory, and both generation processes took over 40 hours on an NVIDIA A6000 GPU. This is due to the large scale of the generation. We conservatively estimate that over 80,000,000 tokens are used in the process. Thus, to speed up the inference speed and save resources, we observe that the few-shot prompt text is always the same. Thus, there is no reason to repeatedly compute them. Instead, we enable auto prefix caching, which reuses them and gives a 2x speedup, as shown in Figure 2. Nonetheless, 40+ hours were used even after using this optimization.

2.5 Evaluation Metrics

In our experiments, we will evaluate the performance of the models using two primary metrics based on CLIP. These metrics assess the models’ ability to balance preserving the original image content with accurately applying the desired edits specified by the instructions.

- **CLIP Image Similarity**: Cosine similarity of CLIP image embeddings between the input image and the edited image, measuring how much the edited image retains the content of the input image.
- **Directional CLIP Similarity**: This metric measures the consistency of the change between the two images (in CLIP space) with the change between the two English *image captions*.

[2]https://translate.google.com

3 Related Works

3.1 InstructPix2Pix: Learning to Follow Image Editing Instructions(Brooks et al., 2023)

InstructPix2Pix is a diffusion-based generative model that is designed for prompt driven image editing. InstructPix2Pix treats image editing as a supervised learning task, that is, given an input image and a natural language instruction describing the desired edit, the model generates the edited image directly in a forward pass. The model is trained on a paired dataset generated by combining GPT-3 for creating edit instructions and captions, and with Stable Diffusion for generating paired images. It utilizes a CLIP-based metric to ensure alignment between captions and images in evaluation. This paper provides a strong baseline model that we may refernece and finetune on.

3.2 ReasonPix2Pix: Instruction Reasoning Dataset for Advanced Image Editing(Jin et al., 2024)

InstructPix2Pix as previously mentioned demonstrate the ability to deal with explicit instruction image editing. However, such approach often struggle with implicit, abstract, or multi-step reasoning tasks. The paper proposed a reasoning-attentive dataset for instruction-based image editing, which is constructed focusing on implicit and reasoning-focused instructions paired with images, and those data that contains significant variances between input and edited images, such as geometric changes and background changes. Finetuning on model with such dataset proved a increase in reasoning ability of image editing and addressing significant change problem in original IntructPix2Pix. This provides a dataset we may sample from for our finetuning and testing dataset for enhancing model’s capability of reasoning.

3.3 PixWizard: Versatile Image-to-Image Visual Assistant with Open-Language Instructions(Lin et al., 2024)

This paper introduces PixWizard, introduces variety of vision tasks such as image editing, text-to-image generation, inpainting, outpainting, image restoration, dense image prediction, and controllable image generation, and combined them into a same framework. It introduces Omni Pixel-to-Pixel Instruction-Tuning Dataset, that contains data tasks above covering multiple domains such as image restoration, image editing, etc, providing diverse training examples for instruction-based learning. Our construction of own finetuning dataset may reference variety of tasks of this dataset and reference the data from this dataset.

3.4 Instruct-GS2GS: Editing 3D Gaussian Splats with Instructions(Vachha and Haque, 2024)

Instruct-GS2GS introduces a method for editing 3D Gaussian Splatting (3DGS) scenes using text-based instructions, building on the iterative dataset update approach of Instruct-NeRF2NeRF(Haque et al., 2023). It leverages Instruct-Pix2Pix diffusion model, presenting image editing on different angles for 3D reconstruction. With Instruct-Pix2Pix, it iteratively update all training dataset images for 3DGS scenes. In the final step of our project, we will incorporate the pipline of Instruct-GS2GS to test the finetuned model's performance comparing to baseline model.

3.5 High-Resolution Image Synthesis with Latent Diffusion Models(Rombach et al., 2022)

The paper proposes Latent Diffusion Models (LDMs). It addresses computational challenges in diffusion models (DMs) for image synthesis. LDMs leverage pretrained autoencoders to operate in a perceptually equivalent, lower-dimensional latent space, reducing computational costs while maintaining synthesis quality. This paper presents our base model, latent diffusion for our project InstructPix2Pix model, which is the model that the project uses in generating image. The model allows us to computationally efficiently apply diffusion for generating images.

3.6 Chinese CLIP: Contrastive Vision-Language Pretraining in Chinese(Yang et al., 2023)

The research on "Chinese CLIP: Contrastive Vision-Language Pretraining in Chinese" introduces a Chinese adaptation of the CLIP model for multimodal tasks. The original CLIP metric does not support encoding of Chinese due to vocabulary base limitation, which does not contain langauge like Chinese. To address the challenges in lack of Chinese metric in evaluation metric and challenge of transferring vision-language foundation models to Chinese contexts, the authors develop Chinese CLIP. Our project incorporates applying multilingual text-image editing performance enhancing where Chinese CLIP can be applied.

3.7 DiffChat: Learning to Chat with Text-to-Image Synthesis Models for Interactive Image Creation(Wang et al., 2024)

The paper introduces "DiffChat," which refines user prompts and generates high-quality outputs based on user instructions. A new dataset, InstructPE, is created for supervised training. The dataset is generated with reinforcement learning with criteria for aesthetics, user preference, and content integrity. The framework employs advanced techniques like Action-space Dynamic Modification (ADM) and Value Estimation with Content Integrity (VCI) for improved performance.

3.8 Qwen2 Technical Report (Yang et al., 2024)

The Qwen2 models is introduced and they represent a significant advancement in large language models (LLMs) and multimodal models. Some key innovations include dense models, a Mixture-of-Experts (MoE) model, and instruction-tuned variants designed for diverse applications such as coding and logical reasoning. Some other advancement include improvements in context length capabilities and robust multilingual support over 30 languages. The model will be used as a LM for text and data generation for our customized dataset.

4 Approach

In this project, we aim to build two solutions to the two challenges, by using inference time processing and using fine-tuning on synthetically generated datasets.

4.1 Reasoning challenge

1. **Inject reasoning into model at inference time**

   To enhance the model's capability in editing with implicit instructions and in reasoning, we performed few-shot chain-of-thought in-context learning. When given the reasoning prompt, our pipeline first uses BLIP2 model (Li et al., 2023) to generate a caption description of the original image. Then, we used Qwen model (Bai et al., 2023) to generate a more detailed editing instruction using the caption and the reasoning prompt. The few-shot prompt used is shown in Figure 11. Since the editing instruction can often be quite vague or unclear, we provide the caption as additional contextual information to assist the LLM in producing the best possible translation. Finally, we fed the original image and the detailed editing instruction to InstructPix2Pix to generate the edited image. The fine-tuning architecture is depicted at the bottom of Figure 1 in the Appendix.

   Since an LLM is used in a inference time pipeline, robustness is key. It is well known that LLMs somtimes have difficulty generating structured output, such as valid JSON strings. To mitigate this issue, we employ LLM reflection techniques to try to correct any potential parse errors, as described in the code overview section. In our experience, when performing over 600 LLM pipeline inferences, several cases needed reflection and need to be corrected. However, no case were uncorrectable.

2. **Fine tuning on synthetic reasoning dataset**

   We incorporated the dataset pattern of Reason-Pix2Pix Dataset (Jin et al., 2024) to extend the InstructPix2Pix dataset. To achieve this, we used the Qwen model to perform few shot in context learning to transform each detailed editing instruction to a more vague reasoning instruction. The new dataset containing the vague reasoning instruction is used to fine-tune the InstructPix2Pix model. The fine-tuning architecture is depicted on the top of Figure 1 in the Appendix.

4.2 Multilingual challenge

1. **LLM-based language translation at inference time**

   In this method, we used a pre-trained Qwen LLM model that is well-versed in multiple languages to translate the user instruction to English. The translated instruction is then used to edit the image using InstructPix2Pix. Again, the BLIP2 model is used to generate a caption of the original image to provide further context information. The few-shot prompt used is shown in Figure 10.

Since an LLM is also used at inference time in this approach, we use the same reflection techniques as described above to solve JSON parse errors.

5 Experiment

In this section, we detail the experiments conducted to evaluate the performance of our extended InstructPix2Pix model on Chinese prompts and on vague prompts. We compare our model with the baseline InstructPix2Pix model and present both quantitative and qualitative results.

5.1 Chinese Instructional Image Editing Task

As introduced in Dataset Section, we constructed a Chinese instructional image editing dataset by leveraging existing images and English prompts in InstructPix2Pix dataset and translating the prompts into Chinese using the `Qwen-2.5` model.

5.1.1 Experimental Setup

A separate evaluation set of 1,000 image-instruction pairs was created to benchmark the models. The evaluation prompts were also translated using `Qwen-2.5`.

- **Baseline Model**: Original InstructPix2Pix trained on English prompts without reasoning capability.
- **Our Inference Time Translation Model**: InstructPix2Pix with inference time translation.

5.2 Image Editing Task with Vague Prompt

We constructed a vague instruction image editing dataset by leveraging existing images and original prompts in InstructPix2Pix dataset and the reasoning prompts that do not provide clear instructions.

5.2.1 Experimental Setup

A separate evaluation set of 1,000 image-instruction pairs was created to benchmark the models. The evaluation reasoning prompts were also generated using `Qwen-2.5`.

- **Baseline Model**: Original InstructPix2Pix (same as in 5.1.1).
- **Our Inference Time Reasoning Model**: InstructPix2Pix with inference time reasoning.
- **Our Fine-tuned Model**: InstructPix2Pix fine-tuned on vague instructions.

5.3 Quantitative Results

We employed quantitative metrics based on CLIP Image Similarity and Directional CLIP Similarity.

We plot the consistency with the input image and the consistency with the edit for all models, as shown in Figure 3 in the Appendix. For both methods, we fix the text guidance scale to 7.5 and vary the image guidance scale s_I in the range $[1.0, 2.2]$. According to Figure 3, for the original prompts, the baseline model outperforms the fine-tuned model, indicating that the fine-tuned model's adaptations may have slightly compromised its general performance on the original dataset.

For the Chinese prompts, both the baseline and the fine-tuned model does not perform well, suggesting that the model alone can not handle the linguistic difference. However, the pipeline model (inference time LLM translation), with inference-time translation, shows significantly improved performance, showing the effectiveness of this approach in handling Chinese tasks. Notably, the pipeline model's performance on Chinese prompts is comparable to the baseline model's performance on English prompts, suggesting that the translation step successfully bridges the gap in linguistic understanding.

For the reasoning prompts, the fine-tuned model outperforms the baseline model and pipeline model (inference time reasoning), showcasing its enhanced capability to handle vague instructions. This improvement highlights the benefit of targeted finetuning for reasoning tasks.

5.4 Qualitative Results

Examples of edited images are presented in Figure 4. This figure showcases side-by-side comparisons of the original images and the edited outputs using selected Chinese and vague prompts. The baseline model often ignores or misinterprets Chinese and reasoning instructions. This is because the text encoder in Stable Diffusion that is used in baseline model is only trained on English dataset. Thus, baseline method cannot interpret Chinese language. Our pipeline model can generally perform well: it understands the Chinese instructions and demonstrates reasoning capabilities. However, our fine-tuned model sometime struggles with images involving humans.

5.5 Timing and Intermediate Results

We trained the model using a machine equipped with an Intel i9-14900K CPU, 64GB of DDR5 RAM, and an NVIDIA A6000 GPU. For our project, we conducted full fine-tuning for all runs, using approximately 39GB out of the available 48GB of GPU memory.

For small-scale training with 10,000 prompts, the process took about 10 minutes per epoch, and we fine-tuned the model for 2 epochs. After making sure that the initial training process is correct by evaluating the model to monitor its performance, we proceeded with larger-scale training using 20,000 prompts, which required 20 minutes per epoch and was fine-tuned for 10 epochs. Finally, we scaled up to 60,000 prompts, corresponding to over 400,000 images. This training took approximately 1 hour per epoch. By running it for 23 epochs, it resulted in a total runtime of about 24 hours.

For all runs, we performed validation every 4 epochs. The validation step incurred negligible runtime compared to the overall training process.

As mentioned before, during training, we continuously evaluated the model to monitor its performance and ensure the effectiveness of the fine-tuning process. The intermediate results are shown in Figure 5 and Figure 6 in the Appendix.

6 Code Overview

Due to the large scale of this project, we implemented significant amounts of code. The code that we wrote is over 800 lines.

6.1 Dataset Generation Code

The dataset generation is built on top of the VLLM inference engine. Figure 12 shows the bash script to launch the VLLM engine to run the Qwen-2.5 32B model as an OpenAI API compatible server. Note that we have shrunk the context size to save GPU memory, and we enabled prefix caching to gain a 2x speedup as previously described.

To connect to the VLLM server with maximal code reuse, we construct a library to easily construct few shot templates and generate using it. As shown in Figure 13, it provides convenience functions to convert JSON objects, query the LLM, and perform logging.

To generate the Chinese and Reasoning dataset (although we don't use the Chinese dataset for fine-tuning, we still use it for evaluation), as shown in Figures 14, 15, we scan through the folder containing all the original prompts, and for each prompt, we generate the dataset by querying the LLM with the prompt and the few-shot template. We then save the generated dataset to a separate JSON file (so we can easily roll back if necessary). Since we parse the LLM output using JSON format, we expect a very small percentage of parse errors. In those cases, we retry for a certain number of times before giving up. The fine-tuning code is fault-tolerant to these missing datapoints.

6.2 Pipeline Generation Code

Instead of inferencing the pipeline one input image at a time, we perform the steps one by one in batches. This allows us to load only one model into GPU memory at a time, which reduces the required memory.

As shown in Figure 16, we first use the BLIP2 model to generate the image caption with the transformers library and the blip-image-captioning-large model at float16 precision. Results are saved in temporary JSON files.

Similar to the dataset generation code, the pipeline LLM code is also built on top of our same LLM inference infrastructure. As shown in Figures 17, 18, they take in the image captions and generate the "processed prompts" from the Chinese and reasoning prompts, plus the image caption. This step is coordinated by a script that scans through all inputs and calls the respective LLM operation, as shown in Figure 19.

Unlike the dataset generation LLM generation code, as described previously, we want to minimize the chance of a JSON parse error. As shown in the bottom of Figures 17, 18, we use LLM reflection techniques to ensure that the LLM output is always valid JSON. Whenever we detect an error, we further prompt the LLM with more instructions, as well as the previous result.

Finally, we generate the pipeline outputs by feeding the LLM processed prompts into the baseline diffusion model, as shown in Figure 20.

6.3 Model Fine-tuning Code

When fine-tuning (and running evaluations on) the diffusion model, we mostly use the code provided by the baseline model code. However, since our generation dataset is a fraction of the size of the original dataset, most data samples are non-existent in the index file (called seeds.json). Thus, when the dataloader tries to load the sample, it will encounter an error. To fix this, we dynamically precompute the index file based on what exists in the directory, as shown in Figure 21. Since this operation is heavily I/O bound, we parallelize the operation among many Python threads to speed up the process.

6.4 Evaluation Code

To qualitatively evaluate the results in batch, we modified lines 82-90 in instruct_pix2pix/generation.py to allow caching of the diffusion model between generations, enabling faster generation speeds. We also wrote Chinese enabled (Chinese font included in the codebase) matplotlib code to format and display the results in a grid, which can be found in visualizations/generate_comparison.py. Model checkpoint link is in the README file.

7 Timeline

This section specifies the time allocation spent on the project, and a general timeline for the project.

1. Initial overview of project and writing proposal, total time spent: **5 Hours during Nov 14 - Nov 15**
2. Reading related works and conducting literature review, total time spent: **5 Hours during Nov 15 - Nov 20**
3. Running baselines and downloading datasets, total time spent: **12 hours during Nov 15 - Nov 20**
4. Running data generation pipeline and generate data, total time spent: **15 hours during Nov 20 - Nov 28**
5. Running Fine-tuning on the dataset, total time spent: **15 Hours during Nov 28 - Dec 5**
6. Evaluation and summarization, total time spent: **15 Hours during Dec 5 - Dec 12**
7. Poster building, code review, and final report writing **10 Hours during Dec 11 - Dec 13**

Tasks	Time	Date Span
1. Overview	5 hrs	Nov 14 - Nov 15
2. Read Related Works	5 hrs	Nov 15 - Nov 20
3. Baseline Running	12 hrs	Nov 15 - Nov 20
4. Dataset Building	15 hrs	Nov 20 - Nov 28
5. Fine-tuning	15 hrs	Nov 28 - Dec 5
6. Evaluation	15 hrs	Dec 5 - Dec 12
7. Poster and Report	10 hrs	Dec 11 - Dec 13

The project spans from mid November to mid December, total of 62 Hours human work.

8 Research Log

This research log outlines how our motivation evolved, the steps taken to set up the project, and the adjustments made to refine our objectives.

8.1 Motivation Gained

The project is first inspired by InstructPix2Pix, which was trained with GPT-3 and an earlier version of the stable diffusion-generated data. Recognizing the limitations of the outdated dataset, we proposed refining it with a customized dataset, which can be used to fine-tune the model. Beyond the common evaluation on images, we aimed to apply the fine-tuned model in at some additional domains. Initially, 3D mesh editing was considered as a downstream application.

8.2 3D Editing Objective Pivoted

While exploring 3D mesh editing, we discovered a precedent study (Vachha and Haque, 2024) that was built on InstructPix2Pix. It had already addressed similar objectives with promising results. This led us to pivot our focus from applying 3D mesh editing to improving InstructPix2Pix itself.

8.3 InstructPix2Pix's Limitations

Inspired by the paper ReasonPix2Pix(Jin et al., 2024), we found that InstructPix2Pix baseline is not capable of understanding implicit prompts. However, the paper does not provide a valid open source dataset or the pipeline code to generate such dataset. Our project goal is to refine the reasoning capabilities of InstructPix2Pix by fine-tuning on a customized dataset.

We also found that InstructPix2Pix lacked support for multilingual prompts. Given our team's multilingual expertise, we decided to focus on enhancing this capability as part of our project's objectives.

8.4 Methods Determined and Conclusion Achieved

Since both issues are text-based, a natural solution was to transform the text inputs in a way that the diffusion model could better understand. The most cost effect way to achieve that is to take advantage of the common sense knowledge from a pretrained LLM model. Because many reasoning instructions require contextual information about the original image, we integrated a BLIP2 image captioning model to provide contextual information about the image.

Despite its advantages, using an LLM had some drawbacks, such as increased computational, memory, and latency overhead during inference. Moreover, it had potential robustness issues with LLM output parsing errors. Thus, we aimed to fine-tune the InstructPix2Pix model to mitigate these issues. We had the option to either fine-tune the CLIP text encoder model or the stable diffusion model. Fine-tuning the CLIP text encoder was considered but then rejected, as it could not fully utilize information from the image modality. Incorporating the BLIP2 model together with it would re-introduce overhead, losing the benefits of this approach. Thus, we chose to fine-tune the stable diffusion model. We have access to ample amount of InstructPix2Pix pretraining dataset, so fine-tuning the stable diffusion model was feasible.

After fine-tuning on reasoning dataset, Chinese prompt remained to be a challenge. Upon investigation, we discovered that the CLIP text encoder's tokenizer lacked Chinese tokens. Fine-tuning with out-of-vocabulary tokens would be ineffective. While we found that the Chinese CLIP model could be an option, it operates in a completely different embedding space from the original CLIP model. This meant that we would have to retrain InstructPix2Pix from scratch, which is infeasible in the time frame of this project and with our resources.

Based on these considerations, we concluded with three models specified the the previous sections to enhance InstructPix2Pix. This focused approach allowed us to refine InstructPix2Pix effectively while balancing feasibility and resource availability.

9 Conclusion

In conclusion, we tackled the two challenges that InstructPix2Pix failed on: handling multilingual prompts and vague prompts. To address these challenges, we introduced a pipeline model incorporating inference-time processing with LLM-based translation and reasoning, and a fine-tuned model trained on generated datasets.

Our experiments demonstrated the effectiveness of these approaches. The pipeline model with inference-time translation showed significantly improved performance on Chinese prompts, achieving results comparable to the baseline model on English prompts. The fine-tuned model outperforms the baseline model in handling vague prompts. The pipeline model for reasoning also showed promising results, which shows that caption generation and few shot instruction enhance the instruction prompt.

There are directions for future work. First, fine-tuning the CLIP model on multilingual datasets together with InstructPix2Pix could improve text-image alignment and overall model performance, particularly for non-English prompts. Second, extending the training data to be more inclusive, covering additional languages and a larger range of implicit instructions and corresponding images, could further enhance the model's adaptability.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report.

Tim Brooks, Aleksander Holynski, and Alexei A. Efros. 2023. Instructpix2pix: Learning to follow image editing instructions.

Ayaan Haque, Matthew Tancik, Alexei A. Efros, Aleksander Holynski, and Angjoo Kanazawa. 2023. Instruct-nerf2nerf: Editing 3d scenes with instructions.

Ying Jin, Pengyang Ling, Xiaoyi Dong, Pan Zhang, Jiaqi Wang, and Dahua Lin. 2024. Reasonpix2pix: Instruction reasoning dataset for advanced image editing.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models.

Weifeng Lin, Xinyu Wei, Renrui Zhang, Le Zhuo, Shitian Zhao, Siyuan Huang, Junlin Xie, Yu Qiao, Peng Gao, and Hongsheng Li. 2024. Pixwizard: Versatile image-to-image visual assistant with open-language instructions.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models.

Cyrus Vachha and Ayaan Haque. 2024. Instruct-gs2gs: Editing 3d gaussian splats with instructions.

Jiapeng Wang, Chengyu Wang, Tingfeng Cao, Jun Huang, and Lianwen Jin. 2024. Diffchat: Learning to chat with text-to-image synthesis models for interactive image creation.

An Yang, Junshu Pan, Junyang Lin, Rui Men, Yichang Zhang, Jingren Zhou, and Chang Zhou. 2023. Chinese clip: Contrastive vision-language pretraining in chinese.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report.

A Appendix

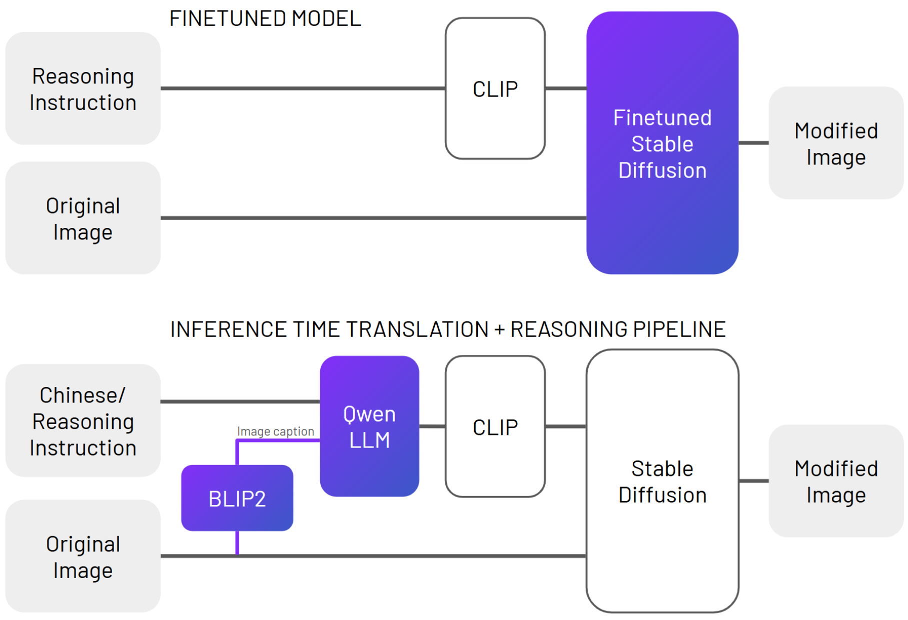


Figure 1: Model architectures for our fine-tuned model and inference time pipeline.

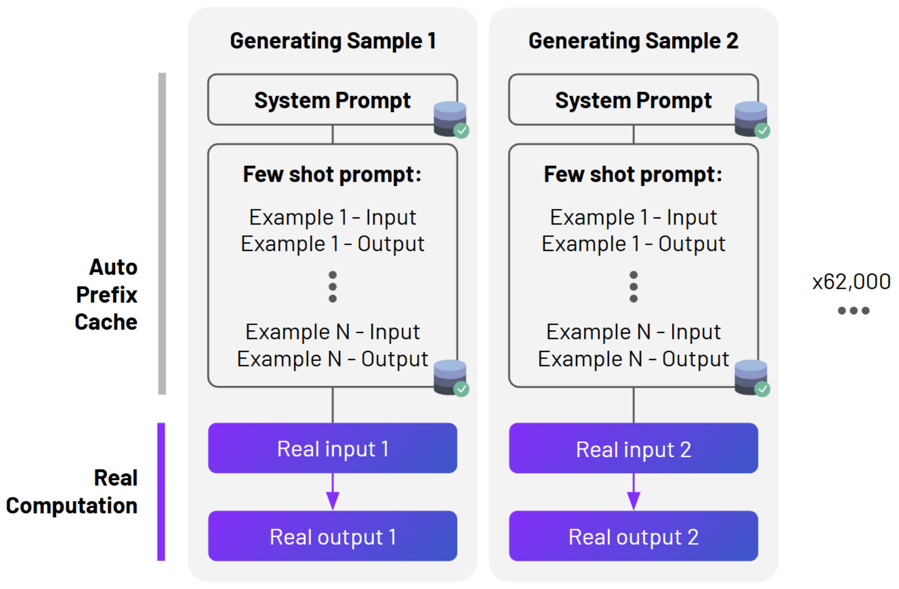


Figure 2: Model architectures for our fine-tuned model and inference time pipeline.

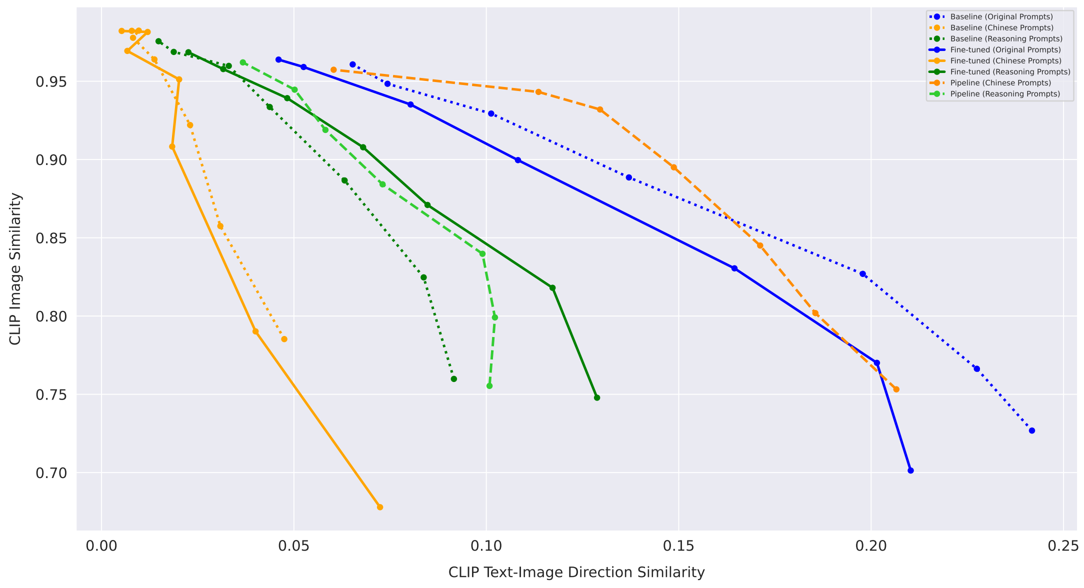


Figure 3: The performance comparison between baseline model, pipeline model (inference time translation or reasoning), and fine-tuned models across different prompts. The x-axis represents the CLIP Text-Image Direction Similarity, while the y-axis shows the CLIP Image Similarity. Fine-tuned models demonstrate better performance compared to baseline on reasoning prompts.

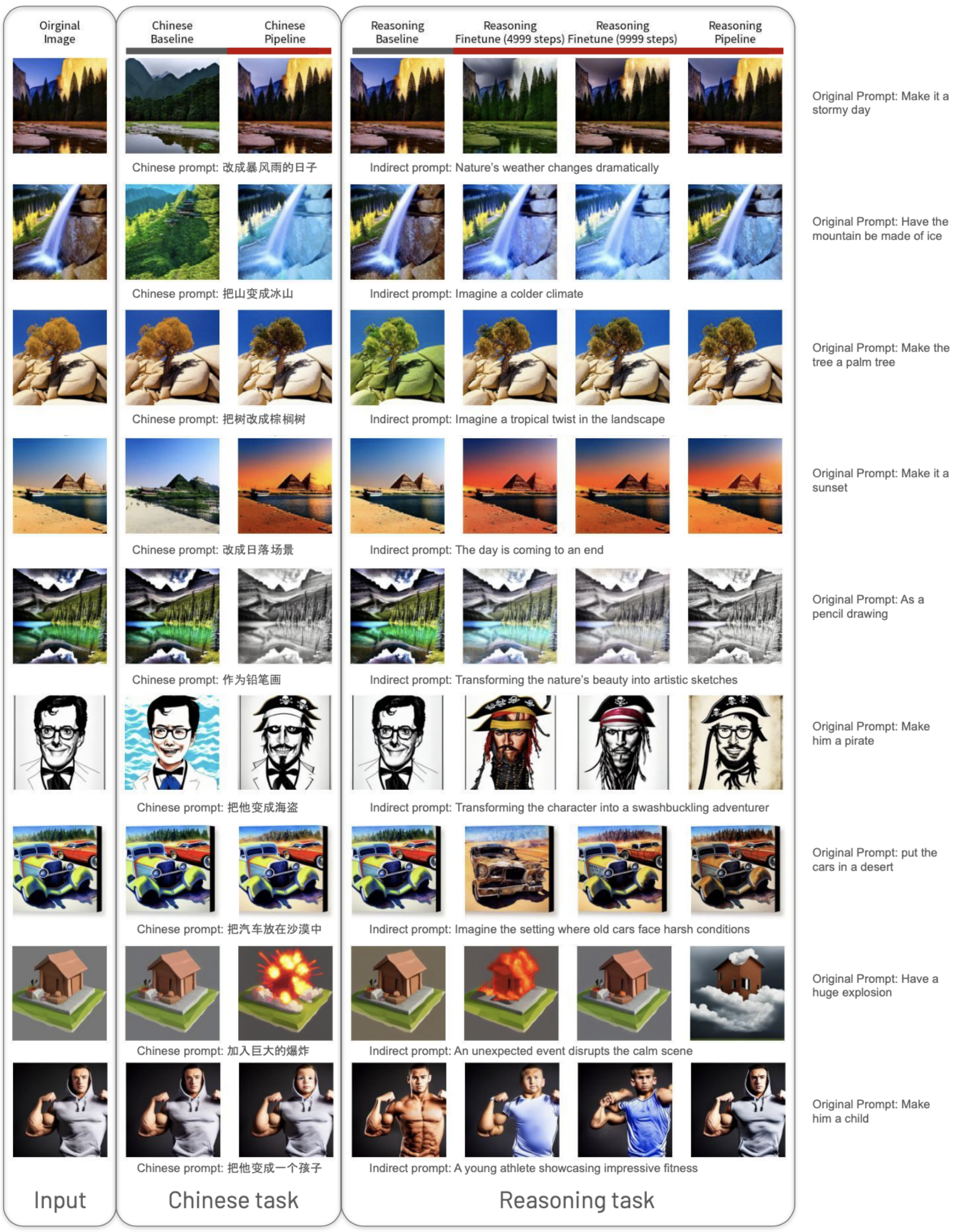

Oirginal Image
Chinese Baseline
Chinese Pipeline
Reasoning Baseline
Reasoning Finetune (4999 steps)
Reasoning Finetune (9999 steps)
Reasoning Pipeline
Chinese prompt: 改成暴风雨的日子
Indirect prompt: Nature's weather changes dramatically
Original Prompt: Make it a stormy day
Chinese prompt: 把山变成冰山
Indirect prompt: Imagine a colder climate
Original Prompt: Have the mountain be made of ice
Chinese prompt: 把树改成棕榈树
Indirect prompt: Imagine a tropical twist in the landscape
Original Prompt: Make the tree a palm tree
Chinese prompt: 改成日落场景
Indirect prompt: The day is coming to an end
Original Prompt: Make it a sunset
Chinese prompt: 作为铅笔画
Indirect prompt: Transforming the nature's beauty into artistic sketches
Original Prompt: As a pencil drawing
Chinese prompt: 把他变成海盗
Indirect prompt: Transforming the character into a swashbuckling adventurer
Original Prompt: Make him a pirate
Chinese prompt: 把汽车放在沙漠中
Indirect prompt: Imagine the setting where old cars face harsh conditions
Original Prompt: put the cars in a desert
Chinese prompt: 加入巨大的爆炸
Indirect prompt: An unexpected event disrupts the calm scene
Original Prompt: Have a huge explosion
Chinese prompt: 把他变成一个孩子
Indirect prompt: A young athlete showcasing impressive fitness
Original Prompt: Make him a child
Input
Chinese task
Reasoning task

Figure 4: Side-by-side comparisons of results generated by baseline model and fine-tuned models across different prompts.

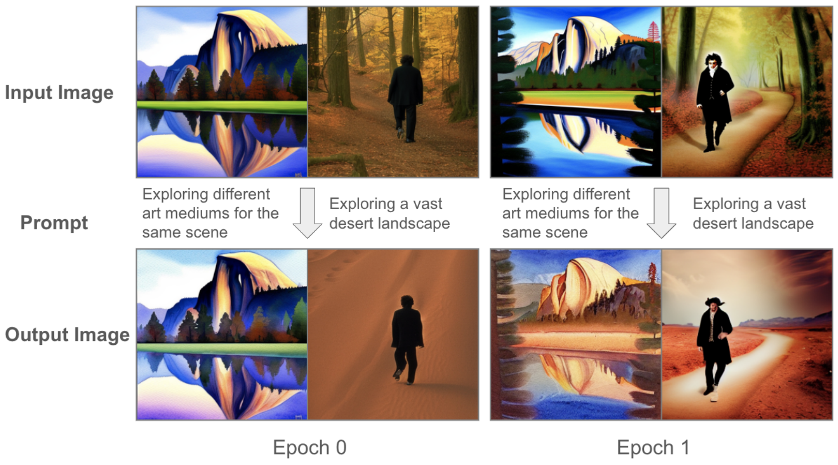

Input Image
Prompt
Output Image
Exploring different art mediums for the same scene
Exploring a vast desert landscape
Exploring different art mediums for the same scene
Exploring a vast desert landscape
Epoch 0
Epoch 1

Figure 5: **Intermediate results:** Resulting images after training on 1 shard for 0 and 1 epochs (2 hours training time).

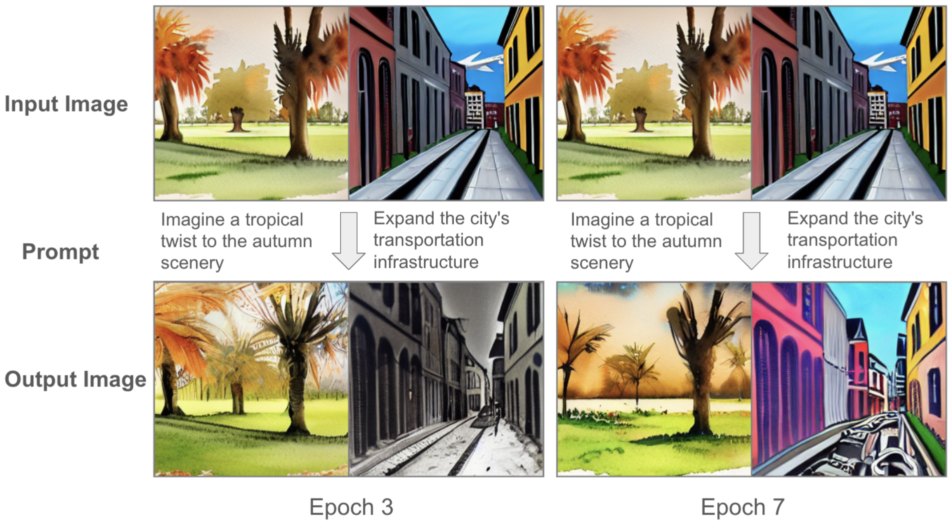

Input Image
Prompt
Output Image
Imagine a tropical twist to the autumn scenery
Expand the city's transportation infrastructure
Imagine a tropical twist to the autumn scenery
Expand the city's transportation infrastructure
Epoch 3
Epoch 7

Figure 6: **Intermediate results:** Resulting images after training on 2 shards for 3 and 7 epochs (3.5 hours training time).

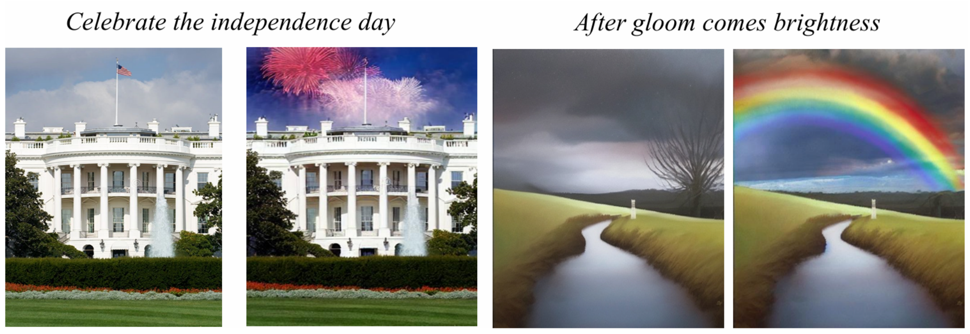

Celebrate the independence day
After gloom comes brightness

Figure 7: Reasoning image editing example (Jin et al., 2024)

```
[system]:
You are a helpful assistant who is an expert on translating English content to Chinese.
Your job is to translate the user's text into Chinese in a JSON format.

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to translate these three pieces of text into Chinese.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of
images.

Please output only the Chinese translation in a JSON format.

{
    "original": "Zhangjiajie Private 5-Day Tour: Tianmen Mountain and Fenghuang County",
    "instruction": "make it at night",
    "modified": "Zhangjiajie Private 5-Day Tour: Tianmen Mountain and Fenghuang County at night"
}

[assistant]:
{
    "original": "张家界私人 5 天游：天门山和凤凰县",
    "instruction": "改为夜间",
    "modified": "张家界私人 5 天游：天门山和凤凰县夜间"
}

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to translate these three pieces of text into Chinese.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of
images.

Please output only the Chinese translation in a JSON format.

{
    "original": "After Wedding Shooting Paris",
    "instruction": "have the couple be in a hot air balloon",
    "modified": "After Wedding Shooting from Hot Air Balloon Paris"
}

[assistant]:
{
    "original": "巴黎婚礼后拍摄",
    "instruction": "让这对夫妇在热气球里",
    "modified": "巴黎热气球上的婚礼后拍摄"
}

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to translate these three pieces of text into Chinese.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of
images.

Please output only the Chinese translation in a JSON format.

{
    "original": "Picture the sky, stars, mountains, night, Northern lights, Norway, North",
    "instruction": "make it a sunset",
    "modified": "Picture the sky, sun, mountains, sunset, Norway, North"
}
```

Figure 8: Chinese image editing dataset generation example

```
[system]:
You are a helpful assistant who is an expert on reasoning and modifying image captions.
Your job is to transform the user's instruction and return in a JSON format.

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to generate a reasoning prompt that transforms the original image caption into the modified image caption.

The reasoning prompt is different from the given instruction, because it does not directly say what to modify.
Instead, it should describe a related sentence where the given instruction would make logical sense.

Please output only the new reasoning prompt in English in a JSON format.

{
    "original": "A pair of high heels on the table",
    "instruction": "make it sneakers",
    "reasoning": "",
    "modified": "A pair of sneakers on the table"
}

[assistant]:
{
    "reasoning": "Walk in a comfortable way"
}

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to generate a reasoning prompt that transforms the original image caption into the modified image caption.

The reasoning prompt is different from the given instruction, because it does not directly say what to modify.
Instead, it should describe a related sentence where the given instruction would make logical sense.

Please output only the new reasoning prompt in English in a JSON format.

{
    "original": "A river and a dark sky",
    "instruction": "change to blue sky and add a rainbow",
    "reasoning": "",
    "modified": "A river and a blue sky with a rainbow"
}

[assistant]:
{
    "reasoning": "After gloom comes brightness"
}

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to generate a reasoning prompt that transforms the original image caption into the modified image caption.

The reasoning prompt is different from the given instruction, because it does not directly say what to modify.
Instead, it should describe a related sentence where the given instruction would make logical sense.

Please output only the new reasoning prompt in English in a JSON format.

{
    "original": "Drawing of five white flowers",
    "instruction": "draw a butterfly on one of the flowers",
    "reasoning": "",
    "modified": "Drawing of five white flowers with a butterfly on one of them"
}

[assistant]:
{
    "reasoning": "A beautiful creature is attracted by the flowers"
}
```

```
[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to generate a reasoning prompt that transforms the original image caption into the modified image caption.

The reasoning prompt is different from the given instruction, because it does not directly say what to modify.
Instead, it should describe a related sentence where the given instruction would make logical sense.

Please output only the new reasoning prompt in English in a JSON format.

{
    "original": "Picture of the white house",
    "instruction": "put fireworks in the sky",
    "reasoning": "",
    "modified": "Picture of the white house with fireworks in the sky"
}

[assistant]:
{
    "reasoning": "Celebrate the independence day"
}

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to generate a reasoning prompt that transforms the original image caption into the modified image caption.

The reasoning prompt is different from the given instruction, because it does not directly say what to modify.
Instead, it should describe a related sentence where the given instruction would make logical sense.

Please output only the new reasoning prompt in English in a JSON format.

{
    "original": "Picture the sky, stars, mountains, night, Northern lights, Norway, North",
    "instruction": "make it a sunset",
    "reasoning": "",
    "modified": "Picture the sky, sun, mountains, sunset, Norway, North"
}
```

Figure 9: Reasoning image editing dataset generation example

```
[system]:
You are a helpful assistant who is an expert on translating text from other languages into English.
Your job is to translate the user's text into English in a JSON format.

[user]:
Below you are given two pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to translate the instruction into English.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of
images.

If the instruction is already in English, then do not make any changes, just repeat the text as it is into the JSON.

Please output only the English translation and reasoning in a JSON format.

{
    "image_caption": "Zhangjiajie Private 5-Day Tour: Tianmen Mountain and Fenghuang County",
    "instruction": "改为夜间"
}

[assistant]:
{
    "translation": "make it at night"
}

[user]:
Below you are given two pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to translate the instruction into English.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of
images.

If the instruction is already in English, then do not make any changes, just repeat the text as it is into the JSON.

Please output only the English translation and reasoning in a JSON format.

{
    "image_caption": "After Wedding Shooting Paris",
    "instruction": "巴黎热气球上的婚礼后拍摄"
}

[assistant]:
{
    "translation": "have the couple be in a hot air balloon"
}

[user]:
Below you are given two pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to translate the instruction into English.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of
images.

If the instruction is already in English, then do not make any changes, just repeat the text as it is into the JSON.

Please output only the English translation and reasoning in a JSON format.

{
    "image_caption": "there are many pumpkins on display under a wooden structure",
    "instruction": "让它下雪"
}
```

Figure 10: Chinese pipeline image editing example

```
[system]:
You are a helpful assistant who is an expert on reasoning and modifying image captions.
Your job is to transform the user's instruction and return in a JSON format.

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to improve the user's instruction, but only if necessary.

A good prompt is one that is detailed, specific, and concrete.
A bad prompt is one that is vague, ambiguous, or overly general.

If and only if the user's instruction is vague, then your job is to make it more specific.
In this case, you may need to use your reasoning skills to infer the user's intent.
You need to transform the vague prompt into something that directly says what to modify.
However, your response also must be logical. Your instruction must make sense in the context of the image and the
original prompt.

Again, if the user's prompt is not vague, please do not make any changes, just repeat the user's prompt as it is into the
JSON.

Please output only your response in English in a JSON format.

{
    "image_caption": "A pair of high heels on the table",
    "instruction": "Walk in a comfortable way"
}

[assistant]:
{
    "new_instruction": "Change it to sneakers"
}

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to improve the user's instruction, but only if necessary.

A good prompt is one that is detailed, specific, and concrete.
A bad prompt is one that is vague, ambiguous, or overly general.

If and only if the user's instruction is vague, then your job is to make it more specific.
In this case, you may need to use your reasoning skills to infer the user's intent.
You need to transform the vague prompt into something that directly says what to modify.
However, your response also must be logical. Your instruction must make sense in the context of the image and the
original prompt.

Again, if the user's prompt is not vague, please do not make any changes, just repeat the user's prompt as it is into the
JSON.

Please output only your response in English in a JSON format.

{
    "image_caption": "A river and a dark sky",
    "instruction": "change to blue sky and add a rainbow"
}

[assistant]:
{
    "new_instruction": "Change to blue sky and add a rainbow"
}

[user]:
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to improve the user's instruction, but only if necessary.

A good prompt is one that is detailed, specific, and concrete.
A bad prompt is one that is vague, ambiguous, or overly general.

If and only if the user's instruction is vague, then your job is to make it more specific.
In this case, you may need to use your reasoning skills to infer the user's intent.
You need to transform the vague prompt into something that directly says what to modify.
```

```
However, your response also must be logical. Your instruction must make sense in the context of the image and the
original prompt.

Again, if the user's prompt is not vague, please do not make any changes, just repeat the user's prompt as it is into the
JSON.

Please output only your response in English in a JSON format.

{
    "image_caption": "Drawing of five white flowers",
    "instruction": "A beautiful creature is attracted by the flowers"
}
```

[assistant]:

```
{
    "new_instruction": "Draw a butterfly on one of the flowers"
}
```

[user]:

```
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to improve the user's instruction, but only if necessary.

A good prompt is one that is detailed, specific, and concrete.
A bad prompt is one that is vague, ambiguous, or overly general.

If and only if the user's instruction is vague, then your job is to make it more specific.
In this case, you may need to use your reasoning skills to infer the user's intent.
You need to transform the vague prompt into something that directly says what to modify.
However, your response also must be logical. Your instruction must make sense in the context of the image and the
original prompt.

Again, if the user's prompt is not vague, please do not make any changes, just repeat the user's prompt as it is into the
JSON.

Please output only your response in English in a JSON format.

{
    "image_caption": "Picture of the white house",
    "instruction": "Celebrate the independence day"
}
```

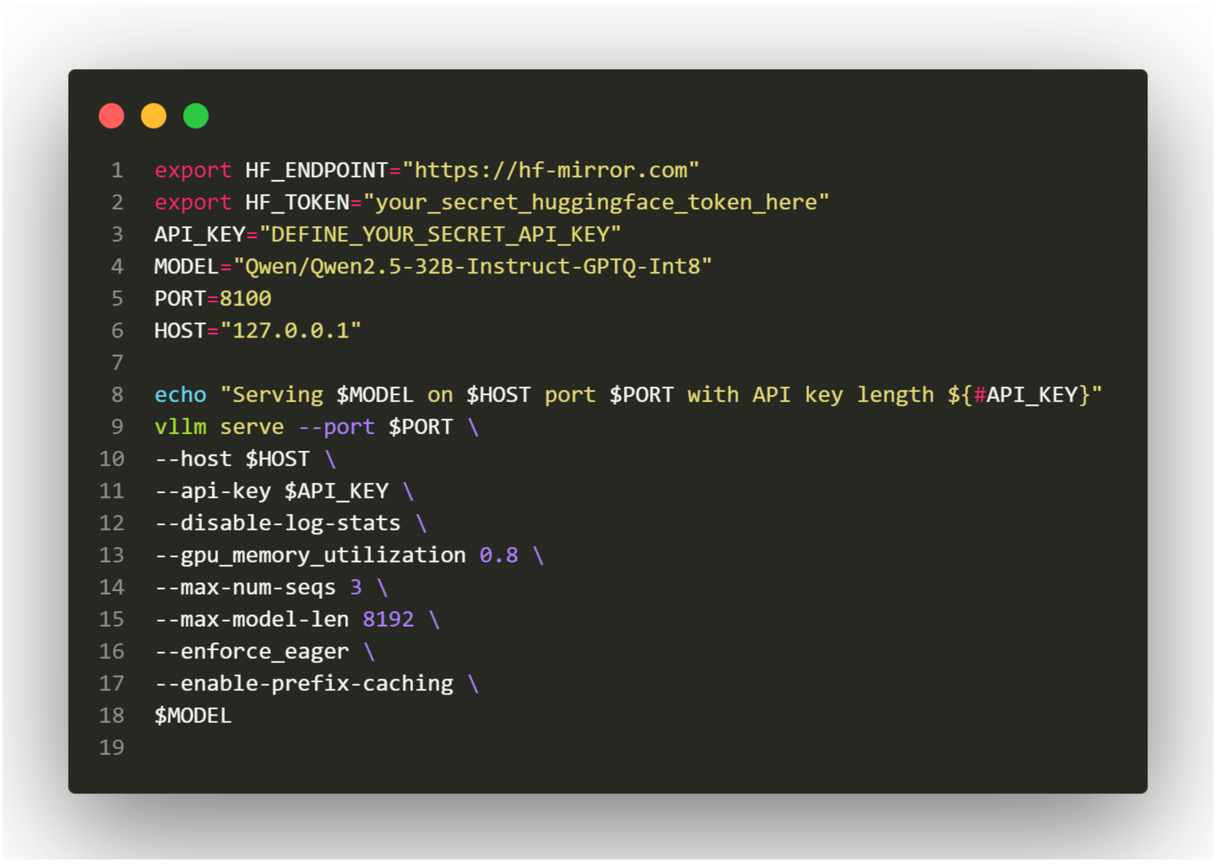
[assistant]:

```
{
    "new_instruction": "Put fireworks in the sky"
}
```

[user]:

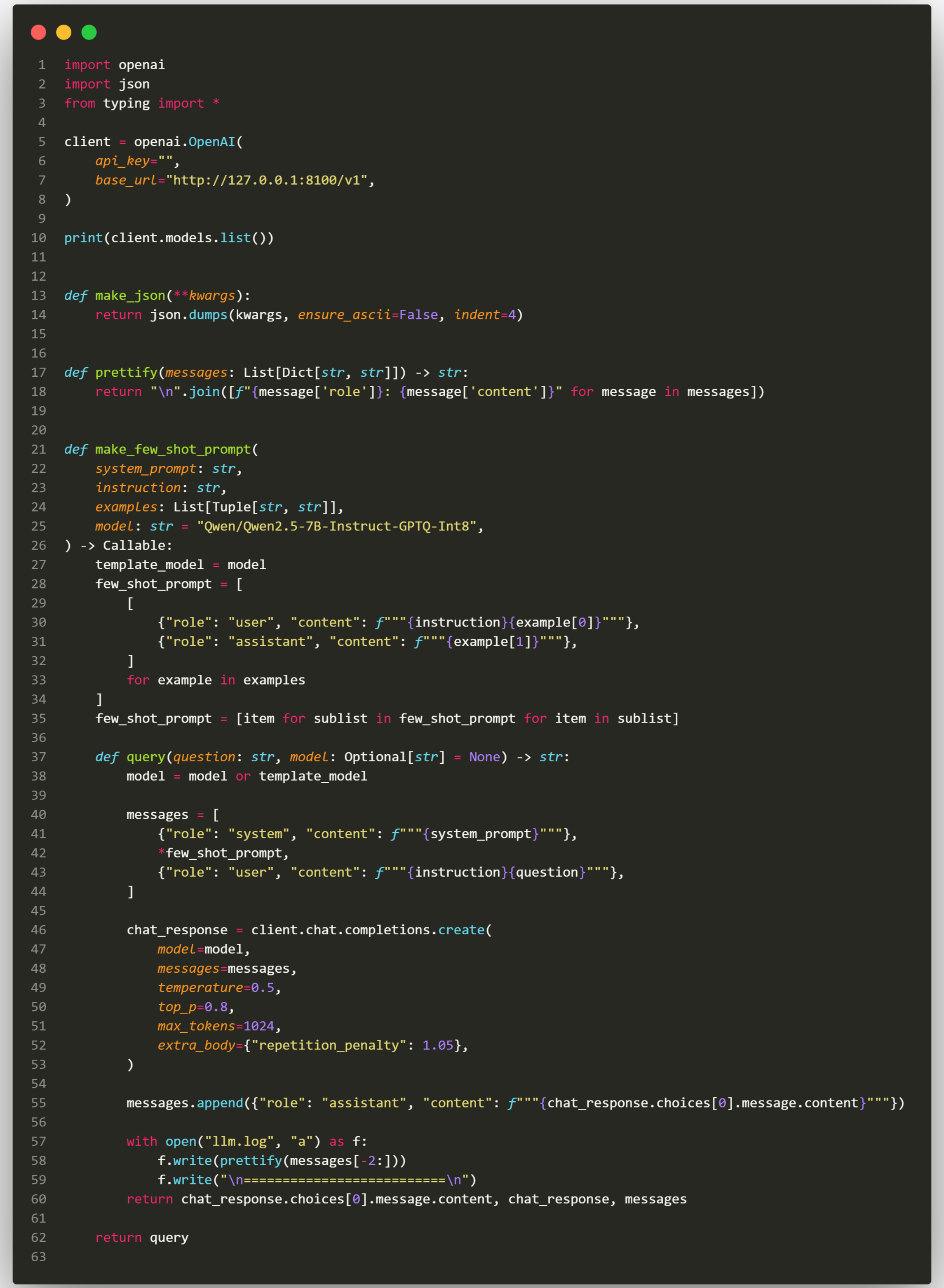
```
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to improve the user's instruction, but only if necessary.

A good prompt is one that is detailed, specific, and concrete.
A bad prompt is one that is vague, ambiguous, or overly general.

If and only if the user's instruction is vague, then your job is to make it more specific.
In this case, you may need to use your reasoning skills to infer the user's intent.
You need to transform the vague prompt into something that directly says what to modify.
However, your response also must be logical. Your instruction must make sense in the context of the image and the
original prompt.

Again, if the user's prompt is not vague, please do not make any changes, just repeat the user's prompt as it is into the
JSON.

Please output only your response in English in a JSON format.

{
    "image_caption": "there are many pumpkins on display under a wooden structure",
    "instruction": "Winter has arrived unexpectedly"
}
```

Figure 11: Reasoning pipeline image editing example



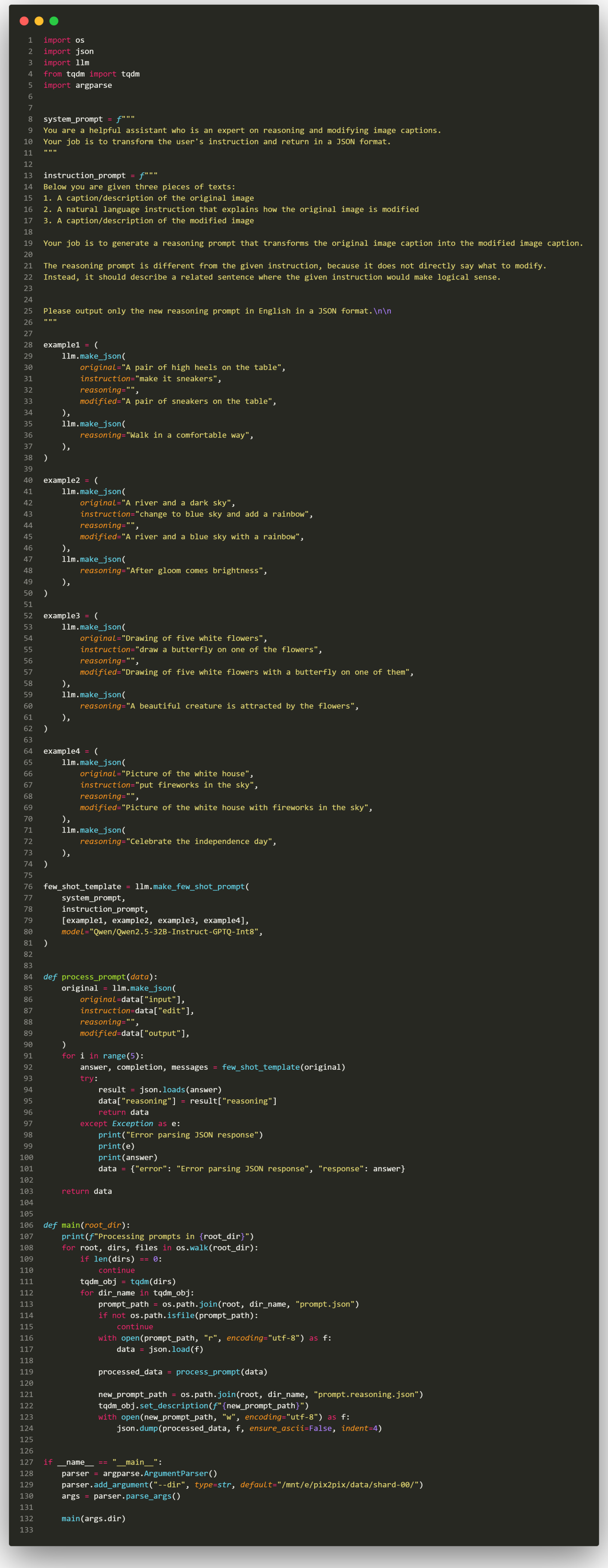
```bash
export HF_ENDPOINT="https://hf-mirror.com"
export HF_TOKEN="your_secret_huggingface_token_here"
API_KEY="DEFINE_YOUR_SECRET_API_KEY"
MODEL="Qwen/Qwen2.5-32B-Instruct-GPTQ-Int8"
PORT=8100
HOST="127.0.0.1"

echo "Serving $MODEL on $HOST port $PORT with API key length ${#API_KEY}"
vllm serve --port $PORT \
--host $HOST \
--api-key $API_KEY \
--disable-log-stats \
--gpu_memory_utilization 0.8 \
--max-num-seqs 3 \
--max-model-len 8192 \
--enforce_eager \
--enable-prefix-caching \
$MODEL
```

Figure 12: VLLM server bash code



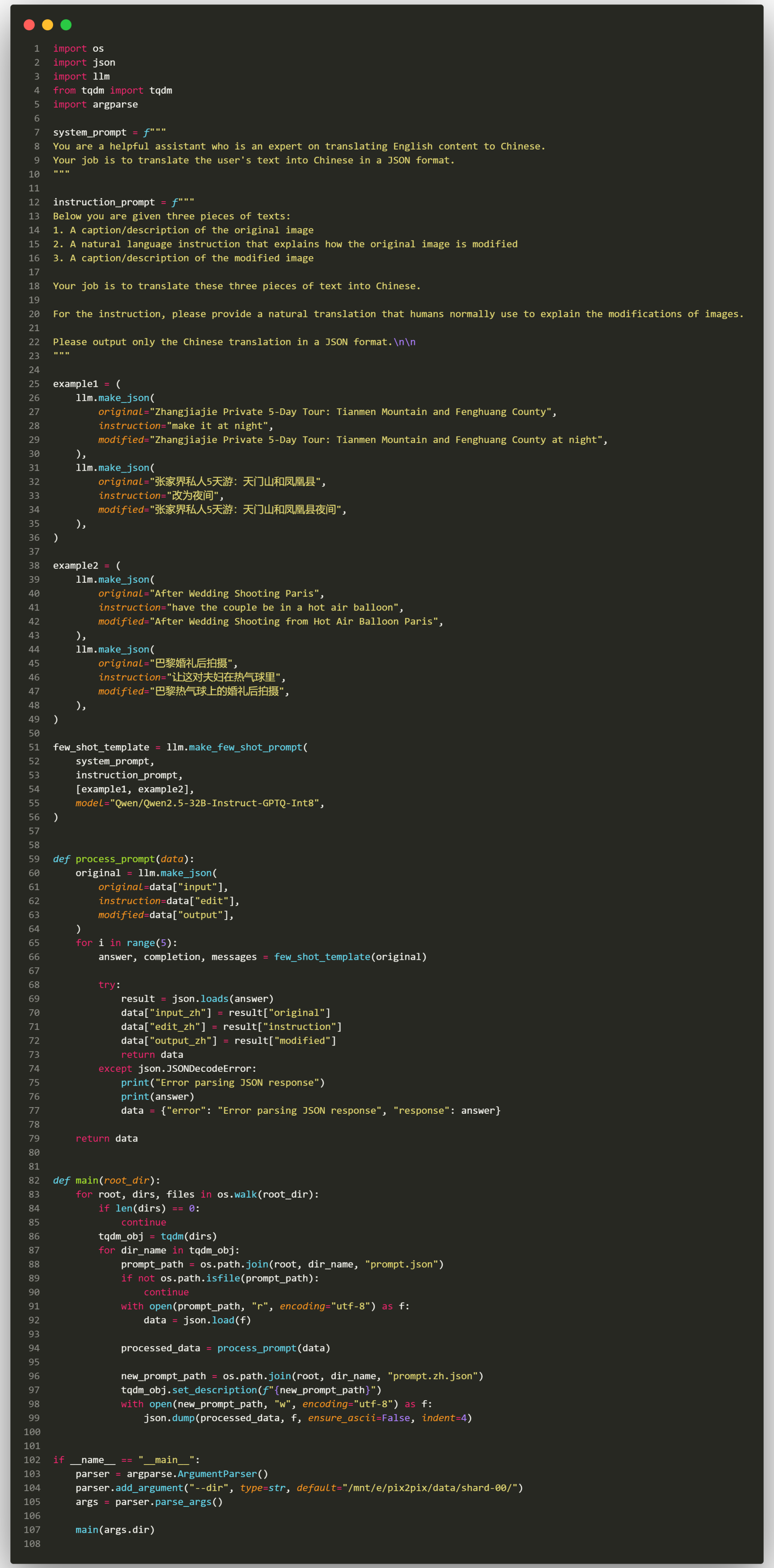
```python
import openai
import json
from typing import *

client = openai.OpenAI(
    api_key="",
    base_url="http://127.0.0.1:8100/v1",
)

print(client.models.list())


def make_json(**kwargs):
    return json.dumps(kwargs, ensure_ascii=False, indent=4)


def prettify(messages: List[Dict[str, str]]) -> str:
    return "\n".join([f"{message['role']}: {message['content']}" for message in messages])


def make_few_shot_prompt(
    system_prompt: str,
    instruction: str,
    examples: List[Tuple[str, str]],
    model: str = "Qwen/Qwen2.5-7B-Instruct-GPTQ-Int8",
) -> Callable:
    template_model = model
    few_shot_prompt = [
        [
            {"role": "user", "content": f"""{instruction}{example[0]}"""},
            {"role": "assistant", "content": f"""{example[1]}"""},
        ]
        for example in examples
    ]
    few_shot_prompt = [item for sublist in few_shot_prompt for item in sublist]

    def query(question: str, model: Optional[str] = None) -> str:
        model = model or template_model

        messages = [
            {"role": "system", "content": f"""{system_prompt}"""},
            *few_shot_prompt,
            {"role": "user", "content": f"""{instruction}{question}"""},
        ]

        chat_response = client.chat.completions.create(
            model=model,
            messages=messages,
            temperature=0.5,
            top_p=0.8,
            max_tokens=1024,
            extra_body={"repetition_penalty": 1.05},
        )

        messages.append({"role": "assistant", "content": f"""{chat_response.choices[0].message.content}"""})

        with open("llm.log", "a") as f:
            f.write(prettify(messages[-2:]))
            f.write("\n==========================\n")
        return chat_response.choices[0].message.content, chat_response, messages

    return query

```

Figure 13: Custom LLM library to facilitate few shot prompting



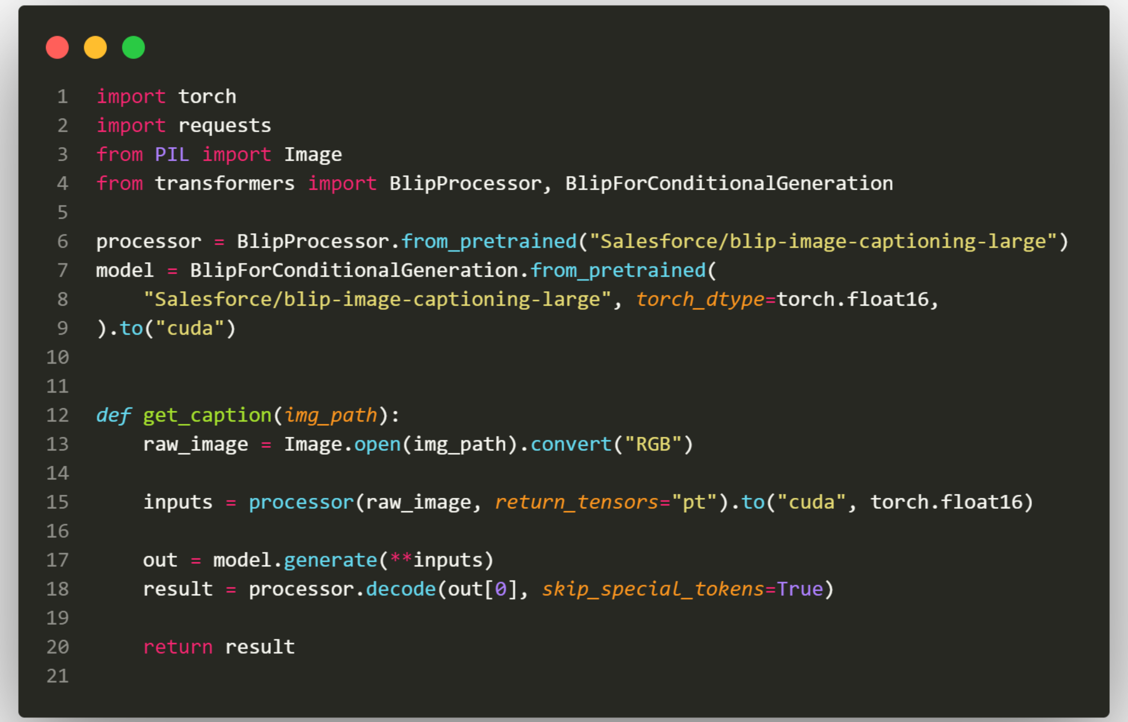
```python
import os
import json
import llm
from tqdm import tqdm
import argparse


system_prompt = f"""
You are a helpful assistant who is an expert on reasoning and modifying image captions.
Your job is to transform the user's instruction and return in a JSON format.
"""

instruction_prompt = f"""
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to generate a reasoning prompt that transforms the original image caption into the modified image caption.

The reasoning prompt is different from the given instruction, because it does not directly say what to modify.
Instead, it should describe a related sentence where the given instruction would make logical sense.


Please output only the new reasoning prompt in English in a JSON format.\n\n
"""

example1 = (
    llm.make_json(
        original="A pair of high heels on the table",
        instruction="make it sneakers",
        reasoning="",
        modified="A pair of sneakers on the table",
    ),
    llm.make_json(
        reasoning="Walk in a comfortable way",
    ),
)

example2 = (
    llm.make_json(
        original="A river and a dark sky",
        instruction="change to blue sky and add a rainbow",
        reasoning="",
        modified="A river and a blue sky with a rainbow",
    ),
    llm.make_json(
        reasoning="After gloom comes brightness",
    ),
)

example3 = (
    llm.make_json(
        original="Drawing of five white flowers",
        instruction="draw a butterfly on one of the flowers",
        reasoning="",
        modified="Drawing of five white flowers with a butterfly on one of them",
    ),
    llm.make_json(
        reasoning="A beautiful creature is attracted by the flowers",
    ),
)

example4 = (
    llm.make_json(
        original="Picture of the white house",
        instruction="put fireworks in the sky",
        reasoning="",
        modified="Picture of the white house with fireworks in the sky",
    ),
    llm.make_json(
        reasoning="Celebrate the independence day",
    ),
)

few_shot_template = llm.make_few_shot_prompt(
    system_prompt,
    instruction_prompt,
    [example1, example2, example3, example4],
    model="Qwen/Qwen2.5-32B-Instruct-GPTQ-Int8",
)


def process_prompt(data):
    original = llm.make_json(
        original=data["input"],
        instruction=data["edit"],
        reasoning="",
        modified=data["output"],
    )
    for i in range(5):
        answer, completion, messages = few_shot_template(original)
        try:
            result = json.loads(answer)
            data["reasoning"] = result["reasoning"]
            return data
        except Exception as e:
            print("Error parsing JSON response")
            print(e)
            print(answer)
            data = {"error": "Error parsing JSON response", "response": answer}

    return data


def main(root_dir):
    print(f"Processing prompts in {root_dir}")
    for root, dirs, files in os.walk(root_dir):
        if len(dirs) == 0:
            continue
        tqdm_obj = tqdm(dirs)
        for dir_name in tqdm_obj:
            prompt_path = os.path.join(root, dir_name, "prompt.json")
            if not os.path.isfile(prompt_path):
                continue
            with open(prompt_path, "r", encoding="utf-8") as f:
                data = json.load(f)

            processed_data = process_prompt(data)

            new_prompt_path = os.path.join(root, dir_name, "prompt.reasoning.json")
            tqdm_obj.set_description(f"{new_prompt_path}")
            with open(new_prompt_path, "w", encoding="utf-8") as f:
                json.dump(processed_data, f, ensure_ascii=False, indent=4)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--dir", type=str, default="/mnt/e/pix2pix/data/shard-00/")
    args = parser.parse_args()

    main(args.dir)

```

Figure 14: Reasoning dataset generation code



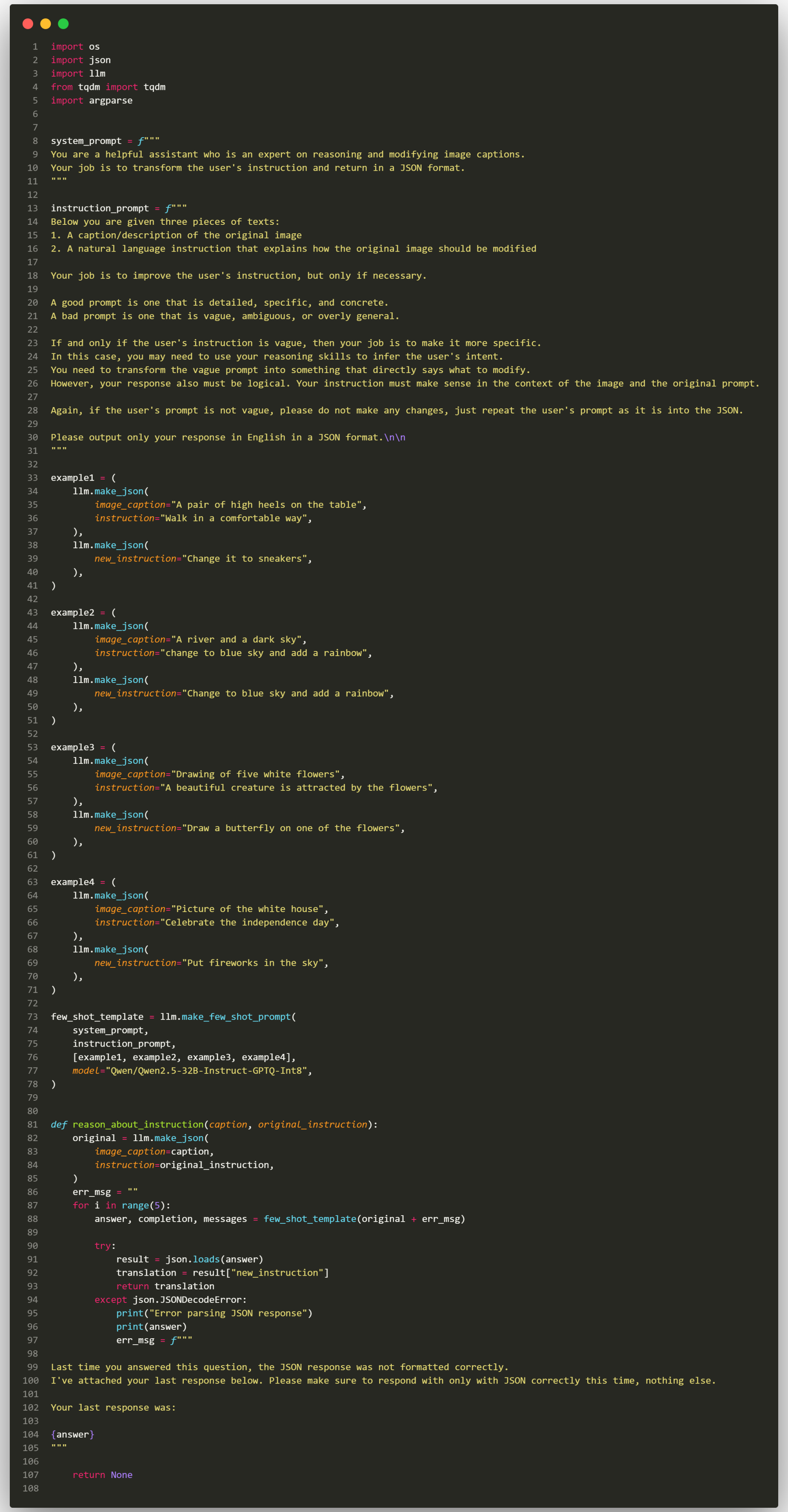
```python
import os
import json
import llm
from tqdm import tqdm
import argparse

system_prompt = f"""
You are a helpful assistant who is an expert on translating English content to Chinese.
Your job is to translate the user's text into Chinese in a JSON format.
"""

instruction_prompt = f"""
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image is modified
3. A caption/description of the modified image

Your job is to translate these three pieces of text into Chinese.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of images.

Please output only the Chinese translation in a JSON format.\n\n
"""

example1 = (
    llm.make_json(
        original="Zhangjiajie Private 5-Day Tour: Tianmen Mountain and Fenghuang County",
        instruction="make it at night",
        modified="Zhangjiajie Private 5-Day Tour: Tianmen Mountain and Fenghuang County at night",
    ),
    llm.make_json(
        original="张家界私人5天游：天门山和凤凰县",
        instruction="改为夜间",
        modified="张家界私人5天游：天门山和凤凰县夜间",
    ),
)

example2 = (
    llm.make_json(
        original="After Wedding Shooting Paris",
        instruction="have the couple be in a hot air balloon",
        modified="After Wedding Shooting from Hot Air Balloon Paris",
    ),
    llm.make_json(
        original="巴黎婚礼后拍摄",
        instruction="让这对夫妇在热气球里",
        modified="巴黎热气球上的婚礼后拍摄",
    ),
)

few_shot_template = llm.make_few_shot_prompt(
    system_prompt,
    instruction_prompt,
    [example1, example2],
    model="Qwen/Qwen2.5-32B-Instruct-GPTQ-Int8",
)


def process_prompt(data):
    original = llm.make_json(
        original=data["input"],
        instruction=data["edit"],
        modified=data["output"],
    )
    for i in range(5):
        answer, completion, messages = few_shot_template(original)

        try:
            result = json.loads(answer)
            data["input_zh"] = result["original"]
            data["edit_zh"] = result["instruction"]
            data["output_zh"] = result["modified"]
            return data
        except json.JSONDecodeError:
            print("Error parsing JSON response")
            print(answer)
            data = {"error": "Error parsing JSON response", "response": answer}

    return data


def main(root_dir):
    for root, dirs, files in os.walk(root_dir):
        if len(dirs) == 0:
            continue
        tqdm_obj = tqdm(dirs)
        for dir_name in tqdm_obj:
            prompt_path = os.path.join(root, dir_name, "prompt.json")
            if not os.path.isfile(prompt_path):
                continue
            with open(prompt_path, "r", encoding="utf-8") as f:
                data = json.load(f)

            processed_data = process_prompt(data)

            new_prompt_path = os.path.join(root, dir_name, "prompt.zh.json")
            tqdm_obj.set_description(f"{new_prompt_path}")
            with open(new_prompt_path, "w", encoding="utf-8") as f:
                json.dump(processed_data, f, ensure_ascii=False, indent=4)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--dir", type=str, default="/mnt/e/pix2pix/data/shard-00/")
    args = parser.parse_args()

    main(args.dir)
```

Figure 15: Chinese dataset generation code



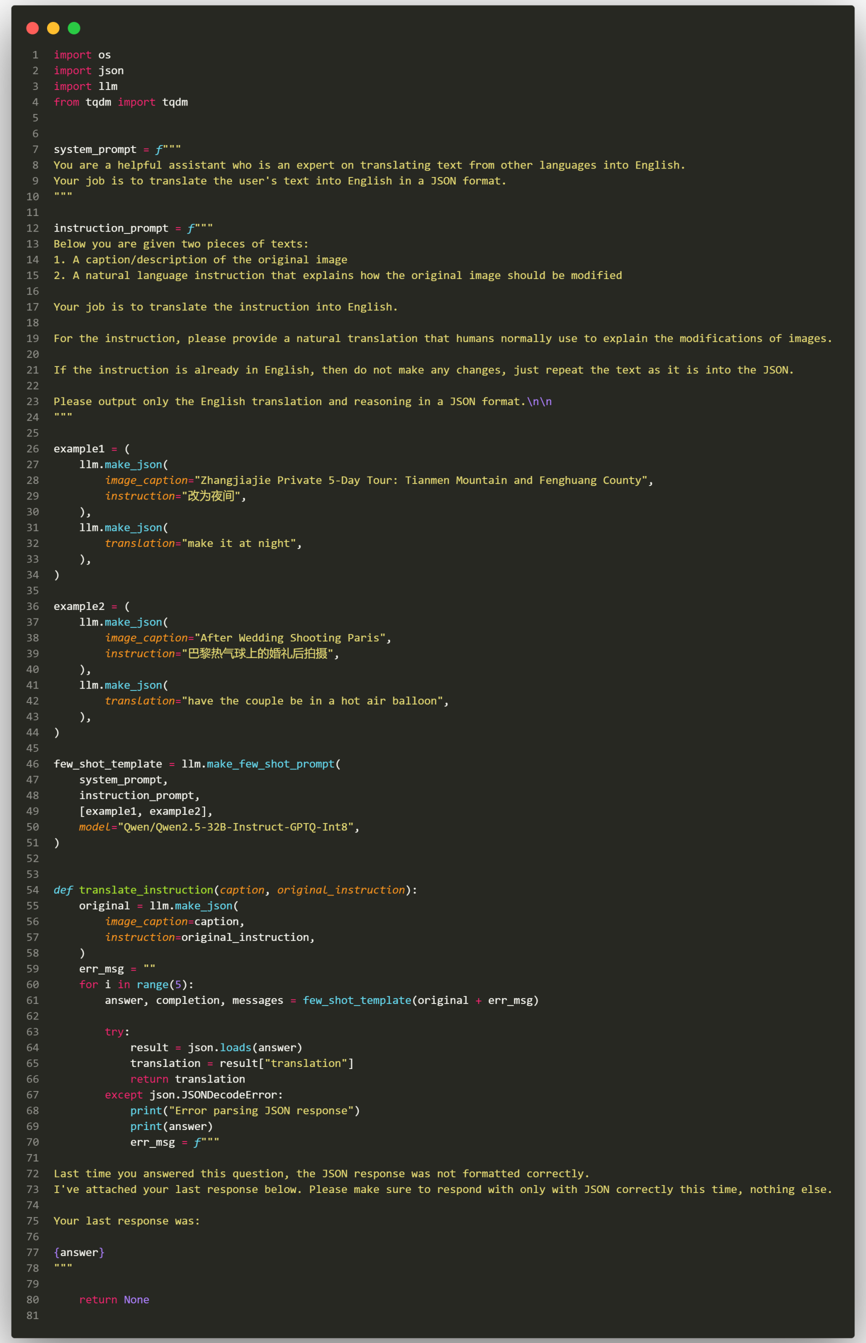
```python
import torch
import requests
from PIL import Image
from transformers import BlipProcessor, BlipForConditionalGeneration

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-large")
model = BlipForConditionalGeneration.from_pretrained(
    "Salesforce/blip-image-captioning-large", torch_dtype=torch.float16,
).to("cuda")


def get_caption(img_path):
    raw_image = Image.open(img_path).convert("RGB")

    inputs = processor(raw_image, return_tensors="pt").to("cuda", torch.float16)

    out = model.generate(**inputs)
    result = processor.decode(out[0], skip_special_tokens=True)

    return result

```

Figure 16: Inference time BLIP2 captioning code



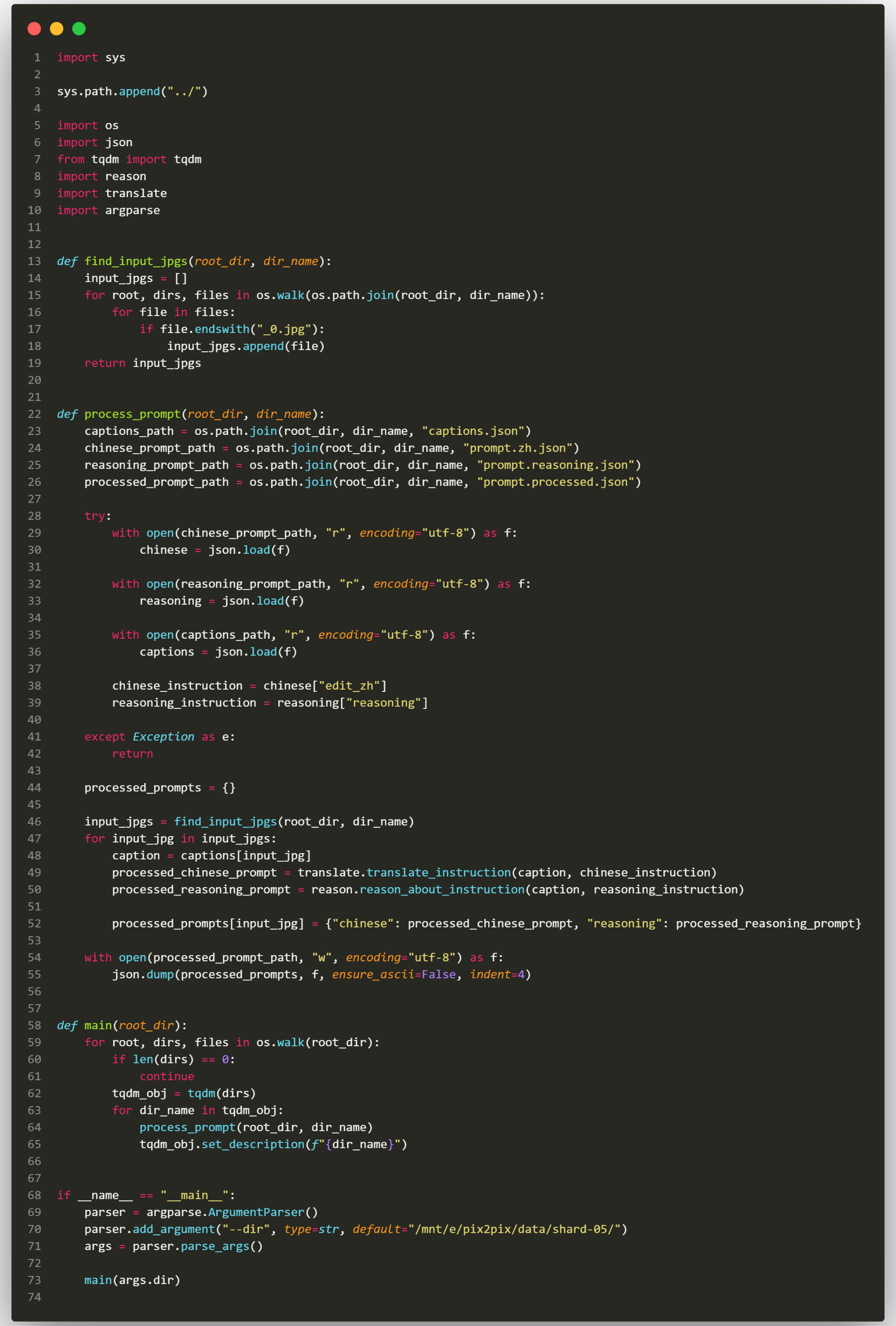
```python
import os
import json
import llm
from tqdm import tqdm
import argparse


system_prompt = f"""
You are a helpful assistant who is an expert on reasoning and modifying image captions.
Your job is to transform the user's instruction and return in a JSON format.
"""

instruction_prompt = f"""
Below you are given three pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to improve the user's instruction, but only if necessary.

A good prompt is one that is detailed, specific, and concrete.
A bad prompt is one that is vague, ambiguous, or overly general.

If and only if the user's instruction is vague, then your job is to make it more specific.
In this case, you may need to use your reasoning skills to infer the user's intent.
You need to transform the vague prompt into something that directly says what to modify.
However, your response also must be logical. Your instruction must make sense in the context of the image and the original prompt.

Again, if the user's prompt is not vague, please do not make any changes, just repeat the user's prompt as it is into the JSON.

Please output only your response in English in a JSON format.\n\n
"""

example1 = (
    llm.make_json(
        image_caption="A pair of high heels on the table",
        instruction="Walk in a comfortable way",
    ),
    llm.make_json(
        new_instruction="Change it to sneakers",
    ),
)

example2 = (
    llm.make_json(
        image_caption="A river and a dark sky",
        instruction="change to blue sky and add a rainbow",
    ),
    llm.make_json(
        new_instruction="Change to blue sky and add a rainbow",
    ),
)

example3 = (
    llm.make_json(
        image_caption="Drawing of five white flowers",
        instruction="A beautiful creature is attracted by the flowers",
    ),
    llm.make_json(
        new_instruction="Draw a butterfly on one of the flowers",
    ),
)

example4 = (
    llm.make_json(
        image_caption="Picture of the white house",
        instruction="Celebrate the independence day",
    ),
    llm.make_json(
        new_instruction="Put fireworks in the sky",
    ),
)

few_shot_template = llm.make_few_shot_prompt(
    system_prompt,
    instruction_prompt,
    [example1, example2, example3, example4],
    model="Qwen/Qwen2.5-32B-Instruct-GPTQ-Int8",
)


def reason_about_instruction(caption, original_instruction):
    original = llm.make_json(
        image_caption=caption,
        instruction=original_instruction,
    )
    err_msg = ""
    for i in range(5):
        answer, completion, messages = few_shot_template(original + err_msg)

        try:
            result = json.loads(answer)
            translation = result["new_instruction"]
            return translation
        except json.JSONDecodeError:
            print("Error parsing JSON response")
            print(answer)
            err_msg = f"""

Last time you answered this question, the JSON response was not formatted correctly.
I've attached your last response below. Please make sure to respond with only with JSON correctly this time, nothing else.

Your last response was:

{answer}
"""

    return None

```

Figure 17: Inference time LLM reasoning code



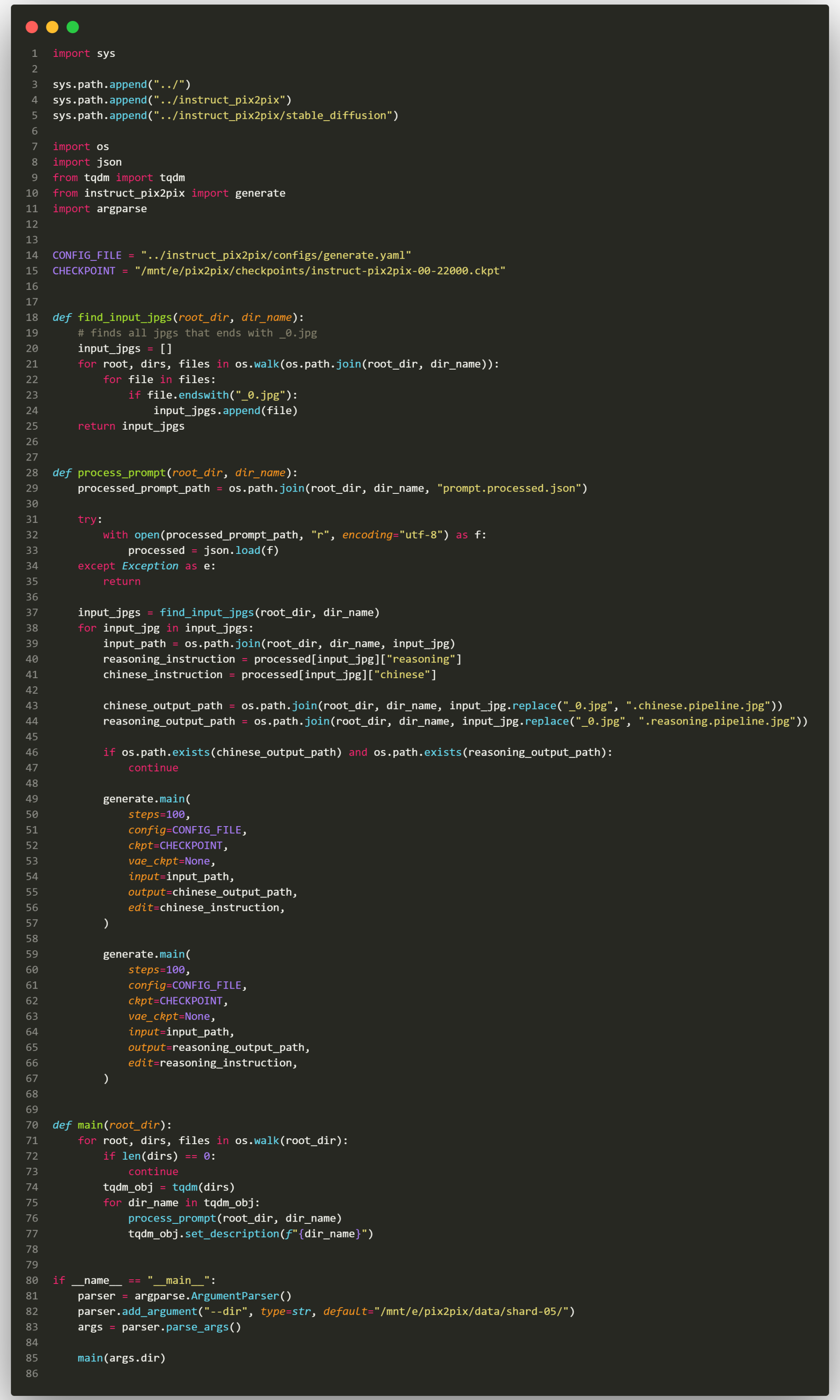
```python
import os
import json
import llm
from tqdm import tqdm


system_prompt = f"""
You are a helpful assistant who is an expert on translating text from other languages into English.
Your job is to translate the user's text into English in a JSON format.
"""

instruction_prompt = f"""
Below you are given two pieces of texts:
1. A caption/description of the original image
2. A natural language instruction that explains how the original image should be modified

Your job is to translate the instruction into English.

For the instruction, please provide a natural translation that humans normally use to explain the modifications of images.

If the instruction is already in English, then do not make any changes, just repeat the text as it is into the JSON.

Please output only the English translation and reasoning in a JSON format.\n\n
"""

example1 = (
    llm.make_json(
        image_caption="Zhangjiajie Private 5-Day Tour: Tianmen Mountain and Fenghuang County",
        instruction="改为夜间",
    ),
    llm.make_json(
        translation="make it at night",
    ),
)

example2 = (
    llm.make_json(
        image_caption="After Wedding Shooting Paris",
        instruction="巴黎热气球上的婚礼后拍摄",
    ),
    llm.make_json(
        translation="have the couple be in a hot air balloon",
    ),
)

few_shot_template = llm.make_few_shot_prompt(
    system_prompt,
    instruction_prompt,
    [example1, example2],
    model="Qwen/Qwen2.5-32B-Instruct-GPTQ-Int8",
)


def translate_instruction(caption, original_instruction):
    original = llm.make_json(
        image_caption=caption,
        instruction=original_instruction,
    )
    err_msg = ""
    for i in range(5):
        answer, completion, messages = few_shot_template(original + err_msg)

        try:
            result = json.loads(answer)
            translation = result["translation"]
            return translation
        except json.JSONDecodeError:
            print("Error parsing JSON response")
            print(answer)
            err_msg = f"""

Last time you answered this question, the JSON response was not formatted correctly.
I've attached your last response below. Please make sure to respond with only with JSON correctly this time, nothing else.

Your last response was:

{answer}
"""

    return None
```

Figure 18: Inference time LLM translation code



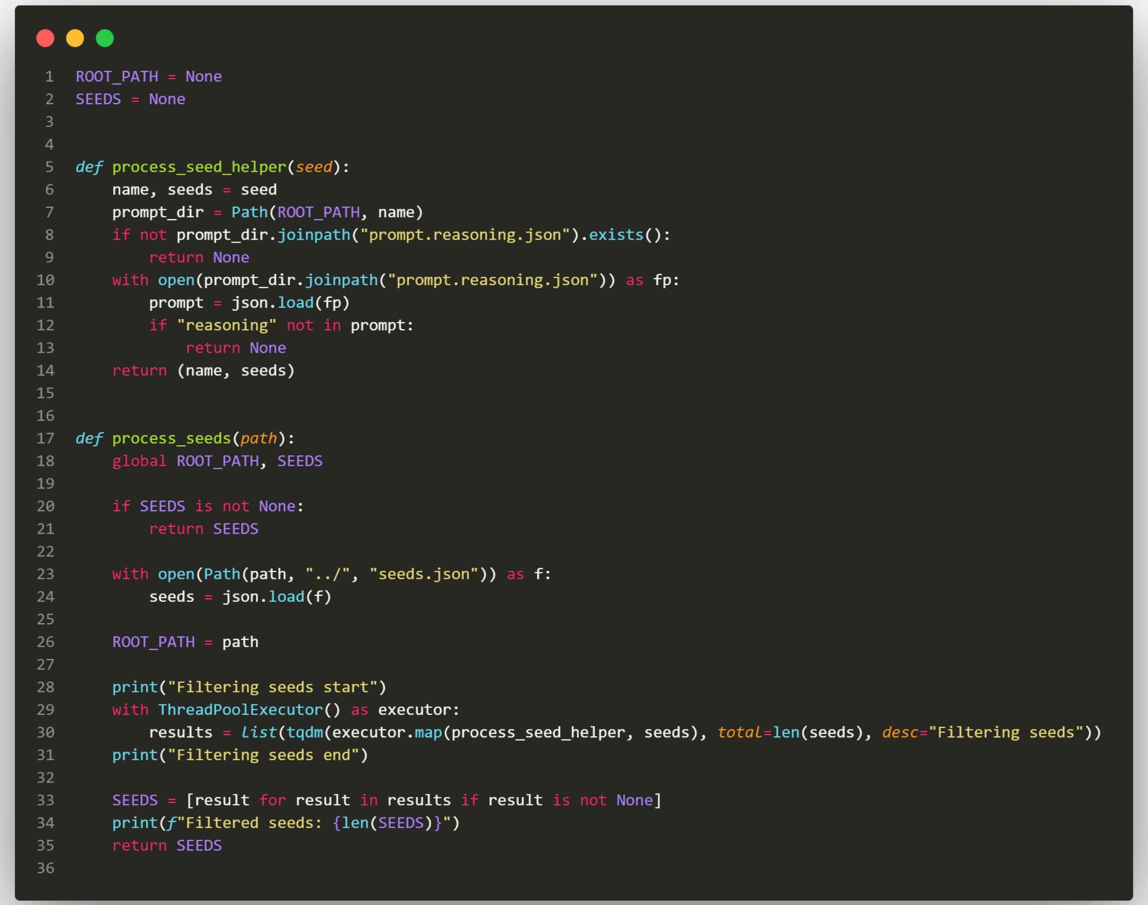
```python
import sys

sys.path.append("../")

import os
import json
from tqdm import tqdm
import reason
import translate
import argparse


def find_input_jpgs(root_dir, dir_name):
    input_jpgs = []
    for root, dirs, files in os.walk(os.path.join(root_dir, dir_name)):
        for file in files:
            if file.endswith("_0.jpg"):
                input_jpgs.append(file)
    return input_jpgs


def process_prompt(root_dir, dir_name):
    captions_path = os.path.join(root_dir, dir_name, "captions.json")
    chinese_prompt_path = os.path.join(root_dir, dir_name, "prompt.zh.json")
    reasoning_prompt_path = os.path.join(root_dir, dir_name, "prompt.reasoning.json")
    processed_prompt_path = os.path.join(root_dir, dir_name, "prompt.processed.json")

    try:
        with open(chinese_prompt_path, "r", encoding="utf-8") as f:
            chinese = json.load(f)

        with open(reasoning_prompt_path, "r", encoding="utf-8") as f:
            reasoning = json.load(f)

        with open(captions_path, "r", encoding="utf-8") as f:
            captions = json.load(f)

        chinese_instruction = chinese["edit_zh"]
        reasoning_instruction = reasoning["reasoning"]

    except Exception as e:
        return

    processed_prompts = {}

    input_jpgs = find_input_jpgs(root_dir, dir_name)
    for input_jpg in input_jpgs:
        caption = captions[input_jpg]
        processed_chinese_prompt = translate.translate_instruction(caption, chinese_instruction)
        processed_reasoning_prompt = reason.reason_about_instruction(caption, reasoning_instruction)

        processed_prompts[input_jpg] = {"chinese": processed_chinese_prompt, "reasoning": processed_reasoning_prompt}

    with open(processed_prompt_path, "w", encoding="utf-8") as f:
        json.dump(processed_prompts, f, ensure_ascii=False, indent=4)


def main(root_dir):
    for root, dirs, files in os.walk(root_dir):
        if len(dirs) == 0:
            continue
        tqdm_obj = tqdm(dirs)
        for dir_name in tqdm_obj:
            process_prompt(root_dir, dir_name)
            tqdm_obj.set_description(f"{dir_name}")


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--dir", type=str, default="/mnt/e/pix2pix/data/shard-05/")
    args = parser.parse_args()

    main(args.dir)
```

Figure 19: Inference time LLM processed prompt batch generation code



```python
import sys

sys.path.append("../")
sys.path.append("../instruct_pix2pix")
sys.path.append("../instruct_pix2pix/stable_diffusion")

import os
import json
from tqdm import tqdm
from instruct_pix2pix import generate
import argparse


CONFIG_FILE = "../instruct_pix2pix/configs/generate.yaml"
CHECKPOINT = "/mnt/e/pix2pix/checkpoints/instruct-pix2pix-00-22000.ckpt"


def find_input_jpgs(root_dir, dir_name):
    # finds all jpgs that ends with _0.jpg
    input_jpgs = []
    for root, dirs, files in os.walk(os.path.join(root_dir, dir_name)):
        for file in files:
            if file.endswith("_0.jpg"):
                input_jpgs.append(file)
    return input_jpgs


def process_prompt(root_dir, dir_name):
    processed_prompt_path = os.path.join(root_dir, dir_name, "prompt.processed.json")

    try:
        with open(processed_prompt_path, "r", encoding="utf-8") as f:
            processed = json.load(f)
    except Exception as e:
        return

    input_jpgs = find_input_jpgs(root_dir, dir_name)
    for input_jpg in input_jpgs:
        input_path = os.path.join(root_dir, dir_name, input_jpg)
        reasoning_instruction = processed[input_jpg]["reasoning"]
        chinese_instruction = processed[input_jpg]["chinese"]

        chinese_output_path = os.path.join(root_dir, dir_name, input_jpg.replace("_0.jpg", ".chinese.pipeline.jpg"))
        reasoning_output_path = os.path.join(root_dir, dir_name, input_jpg.replace("_0.jpg", ".reasoning.pipeline.jpg"))

        if os.path.exists(chinese_output_path) and os.path.exists(reasoning_output_path):
            continue

        generate.main(
            steps=100,
            config=CONFIG_FILE,
            ckpt=CHECKPOINT,
            vae_ckpt=None,
            input=input_path,
            output=chinese_output_path,
            edit=chinese_instruction,
        )

        generate.main(
            steps=100,
            config=CONFIG_FILE,
            ckpt=CHECKPOINT,
            vae_ckpt=None,
            input=input_path,
            output=reasoning_output_path,
            edit=reasoning_instruction,
        )


def main(root_dir):
    for root, dirs, files in os.walk(root_dir):
        if len(dirs) == 0:
            continue
        tqdm_obj = tqdm(dirs)
        for dir_name in tqdm_obj:
            process_prompt(root_dir, dir_name)
            tqdm_obj.set_description(f"{dir_name}")


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--dir", type=str, default="/mnt/e/pix2pix/data/shard-05/")
    args = parser.parse_args()

    main(args.dir)
```

Figure 20: Diffusion batch generation code on LLM processed prompts for pipeline model



```python
ROOT_PATH = None
SEEDS = None


def process_seed_helper(seed):
    name, seeds = seed
    prompt_dir = Path(ROOT_PATH, name)
    if not prompt_dir.joinpath("prompt.reasoning.json").exists():
        return None
    with open(prompt_dir.joinpath("prompt.reasoning.json")) as fp:
        prompt = json.load(fp)
        if "reasoning" not in prompt:
            return None
    return (name, seeds)


def process_seeds(path):
    global ROOT_PATH, SEEDS

    if SEEDS is not None:
        return SEEDS

    with open(Path(path, "../", "seeds.json")) as f:
        seeds = json.load(f)

    ROOT_PATH = path

    print("Filtering seeds start")
    with ThreadPoolExecutor() as executor:
        results = list(tqdm(executor.map(process_seed_helper, seeds), total=len(seeds), desc="Filtering seeds"))
    print("Filtering seeds end")

    SEEDS = [result for result in results if result is not None]
    print(f"Filtered seeds: {len(SEEDS)}")
    return SEEDS

```

Figure 21: Instruct-pix2pix fine-tuning dataset loader code