Fantastic LLMs and How to Train/Run Them

By Pete

Feb 19, 2024

Intelligent Computer Entertainment

RITSUMEIKAN

Agenda

Train Them

- **`transformers` for Training**
- **Google Colab**
- **Kaggle Notebook**

Run Them

- **Full-Size Models**
  - `transformers` for Inference
- **Quantized Models**
  - LM Studio
  - Ollama
- **API**
  - Gemini Pro

Intelligent Computer Entertainment

Intelligent Computer Entertainment

Train Them!

Intelligent Computer Entertainment

A Very Brief Introduction to `transformers`

Train Them!

What is 🤗 Hugging Face's `transformers`?

- High-level APIs for working with SOTA pre-trained models
  - Specifically, Transformer-based models, e.g., ViT, AST, and ViViT
- Optimized and works with TensorFlow, PyTorch, and Flax
  - Able to opt into a lower level at any time!
- Huge ecosystems!
  - `tokenizers`: for tokenization
  - `datasets`: for optimized data loading and works well with `transformers`
  - `accelerate`: for automatically choosing optimal accelerators, both training and inference with `device_map = 'auto'`

ICE
Intelligent Computer Entertainment

`transformers`: Fine-Tuning A Pre-Trained Model

Step 1: Prepare your dataset

• Ideally, prepare it with `datasets`

Step 2: Choose and download a pre-trained model

• Check available models on Hugging Face Hub

Step 3: Configure hyperparameters

• Check available hyperparameters on this page

Step 4: Prepare evaluation metrics

• For typical metrics, use Hugging Face's `evaluate`

Step 5: Train it

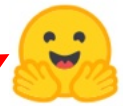
`transformers`: Fine-Tuning A Pre-Trained Model

⚠️ Caution!

Install all **required** dependencies first,
including PyTorch, TensorFlow, or Flax

`transformers`: Fine-Tuning A Pre-Trained Model

Step 1: Prepare your dataset



Accelerate

```
import numpy as np
import evaluate
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification,
       TrainingArguments, Trainer

# Load the dataset
dataset = load_dataset("yelp_review_full")
# Prepare the tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
```

`transformers`: Fine-Tuning A Pre-Trained Model

Step 1: Prepare your dataset

```
# How do you want to tokenize a string: padding, truncation, etc.
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length",
truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)

small_train_dataset =
      tokenized_datasets["train"].shuffle(seed=42).select(range(1000))

small_eval_dataset =
      tokenized_datasets["test"].shuffle(seed=42).select(range(1000))
```

`transformers`: Fine-Tuning A Pre-Trained Model

Step 2: Choose and download a pre-trained model

Step 3: Configure hyperparameters

```
# Download the pre-trained model
model = AutoModelForSequenceClassification.from_pretrained(
       "bert-base-cased", num_labels=5)

# Prepare hyperparameters
training_args = TrainingArguments(output_dir="test_trainer”,
       evaluation_strategy="epoch")
```

`transformers`: Fine-Tuning A Pre-Trained Model

Step 4: Prepare evaluation metrics

```
# Select an evaluation metric
metric = evaluate.load("accuracy")

# Define how to evaluate each (epoch|step)
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

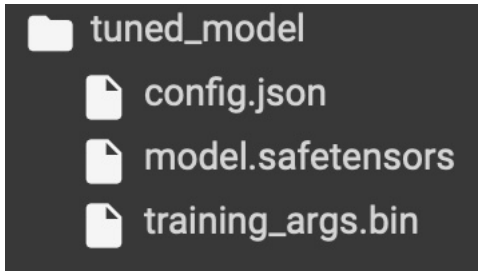
`transformers`: Fine-Tuning A Pre-Trained Model

Step 5: Train it

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
)

trainer.train()

trainer.save_model("tuned_model")
```



`transformers`: Fine-Tuning A Pre-Trained Model

Resources

- Full tutorial: https://huggingface.co/docs/transformers/en/training
- Opt for lower-level interfaces by
  - **TensorFlow**: simply import the model class with prefix "TF"
  - **PyTorch**: do not use the Trainer interface and create your own training loop
- Build a custom model by simply extending the `PreTrainedModel` abstract class
- How to prepare your own dataset using `datasets`: https://huggingface.co/docs/datasets/en/create_dataset

Intelligent Computer Entertainment

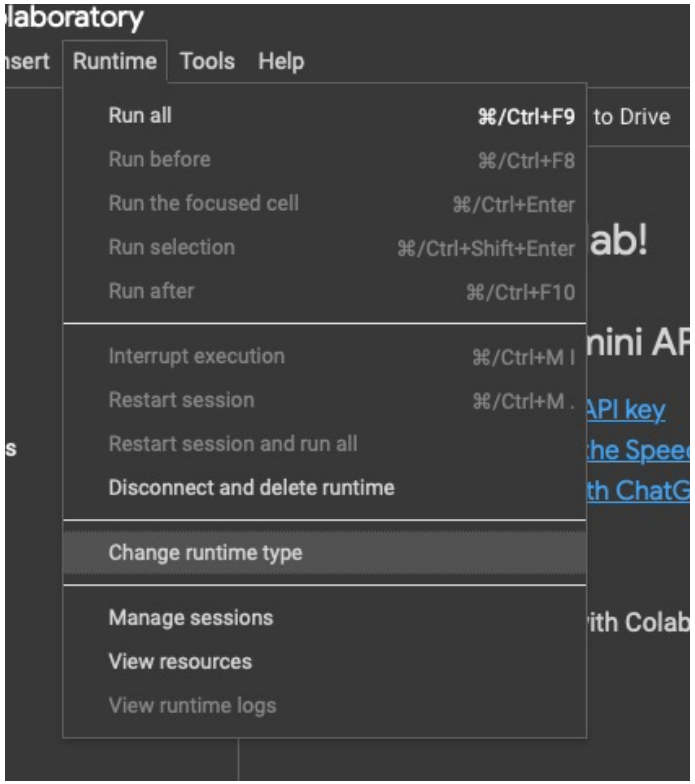
Google Colaboratory 101

Train Them!

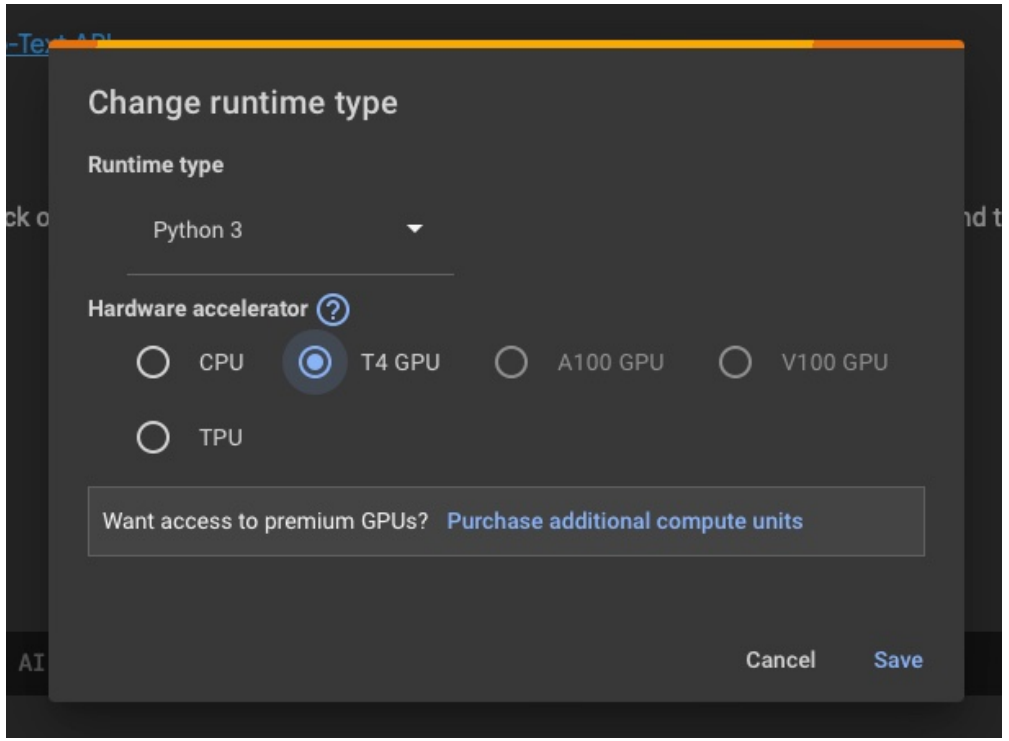
https://colab.research.google.com

- Google Account required
- Jupyter notebook
- Connected to Google Drive
- Free GPUs! (w/ fluctuation)
  - Opt in is required
- If notebook session is not active, you may get cut off
- At most 12 hours of running

How to Opt in GPU?



Step 1: Change runtime type



Step 2: Select T4 GPU

Resources

- Example: https://colab.research.google.com/drive/18cNN0Afo0JRNW0XHwwcP1pmaHqGCUREI?usp=sharing
- Basics of Jupyter Notebook: https://www.datacamp.com/tutorial/tutorial-jupyter-notebook
- How to download your own files?
  - Google Drive
    - `from google.colab import drive`
    - `drive.mount('/content/drive')`
  - Curl
    - !curl https://dummyjson.com/products/1 -o data1.json

Intelligent Computer Entertainment

Intelligent Computer Entertainment

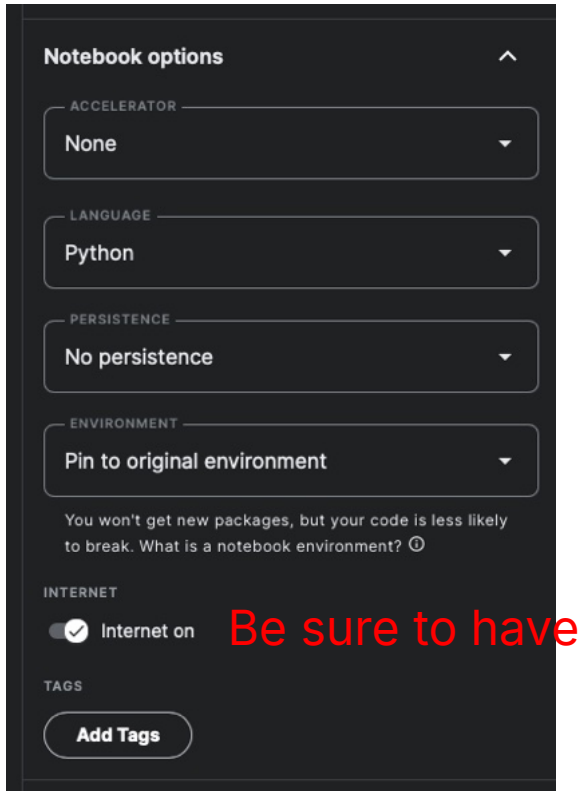
Kaggle 101

Train Them!

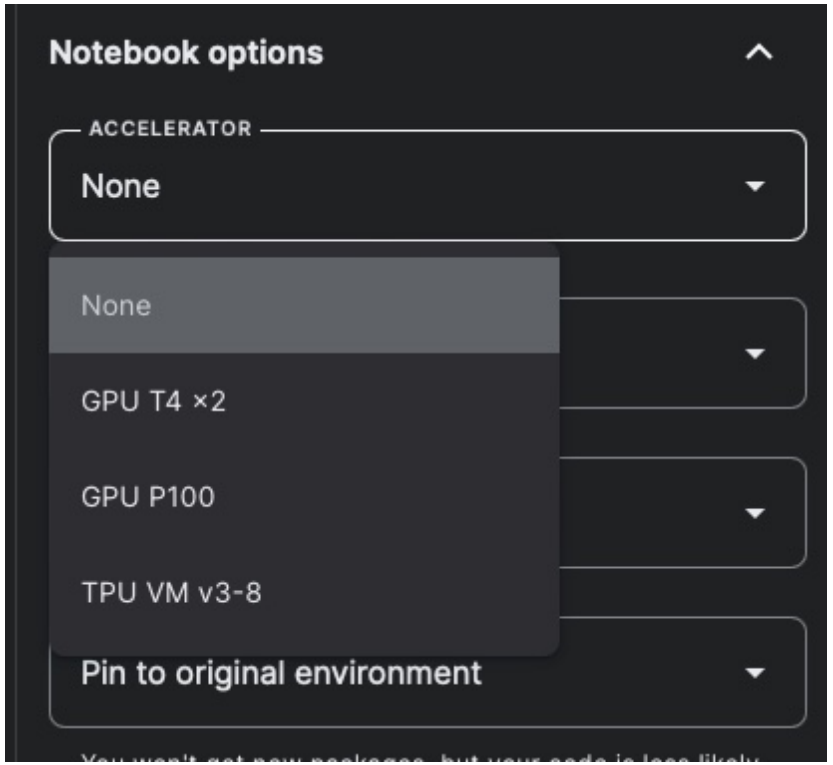
https://www.kaggle.com/code

- Mobile phone number verification may be required
- Better GPUs and more detailed configurations
  - Upto 40 GPU hours/week
  - Opt in is required
- Can run as long as needed

How to Opt in GPU?



Be sure to have “Internet on”



Step 1: On the right side bar, scroll to “Notebook options”

Step 2: Select GPT T4x2 or GPU P100

Intelligent Computer Entertainment

Run Them!

HuggingFace

Pipelines

- Optimized for inference only
- Very high-level of abstraction
- Support tasks across modalities
  - Not all tasks are supported
- For chat models, prompt templates are automatically managed

AutoModels

- Support both training and inference
- High-level of abstraction
- Require basic understandings of the model
- For chat models, prompt templates must be manually managed

Pipelines for LLMs

🤗 Accelerate

- Pre-trained

```
genai = pipeline("text-generation", model=model_name, device_map='auto')
text_completion = genai("prompt")
res = text_completion[-1]['generated_text'].replace("prompt", '')
```

- Chat

```
genai = pipeline("conversational", model=model_name, device_map='auto')
conversation = Conversation("prompt")
chat_completion = genai(conversation)
res = chat_completion.generated_responses[- 1]
```

AutoModel

```
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

prompt = "Hi!"
inputs = tokenizer(prompt, return_tensors=”pt")
generate_ids = model.generate(inputs.input_ids, max_length=30)
res = tokenizer.batch_decode(generate_ids,
skip_special_tokens=True, clean_up_tokenization_spaces=False)[0]
```

Intelligent Computer Entertainment

Ollama: Windows, macOS

Get up and running with large language models, locally.

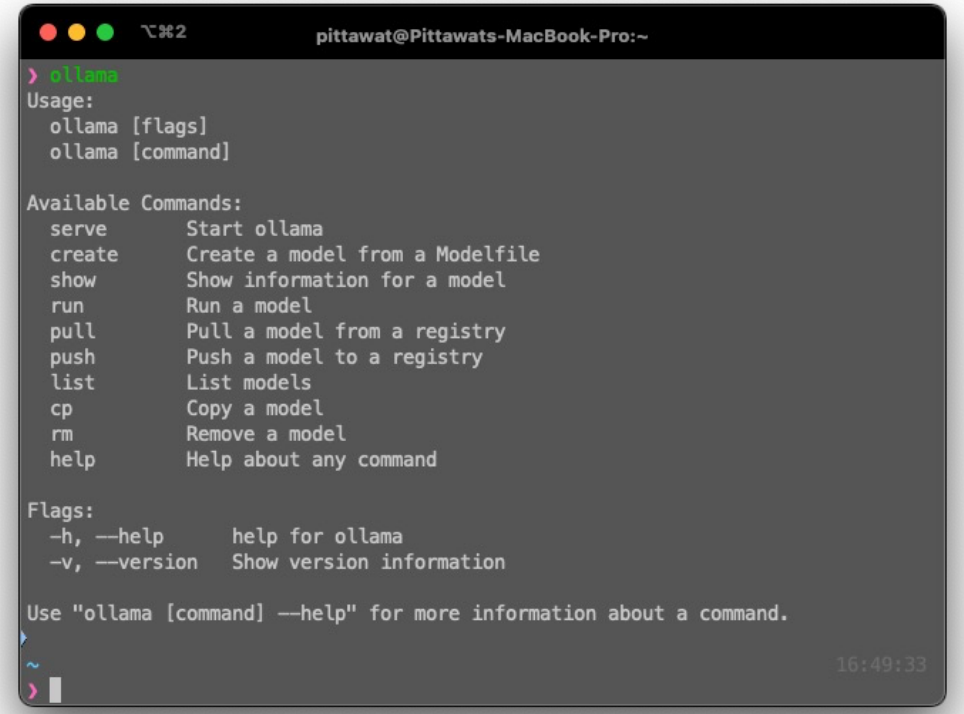
Run Llama 2, Code Llama, and other models.
Customize and create your own.

Download ↓

Available for macOS, Linux, and Windows (preview)

1. Start it

Step 1: Click on the icon



Step 2: Check if the application is ready

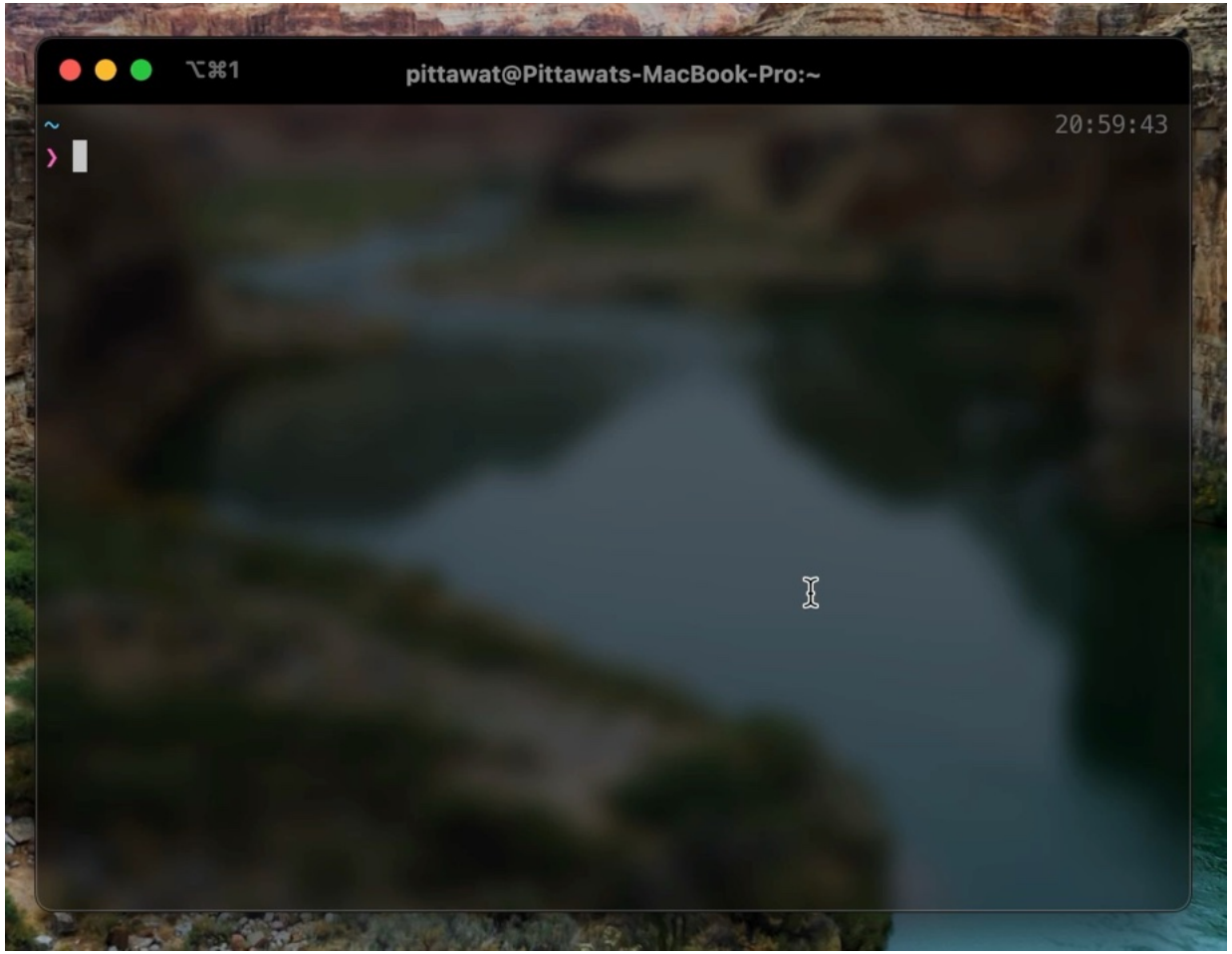
2. Download a Model

$ ollama run <model_name>

Example

$ ollama run mistral

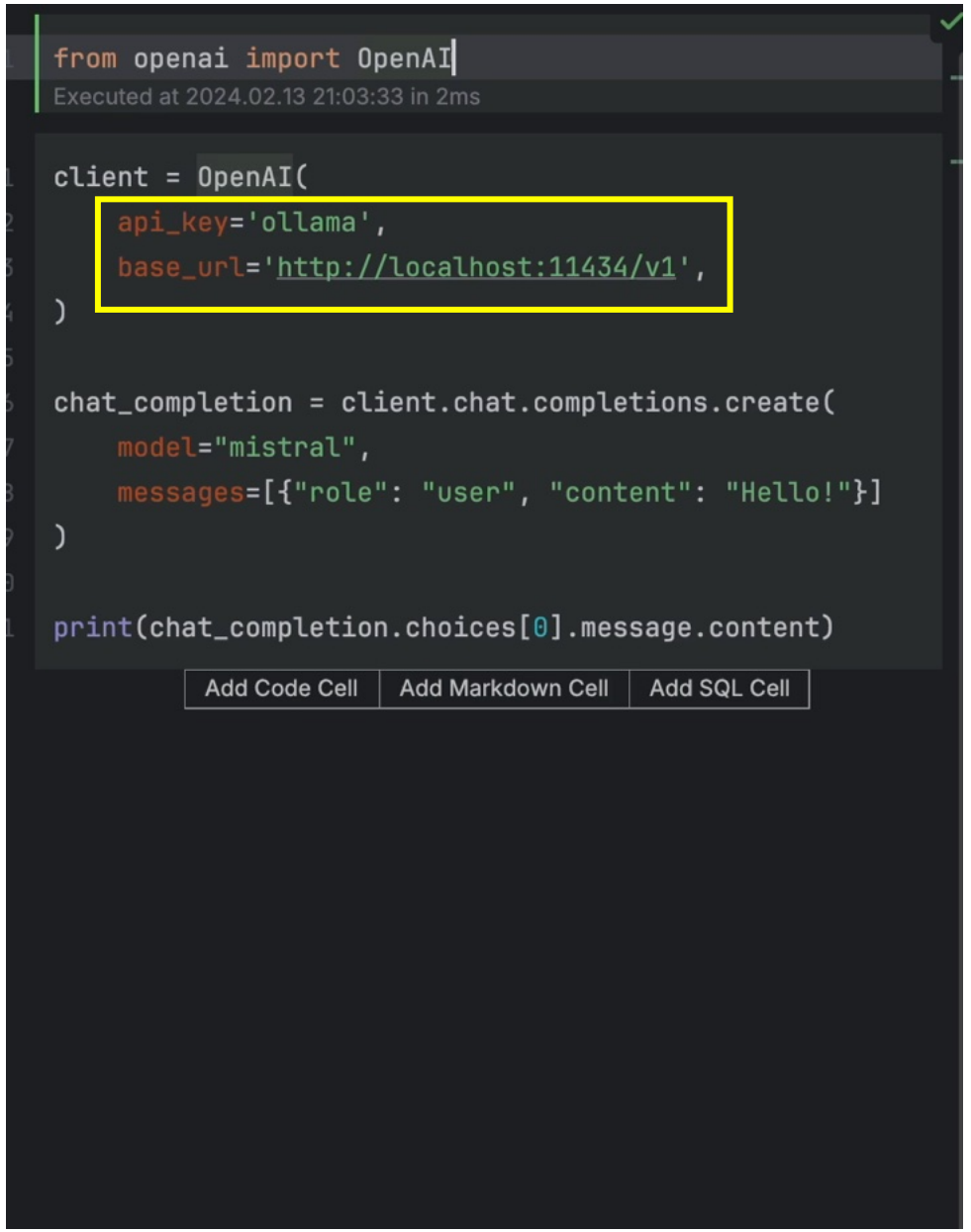
3. Interact with it!



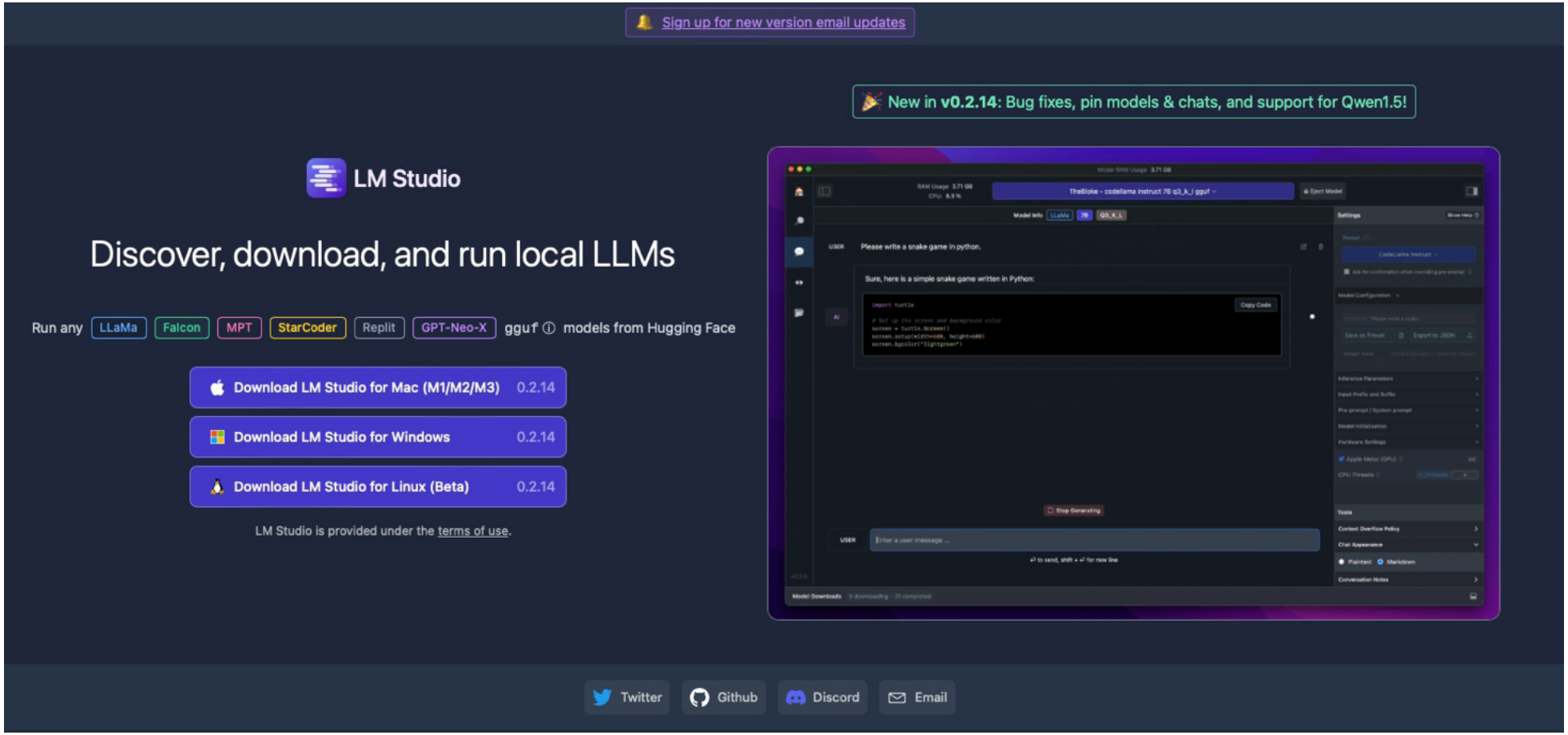
Intelligent Computer Entertainment

3. Interact with it!

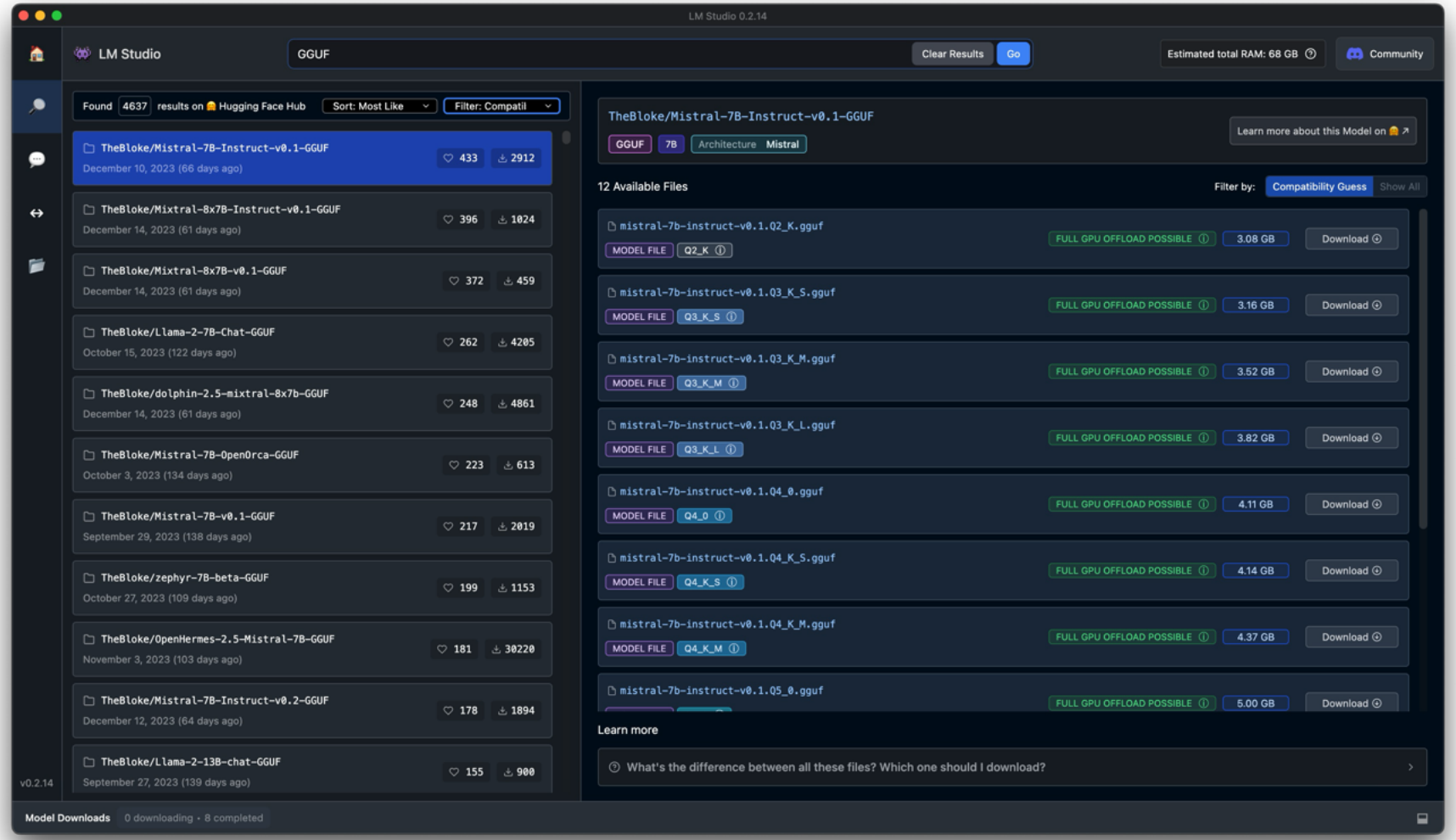
Via OpenAI Client!



LM Studio: https://lmstudio.ai

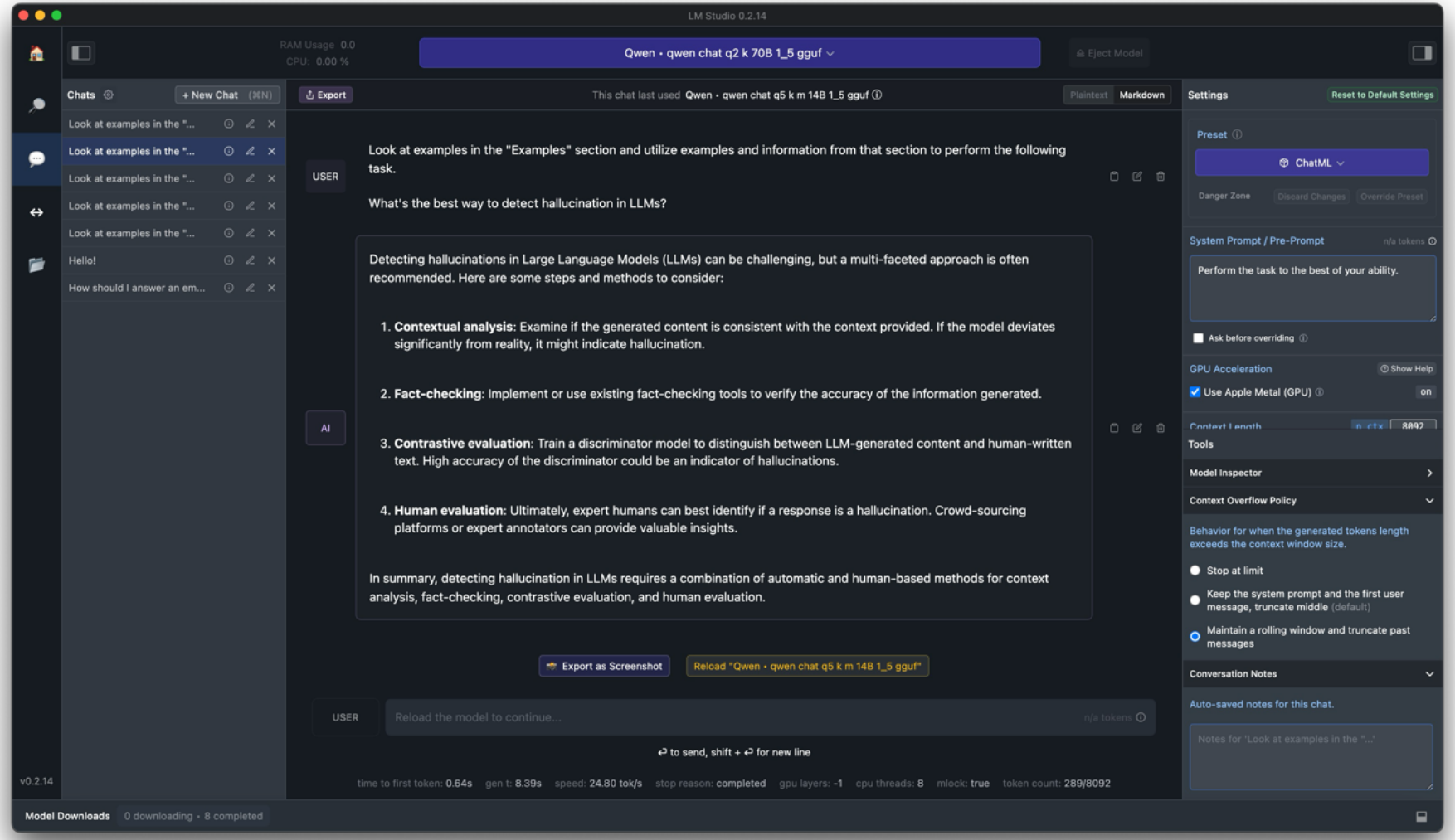


1. Download a Model



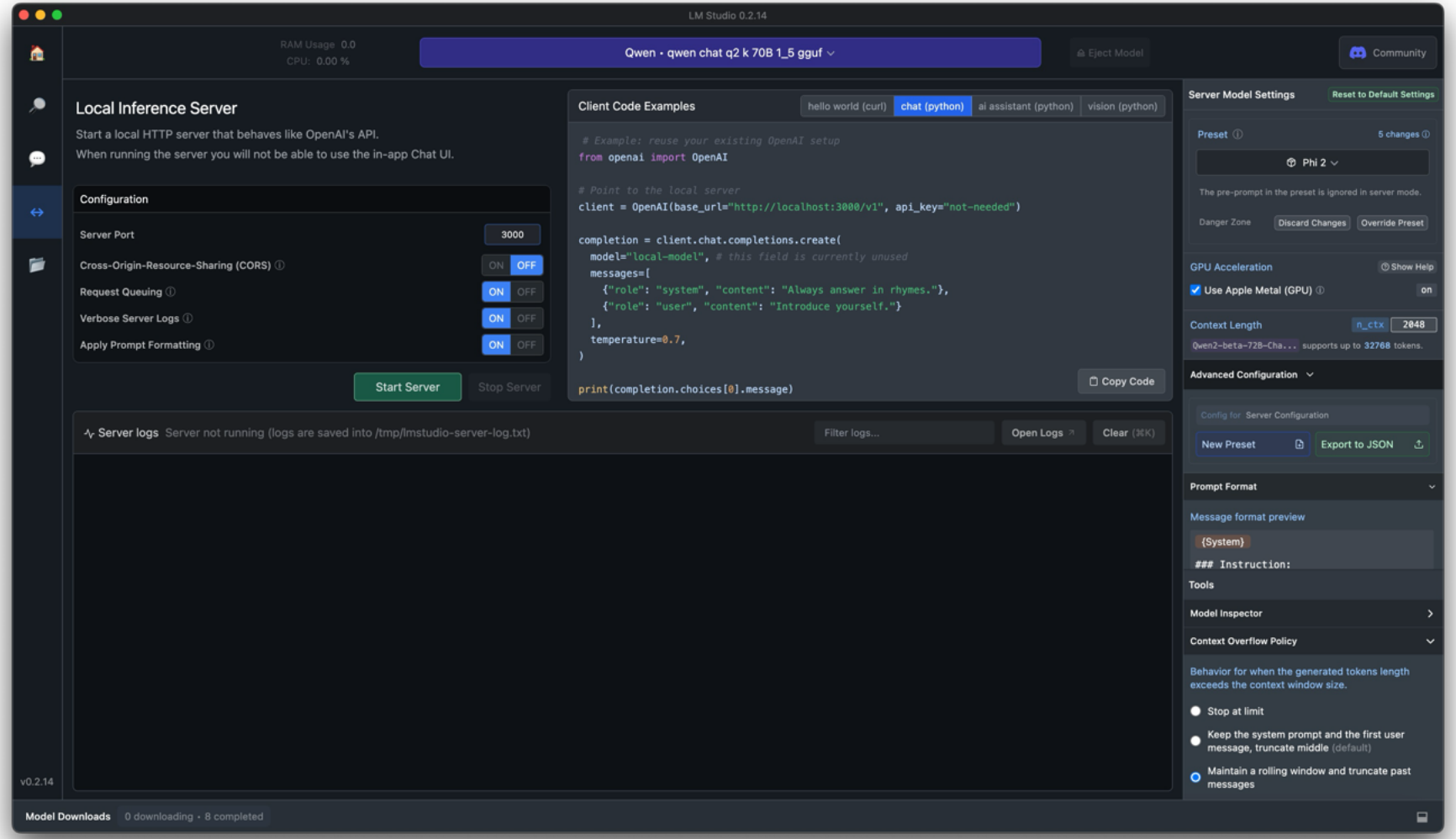
Intelligent Computer Entertainment

2. Chat



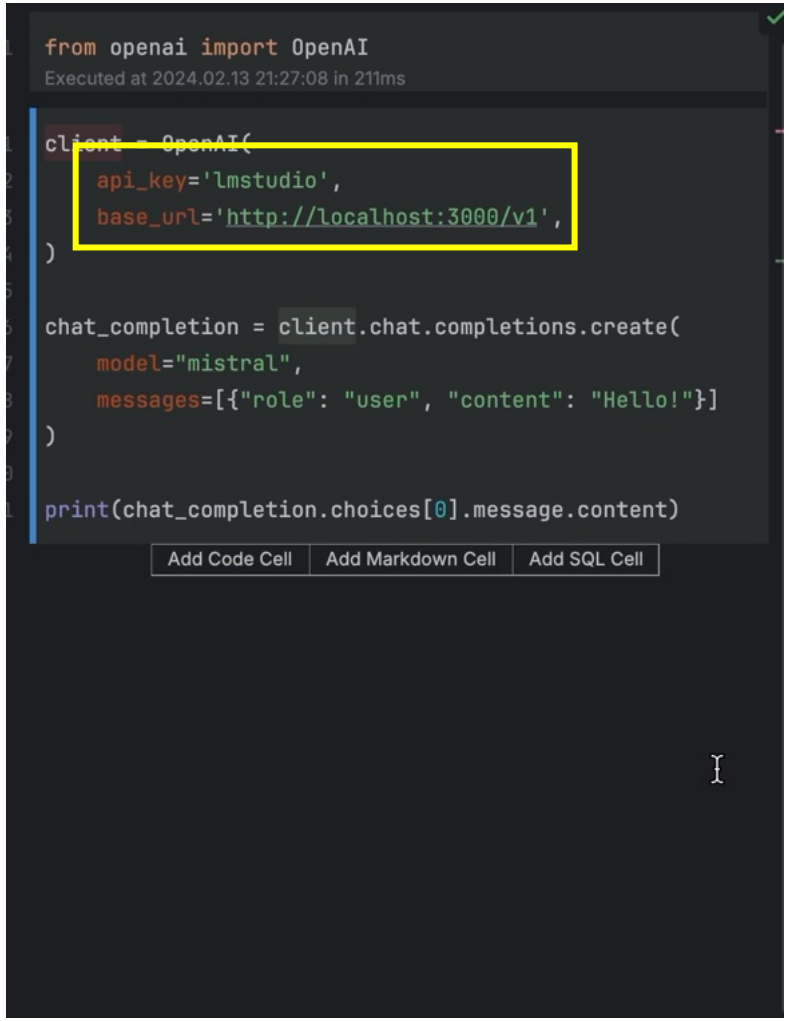
Intelligent Computer Entertainment

3. Run a Server



Intelligent Computer Entertainment

4. Interact via OpenAI Client



Intelligent Computer Entertainment

Gemini Pro

https://github.com/google/generative-ai-python

```
import google.generativeai as genai

genai.configure(api_key=os.environ["API_KEY"])

model = genai.GenerativeModel("gemini-pro")
response = model.generate_content("Hi!")
print(response.text)
```

Rate limit: 60 requests/minute