OpenAI API Documentation (Formatted)

Function Calling

Function calling Learn how to connect large language models to external tools. Introduction In an API call, you can describe functions and have the model intelligently choose to output a JSON object containing arguments to call one or many functions. The Chat Completions API does not call the function; instead, the model generates JSON that you can use to call the function in your code. The latest models (gpt-3.5-turbo-1006 and gpt-4-1106-preview) have been trained to both detect when a function should to be called (depending on the input) and to respond with JSON that adheres to the function signature more closely than previous models. With this capability also comes potential risks. We strongly recommend building in user confirmation flows before taking actions that impact the world on behalf of users (sending an email, posting something online, making a purchase, etc). This guide is focused on function calling with the Chat Completions API, for details on function calling in the Assistants API, please see the Assistants Tools page. Common use cases Function calling allows you to more reliably get structured data back from the model. For example, you can: • Create assistants that answer questions by calling external APIs (e.g. like ChatGPT Plugins) ■ e.g. define functions like send_email(to: string, body: string), or get_current_weather(location: string, unit: 'celsius' | 'fahrenheit') • Convert natural language into API calls ■ e.g. convert "Who are my top customers?" to get_customers(min_revenue: int, created_before: string, limit: int) and call your internal API • Extract structured data from text ■ e.g. define a function called extract_data(name: string, birthday: string), or sql_query(query: string) ...and much more! The basic sequence of steps for function calling is as follows: 1. Call the model with the user query and a set of functions defined in the functions parameter. 2. The model can choose to call one or more functions; if so, the content will be a stringified JSON object adhering to your custom schema (note: the model may hallucinate parameters). Stella Page 22 hallucinate parameters). 3. Parse the string into JSON in your code, and call your function with the provided arguments if they exist. 4. Call the model again by appending the function response as a new message, and let the model summarize the results back to the user. Supported models Not all model versions are trained with function calling data. Function calling is supported with the following models: • gpt-4 • gpt-4-1106-preview • gpt-4-0613 • gpt-3.5-turbo • gpt-3.5-turbo-1106 • gpt-3.5-turbo-0613 In addition, parallel function calls is supported on the following models: • gpt-4-1106-preview • gpt-3.5-turbo-1106 Parallel function calling Parallel function call is helpful for cases where you want to call multiple functions in one turn. For example, you may want to call functions to get the weather in 3 different locations at the same time. In this case, the model will call multiple functions in a single response. And you can pass back the results of each function call by referencing the tool_call_id in the response matching the ID of each tool call. In this example, we define a single function get_current_weather. The model calls the function multiple times, and after sending the function response back to the model, we let it decide the next step. It responded with a user-facing message which was telling the user the temperature in Boston, San Francisco, and Tokyo. Depending on the query, it may choose to call a function again. If you want to force the model to call a specific function you can do so by setting tool_choice with a specific function name. You can also force the model to generate a user-facing message by setting tool_choice: "none". Note that the default behavior (tool_choice: "auto") is for the model to decide on its own whether to call a function and if so which function to call. Example with one function called in parallel You can find more examples of function calling in the OpenAI cookbook: Function calling Learn from more examples demonstrating function calling Tokens Under the hood, functions are injected into the system message in a syntax the model has been trained on. This means functions count against the model's context limit and are billed as input tokens. If running into context limits, we suggest limiting the number of functions or the length of documentation you provide for function parameters. It is also possible to use fine-tuning to reduce the number of tokens used if you have Stella Page 23 It is also possible to use fine-tuning to reduce the number of tokens used if you have many functions defined. From Embeddings What are embeddings? OpenAI’s■text■embeddings■measure■the■relatedness■of■text■strings.■Embeddings■are■

commonly used for: • Search (where results are ranked by relevance to a query string) • Clustering (where text strings are grouped by similarity) • Recommendations (where items with related text strings are recommended) • Anomaly detection (where outliers with little relatedness are identified) • Diversity measurement (where similarity distributions are analyzed) • Classification (where text strings are classified by their most similar label) An embedding is a vector (list) of floating point numbers. The distance between two vectors measures their relatedness. Small distances suggest high relatedness and large distances suggest low relatedness. Visit our pricing page to learn about Embeddings pricing. Requests are billed based on the number of tokens in the input sent. To see embeddings in action, check out our code samples • Classification • Topic clustering • Search • Recommendations Browse■Samples■ How to get embeddings To get an embedding, send your text string to the embeddings API endpoint along with a choice of embedding model ID (e.g., text-embedding-ada-002). The response will contain an embedding, which you can extract, save, and use. Example requests: Example: Getting embeddings curl Copy■ 1 2 3 4 5 6 7 curl https://api.openai.com/v1/embeddings \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ Stella Page 24 -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "input": "Your text string goes here", "model": "text-embedding-ada-002" }' Example response: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 { "data": [{ "embedding": [-0.006929283495992422, -0.005336422007530928, ... -4.547132266452536e-05, -0.024047505110502243], "index": 0, "object": "embedding" }], "model": "text-embedding-ada-002", "object": "list", "usage": { "prompt_tokens": 5, "total_tokens": 5 } } See more Python code examples in the OpenAI Cookbook. When using OpenAI embeddings, please keep in mind their limitations and risks. Embedding models OpenAI offers one second-generation embedding model (denoted by -002 in the model ID) and 16 first-generation models (denoted by -001 in the model ID). Stella Page 25 We recommend using text-embedding-ada-002■for■nearly■all■use■cases.■It’s■better,■ cheaper, and simpler to use. Read the blog post announcement. MODEL GENERATION TOKENIZER MAX INPUT TOKENS KNOWLEDGE CUTOFF V2 cl100k_base 8191 Sep 2021 V1 GPT-2/GPT-3 2046 Aug 2020 Usage is priced per input token, at a rate of $0.0004 per 1000 tokens, or about ~3,000 pages per US dollar (assuming ~800 tokens per page): MODEL ROUGH PAGES PER DOLLAR EXAMPLE PERFORMANCE ON BEIR SEARCH EVAL text-embedding- ada-002 3000 53.9 *-davinci-*-001 6 52.8 *-curie-*-001 60 50.9 *-babbage-*-001 240 50.4 *-ada-*-001 300 49.0 Second-generation models MODEL NAME TOKENIZER MAX INPUT TOKENS OUTPUT DIMENSIONS text-embedding-ada-002 cl100k_base 8191 1536 First-generation models (not recommended) Use cases Here we show some representative use cases. We will use the Amazon fine-food reviews dataset for the following examples. Obtaining the embeddings The dataset contains a total of 568,454 food reviews Amazon users left up to October 2012. We will use a subset of 1,000 most recent reviews for illustration purposes. The reviews are in English and tend to be positive or negative. Each review has a ProductId, UserId, Score, review title (Summary) and review body (Text). For example: PRODUCT ID USER ID SCORE SUMMARY TEXT B001E4KFG0 A3SGXH7AUHU8G W 5 Good Quality Dog Food I have bought several of the Vitality canned... B00813GRG 4 A1D87F6ZCVE5NK 1 Not as Advertised Product arrived labeled as Jumbo Salted Peanut... We will combine the review summary and review text into a single combined text. The model will encode this combined text and output a single vector embedding. Get embeddings from dataset 1 2 3 4 5 6 Stella Page 26 6 7 8 9 from openai import OpenAI client = OpenAI() def get_embedding(text, model="text-embedding-ada-002"): text = text.replace("\n", " ") return client.embeddings.create(input = [text], model=model)['data'][0]['embedding'] df['ada_embedding'] = df.combined.apply(lambda x: get_embedding(x, model='text- embedding-ada-002')) df.to_csv('output/embedded_1k_reviews.csv', index=False) To load the data from a saved file, you can run the following: 1 2 3 4 import pandas as pd df = pd.read_csv('output/embedded_1k_reviews.csv') df['ada_embedding'] = df.ada_embedding.apply(eval).apply(np.array) Data visualization in 2D Embedding as a text feature encoder for ML algorithms Classification using the embedding features Zero-shot classification Obtaining user and product embeddings for cold-start recommendation Clustering Text search using embeddings Code search using embeddings Recommendations using embeddings Limitations & risks Our embedding models may be unreliable or pose social risks in certain cases, and may cause harm in the absence of mitigations. Social bias Limitation: The models encode social biases, e.g. via stereotypes or negative sentiment towards certain groups. We found evidence of bias in our models via running the SEAT (May et al, 2019) and the Winogender (Rudinger et al, 2018) benchmarks. Together, these benchmarks consist of 7 tests that measure whether models contain implicit biases when applied to gendered names, regional names, and some stereotypes. For example, we found that our models more strongly associate (a) European American names with positive sentiment, when compared to African American names, and (b) negative stereotypes with black women. These benchmarks are limited in several ways: (a) they may not generalize to your Stella Page 27 These benchmarks are limited in several ways: (a) they may not generalize to your particular use case, and (b) they only test for a very small slice of possible social bias. These tests are preliminary, and we recommend running tests for your specific use cases. These results should be taken as evidence of the existence of the phenomenon, not a definitive characterization of it for your use case. Please see our usage policies for more details and guidance. Please contact our support team via chat if you have any questions; we are happy to advise on this. Blindness to recent events Limitation: Models lack knowledge of events that occurred after August 2020. Our models are trained on datasets that contain some information about real world events up until 8/2020. If you rely on the models representing recent events, then they may not perform well. Frequently asked questions How can I tell how many tokens a string has before I embed it? In Python, you can split a string into tokens with OpenAI's tokenizer tiktoken. Example code: 1 2 3 4 5 6 7 8 9 import tiktoken def num_tokens_from_string(string: str, encoding_name: str) -> int: """Returns the number of tokens in a text string.""" encoding = tiktoken.get_encoding(encoding_name) num_tokens = len(encoding.encode(string)) return num_tokens num_tokens_from_string("tiktoken is great!", "cl100k_base") For second-generation embedding models like text-embedding-ada-002, use the cl100k_base encoding. More details and example code are in the OpenAI Cookbook guide how to count tokens with tiktoken. How can I retrieve K nearest embedding vectors quickly? For searching over many vectors quickly, we recommend using a vector database. You can find examples of working with vector databases and the OpenAI API in our Cookbook on GitHub. Vector database options include: Stella Page 28 Vector database options include: • Chroma, an open-source embeddings store • Elasticsearch, a popular search/analytics engine and vector database • Milvus, a vector database built for scalable similarity search • Pinecone, a fully managed vector database • Qdrant, a vector search engine • Redis as a vector database • Typesense, fast open source vector search • Weaviate, an open-source vector search engine • Zilliz, data infrastructure, powered by Milvus Which distance function should I use? We recommend cosine similarity.■The■choice■of■distance■function■typically■doesn’t■ matter much. OpenAI embeddings are normalized to length 1, which means that: • Cosine similarity can be computed slightly faster using just a dot product • Cosine similarity and Euclidean distance will result in the identical rankings Can I share my embeddings online? Customers own their input and output from our models, including in the case of embeddings. You are responsible for ensuring that the content you input to our API does not violate any applicable law or our Terms of Use. From Fine-tuning Learn how to customize a model for your application. Introduction This guide is intended for users of the new OpenAI fine-tuning API. If you are a legacy fine-tuning user, please refer to our legacy fine-tuning guide. Fine-tuning lets you get more out of the models available through the API by providing: • Higher quality results than prompting • Ability to train on more examples than can fit in a prompt • Token savings due to shorter prompts • Lower latency requests OpenAI's text generation models have been pre-trained on a vast amount of text. To use the models effectively, we include instructions and sometimes several examples in a prompt. Using demonstrations to show how to perform a task is often called "few-shot learning." Fine-tuning improves on few-shot learning by training on many more examples than can fit in the prompt, letting you achieve better results on a wide number of tasks. Once a model has been fine-tuned, you won't need to provide as many examples in the Stella Page 29 model has been fine-tuned, you won't need to provide as many examples in the prompt. This saves costs and enables lower-latency requests. At a high level, fine-tuning involves the following steps: 1. Prepare and upload training data 2. Train a new fine-tuned model 3. Evaluate results and go back to step 1 if needed 4. Use your fine-tuned model Visit our pricing page to learn more about how fine-tuned model training and usage are billed. What models can be fine-tuned? We are working on enabling fine-tuning for GPT-4 and expect this feature to be available later this year. Fine-tuning is currently available for the following models: • gpt-3.5-turbo-1006 (recommended) • babbage-002 • davinci-002 • gpt-4-0613 (experimental — eligible users can request here) You can also fine-tune a fine-tuned model which is useful if you acquire additional data and don't want to repeat the previous training steps. We expect gpt-3.5-turbo to be the right model for most users in terms of results and ease of use, unless you are migrating a legacy fine-tuned model. When to use fine-tuning Fine-tuning OpenAI text generation models can make them better for specific applications, but it requires a careful investment of time and effort. We recommend first attempting to get good results with prompt engineering, prompt chaining (breaking complex tasks into multiple prompts), and function calling, with the key reasons being: • There are many tasks at which our models may not initially appear to perform well, but results can be improved with the right prompts - thus fine-tuning may not be necessary • Iterating over prompts and other tactics has a much faster feedback loop than iterating with fine-tuning, which requires creating datasets and running training jobs • In cases where fine-tuning is still necessary, initial prompt engineering work is not wasted - we typically see best results when using a good prompt in the fine-tuning data (or combining prompt chaining / tool use with fine-tuning) Our prompt engineering guide provides a background on some of the most effective strategies and tactics for getting better performance without fine-tuning. You may find it helpful to iterate quickly on prompts in our playground. Common use cases Some common use cases where fine-tuning can improve results: • Setting the style, tone, format, or other qualitative aspects Improving reliability at producing a desired output Stella Page 30 • Improving reliability at producing a desired output • Correcting failures to follow complex prompts • Handling many edge cases in specific ways • Performing■a■new■skill■or■task■that’s■hard■to■articulate■in■a■prompt One high-level■way■to■think■about■these■cases■is■when■it’s■easier■to■"show,■not■tell".■In■ the sections to come, we will explore how to set up data for fine-tuning and various examples where fine-tuning improves the performance over the baseline model. Another scenario where fine-tuning is effective is in reducing costs and / or latency, by replacing GPT-4 or by utilizing shorter prompts, without sacrificing quality. If you can achieve good results with GPT-4, you can often reach similar quality with a fine- tuned gpt-3.5-turbo model by fine-tuning on the GPT-4 completions, possibly with a shortened instruction prompt. Preparing your dataset Once you have determined that fine-tuning■is■the■right■solution■(i.e.■you’ve■optimized■ your prompt as far as it can take you and identified problems that the model still has), you’ll■need■to■prepare■data■for■training■the ■model.■You■should■create■a■diverse■set■of■ demonstration conversations that are similar to the conversations you will ask the model to respond to at inference time in production. Each example in the dataset should be a conversation in the same format as our Chat Completions API, specifically a list of messages where each message has a role, content, and optional name. At least some of the training examples should directly target cases where the prompted model is not behaving as desired, and the provided assistant messages in the data should be the ideal responses you want the model to provide. Example format In this example, our goal is to create a chatbot that occasionally gives sarcastic responses, these are three training examples (conversations) we could create for a dataset: 1 2 3 {"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already."}]} {"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"}, {"role": "assistant", "content": "Oh, just some guy named William Shakespeare. Ever heard of him?"}]} {"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "How far is the Moon from Earth?"}, {"role": "assistant", "content": "Around 384,400 kilometers. Give or take a few, like that really matters."}]} The conversational chat format is required to fine-tune gpt-3.5-turbo. For babbage-002 and davinci-002, you can follow the prompt completion pair format used for legacy fine-tuning as shown below. 1 2 Stella Page 31 2 3 {"prompt": "", "completion": ""} {"prompt": "", "completion": ""} {"prompt": "", "completion": ""} Crafting prompts We generally recommend taking the set of instructions and prompts that you found worked best for the model prior to fine-tuning, and including them in every training example. This should let you reach the best and most general results, especially if you have relatively few (e.g. under a hundred) training examples. If you would like to shorten the instructions or prompts that are repeated in every example to save costs, keep in mind that the model will likely behave as if those instructions were included, and it may be hard to get the model to ignore those "baked- in" instructions at inference time. It may take more training examples to arrive at good results, as the model has to learn entirely through demonstration and without guided instructions. Example count recommendations To fine-tune a model, you are required to provide at least 10 examples. We typically see clear improvements from fine-tuning on 50 to 100 training examples with gpt-3.5-turbo but the right number varies greatly based on the exact use case. We recommend starting with 50 well-crafted demonstrations and seeing if the model shows signs of improvement after fine-tuning. In some cases that may be sufficient, but even if the model is not yet production quality, clear improvements are a good sign that providing more data will continue to improve the model. No improvement suggests that you may need to rethink how to set up the task for the model or restructure the data before scaling beyond a limited example set. Train and test splits After collecting the initial dataset, we recommend splitting it into a training and test portion. When submitting a fine-tuning job with both training and test files, we will provide statistics on both during the course of training. These statistics will be your initial signal of how much the model is improving. Additionally, constructing a test set early on will be useful in making sure you are able to evaluate the model after training, by generating samples on the test set. Token limits Each training example is limited to 4096 tokens. Examples longer than this will be truncated to the first 4096 tokens when training. To be sure that your entire training example fits in context, consider checking that the total token counts in the message contents are under 4,000. You can compute token counts using our counting tokens notebook from the OpenAI cookbook. Estimate costs Please refer to the pricing page for details on cost per 1k input and output tokens (we do to charge for tokens that are part of the validation data). To estimate the costs for a specific fine-tuning job, use the following formula: Stella Page 32 base cost per 1k tokens * number of tokens in the input file * number of epochs trained For a training file with 100,000 tokens trained over 3 epochs, the expected cost would be ~$2.40 USD. Check data formatting Once you have compiled a dataset and before you create a fine-tuning job, it is important to check the data formatting. To do this, we created a simple Python script which you can use to find potential errors, review token counts, and estimate the cost of a fine-tuning job. Fine-tuning data format validation Learn about fine-tuning data formatting Upload a training file Once you have the data validated, the file needs to be uploaded using the Files API in order to be used with a fine-tuning jobs: python Copy■ 1 2 3 4 5 6 7 from openai import OpenAI client = OpenAI() client.files.create(file=open("mydata.jsonl", "rb"), purpose="fine-tune") After you upload the file, it may take some time to process. While the file is processing, you can still create a fine-tuning job but it will not start until the file processing has completed. Create a fine-tuned model After ensuring you have the right amount and structure for your dataset, and have uploaded the file, the next step is to create a fine-tuning job. We support creating fine- tuning jobs via the fine-tuning UI or programmatically. To start a fine-tuning job using the OpenAI SDK: python Copy■ 1 2 3 4 5 6 7 from openai import OpenAI client = OpenAI() client.fine_tuning.jobs.create(training_file="file-abc123", Stella Page 33 training_file="file-abc123", model="gpt-3.5-turbo") In this example, model is the name of the model you want to fine-tune (gpt-3.5- turbo, babbage-002, davinci-002, or an existing fine-tuned model) and training_file is the file ID that was returned when the training file was uploaded to the OpenAI API. You can customize your fine-tuned model's name using the suffix parameter. To set additional fine-tuning parameters like the validation_file or hyperparameters, please refer to the API specification for fine-tuning. After you've started a fine-tuning job, it may take some time to complete. Your job may be queued behind other jobs in our system, and training a model can take minutes or hours depending on the model and dataset size. After the model training is completed, the user who created the fine-tuning job will receive an email confirmation. In addition to creating a fine-tuning job, you can also list existing jobs, retrieve the status of a job, or cancel a job. python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 from openai import OpenAI client = OpenAI() # List 10 fine-tuning jobs client.fine_tuning.jobs.list(limit=10) # Retrieve the state of a fine-tune client.fine_tuning.jobs.retrieve("ftjob-abc123") # Cancel a job client.fine_tuning.jobs.cancel("ftjob-abc123") # List up to 10 events from a fine-tuning job client.fine_tuning.jobs.list_events(id="ftjob-abc123", limit=10) # Delete a fine-tuned model (must be an owner of the org the model was created in) client.models.delete("ft:gpt-3.5-turbo:acemeco:suffix:abc123") Use a fine-tuned model When a job has succeeded, you will see the fine_tuned_model field populated with the name of the model when you retrieve the job details. You may now specify this model as a parameter to in the Chat Completions (for gpt-3.5-turbo) or legacy Completions API (for babbage-002 and davinci-002), and make requests to it using the Playground. Stella Page 34 After your job is completed, the model should be available right away for inference use. In some cases, it may take several minutes for your model to become ready to handle requests. If requests to your model time out or the model name cannot be found, it is likely because your model is still being loaded. If this happens, try again in a few minutes. python Copy■ 1 2 3 4 5 6 7 8 9 10 11 from openai import OpenAI client = OpenAI() response = client.chat.completions.create(model="ft:gpt-3.5-turbo:my-org:custom_suffix:id", messages=[{"role": "system", "content": "You are a helpful assistant."}, {"role": "user", "content": "Hello!"}]) print(completion.choices[0].message) You can start making requests by passing the model name as shown above and in our GPT guide. Analyzing your fine-tuned model We provide the following training metrics computed over the course of training: training loss, training token accuracy, test loss, and test token accuracy. These statistics are meant to provide a sanity check that training went smoothly (loss should decrease, token accuracy should increase). While an active fine-tuning jobs is running, you can view an event object which contains some useful metrics: 1 2 3 4 5 6 7 8 9 10 11 12 13 { "object": "fine_tuning.job.event", "id": "ftevent-abc-123", Stella Page 35 "id": "ftevent-abc-123", "created_at": 1693582679, "level": "info", "message": "Step 100/100: training loss=0.00", "data": { "step": 100, "train_loss": 1.805623287509661e-5, "train_mean_token_accuracy": 1.0 }, "type": "metrics" } After a fine-tuning job has finished, you can also see metrics around how the training process went by querying a fine-tuning job, extracting a file ID from the result_files, and then retrieving that files content. Each results CSV file has the following columns: step, train_loss, train_accuracy, valid_loss, and valid_mean_token_accuracy. 1 2 3 4 5 6 step,train_loss,train_accuracy,valid_loss,valid_mean_token_accuracy 1,1.52347,0.0,, 2,0.57719,0.0,, 3,3.63525,0.0,, 4,1.72257,0.0,, 5,1.52379,0.0,, While metrics can he helpful, evaluating samples from the fine-tuned model provides the most relevant sense of model quality. We recommend generating samples from both the base model and the fine-tuned model on a test set, and comparing the samples side by side. The test set should ideally include the full distribution of inputs that you might send to the model in a production use case. If manual evaluation is too time- consuming, consider using our Evals library to automate future evaluations. Iterating on data quality If the results from a fine-tuning job are not as good as you expected, consider the following ways to adjust the training dataset: • Collect examples to target remaining issues ■ If■the■model■still■isn’t■good■at■certain■aspects,■add■training■examples■that■ directly show the model how to do these aspects correctly • Scrutinize existing examples for issues ■ If your model has grammar, logic, or style issues, check if your data has any of the same issues. For instance, if the model now says "I will schedule this meeting■for■you"■(when■it■shouldn’t),■see■if■existing■examples■teach■the■model■ to■say■it■can■do■new■things■that■it■can’t■do • Consider the balance and diversity of data ■ If 60% of the assistant responses in the data says "I cannot answer this", but at inference time only 5% of responses should say that, you will likely get an overabundance of refusals • Make sure your training examples contain all of the information needed for the response Stella Page 36 response ■ If we want the model to compliment a user based on their personal traits and a training example includes assistant compliments for traits not found in the preceding conversation, the model may learn to hallucinate information • Look at the agreement / consistency in the training examples ■ If■multiple■people■created■the■training■data,■it’s■likely■that■model■performance■ will be limited by the level of agreement / consistency between people. For instance, in a text extraction task, if people only agreed on 70% of extracted snippets, the model would likely not be able to do better than this • Make sure your all of your training examples are in the same format, as expected for inference Iterating on data quantity Once■you’re■satisfied■with■the■quality■and■distribution ■of■the■examples,■you■can■consider■ scaling up the number of training examples. This tends to help the model learn the task better, especially around possible "edge cases". We expect a similar amount of improvement every time you double the number of training examples. You can loosely estimate the expected quality gain from increasing the training data size by: • Fine-tuning on your current dataset • Fine-tuning on half of your current dataset • Observing the quality gap between the two In general, if you have to make a trade-off, a smaller amount of high-quality data is generally more effective than a larger amount of low-quality data. Iterating on hyperparameters We allow you to specify the following hyperparameters: • epochs • learning rate multiplier • batch size We recommend initially training without specifying any of these, allowing us to pick a default for you based on dataset size, then adjusting if you observe the following: • If the model does not follow the training data as much as expected increase the number of epochs by 1 or 2 ■ This is more common for tasks for which there is a single ideal completion (or a small set of ideal completions which are similar). Some examples include classification, entity extraction, or structured parsing.

These are often tasks for which you can compute a final accuracy metric against a reference answer. • If the model becomes less diverse than expected decrease the number of epochs by 1 or 2 ■ This is more common for tasks for which there are a wide range of possible good completions • If the model does not appear to be converging, increase the learning rate multiplier You can set the hyperparameters as is shown below: python Copy■ 1 Stella Page 37 1 2 3 4 5 6 7 8 9 10 from openai import OpenAI client = OpenAI() client.fine_tuning.jobs.create(training_file="file-abc123", model="gpt-3.5-turbo", hyperparameters={ "n_epochs":2 }) Fine-tuning examples Now that we have explored the basics of the fine-tuning■API,■let's■look■at■going■through■ the fine-tuning lifecycle for a few different use cases. Style and tone In this example, we will explore how to build a fine-tuned model which gets the model follow specific style and tone guidance beyond what is possible with prompting alone. To begin, we create a sample set of messages showing what the model should which in this case is misspelled words. 1 2 3 {"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already."}]} {"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"}, {"role": "assistant", "content": "Oh, just some guy named William Shakespeare. Ever heard of him?"}]} {"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "How far is the Moon from Earth?"}, {"role": "assistant", "content": "Around 384,400 kilometers. Give or take a few, like that really matters."}]} If you want to follow along and create a fine-tuned model yourself, you will need at least 10 examples. After getting the data that will potentially improve the model, the next step is to check if the data meets all the formatting requirements. Now that we have the data formatted and validated, the final training step is to kick off a job to create the fine-tuned model. You can do this via the OpenAI CLI or one of our SDKs as shown below: 1 2 Stella Page 38 2 3 4 5 6 7 8 9 10 11 12 from openai import OpenAI client = OpenAI() file = client.files.create(file=open("marv.jsonl", "rb"), purpose="fine-tune") client.fine_tuning.jobs.create(training_file=file.id, model="gpt-3.5-turbo") Once the training job is done, you will be able to use your fine-tuned model. Collapse■ Structured output Another type of use case which works really well with fine-tuning is getting the model to provide structured information, in this case about sports headlines: 1 2 3 4 {"messages": [{"role": "system", "content": "Given a sports headline, provide the following fields in a JSON dict, where applicable: "player" (full name)", "team", "sport", and "gender".},{"role": "user", "content": "Sources: Colts grant RB Taylor OK to seek trade"}, {"role": "assistant", "content": "{"player": "Jonathan Taylor", "team": "Colts", "sport": "football", "gender": "male" }"},]} {"messages": [{"role": "system", "content": "Given a sports headline, provide the following fields in a JSON dict, where applicable: "player" (full name)", "team", "sport", and "gender".},{"role": "user", "content": "OSU 'split down middle' on starting QB battle"}, {"role": "assistant", "content": "{"player": null, "team": "OSU", "sport": "football", "gender": null }"},]} If you want to follow along and create a fine-tuned model yourself, you will need at least 10 examples. After getting the data that will potentially improve the model, the next step is to check if the data meets all the formatting requirements. Now that we have the data formatted and validated, the final training step is to kick off a job to create the fine-tuned model. You can do this via the OpenAI CLI or one of our SDKs as shown below: Stella Page 39 1 2 3 4 5 6 7 8 9 10 11 12 from openai import OpenAI client = OpenAI() file = client.files.create(file=open("sports-context.jsonl", "rb"), purpose="fine-tune") client.fine_tuning.jobs.create(training_file=file.id, model="gpt-3.5-turbo") Once the training job is done, you will be able to use your fine-tuned model and make a request that looks like the following: 1 2 3 4 5 6 7 8 9 completion = client.chat.completions.create(model="ft:gpt-3.5-turbo:my-org:custom_suffix:id", messages=[{"role": "system", "content": "Given a sports headline, provide the following fields in a JSON dict, where applicable: player (full name), team, sport, and gender"}, {"role": "user", "content": "Richardson wins 100m at worlds to cap comeback"}]) print(completion.choices[0].message) Based on the formatted training data, the response should look like the following: {"player": "Sha'Carri Richardson", "team": null", "sport": "track and field", "gender": "female"} Collapse■ Function calling The chat completions API supports function calling. Including a long list of functions in the completions API can consume a considerable number of prompt tokens and Stella Page 40 the completions API can consume a considerable number of prompt tokens and sometimes the model hallucinates or does not provide valid JSON output. Fine-tuning a model with function calling examples can allow you to: • Get similarly formatted responses even when the full function definition isn't present • Get more accurate and consistent outputs Format your examples as shown, with each line including a list of "messages" and an optional list of "functions": 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 { "messages": [{"role": "user", "content": "What is the weather in San Francisco?"}, {"role": "assistant", "function_call": {"name": "get_current_weather", "arguments": "{\"location\": \"San Francisco, USA\", \"format\": \"celcius\"}"}], "functions": [{ "name": "get_current_weather", "description": "Get the current weather", "parameters": { "type": "object", "properties": { "location": {"type": "string", "description": "The city and country, eg. San Francisco, USA"}, "format": {"type": "string", "enum": ["celsius", "fahrenheit"]} }, "required": ["location", "format"] } }] } If you want to follow along and create a fine-tuned model yourself, you will need at least 10 examples. If your goal is to use less tokens, some useful techniques are: • Omit function and parameter descriptions: remove the description field from function and parameters Stella Page 41 function and parameters • Omit parameters: remove the entire properties field from the parameters object • Omit function entirely: remove the entire function object from the functions array If your goal is to maximize the correctness of the function calling output, we recommend using the same function definitions for both training and querying the fine-tuned model. Fine-tuning on function calling can also be used to customize the model's response to function outputs. To do this you can include a function response message and an assistant message interpreting that response: 1 2 3 4 5 6 7 8 9 { "messages": [{"role": "user", "content": "What is the weather in San Francisco?"}, {"role": "assistant", "function_call": {"name": "get_current_weather", "arguments": "{\"location\": \"San Francisco, USA\", \"format\": \"celcius\"}"}} {"role": "function", "name": "get_current_weather", "content": "21.0"}, {"role": "assistant", "content": "It is 21 degrees celsius in San Francisco, CA"}], "functions": [...] // same as before } Collapse■ Migration of legacy models For users migrating from /v1/fine-tunes to the updated /v1/fine_tuning/jobs API and newer models, the main difference you can expect is the updated API. The legacy prompt completion pair data format has been retained for the updated babbage-002 and davinci-002 models to ensure a smooth transition. The new models will support fine-tuning with 4k token context and have a knowledge cutoff of September 2021. For most tasks, you should expect to get better performance from gpt-3.5-turbo than from the GPT base models. FAQ When should I use fine-tuning vs embeddings with retrieval? Embeddings with retrieval is best suited for cases when you need to have a large database of documents with relevant context and information.

By■default■OpenAI’s■models■are■trained■to■be■helpful■generalist■assistants.■Fine-tuning can be used to make a model which is narrowly focused, and exhibits specific ingrained behavior patterns. Retrieval strategies can be used to make new information available to a model by providing it with relevant context before generating its response. Retrieval strategies are not an alternative to fine-tuning and can in fact be complementary to it. Stella Page 42 When can I fine-tune GPT-4 or GPT-3.5-Turbo-16k? We plan to release support for fine-tuning both of these models later this year. How do I know if my fine-tuned model is actually better than the base model? We recommend generating samples from both the base model and the fine-tuned model on a test set of chat conversations, and comparing the samples side by side. For more comprehensive evaluations, consider using the OpenAI evals framework to create an eval specific to your use case. Can I continue fine-tuning a model that has already been fine- tuned? Yes, you can pass the name of a fine-tuned model into the model parameter when creating a fine-tuning job. This will start a new fine-tuning job using the fine-tuned model as the starting point. How can I estimate the cost of fine-tuning a model? Please refer to the estimate cost section above. Does the new fine-tuning endpoint still work with Weights & Biases for tracking metrics? No, we do not currently support this integration but are working to enable it in the near future. How many fine-tuning jobs can I have running at once? Please refer to our rate limit guide for the most up to date information on the limits. How do rate limits work on fine-tuned models? A fine-tuned model pulls from the same shared rate limit as the model it is based off of. For example, if you use half your TPM rate limit in a given time period with the standard gpt-3.5-turbo model, any model(s) you fine-tuned from gpt-3.5-turbo would only have the remaining half of the TPM rate limit accessible since the capacity is shared across all models of the same type. Put another way, having fine-tuned models does not give you more capacity to use our models from a total throughput perspective. From Image generation Learn how to generate or manipulate images with our DALL·E models. Introduction The Images API provides three methods for interacting with images: 1. Creating images from scratch based on a text prompt (DALL·E 3 and DALL·E 2) 2. Creating edited versions of images by having the model replace some areas of a Stella Page 43 2. Creating edited versions of images by having the model replace some areas of a pre-existing image, based on a new text prompt (DALL·E 2 only) 3. Creating variations of an existing image (DALL·E 2 only) This guide covers the basics of using these three API endpoints with useful code samples. To try DALL·E 3, head to ChatGPT. To try DALL·E 2, check out the DALL·E preview app. Usage Generations The image generations endpoint allows you to create an original image given a text prompt. When using DALL·E 3, images can have a size of 1024x1024, 1024x1792 or 1792x1024 pixels. By default, images are generated at standard quality, but when using DALL·E 3 you can set quality: "hd" for enhanced detail. Square, standard quality images are the fastest to generate. You can request 1 image at a time with DALL·E 3 (request more by making parallel requests) or up to 10 images at a time using DALL·E 2 with the n parameter. When you send a generation request to DALL·E 3, we will automatically re-write it for safety reasons, and to add more detail (because more detailed prompts generally result in higher quality images). Generate an image python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 from openai import OpenAI client = OpenAI() response = client.images.generate(model="dall-e-3", prompt="a white siamese cat", size="1024x1024", quality="standard", n=1,) image_url = response.data[0].url What is new with DALL·E 3 Explore what is new with DALL·E 3 in the OpenAI Cookbook Example DALL·E 3 generations PROMPT GENERATION Stella Page 44 PROMPT GENERATION A photograph of a white Siamese cat. Each image can be returned as either a URL or Base64 data, using the response_format parameter. URLs will expire after an hour. Edits (DALL·E 2 only) Also known as "inpainting", the image edits endpoint allows you to edit or extend an image by uploading an image and mask indicating which areas should be replaced. The transparent areas of the mask indicate where the image should be edited, and the prompt should describe the full new image, not just the erased area. This endpoint can enable experiences like the editor in our DALL·E preview app. Edit an image python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 from openai import OpenAI client = OpenAI() response = client.images.edit((model="dall-e-2", image=open("sunlit_lounge.png", "rb"), mask=open("mask.png", "rb"), prompt="A sunlit indoor lounge area with a pool containing a flamingo", n=1, size="1024x1024") image_url = response.data[0].url IMAGE MASK OUTPUT Prompt: a sunlit indoor lounge area with a pool containing a flamingo The uploaded image and mask must both be square PNG images less than 4MB in size, and also must have the same dimensions as each other. The non-transparent areas■of■the■mask■are■not■used■when■generating■the■output,■so■they■don't■necessarily■ need to match the original image like the example above. Variations (DALL·E 2 only) The image variations endpoint allows you to generate a variation of a given image. Generate an image variation python Copy■ 1 Stella Page 45 1 2 3 4 5 6 7 8 9 10 from openai import OpenAI client = OpenAI() response = client.images.create_variation(image=open("image_edit_original.png", "rb"), n=2, size="1024x1024") image_url = response.data[0].url IMAGE OUTPUT Similar to the edits endpoint, the input image must be a square PNG image less than 4MB in size. Content moderation Prompts and images are filtered based on our content policy, returning an error when a prompt or image is flagged. Language-specific tips Node.js■Python■ Using in-memory image data The Node.js examples in the guide above use the fs module to read image data from disk. In some cases, you may have your image data in memory instead. Here's an example API call that uses image data stored in a Node.js Buffer object: 1 2 3 4 5 6 7 8 9 // This is the Buffer object that contains your image data const buffer = [your image data]; // Set a `name` that ends with .png so that the API knows it's a PNG image buffer.name = "image.png"; const response = await openai.createImageVariation(buffer, 1, "1024x1024"); Working with TypeScript Stella Page 46 Working with TypeScript If you're using TypeScript, you may encounter some quirks with image file arguments. Here's an example of working around the type mismatch by explicitly casting the argument: 1 2 3 4 5 6 // Cast the ReadStream to `any` to appease the TypeScript compiler const response = await openai.createImageVariation(fs.createReadStream("image.png") as any, 1, "1024x1024"); And here's a similar example for in-memory image data: 1 2 3 4 5 6 7 8 9 10 11 // This is the Buffer object that contains your image data const buffer: Buffer = [your image data]; // Cast the buffer to `any` so that we can set the `name` property const file: any = buffer; // Set a `name` that ends with .png so that the API knows it's a PNG image file.name = "image.png"; const response = await openai.createImageVariation(file, 1, "1024x1024"); Error handling API requests can potentially return errors due to invalid inputs, rate limits, or other issues. These errors can be handled with a try...catch statement, and the error details can be found in either error.response or error.message: 1 2 3 4 5 6 7 Stella Page 47 7 8 9 10 11 12 13 14 15 try { const response = await openai.createImageVariation(fs.createReadStream("image.png"), 1, "1024x1024");

console.log(response.data.data[0].url); } catch (error) { if (error.response) { console.log(error.response.status); console.log(error.response.data); } else { console.log(error.message); } } From Vision Learn how to use GPT-4 to understand images Introduction GPT-4 with Vision, sometimes referred to as GPT-4V or gpt-4-vision-preview in the API, allows the model to take in images and answer questions about them. Historically, language model systems have been limited by taking in a single input modality, text. For many use cases, this constrained the areas where models like GPT-4 could be used. GPT-4 with vision is currently available to all developers who have access to GPT-4 via the gpt-4-vision-preview model and the Chat Completions API which has been updated to support image inputs. Note that the Assistants API does not currently support image inputs. It is important to note the following: • GPT-4 with vision is not a model that behaves differently from GPT-4, with the small exception of the system prompt we use for the model • GPT-4 with vision is not a different model that does worse at text tasks because it has vision, it is simply GPT-4 with vision added • GPT-4 with vision is an augmentative set of capabilities for the model Quick start Images can are made available to the model in two main ways: by passing a link to the image or by passing the base64 encoded image directly in the request. Images can be passed in the user, system and assistant messages. Currently we don't support images in the first system message but this may change in the future. Stella Page 48 the first system message but this may change in the future. Speak this text python Copy■ from openai import OpenAI client = OpenAI() response = client.chat.completions.create(model="gpt-4-vision-preview", messages=[{ "role": "user", "content": [{"type": "text", "text": "What's■in■this■image?"}, { "type": "image_url", "image_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Gfp-wisconsin-madison-the-nature-boardwalk.jpg/2560px-Gfp-wisconsin-madison-the-nature-boardwalk.jpg", },], }], max_tokens=300,) print(response.choices[0]) The model is best at answering general questions about what is present in the images. While it does understand the relationship between objects in images, it is not yet optimized to answer detailed questions about the location of certain objects in an image. For example, you can ask it what color a car is or what some ideas for dinner might be based on what is in you fridge, but if you show it an image of a room and ask it where the chair is, it may not answer the question correctly. It is important to keep in mind the limitations of the model as you explore what use- cases visual understanding can be applied to. Stella Page 49 Video understanding with vision Learn how to use use GPT-4 with Vision to understand videos in the OpenAI Cookbook Uploading base 64 encoded images If you have an image or set of images locally, you can pass those to the model in base 64 encoded format, here is an example of this in action: import base64 Stella Page 50 import base64 import requests # OpenAI API Key api_key = "YOUR_OPENAI_API_KEY" # Function to encode the image def encode_image(image_path): with open(image_path, "rb") as image_file: return base64.b64encode(image_file.read()).decode('utf-8') # Path to your image image_path = "path_to_your_image.jpg" # Getting the base64 string base64_image = encode_image(image_path) headers = { "Content-Type": "application/json", "Authorization": f"Bearer {api_key}" } payload = { "model": "gpt-4-vision-preview", "messages": [{ "role": "user", "content": [{ "type": "text", "text": "What's■in■this■image?" }, { "type": "image_url", "image_url": { "url": f"data:image/jpeg;base64,{base64_image}" } }] }], "max_tokens": 300 } response = requests.post("https://api.openai.com/v1/chat/completions", headers=headers, json=payload) print(response.json()) Multiple image inputs Stella Page 51 Multiple image inputs The Chat Completions API is capable of taking in and processing multiple image inputs in both base64 encoded format or as an image URL. The model will process each image and use the information from all of them to answer the question. Multiple image inputs python Copy■ from openai import OpenAI client = OpenAI() response = client.chat.completions.create(model="gpt-4-vision-preview", messages=[{ "role": "user", "content": [{ "type": "image_url", "image_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Gfp-wisconsin-madison-the-nature-boardwalk.jpg/2560px-Gfp-wisconsin-madison-the-nature-boardwalk.jpg", }, { "type": "text", "text": "What's■in■these■images?■Is■there■any■difference■between■them?", }, { "type": "image_url", "image_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Gfp-wisconsin-madison-the-nature-boardwalk.jpg/2560px-Gfp-wisconsin-madison-the-nature-boardwalk.jpg", madison-the-nature-boardwalk.jpg/2560px-Gfp-wisconsin-madison-the-nature-boardwalk.jpg", },], }], max_tokens=300,) print(response.choices[0]) Here the model is shown two copies of the same image and can answer questions about both or each of the images independently. Low or high fidelity image understanding By controlling the detail parameter, which has two options, low or high, you have control over how the model processes the image and generates its textual understanding. • low will■disable■the■“high■res”■model.■The■model■will■receive■a■low-res 512 x 512 version of the image, and represent the image with a budget of 65 tokens. This allows the API to return faster responses and consume fewer input tokens for use cases that do not require high detail. • high will■enable■“high■res”■mode,■which■first■allows■the■model■to■see■the■low■res■ image and then creates detailed crops of input images as 512px squares based on the input image size. Each of the detailed crops uses twice the token budget (65 tokens) for a total of 129 tokens. Managing images The Chat Completions API, unlike the Assistants API, is not stateful. That means you have to manage the messages (including images) you pass to the model yourself. If you want to pass the same image to the model multiple times, you will have to pass the image each time you make a request to the API. For long running conversations, we suggest passing images via URL's instead of base64. The latency of the model can also be improved by downsizing your images ahead of time to be less than the maximum size they are expected them to be. For low res mode, we expect a 512px x 512px image. For high rest mode, the short side of the image should be less than 768px and the long side should be less than 2,000px. After an image has been processed by the model, it is deleted from OpenAI servers and not retained. We do not use data uploaded via the OpenAI API to train our models. Limitations While GPT-4 with vision is powerful and can be used in many situations, it is important to understand the limitations of the model. Here are some of the limitations we are aware of: • Medical images: The model is not suitable for interpreting specialized medical images like CT scans and shouldn't be used for medical advice. • Non-English: The model may not perform optimally when handling images with text of non-Latin alphabets, such as Japanese or Korean. • Big text: Enlarge text within the image to improve readability, but avoid cropping important details. • Rotation: The model may misinterpret rotated / upside-down text or images. Stella Page 53 • Rotation: The model may misinterpret rotated / upside-down text or images. • Visual elements: The model may struggle to understand graphs or text where colors or styles like solid, dashed, or dotted lines vary. • Spatial reasoning: The model struggles with tasks requiring precise spatial localization, such as identifying chess positions. • Accuracy: The model may generate incorrect descriptions or captions in certain scenarios. • Image shape: The model struggles with panoramic and fisheye images. • Metadata and resizing: The model doesn't process original file names or metadata, and images are resized before analysis, affecting their original dimensions. • Counting: May give approximate counts for objects in images. • CAPTCHAS: For safety reasons, we have implemented a system to block the submission of CAPTCHAs. Calculating costs Image inputs are metered and charged in tokens, just as text inputs are. The token cost of a given image is determined by two factors: its size, and the detail option on each image_url block. All images with detail: low cost 85 tokens each. detail: high images are first scaled to fit within a 2048 x 2048 square, maintaining their aspect ratio. Then, they are scaled such that the shortest side of the image is 768px long. Finally, we count how many 512px squares the image consists of. Each of those squares costs 170 tokens. Another 85 tokens are always added to the final total. Here are some examples demonstrating the above. • A 1024 x 1024 square image in detail: high mode costs 765 tokens ■ 1024 is less than 2048, so there is no initial resize. ■ The shortest side is 1024, so we scale the image down to 768 x 768. ■ 4 512px square tiles are needed to represent the image, so the final token cost is 170 * 4 + 85 = 765. • A 2048 x 4096 image in detail: high mode costs 1105 tokens ■ We scale down the image to 1024 x 2048 to fit within the 2048 square. ■ The shortest side is 1024, so we further scale down to 768 x 1536. ■ 6 512px tiles are needed, so the final token cost is 170 * 6 + 85 = 1105. • A 4096 x 8192 image in detail: low most costs 85 tokens ■ Regardless of input size, low detail images are a fixed cost. FAQ Can I fine-tune the image capabilities in gpt-4? No, we do not support fine-tuning the image capabilities of gpt-4 at this time. Can I use gpt-4 to generate images? No, you can use dall-e-3 to generate images and gpt-4-vision-preview to understand images. What type of files can I upload? We currently support PNG (.png), JPEG (.jpeg and .jpg), WEBP (.webp), and non- Stella Page 54 We currently support PNG (.png), JPEG (.jpeg and .jpg), WEBP (.webp), and non- animated GIF (.gif). Is there a limit to the size of the image I can upload? Yes, we restrict image uploads to 20MB per image. Can I delete an image I uploaded? No, we will delete the image for you automatically after it has been processed by the model. Where can I learn more about the considerations of GPT-4 with Vision? You can find details about our evaluations, preparation, and mitigation work in the GPT-4 with Vision system card. We have further implemented a system to block the submission of CAPTCHAs. How do rate limits for GPT-4 with Vision work? We process images at the token level, so each image we process counts towards your tokens per minute (TPM) limit. See the calculating costs section for details on the formula used to determine token count per image. Can GPT-4 with Vision understand image metadata? No, the model does not receive image metadata. What happens if my image is unclear? If an image is ambiguous or unclear, the model will do its best to interpret it. However, the results may be less accurate. A good rule of thumb is that if an average human cannot see the info in an image at the resolutions used in low/high res mode, then the model cannot either. From Text to speech Learn how to turn text into lifelike spoken audio Introduction The Audio API provides a text to speech endpoint, speech, based on our TTS (text-to- speech) model. It comes with 6 build in voices and can be used to: • Narrate a written blog post • Produce spoken audio in multiple languages • Give real time audio output using streaming Here is an example of the alloy voice: Please note that our Usage Policies require you to provide a clear disclosure to end users that the TTS voice they are hearing is AI-generated and not a human voice. Quick start Stella Page 55 Quick start The speech endpoint takes in three key inputs: the model name, the text that should be turned into audio, and the voice to be used for the audio generation. A simple request would look like the following: Speak this text python Copy■ from pathlib import Path from openai import OpenAI client = OpenAI() speech_file_path = Path(__file__).parent / "speech.mp3" response = client.audio.speech.create(model="tts-1", voice="alloy", input="Today is a wonderful day to build something people love!") response.stream_to_file(speech_file_path) By default, the endpoint will output a MP3 file of the spoken audio but it can also be configured to output any of our supported formats. Audio quality For real-time applications, the standard tts-1 model provides the lowest latency but at a lower quality than the tts-1-hd model. Due to the way the audio is generated, tts-1 is likely to generate content that has more static in certain situations than tts-1-hd. In some cases, the audio may not have noticeable differences depending on your listening device and the individual person. Voice options Experiment with different voices (alloy, echo, fable, onyx, nova, and shimmer) to find one that matches your desired tone and audience. Alloy Echo Fable Onyx Nova Shimmer Supported output formats Stella Page 56 Supported output formats The default response format is "mp3", but other formats like "opus", "aac", or "flac" are available. • Opus: For internet streaming and communication, low latency. • AAC: For digital audio compression, preferred by YouTube, Android, iOS. • FLAC: For lossless audio compression, favored by audio enthusiasts for archiving. Streaming real time audio The Speech API provides support for real time audio streaming using chunk transfer encoding. This means that the audio is able to be played before the full file has been generated and made accessible. import io from openai import OpenAI from pydub import AudioSegment from pydub.playback import play client = OpenAI() def stream_and_play(text): response = client.audio.speech.create(model="tts-1", voice="alloy", input=text,) # Convert the binary response content to a byte stream Stella Page 57 # Convert the binary response content to a byte stream byte_stream = io.BytesIO(response.content) # Read the audio data from the byte stream audio = AudioSegment.from_file(byte_stream, format="mp3") # Play the audio play(audio) if __name__ == "__main__": text = input("Enter text: ") stream_and_play(text) FAQ How can I control the emotional range of the generated audio? There is no direct mechanism to control the emotional output of the audio generated. Certain factors may influence the output audio like capitalization or grammar but our internal tests with these have yielded mixed results. Can I create a custom copy of my own voice? No, this is not something we support. Do I own the outputted audio files? Yes, like with all outputs from our API, the person who created them owns the output. You are still required to inform end users that they are hearing audio generated by AI and not a real person talking to them. From Speech to text Learn how to turn audio into text Introduction The Audio API provides two speech to text endpoints, transcriptions and translations, based on our state-of-the-art open source large-v2 Whisper model. They can be used to: • Transcribe audio into whatever language the audio is in. • Translate and transcribe the audio into english. File uploads are currently limited to 25 MB and the following input file types are supported: mp3, mp4, mpeg, mpga, m4a, wav, and webm. Quickstart Transcriptions The transcriptions API takes as input the audio file you want to transcribe and the desired output file format for the transcription of the audio. We currently support multiple input and output file formats. Stella Page 58 Transcribe audio python Copy■ 1 2 3 4 5 6 7 8 from openai import OpenAI client = OpenAI() audio_file= open("/path/to/file/audio.mp3", "rb") transcript = client.audio.transcriptions.create(model="whisper-1", file=audio_file) By default, the response type will be json with the raw text included. { "text": "Imagine the wildest idea that you've ever had, and you're curious about how it might scale to something that's a 100, a 1,000 times bigger. .... } The Audio API also allows you to set additional parameters in a request. For example, if you want to set the response_format as text, your request would look like the following: Additional options python Copy■ 1 2 3 4 5 6 7 8 9 from openai import OpenAI client = OpenAI() audio_file = open("speech.mp3", "rb") transcript = client.audio.transcriptions.create(model="whisper-1", file=audio_file, response_format="text") The API Reference includes the full list of available parameters. Translations The translations API takes as input the audio file in any of the supported languages and transcribes, if necessary, the audio into English. This differs from our /Transcriptions endpoint since the output is not in the original input language and is instead translated to English text. Translate audio python Stella Page 59 python Copy■ 1 2 3 4 5 6 7 8 from openai import OpenAI client = OpenAI() audio_file= open("/path/to/file/german.mp3", "rb") transcript = client.audio.translations.create(model="whisper-1", file=audio_file) In this case, the inputted audio was german and the outputted text looks like: Hello, my name is Wolfgang and I come from Germany. Where are you heading today? We only support translation into english at this time. Supported languages We currently support the following languages through both the transcriptions and translations endpoint: Afrikaans, Arabic, Armenian, Azerbaijani, Belarusian, Bosnian, Bulgarian, Catalan, Chinese, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, Galician, German, Greek, Hebrew, Hindi, Hungarian, Icelandic, Indonesian, Italian, Japanese, Kannada, Kazakh, Korean, Latvian, Lithuanian, Macedonian, Malay, Marathi, Maori, Nepali, Norwegian, Persian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak, Slovenian, Spanish, Swahili, Swedish, Tagalog, Tamil, Thai, Turkish, Ukrainian, Urdu, Vietnamese, and Welsh. While the underlying model was trained on 98 languages, we only list the languages that exceeded <50% word error rate (WER) which is an industry standard benchmark for speech to text model accuracy. The model will return results for languages not listed above but the quality will be low. Longer inputs By default, the Whisper API only supports files that are less than 25 MB. If you have an audio file that is longer than that, you will need to break it up into chunks of 25 MB's or less or used a compressed audio format. To get the best performance, we suggest that you avoid breaking the audio up mid-sentence as this may cause some context to be lost. One way to handle this is to use the PyDub open source Python package to split the audio: 1 2 3 4 5 Stella Page 60 5 6 7 8 9 10 from pydub import AudioSegment song = AudioSegment.from_mp3("good_morning.mp3") # PyDub handles time in milliseconds ten_minutes = 10 * 60 * 1000 first_10_minutes = song[:ten_minutes] first_10_minutes.export("good_morning_10.mp3", format="mp3") OpenAI makes no guarantees about the usability or security of 3rd party software like PyDub. Prompting You can use a prompt to improve the quality of the transcripts generated by the Whisper API. The model will try to match the style of the prompt, so it will be more likely to use capitalization and punctuation if the prompt does too. However, the current prompting system is much more limited than our other language models and only provides limited control over the generated audio. Here are some examples of how prompting can help in different scenarios: 1. Prompts can be very helpful for correcting specific words or acronyms that the model often misrecognizes in the audio. For example, the following prompt improves the transcription of the words DALL·E and GPT-3, which were previously written as "GDP 3" and "DALI": "The transcript is about OpenAI which makes technology like DALL·E, GPT-3, and ChatGPT with the hope of one day building an AGI system that benefits all of humanity" 2. To preserve the context of a file that was split into segments, you can prompt the model with the transcript of the preceding segment. This will make the transcript more accurate, as the model will use the relevant information from the previous audio. The model will only consider the final 224 tokens of the prompt and ignore anything earlier. For multilingual inputs, Whisper uses a custom tokenizer. For English only inputs, it uses the standard GPT-2 tokenizer which are both accessible through the open source Whisper Python package. 3. Sometimes the model might skip punctuation in the transcript. You can avoid this by using a simple prompt that includes punctuation: "Hello, welcome to my lecture." 4. The model may also leave out common filler words in the audio. If you want to keep the filler words in your transcript, you can use a prompt that contains them:

"Umm, let me think like, hmm... Okay, here's what I'm, like, thinking." 5. Some languages can be written in different ways, such as simplified or traditional Chinese. The model might not always use the writing style that you want for your transcript by default. You can improve this by using a prompt in your preferred writing style. Improving reliability As we explored in the prompting section, one of the most common challenges faced when using Whisper is the model often does not recognize uncommon words or acronyms. To address this, we have highlighted different techniques which improve the Stella Page 61 acronyms. To address this, we have highlighted different techniques which improve the reliability of Whisper in these cases: Using the prompt parameter The first method involves using the optional prompt parameter to pass a dictionary of the correct spellings. Since it wasn't trained using instruction-following techniques, Whisper operates more like a base GPT model. It's important to keep in mind that Whisper only considers the first 244 tokens of the prompt. transcribe(filepath, prompt="ZyntriQix, Digique Plus, CynapseFive, VortiQore V8, EchoNix Array, OrbitalLink Seven, DigiFractal Matrix, PULSE, RAPT, B.R.I.C.K., Q.U.A.R.T.Z., F.L.I.N.T.") While it will increase reliability, this technique is limited to only 244 characters so your list of SKUs would need to be relatively small in order for this to be a scalable solution. Collapse■
Post-processing with GPT-4 The second method involves a post-processing step using GPT-4 or GPT-3.5-Turbo. We start by providing instructions for GPT-4 through the system_prompt variable. Similar to what we did with the prompt parameter earlier, we can define our company and product names. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 system_prompt = "You are a helpful assistant for the company ZyntriQix. Your task is to correct any spelling discrepancies in the transcribed text. Make sure that the names of the following products are spelled correctly: ZyntriQix, Digique Plus, CynapseFive, VortiQore V8, EchoNix Array, OrbitalLink Seven, DigiFractal Matrix, PULSE, RAPT, B.R.I.C.K., Q.U.A.R.T.Z., F.L.I.N.T. Only add necessary punctuation such as periods, commas, and capitalization, and use only the context provided." def generate_corrected_transcript(temperature, system_prompt, audio_file): response = client.chat.completions.create(model="gpt-4", Stella Page 62 model="gpt-4", temperature=temperature, messages=[{ "role": "system", "content": system_prompt }, { "role": "user", "content": transcribe(audio_file, "") }]) return response['choices'][0]['message']['content'] corrected_text = generate_corrected_transcript(0, system_prompt, fake_company_filepath) If you try this on your own audio file, you can see that GPT-4 manages to correct many misspellings in the transcript. Due to its larger context window, this method might be more scalable than using Whisper's prompt parameter and is more reliable since GPT-4 can be instructed and guided in ways that aren't possible with Whisper given the lack of instruction following. From Moderation Overview The moderations endpoint is a tool you can use to check whether content complies with OpenAI's usage policies. Developers can thus identify content that our usage policies prohibits and take action, for instance by filtering it. The models classifies the following categories: CATEGORY DESCRIPTION hate Content that expresses, incites, or promotes hate based on race, gender, ethnicity, religion, nationality, sexual orientation, disability status, or caste. Hateful content aimed at non-protected groups (e.g., chess players) is harrassment. hate/threat ening Hateful content that also includes violence or serious harm towards the targeted group based on race, gender, ethnicity, religion, nationality, sexual orientation, disability status, or caste. harassment Content that expresses, incites, or promotes harassing language towards any target. harassment /threatening Harassment content that also includes violence or serious harm towards any target. self-harm Content that promotes, encourages, or depicts acts of self-harm, such as suicide, cutting, and eating disorders. Stella Page 63 suicide, cutting, and eating disorders. self- harm/intent Content where the speaker expresses that they are engaging or intend to engage in acts of self-harm, such as suicide, cutting, and eating disorders. self- harm/instru ctions Content that encourages performing acts of self-harm, such as suicide, cutting, and eating disorders, or that gives instructions or advice on how to commit such acts. sexual Content meant to arouse sexual excitement, such as the description of sexual activity, or that promotes sexual services (excluding sex education and wellness). sexual/min ors Sexual content that includes an individual who is under 18 years old. violence Content that depicts death, violence, or physical injury. violence/gr aphic Content that depicts death, violence, or physical injury in graphic detail. The moderation endpoint is free to use when monitoring the inputs and outputs of OpenAI APIs. We currently disallow other use cases. Accuracy may be lower on longer pieces of text. For higher accuracy, try splitting long pieces of text into smaller chunks each less than 2,000 characters. We are continuously working to improve the accuracy of our classifier. Our support for non-English

languages is currently limited. Quickstart To obtain a classification for a piece of text, make a request to the moderation endpoint as demonstrated in the following code snippets: Example: Getting moderations curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/moderations \ -X POST \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{"input": "Sample text goes here"}' Below is an example output of the endpoint. It returns the following fields: • flagged: Set to true if the model classifies the content as violating OpenAI's usage policies, false otherwise. • categories: Contains a dictionary of per-category binary usage policies violation flags. For each category, the value is true if the model flags the corresponding category as violated, false otherwise. • category_scores: Contains a dictionary of per-category raw scores output by the model, denoting the model's confidence that the input violates the OpenAI's policy for the category. The value is between 0 and 1, where higher values denote higher confidence. The scores should not be interpreted as probabilities. 1 Stella Page 64 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 { "id": "modr-XXXXX", "model": "text-moderation-005", "results": [{ "flagged": true, "categories": { "sexual": false, "hate": false, "harassment": false, "self-harm": false, "sexual/minors": false, "hate/threatening": false, "violence/graphic": false, "self-harm/intent": false, "self-harm/instructions": false, "harassment/threatening": true, "violence": true, }, "category_scores": { "sexual": 1.2282071e-06, "hate": 0.010696256, "harassment": 0.29842457, Stella Page 65 "harassment": 0.29842457, "self-harm": 1.5236925e-08, "sexual/minors": 5.7246268e-08, "hate/threatening": 0.0060676364, "violence/graphic": 4.435014e-06, "self-harm/intent": 8.098441e-10, "self-harm/instructions": 2.8498655e-11, "harassment/threatening": 0.63055265, "violence": 0.99011886, } }] } OpenAI will continuously upgrade the moderation endpoint's underlying model. Therefore, custom policies that rely on category_scores may need recalibration over time. From Assistants API Beta The Assistants API allows you to build AI assistants within your own applications. An Assistant has instructions and can leverage models, tools, and knowledge to respond to user queries. The Assistants API currently supports three types of tools: Code Interpreter, Retrieval, and Function calling. In the future, we plan to release more OpenAI-built tools, and allow you to provide your own tools on our platform. At a high level, a typical integration of the Assistants API has the following flow: 1. Create an Assistant in the API by defining it custom instructions and picking a model. If helpful, enable tools like Code Interpreter, Retrieval, and Function calling. 2. Create a Thread when a user starts a conversation. 3. Add Messages to the Thread as the user ask questions. 4. Run the Assistant on the Thread to trigger responses. This automatically calls the relevant tools. The Assistants API is in beta and we are actively working on adding more functionality. Share your feedback in our Developer Forum! This starter guide walks through the key steps to create and run an Assistant that uses Code Interpreter. Step 1: Create an Assistant An■Assistant■represents■an ■entity■that■can■be■configured■to■respond■to■users'■Messages■ using several parameters like: • Instructions: how the Assistant and model should behave or respond • Model: you can specify any GPT-3.5 or GPT-4 models, including fine-tuned models. The Retrieval tool requires gpt-3.5-turbo-1106 and gpt-4-1106-preview models. • Tools: the API supports Code Interpreter and REtrieval that are built and hosted by OpenAI. • Functions: the API allows you to define custom function signatures, with similar behavior as our function calling feature. In this example, we're creating an Assistant that is a personal math tutor, with the Code Stella Page 66 In this example, we're creating an Assistant that is a personal math tutor, with the Code Interpreter tool enabled: Calls to the Assistants API require that you pass a Beta header. This is handled automatically■if■you're■using■OpenAI's■official■Python■and■Node.js■SDKs. OpenAI-Beta: assistants=v1 python Copy■ 1 2 3 4 5 6 assistant = client.beta.assistants.create(name="Math Tutor", instructions="You are a personal math tutor. Write and run code to answer math questions.", tools=[{"type": "code_interpreter"}], model="gpt-4-1106-preview") Step 2: Create a Thread A Thread represents a conversation. We recommend creating one Thread per user as soon as the user initiates the conversation. Pass any user-specific context and files in this thread by creating Messages. python Copy■ thread = client.beta.threads.create() Threads■don't■have■a■size■limit .■You■can■pass■as■many■Messages■as■you■want■to■a■ Thread. The API will ensure that requests to the model fit within the maximum context window, using relevant optimization techniques such as truncation. Step 3: Add a Message to a Thread A Message contains the user's text, and optionally, any files that the user uploads. Image files aren't supported today, but we plan to add support for them in the coming months. python Copy■ 1 2 3 4 5 message = client.beta.threads.messages.create(thread_id=thread.id, role="user", content="I need to solve the equation `3x + 11 = 14`. Can you help me?") Now if you list Messages in Thread, you will see that this message is added to the thread on creation: 1 2 3 4 5 Stella Page 67 5 6 7 8 9 10 11 12 13 14 15 16 17 { "object": "list", "data": [{ "created_at": 1696995451, "id": "msg_4rb1Skx3XgQZEe4PHVRFQhr0", "object": "thread.message", "thread_id": "thread_34p0sfdas0823smfv", "role": "user", "content": [{ "type": "text", "text": { "value": "I need to solve the equation `3x + 11 = 14`. Can you help me?", "annotations": [] } }], ... Step 4: Run the Assistant For the Assistant to respond to the user message, you need to create a Run. This makes the Assistant read the Thread and decide whether to call tools or simply use the model to best answer the user query. As the run progresses, the assistant appends Messages to the thread with the role="assistant" . You can optionally pass additional instructions to the Assistant while creating the Run: python Copy■ 1 2 3 4 5 run = client.beta.threads.runs.create(thread_id=thread.id, assistant_id=assistant.id, instructions="Please address the user as Jane Doe. The user has a premium account.") Step 5: Display the Assistant's Response This creates a Run in a queued status. You can periodically retrieve the Run to check on its status to see if it has moved to completed. python Copy■ 1 2 Stella Page 68 2 3 4 run = client.beta.threads.runs.retrieve(thread_id=thread.id, run_id=run.id) Once the Run completes, you can retrieve the Messages added by the Assistant to the Thread. python Copy■ 1 2 3 messages = client.beta.threads.messages.list(thread_id=thread.id) And finally, display them to the user! During this Run, the Assistant added two new Messages to the Thread. ROLE CONTENT user I need to solve the equation 3x + 11 = 14. Can you help me? assista nt Certainly, Jane Doe. To solve the equation (3x + 11 = 14) for (x), you'll want to isolate (x) on one side of the equation. Here's how you can do that: 1.Subtract 11 from both sides of the equation to get (3x = 3). 2.Then, divide both sides by 3 to solve for (x). Let me calculate the value of (x) for you. assista nt The solution to the equation (3x + 11 = 14) is (x = 1). You can also retrieve the Run Steps of this Run if you'd like to explore or display the inner workings of the Assistant and its tools. Next 1. Dive deeper into How Assistants work 2. Learn more about Tools From How Assistants work Beta The Assistants API is designed to help developers build powerful AI assistants capable of performing a variety of tasks. The Assistants API is in beta and we are actively working on adding more functionality. Share your feedback in our Developer Forum! 1. Assistants■can■call■OpenAI's models with specific instructions to tune their personality and capabilities. 2. Assistants can access multiple tools in parallel. These can be both OpenAI- hosted tools — like Code interpreter and Knowledge retrieval — or tools you build / Stella Page 69 hosted tools — like Code interpreter and Knowledge retrieval — or tools you build / host (via Function calling). 3. Assistants can access persistent Threads. Threads simplify AI application development by storing message history and truncating it when the conversation gets■too■long■for■the■model's■context■length.■You■create■a■Thread■once, ■and■simply■ append Messages to it as your users reply. 4. Assistants can access Files in several formats — either as part of their creation or as part of Threads between Assistants and users. When using tools, Assistants can also create files (e.g., images, spreadsheets, etc) and cite files they reference in the Messages they create. Objects OBJECT WHAT IT REPRESENTS Assista nt Purpose-built■AI■that■uses■OpenAI's models and calls tools Thread A conversation session between an Assistant and a user. Threads store Messages■and■automatically■handle■truncation■to■fit■content■into■a■model's■ context. Messa ge A message created by an Assistant or a user. Messages can include text, images, and other files. Messages stored as a list on the Thread. Run An■invocation■of■an■Assistant■on■a■Thread.■The■Assistant■uses■it's■ configuration■and■the■Thread's■Messages■to■perform■tasks■by■calling■models■ and tools. As part of a Run, the Assistant appends Messages to the Thread. Run Step A detailed list of steps the Assistant took as part of a Run. An Assistant can call■tools■or■create■Messages■during■it's■run.■Examining■Run■Steps■allows■you■ to■introspect■how■the■Assistant■is■getting■to■it's■final■results. Creating Assistants We■recommend■using■OpenAI's latest models with the Assistants API for best results and maximum compatibility with tools. To get started, creating an Assistant only requires specifying the model to use. But you can further customize the behavior of the Assistant: 1. Use the instructions parameter to guide the personality of the Assistant and define it's■goals.■Instructions■are■similar■to■system■messages■in■the■Chat■Completions■ API. 2. Use the tools parameter to give the Assistant access to up to 128 tools. You can give it access to OpenAI-hosted tools like code_interpreter and retrieval, or call a third-party tools via a function calling. 3. Use the file_ids parameter to give the tools like code_interpreter and retrieval access to files. Files are uploaded using the File upload endpoint and must have the purpose set to assistants to be used with this API. For example, to create an Assistant that can create data visualization based on a .csv file, first upload a file. python Copy■ 1 2 3 4 Stella Page 70 4 file = client.files.create(file=open("speech.py", "rb"), purpose='assistants') And then create the Assistant with the uploaded file. python Copy■ 1 2 3 4 5 6 7 assistant = client.beta.assistants.create(name="Data visualizer", description="You are great at creating beautiful data visualizations. You analyze data present in .csv files, understand trends, and come up with data visualizations relevant to those trends. You also share a brief text summary of the trends observed.", model="gpt-4-1106-preview", tools=[{"type": "code_interpreter"}], file_ids=[file.id]) You can attach a maximum of 20 files per Assistant, and they can be at most 512 MB each. In addition, the size of all the files uploaded by your organization should not exceed 100GB. You can request an increase in this storage limit using our help center. You can also use the AssistantFile object to create, delete, or view associations between Assistant and File objects. Note that deleting an AssistantFile doesn’t■delete■the■original■ File object, it simply deletes the association between that File and the Assistant. To delete a File, use the File delete endpoint instead. Managing Threads and Messages Threads and Messages represent a conversation session between an Assistant and a user. There is no limit to the number of Messages you can store in a Thread. Once the size of the Messages exceeds the context window of the model, the Thread smartly truncates them to fit. You can create a Thread with an initial list of Messages like this: python Copy■ 1 2 3 4 5 6 7 8 9 thread = client.beta.threads.create(messages=[{ "role": "user", "content": "Create 3 data visualizations based on the trends in this file.", "file_ids": [file.id] }]) Messages can contain text, images, or files. At the moment, user-created Messages Stella Page 71 Messages can contain text, images, or files. At the moment, user-created Messages cannot contain image files but we plan to add support for this in the future. Message annotations Messages created by Assistants may contain annotations within the content array of the object. Annotations provide information around how you should annotate the text in the Message. There are two types of Annotations: 1. file_citation: File citations are created by the retrieval tool and define references to a specific quote in a specific file that was uploaded and used by the Assistant to generate the response. 2. file_path: File path annotations are created by the code_interpreter tool and contain references to the files generated by the tool. When annotations are present in the Message object, you'll see illegible model- generated substrings in the text that you should replace with the annotations. These strings may look something like ■13†source■ or sandbox:/mnt/data/file.csv.■Here’s■an■ example python code snippet that replaces these strings with information present in the annotations. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 # Retrieve the message object message = client.beta.threads.messages.retrieve(thread_id="...", message_id="...") # Extract the message content Stella Page 72 # Extract the message content message_content = message.content[0].text annotations = message_content.annotations citations = [] # Iterate over the annotations and add footnotes for index, annotation in enumerate(annotations): # Replace the text with a footnote message_content.value = message_content.value.replace(annotation.text, f' [{index}]') # Gather citations based on annotation attributes if (file_citation := getattr(annotation, 'file_citation', None)): cited_file = client.files.retrieve(file_citation.file_id) citations.append(f'[{index}] {file_citation.quote} from {cited_file.filename}') elif (file_path := getattr(annotation, 'file_path', None)): cited_file = client.files.retrieve(file_path.file_id) citations.append(f'[{index}] Click to download {cited_file.filename}') # Note: File download functionality not implemented above for brevity # Add footnotes to the end of the message before displaying to user message_content.value += '\n' + '\n'.join(citations) Runs and Run Steps When you have all the context you need from your user in the Thread, you can run the Thread with an Assistant of your choice. python Copy■ 1 2 3 4 run = client.beta.threads.runs.create(thread_id=thread.id, assistant_id=assistant.id) By default, a Run will use the model and tools configuration specified in Assistant object, but you can override most of these when creating the Run for added flexibility: python Copy■ 1 2 3 4 5 6 7 run = client.beta.threads.runs.create(thread_id=thread.id, assistant_id=assistant.id, model="gpt-4-1106-preview", instructions="additional instructions", tools=[{"type": "code_interpreter"}, {"type": "retrieval"}]) Note: file_ids associated with the Assistant cannot be overridden during Run creation. Stella Page 73 Note: file_ids associated with the Assistant cannot be overridden during Run creation. You must use the modify Assistant endpoint to do this. Run lifecycle Run objects can have multiple statuses. STATUS DEFINITION queued When Runs are first created or when you complete the required_action, they are moved to a queued status. They should almost immediately move to in_progress. in_progr ess While in_progress, the Assistant uses the model and tools to perform steps. You can view progress being made by the Run by examining the Run Steps. complet ed The Run successfully completed! You can now view all Messages the Assistant added to the Thread, and all the steps the Run took. You can also continue the conversation by adding more user Messages to the Thread and creating another Run. requires _action When using the Function calling tool, the Run will move to a required_action state once the model determines the names and arguments of the functions to be called. You must then run those functions and submit the outputs before the run proceeds. If the outputs are not provided before the expires_at timestamp passes (roughly 10 mins past creation), the run will move to an expired status. expired This happens when the function calling outputs were not submitted before expires_at and the run expires. Additionally, if the runs take too long to execute and go beyond the time stated in expires_at, our systems will expire the run. cancelli ng You can attempt to cancel an in_progress run using the Cancel Run endpoint. Once the attempt to cancel succeeds, status of the Run moves to cancelled. Cancellation is attempted but not guaranteed. cancelle d Run was successfully cancelled. failed You can view the reason for the failure by looking at the last_error object in the Run. The timestamp for the failure will be recorded under failed_at. Polling for updates In order to keep the status of your run up to date, you will have to periodically retrieve the Run object. You can check the status of the run each time you retrieve the object to determine what your application should do next. We plan to add support for streaming to make this simpler in the near future. Thread locks Stella Page 74 When a Run is in_progress and not in a terminal state, the Thread is locked. This means that: • New Messages cannot be added to the Thread. • New Runs cannot be created on the Thread. Run steps Run step statuses have the same meaning as Run statuses. Most of the interesting detail in the Run Step object lives in the step_details field. There can be two types of step details: 1. message_creation: This Run Step is created when the Assistant creates a Message on the Thread. 2. tool_calls: This Run Step is created when the Assistant calls a tool. Details around this are covered in the relevant sections of the Tools guide. Limitations During this beta, there are several known limitations we are looking to address in the coming weeks and months. We will publish a changelog on this page when we add support for additional functionality. • Support for streaming output (including Messages and Run Steps). • Support for notifications to share object status updates without the need for polling. • Support for DALL·E as a tool. • Support for user message creation with images. Next 1. Learn more about Tools From Tools Beta Give Assistants access to OpenAI-hosted tools like Code Interpreter and Knowledge Retrieval, or build your own tools using Function calling. The Assistants API is in beta and we are actively working on adding more functionality. Share your feedback in our Developer Forum! Code Interpreter Code Interpreter allows the Assistants API to write and run Python code in a sandboxed execution environment. This tool can process files with diverse data and formatting, and generate files with data Stella Page 75 environment. This tool can process files with diverse data and formatting, and generate files with data and images of graphs. Code Interpreter allows your Assistant to run code iteratively to solve challenging code and math problems. When your Assistant writes code that fails to run, it can iterate on this code by attempting to run different code until the code execution succeeds. Enabling Code Interpreter Pass the code_interpreterin the tools parameter of the Assistant object to enable Code Interpreter: python Copy■ 1 2 3 4 5 assistant = client.beta.assistants.create(instructions="You are a personal math tutor. When asked a math question, write and run code to answer the question.", model="gpt-4-1106-preview", tools=[{"type": "code_interpreter"}]) The model then decides when to invoke Code Interpreter in a Run based on the nature of the user request. This behavior can be promoted by prompting in the Assistant's instructions (e.g., “write code to solve this problem”). Passing files to Code Interpreter Code Interpreter can parse data from files. This is useful when you want to provide a large volume of data to the Assistant or allow your users to upload their own files for analysis. Files that are passed at the Assistant level are accessible by all Runs with this Assistant: python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 # Upload a file with an "assistants" purpose file = client.files.create(file=open("speech.py", "rb"), purpose='assistants') # Create an assistant using the file ID assistant = client.beta.assistants.create(instructions="You are a personal math tutor. When asked a math question, write and run code to answer the question.", model="gpt-4-1106-preview", tools=[{"type": "code_interpreter"}], file_ids=[file.id]) Stella Page 76) Files can also be passed at the Thread level. These files are only accessible in the specific Thread. Upload the File using the File upload endpoint and then pass the File ID as part of the Message creation request: python Copy■ 1 2 3 4 5 6 7 8 9 thread = client.beta.threads.create(messages=[{ "role": "user", "content": "I need to solve the equation `3x + 11 = 14`. Can you help me?", "file_ids": [file.id] }]) Files have a maximum size of 512 MB. Code Interpreter supports a variety of file formats including .csv, .pdf, .json and many more. More details on the file extensions (and their corresponding MIME-types) supported can be found in the Supported files section below. Reading images and files generated by Code Interpreter Code Interpreter in the API also outputs files, such as generating image diagrams, CSVs, and PDFs. There are two types of files that are generated: 1. Images 2. Data files (e.g. a csv file with data generated by the Assistant) When Code Interpreter generates an image, you can look up and download this file in the file_id field of the Assistant Message response: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 { "id": "msg_OHGpsFRGFYmz69MM1u8KYCwf", "object": "thread.message", "created_at": 1698964262, Stella Page 77 "created_at": 1698964262, "thread_id": "thread_uqorHcTs46BZhYMyPn6Mg5gW", "role": "assistant", "content": [{ "type": "image_file", "image_file": { "file_id": "file-WsgZPYWAauPuW4uvcgNUGcb" } }] # ... } The file content can then be downloaded by passing the file ID to the Files API: python Copy■ content = client.files.retrieve_content(file.id) When Code Interpreter references a file path (e.g., ”Download this csv file”), file paths are listed as annotations. You can convert these annotations into links to download the file: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 { "id": "msg_3jyIh3DgunZSNMCOORflDyih", "object": "thread.message", "created_at": 1699073585, "thread_id": "thread_ZRvNTPOoYVGssUZr3G8cRRzE", "role": "assistant", "content": [{ "type": "text", "text": { "value": "The rows of the CSV file have been shuffled and saved to a new CSV file. You can download the shuffled CSV file from the following link:\n\n[Download Shuffled CSV File](sandbox:/mnt/data/shuffled_file.csv)", "annotations": [Stella Page 78 "annotations": [{ "type": "file_path", "text": "sandbox:/mnt/data/shuffled_file.csv", "start_index": 167, "end_index": 202, "file_path": { "file_id": "file-oSgJAzAnnQkVB3u7yCoE9CBe" } } ... Input and output logs of Code Interpreter By listing the steps of a Run that called Code Interpreter, you can inspect the code input and outputs logs of Code Interpreter: python Copy■ 1 2 3 4 run_steps = client.beta.threads.runs.steps.list(thread_id=thread.id, run_id=run.id) 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 { "object": "list", "data": [{ "id": "step_DQfPq3JPu8hRKW0ctAraWC9s", "object": "assistant.run.step", "type": "tool_calls", "run_id": "run_kme4a442kme4a442", Stella Page 79 "run_id": "run_kme4a442kme4a442", "thread_id": "thread_34p0sfdas0823smfv", "status": "completed", "step_details": { "type": "tool_calls", "tool_calls": [{ "type": "code", "code": { "input": "# Calculating 2 + 2\nresult = 2 + 2\nresult", "outputs": [{ "type": "logs", "logs": "4" } ... } Knowledge Retrieval Retrieval augments the Assistant with knowledge from outside its model, such as proprietary product information or documents provided by your users. Once a file is uploaded and passed to the Assistant, OpenAI will automatically chunk your documents, index and store the embeddings, and implement vector search to retrieve relevant content to answer user queries. Enabling Retrieval Pass the retrieval in the tools parameter of the Assistant to enable Retrieval: python Copy■ 1 2 3 4 5 assistant = client.beta.assistants.create(instructions="You are a customer support chatbot. Use your knowledge base to best respond to customer queries.", model="gpt-4-1106-preview", tools=[{"type": "retrieval"}]) How it works The model then decides when to retrieve content based on the user Messages. The Assistants API automatically chooses between two retrieval techniques: 1. it either passes the file content in the prompt for short documents, or 2. performs a vector search for longer documents Retrieval currently optimizes for quality by adding all relevant content to the context of model calls. We plan to introduce other retrieval strategies to enable developers to choose a different tradeoff between retrieval quality and model usage cost. Uploading files for retrieval Similar to Code Interpreter, files can be passed at the Assistant-level or at the Thread-level python Copy■ Stella Page 80 Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 # Upload a file with an "assistants" purpose file = client.files.create(file=open("knowledge.pdf", "rb"), purpose='assistants') # Add the file to the assistant assistant = client.beta.assistants.create(instructions="You are a customer support chatbot. Use your knowledge base to best respond to customer queries.", model="gpt-4-1106-preview", tools=[{"type": "retrieval"}], file_ids=[file.id]) Files can also be added to a Message in a Thread. These files are only accessible within this specific thread. After having uploaded a file, you can pass the ID of this File when creating the Message: python Copy■ 1 2 3 4 5 6 message = client.beta.threads.messages.create(thread_id=thread.id, role="user", content="I can't find in the PDF manual how to turn off this device.", file_ids=[file.id]) Maximum file size is

512MB. Retrieval supports a variety of file formats including .pdf, .md, .docx and many more. More details on the file extensions (and their corresponding MIME-types) supported can be found in the Supported files section below. Deleting files To remove a file from the assistant, you can detach the file from the assistant: python Copy■ 1 2 3 4 file_deletion_status = client.beta.assistants.files.delete(assistant_id=assistant.id, file_id=file.id Stella Page 81 file_id=file.id) Detaching the file from the assistant removes the file from the retrieval index as well. File citations When Code Interpreter outputs file paths in a Message, you can convert them to corresponding file downloads using the annotations field. See the Annotations section for an example of how to do this. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 { "id": "msg_3jyIh3DgunZSNMCOORflDyih", "object": "thread.message", "created_at": 1699073585, "thread_id": "thread_ZRvNTPOoYVGssUZr3G8cRRzE", "role": "assistant", "content": [{ "type": "text", "text": { "value": "The rows of the CSV file have been shuffled and saved to a new CSV file. You can download the shuffled CSV file from the following link:\n\n[Download Shuffled CSV File] (sandbox:/mnt/data/shuffled_file.csv)", "annotations": [{ "type": "file_path", "text": "sandbox:/mnt/data/shuffled_file.csv", "start_index": 167, Stella Page 82 "start_index": 167, "end_index": 202, "file_path": { "file_id": "file-oSgJAzAnnQkVB3u7yCoE9CBe" } }] } }], "file_ids": ["file-oSgJAzAnnQkVB3u7yCoE9CBe"], ... }, Function calling Similar to the Chat Completions API, the Assistants API supports function calling. Function calling allows you to describe functions to the Assistants and have it intelligently return the functions that need to be called along with their arguments. The Assistants API will pause execution during a Run when it invokes functions, and you can supply the results of the function call back to continue the Run execution. Defining functions First, define your functions when creating an Assistant: python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 assistant = client.beta.assistants.create(Stella Page 83 assistant = client.beta.assistants.create(instructions="You are a weather bot. Use the provided functions to answer questions.", model="gpt-4-1106-preview", tools=[{ "type": "function", "function": { "name": "getCurrentWeather", "description": "Get the weather in location", "parameters": { "type": "object", "properties": { "location": {"type": "string", "description": "The city and state e.g. San Francisco, CA"}, "unit": {"type": "string", "enum": ["c", "f"]} }, "required": ["location"] } } }, { "type": "function", "function": { "name": "getNickname", "description": "Get the nickname of a city", "parameters": { "type": "object", "properties": { "location": {"type": "string", "description": "The city and state e.g. San Francisco, CA"}, }, "required": ["location"] } } }]) Reading the functions called by the Assistant When you initiate a Run with a user Message that triggers the function, the Run will enter a requires_action status. The model can provide multiple functions to call at once via the parallel function calling feature: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 Stella Page 84 23 24 25 26 27 28 29 30 { "id": "run_3HV7rrQsagiqZmYynKwEdcxS", "object": "thread.run", "assistant_id": "asst_rEEOF3OGMan2ChvEALwTQakP", "thread_id": "thread_dXgWKGf8Cb7md8p0wKiMDGKc", "status": "requires_action", "required_action": { "type": "submit_tool_outputs", "submit_tool_outputs": { "tool_calls": [{ "tool_call_id": "call_Vt5AqcWr8QsRTNGv4cDIpsmA", "type": "function", "function": { "name": "getCurrentWeather", "arguments": "{\"location\":\"San Francisco\"}" } }, { "tool_call_id": "call_45y0df8230430n34f8saa", "type": "function", "function": { "name": "getNickname", "arguments": "{\"location\":\"Los Angeles\"}" } }] } }, ... Submitting functions outputs You can then complete the Run by submitting the output from the function(s) you call. Pass the tool_call_id referenced in the required_action object above to match output to each function call. python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 run = client.beta.threads.runs.submit_tool_outputs(Stella Page 85 run = client.beta.threads.runs.submit_tool_outputs(thread_id=thread.id, run_id=run.id, tool_outputs=[{ "tool_call_id": call_ids[0], "output": "22C", }, { "tool_call_id": call_ids[1], "output": "LA", },]) After submitting outputs, the run will enter the queued state before it continues it’s execution. Supported files For text/ MIME types, the encoding must be one of utf-8, utf-16, or ascii. FILE FORMAT MIME TYPE CODE INTERPRETER RETRIEVA L .c text/x-c .cpp text/x-c++ .csv application/csv .docx application/vnd.openxmlformats- officedocument.wordprocessingml.document .html text/html .java text/x-java .json application/json .md text/markdown .pdf application/pdf .php text/x-php .pptx application/vnd.openxmlformats- officedocument.presentationml.presentation .py text/x-python .py text/x-script.python .rb text/x-ruby .tex text/x-tex .txt text/plain .css text/css .jpeg image/jpeg .jpg image/jpeg .js text/javascript .gif image/gif .png image/png .tar application/x-tar .ts application/typescript Stella Page 86 .ts application/typescript .xlsx application/vnd.openxmlformats- officedocument.spreadsheetml.sheet .xml application/xml or "text/xml" .zip application/zip From Prompt engineering This guide shares strategies and tactics for getting better results from large language models (sometimes referred to as GPT models) like GPT-4. The methods described here can sometimes be deployed in combination for greater effect. We encourage experimentation to find the methods that work best for you. Some of the examples demonstrated here currently work only with our most capable model, gpt-4. In general, if you find that a model fails at a task and a more capable model is available, it's often worth trying again with the more capable model. Six strategies for getting better results Write clear instructions These■models■can’t■read■your■mind.■If■outputs■are■too■long,■ask■for■brief■replies.■If■ outputs are too simple, ask for expert-level writing. If you dislike the format, demonstrate the■format■you’d ■like■to■see.■The■less■the■model■has■to■guess■at■what■you■want,■the■ more■likely■you’ll■get■it. Tactics: • Include details in your query to get more relevant answers • Ask the model to adopt a persona • Use delimiters to clearly indicate distinct parts of the input • Specify the steps required to complete a task • Provide examples • Specify the desired length of the output Provide reference text Language models can confidently invent fake answers, especially when asked about esoteric topics or for citations and URLs. In the same way that a sheet of notes can help a student do better on a test, providing reference text to these models can help in answering with fewer fabrications. Tactics: • Instruct the model to answer using a reference text • Instruct the model to answer with citations from a reference text Split complex tasks into simpler subtasks Just as it is good practice in software engineering to decompose a complex system into a set of modular components, the same is true of tasks submitted to a language model. Complex tasks tend to have higher error rates than simpler tasks. Furthermore, complex Stella Page 87 Complex tasks tend to have higher error rates than simpler tasks. Furthermore, complex tasks can often be re-defined as a workflow of simpler tasks in which the outputs of earlier tasks are used to construct the inputs to later tasks. Tactics: • Use intent classification to identify the most relevant instructions for a user query • For dialogue applications that require very long conversations, summarize or filter previous dialogue • Summarize long documents piecewise and construct a full summary recursively Give the model time to "think" If asked to multiply 17 by 28, you might not know it instantly, but can still work it out with time. Similarly, models make more reasoning errors when trying to answer right away, rather than taking time to work out an answer. Asking for a "chain of thought" before an answer can help the model reason its way toward correct answers more reliably. Tactics: • Instruct the model to work out its own solution before rushing to a conclusion • Use inner monologue or a sequence of queries to hide the model's reasoning process • Ask the model if it missed anything on previous passes Use external tools Compensate for the weaknesses of the model by feeding it the outputs of other tools. For example, a text retrieval system (sometimes called RAG or retrieval augmented generation) can tell the model about relevant documents. A code execution engine like OpenAI's Code Interpreter can help the model do math and run code. If a task can be done more reliably or efficiently by a tool rather than by a language model, offload it to get the best of both. Tactics: • Use embeddings-based search to implement efficient knowledge retrieval • Use code execution to perform more accurate calculations or call external APIs • Give the model access to specific functions Test changes systematically Improving performance is easier if you can measure it. In some cases a modification to a prompt will achieve better performance on a few isolated examples but lead to worse overall performance on a more representative set of examples. Therefore to be sure that a change is net positive to performance it may be necessary to define a comprehensive test suite (also known an as an "eval"). Tactic: • Evaluate model outputs with reference to gold-standard answers Tactics Each of the strategies listed above can be instantiated with specific tactics. These tactics are meant to provide ideas for things to try. They are by no means fully comprehensive, and you should feel free to try creative ideas not represented here. Stella Page 88 Strategy: Write clear instructions Tactic: Include details in your query to get more relevant answers In order to get a highly relevant response, make sure that requests provide any important details or context. Otherwise you are leaving it up to the model to guess what you mean. Worse Better How do I add numbers in Excel? How do I add up a row of dollar amounts in Excel? I want to do this automatically for a whole sheet of rows with all the totals ending up on the right in a column called "Total". Who’s■president? Who was the president of Mexico in 2021, and how frequently are elections held? Write code to calculate the Fibonacci sequence. Write a TypeScript function to efficiently calculate the Fibonacci sequence. Comment the code liberally to explain what each piece does and why it's written that way. Summarize the meeting notes. Summarize the meeting notes in a single paragraph. Then write a markdown list of the speakers and each of their key points. Finally, list the next steps or action items suggested by the speakers, if any. Tactic: Ask the model to adopt a persona The system message can be used to specify the persona used by the model in its replies. SYSTEM When I ask for help to write something, you will reply with a document that contains at least one joke or playful comment in every paragraph. USER Write a thank you note to my steel bolt vendor for getting the delivery in on time and in short notice. This made it possible for us to deliver an important order. Open in Playground Tactic: Use delimiters to clearly indicate distinct parts of the input Delimiters like triple quotation marks, XML tags, section titles, etc. can help demarcate sections of text to be treated differently. USER Summarize the text delimited by triple quotes with a haiku. """insert text here""" Open in Playground SYSTEM You will be provided with a pair of articles (delimited with XML tags) about the same topic. First summarize the arguments of each article. Then indicate which of them makes a better argument and explain why. USER insert first article here insert second article here Open in Playground Stella Page 89 Open in Playground SYSTEM You will be provided with a thesis abstract and a suggested title for it. The thesis title should give the reader a good idea of the topic of the thesis but should also be eye- catching. If the title does not meet these criteria, suggest 5 alternatives. USER Abstract: insert abstract here Title: insert title here Open in Playground For straightforward tasks such as these, using delimiters might not make a difference in the output quality. However, the more complex a task is the more important it is to disambiguate■task■details .■Don't■make■the■model■work■to■understand■exactly■what■you■ are asking of them. Tactic: Specify the steps required to complete a task Some tasks are best specified as a sequence of steps. Writing the steps out explicitly can make it easier for the model to follow them. SYSTEM Use the following step-by-step instructions to respond to user inputs. Step 1 - The user will provide you with text in triple quotes. Summarize this text in one sentence with a prefix that says "Summary: ". Step 2 - Translate the summary from Step 1 into Spanish, with a prefix that says "Translation: ". USER """insert text here""" Open in Playground Tactic: Provide examples Providing general instructions that apply to all examples is generally more efficient than demonstrating all permutations of a task by example, but in some cases providing examples may be easier. For example, if you intend for the model to copy a particular style of responding to user queries which is difficult to describe explicitly. This is known as "few-shot" prompting. SYSTEM Answer in a consistent style. USER Teach me about patience. ASSISTANT The river that carves the deepest valley flows from a modest spring; the grandest symphony originates from a single note; the most intricate tapestry begins with a solitary thread. USER Teach me about the ocean. Open in Playground Tactic: Specify the desired length of the output You can ask the model to produce outputs that are of a given target length. The targeted output length can be specified in terms of the count of words, sentences, Stella Page 90 targeted output length can be specified in terms of the count of words, sentences, paragraphs, bullet points, etc. Note however that instructing the model to generate a specific number of words does not work with high precision. The model can more reliably generate outputs with a specific number of paragraphs or bullet points. USER Summarize the text delimited by triple quotes in about 50 words. """insert text here""" Open in Playground USER Summarize the text delimited by triple quotes in 2 paragraphs. """insert text here""" Open in Playground USER Summarize the text delimited by triple quotes in 3 bullet points. """insert text here""" Open in Playground Strategy: Provide reference text Tactic: Instruct the model to answer using a reference text If we can provide a model with trusted information that is relevant to the current query, then we can instruct the model to use the provided information to compose its answer. SYSTEM Use the provided articles delimited by triple quotes to answer questions. If the answer cannot be found in the articles, write "I could not find an answer." USER Question: Open in Playground Given that all models have limited context windows, we need some way to dynamically lookup information that is relevant to the question being asked. Embeddings can be used to implement efficient knowledge retrieval. See the tactic "Use embeddings-based search to implement efficient knowledge retrieval" for more details on how to implement this. Tactic: Instruct the model to answer with citations from a reference text If the input has been supplemented with relevant knowledge, it's straightforward to request that the model add citations to its answers by referencing passages from provided documents. Note that citations in the output can then be verified programmatically by string matching within the provided documents. SYSTEM You will be provided with a document delimited by triple quotes and a question. Your task is to answer the question using only the provided document and to cite the passage(s) of the document used to answer the question. If the document does not contain the information needed to answer this question then simply write: "Insufficient information." If an answer to the question is provided, it must be annotated with a citation.■Use■the■following■format■for■to■cite■relevant■passages■({"citation":■…}).■ Stella Page 91 citation.■Use■the■following■format■for■to■cite■relevant■passages■({"citation":■…}).■ USER """""" Question: Open in Playground Strategy: Split complex tasks into simpler subtasks Tactic: Use intent classification to identify the most relevant instructions for a user query For tasks in which lots of independent sets of instructions are needed to handle different cases, it can be beneficial to first classify the type of query and to use that classification to determine which instructions are needed. This can be achieved by defining fixed categories and hardcoding instructions that are relevant for handling tasks in a given category. This process can also be applied recursively to decompose a task into a sequence of stages. The advantage of this approach is that each query will contain only those instructions that are required to perform the next stage of a task which can result in lower error rates compared to using a single query to perform the whole task. This can also result in lower costs since larger prompts cost more to run (see pricing information). Suppose for example that for a customer service application, queries could be usefully classified as follows: SYSTEM You will be provided with customer service queries. Classify each query into a primary category and a secondary category. Provide your output in json format with the keys: primary and secondary. Primary categories: Billing, Technical Support, Account Management, or General Inquiry. Billing secondary categories: - Unsubscribe or upgrade - Add a payment method - Explanation for charge - Dispute a charge Technical Support secondary categories: - Troubleshooting - Device compatibility - Software updates Account Management secondary categories: - Password reset - Update personal information - Close account - Account security General Inquiry secondary categories: - Product information - Pricing - Feedback - Speak to a human USER I need to get my internet working again. Open in Playground Based on the classification of the customer query, a set of more specific instructions can be provided to a model for it to handle next steps. For example, suppose the customer requires help with "troubleshooting". SYSTEM You will be provided with customer service inquiries that require troubleshooting in a technical support context. Help the user by: - Ask them to check that all cables to/from the router are connected. Note that it is common for cables to come loose over time. - If all cables are connected and the issue persists, ask them which router model they are using - Now you will advise them how to restart their device: -- If the model number is MTD-327J, advise them to push the red button and hold it for 5 seconds, then wait 5 minutes before testing the connection. -- If the model number is MTD-327S, advise them to unplug and replug it, then wait 5 minutes before testing the connection. - If the customer's issue persists after restarting the device and waiting 5 minutes, connect them to IT support by outputting {"IT support requested"}. - If the user starts asking Stella Page 92 them to IT support by outputting {"IT support requested"}. - If the user starts asking questions that are unrelated to this topic then confirm if they would like to end the current chat about troubleshooting and classify their request according to the following scheme: USER I need to get my internet working again. Open in Playground Notice that the model has been instructed to emit special strings to indicate when the state of the conversation changes. This enables us to turn our system into a state machine where the state determines which instructions are injected. By keeping track of state, what instructions are relevant at that state, and also optionally what state transitions are allowed from that state, we can put guardrails around the user experience that would be hard to achieve with a less structured approach. Tactic: For dialogue applications that require very long conversations, summarize or filter previous dialogue Since models have a fixed context length, dialogue between a user and an assistant in which the entire conversation is included in the context window cannot continue indefinitely. There are various workarounds to this problem, one of which is to summarize previous turns in the conversation. Once the size of the input reaches a predetermined threshold length, this could trigger a query that summarizes part of the conversation and the summary of the prior conversation could be included as part of the system message. Alternatively, prior conversation could be summarized asynchronously in the background throughout the entire conversation. An alternative solution is to dynamically select previous parts of the conversation that are most relevant to the current query. See the tactic "Use embeddings-based search to implement efficient knowledge retrieval". Tactic: Summarize long documents piecewise and construct a full summary recursively Since models have a fixed context length, they cannot be used to summarize a text longer than the context length minus the length of the generated summary in a single query. To summarize a very long document such as a book we can use a sequence of queries to summarize each section of the document. Section summaries can be concatenated and summarized producing summaries of summaries. This process can proceed recursively■until■an■entire■document■is■summarized.■If■it's■necessary■to■use■information■ about earlier sections in order to make sense of later sections, then a further trick that can be useful is to include a running summary of the text that precedes any given point in the book while summarizing content at that point. The effectiveness of this procedure for summarizing books has been studied in previous research by OpenAI using variants of GPT-3. Strategy: Give models time to "think" Tactic: Instruct the model to work out its own solution before rushing to a conclusion Stella Page 93 a conclusion Sometimes we get better results when we explicitly instruct the model to reason from first principles before coming to a conclusion. Suppose for example we want a model to evaluate■a■student's■solution■to■a■math■problem.■The■most■obvious■way■to■approach■this■ is to simply ask the model if the student's solution is correct or not. SYSTEM Determine if the student's solution is correct or not. USER Problem Statement: I'm building a solar power installation and I need help working out the financials. - Land costs $100 / square foot - I can buy solar panels for $250 / square foot - I negotiated a contract for maintenance that will cost me a flat $100k per year, and an additional $10 / square foot What is the total cost for the first year of operations as a function of the number of square feet. Student's Solution: Let x be the size of the installation in square feet. 1. Land cost: 100x 2. Solar panel cost: 250x 3. Maintenance cost: 100,000 + 100x Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000 ASSISTANT The student's solution is correct. Open in Playground But the student's solution is actually not correct! We can get the model to successfully notice this by prompting the model to generate its own solution first. SYSTEM First work out your own solution to the problem. Then compare your solution to the student's solution and evaluate if the student's solution is correct or not. Don't decide if the student's solution is correct until you have done the problem yourself. USER Problem Statement: I'm building a solar power installation and I need help working out the financials. - Land costs $100 / square foot - I can buy solar panels for $250 / square foot - I negotiated a contract for maintenance that will cost me a flat $100k per year, and an additional $10 / square foot What is the total cost for the first year of operations as a function of the number of square feet. Student's Solution: Let x be the size of the installation in square feet. 1. Land cost: 100x 2. Solar panel cost: 250x 3. Maintenance cost: 100,000 + 100x Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000 ASSISTANT Let x be the size of the installation in square feet. 1. Land cost: 100x 2. Solar panel cost: 250x 3. Maintenance cost: 100,000 + 10x Total cost: 100x + 250x + 100,000 + 10x = 360x + 100,000 The student's solution is incorrect. They made an error in the maintenance cost calculation, using 100x instead of 10x. The correct total cost for the first year of operations is 360x + 100,000. Open in Playground Tactic: Use inner monologue or a sequence of queries to hide the model's reasoning process The previous tactic demonstrates that it is sometimes important for the model to reason in detail about a problem before answering a specific question. For some applications, the reasoning process that a model uses to arrive at a final answer would be inappropriate to share with the user. For example, in tutoring applications we may want to ■encourage■students■to■work■out■their■own■answers,■but■a■model's■reasoning■process■ about■the■student's■solution■could■reveal■the■answer■to■the■student. Stella Page 94 about■the■student's■solution■could■reveal■the■answer■to■the■student. Inner monologue is a tactic that can be used to mitigate this. The idea of inner monologue is to instruct the model to put parts of the output that are meant to be hidden from the user into a structured format that makes parsing them easy. Then before presenting the output to the user, the output is parsed and only part of the output is made visible. SYSTEM Follow these steps to answer the user queries. Step 1 - First work out your own solution to the problem. Don't rely on the student's solution since it may be incorrect. Enclose all your work for this step within triple quotes ("""). Step 2 - Compare your solution to the student's solution and evaluate if the student's solution is correct or not. Enclose all your work for this step within triple quotes ("""). Step 3 - If the student made a mistake, determine what hint you could give the student without giving away the answer. Enclose all your work for this step within triple quotes ("""). Step 4 - If the student made a mistake, provide the hint from the previous step to the student (outside of triple quotes). Instead of writing "Step 4 - ..." write "Hint:". USER Problem Statement: Student Solution: Open in Playground Alternatively, this can be achieved with a sequence of queries in which all except the last have their output hidden from the end user. First, we can ask the model to solve the problem on its own. Since this initial query doesn't■require■the■student’s■solution,■it■can■be■omitted.■This■provides■the■additional■ advantage■that■there■is■no■chance■that■the■model’s■solution■will■be■biased■by■the■ student’s■attempted■solution. USER Open in Playground Next, we can have the model use all available information to assess the correctness of the■student’s■solution. SYSTEM Compare your solution to the student's solution and evaluate if the student's solution is correct or not. USER Problem statement: """""" Your solution: """"""■Student’s■solution:■"""""" Open in Playground Finally, we can let the model use its own analysis to construct a reply in the persona of a helpful tutor. SYSTEM You are a math tutor. If the student made an error, offer a hint to the student in a way

You are a math tutor. If the student made an error, offer a hint to the student in a way that does not reveal the answer. If the student did not make an error, simply offer them an encouraging comment. USER Problem statement: """""" Your solution: """"""■Student’s■solution:■""""""■Analysis:■ """""" Open in Playground Tactic: Ask the model if it missed anything on previous passes Suppose that we are using a model to list excerpts from a source which are relevant to a particular question. After listing each excerpt the model needs to determine if it should start writing another or if it should stop. If the source document is large, it is common for a model to stop too early and fail to list all relevant excerpts. In that case, better performance can often be obtained by prompting the model with followup queries to find any excerpts it missed on previous passes. SYSTEM You will be provided with a document delimited by triple quotes. Your task is to select excerpts which pertain to the following question: "What significant paradigm shifts have occurred in the history of artificial intelligence." Ensure that excerpts contain all relevant context needed to interpret them - in other words don't extract small snippets that are missing important context. Provide output in JSON format as follows: [{"excerpt": "..."}, ... {"excerpt": "..."}] USER """""" ASSISTANT [{"excerpt": "the model writes an excerpt here"}, ... {"excerpt": "the model writes another excerpt here"}] USER Are there more relevant excerpts? Take care not to repeat excerpts. Also ensure that excerpts contain all relevant context needed to interpret them - in other words don't extract small snippets that are missing important context. Open in Playground Strategy: Use external tools Tactic: Use embeddings-based search to implement efficient knowledge retrieval A model can leverage external sources of information if provided as part of its input. This can help the model to generate more informed and up-to-date responses. For example, if a user asks a question about a specific movie, it may be useful to add high quality■information■about■the■movie■(e.g.■actors,■director,■etc…)■to■the■model’s■input.■ Embeddings can be used to implement efficient knowledge retrieval, so that relevant information can be added to the model input dynamically at run-time. A text embedding is a vector that can measure the relatedness between text strings. Similar or relevant strings will be closer together than unrelated strings. This fact, along with the existence of fast vector search algorithms means that embeddings can be used to implement efficient knowledge retrieval. In particular, a text corpus can be split up into chunks, and each chunk can be embedded and stored. Then a given query can be

into chunks, and each chunk can be embedded and stored. Then a given query can be embedded and vector search can be performed to find the embedded chunks of text from the corpus that are most related to the query (i.e. closest together in the embedding space). Example implementations can be found in the OpenAI Cookbook. See the tactic “Instruct■the■model■to■use■retrieved■knowledge■to■answer■queries” for an example of how to use knowledge retrieval to minimize the likelihood that a model will make up incorrect facts. Tactic: Use code execution to perform more accurate calculations or call external APIs Language models cannot be relied upon to perform arithmetic or long calculations accurately on their own. In cases where this is needed, a model can be instructed to write and run code instead of making its own calculations. In particular, a model can be instructed to put code that is meant to be run into a designated format such as triple backtick. After an output is produced, the code can be extracted and run. Finally, if necessary, the output from the code execution engine (i.e. Python interpreter) can be provided as an input to the model for the next query. SYSTEM You can write and execute Python code by enclosing it in triple backticks, e.g. ```code goes here```. Use this to perform calculations. USER Find all real-valued roots of the following polynomial: 3*x**5 - 5*x**4 - 3*x**3 - 7*x - 10. Open in Playground Another good use case for code execution is calling external APIs. If a model is instructed in the proper use of an API, it can write code that makes use of it. A model can be instructed in how to use an API by providing it with documentation and/or code samples showing how to use the API. SYSTEM You can write and execute Python code by enclosing it in triple backticks. Also note that you have access to the following module to help users send messages to their friends: ```python import message message.write(to="John", message="Hey, want to meetup after work?")``` Open in Playground WARNING: Executing code produced by a model is not inherently safe and precautions should be taken in any application that seeks to do this. In particular, a sandboxed code execution environment is needed to limit the harm that untrusted code could cause. Tactic: Give the model access to specific functions The Chat Completions API allows passing a list of function descriptions in requests. This enables models to generate function arguments according to the provided schemas. Generated function arguments are returned by the API in JSON format and can be used to execute function calls. Output provided by function calls can then be fed back into a model in the following request to close the loop. This is the recommended way of using OpenAI models to call external functions. To learn more see the function Stella Page 97 way of using OpenAI models to call external functions. To learn more see the function calling section in our introductory text generation guide and more function calling examples in the OpenAI Cookbook. Strategy: Test changes systematically Sometimes it can be hard to tell whether a change — e.g., a new instruction or a new design — makes your system better or worse. Looking at a few examples may hint at which is better, but with small sample sizes it can be hard to distinguish between a true improvement or random luck. Maybe the change helps performance on some inputs, but hurts performance on others. Evaluation procedures (or "evals") are useful for optimizing system designs. Good evals are: • Representative of real-world usage (or at least diverse) • Contain many test cases for greater statistical power (see table below for guidelines) • Easy to automate or repeat DIFFERENCE TO DETECT SAMPLE SIZE NEEDED FOR 95% CONFIDENCE 30% ~10 10% ~100 3% ~1,000 1% ~10,000 Evaluation of outputs can be done by computers, humans, or a mix. Computers can automate evals with objective criteria (e.g., questions with single correct answers) as well as some subjective or fuzzy criteria, in which model outputs are evaluated by other model queries. OpenAI Evals is an open-source software framework that provides tools for creating automated evals. Model-based evals can be useful when there exists a range of possible outputs that would be considered equally high in quality (e.g. for questions with long answers). The boundary between what can be realistically evaluated with a model-based eval and what requires a human to evaluate is fuzzy and is constantly shifting as models become more capable. We encourage experimentation to figure out how well model-based evals can work for your use case. Tactic: Evaluate model outputs with reference to gold-standard answers Suppose it is known that the correct answer to a question should make reference to a specific set of known facts. Then we can use a model query to count how many of the required facts are included in the answer. For example, using the following system message: SYSTEM You will be provided with text delimited by triple quotes that is supposed to be the answer to a question. Check if the following pieces of information are directly contained in the answer: - Neil Armstrong was the first person to walk on the moon. - The date Neil Armstrong first walked on the moon was July 21, 1969. For each of these points perform the following steps: 1 - Restate the point. 2 - Provide a citation from the answer which is closest to this point. 3 - Consider if someone reading the citation who doesn't know the topic could directly infer the point. Explain why or why not before making up Stella Page 98 know the topic could directly infer the point. Explain why or why not before making up your mind. 4 - Write "yes" if the answer to 3 was yes, otherwise write "no". Finally, provide a count of how many "yes" answers there are. Provide this count as {"count": }. Here's an example input where both points are satisfied: SYSTEM USER """Neil Armstrong is famous for being the first human to set foot on the Moon. This historic event took place on July 21, 1969, during the Apollo 11 mission.""" Open in Playground Here's an example input where only one point is satisfied: SYSTEM USER """Neil Armstrong made history when he stepped off the lunar module, becoming the first person to walk on the moon.""" Open in Playground Here's an example input where none are satisfied: SYSTEM USER """In the summer of '69, a voyage grand, Apollo 11, bold as legend's hand. Armstrong took a step, history unfurled, "One small step," he said, for a new world.""" Open in Playground There are many possible variants on this type of model-based eval. Consider the following variation which tracks the kind of overlap between the candidate answer and the gold-standard answer, and also tracks whether the candidate answer contradicts any part of the gold-standard answer. SYSTEM Use the following steps to respond to user inputs. Fully restate each step before proceeding. i.e. "Step 1: Reason...". Step 1: Reason step-by-step about whether the information in the submitted answer compared to the expert answer is either: disjoint, equal, a subset, a superset, or overlapping (i.e. some intersection but not subset/superset). Step 2: Reason step-by-step about whether the submitted answer contradicts any aspect of the expert answer. Step

3: Output a JSON object structured like: {"type_of_overlap": "disjoint" or "equal" or "subset" or "superset" or "overlapping", "contradiction": true or false} Here's an example input with a substandard answer which nonetheless does not contradict the expert answer: SYSTEM USER Stella Page 99 USER Question: """What event is Neil Armstrong most famous for and on what date did it occur? Assume UTC time.""" Submitted Answer: """Didn't he walk on the moon or something?""" Expert Answer: """Neil Armstrong is most famous for being the first person to walk on the moon. This historic event occurred on July 21, 1969.""" Open in Playground Here's an example input with answer that directly contradicts the expert answer: SYSTEM USER Question: """What event is Neil Armstrong most famous for and on what date did it occur? Assume UTC time.""" Submitted Answer: """On the 21st of July 1969, Neil Armstrong became the second person to walk on the moon, following after Buzz Aldrin.""" Expert Answer: """Neil Armstrong is most famous for being the first person to walk on the moon. This historic event occurred on July 21, 1969.""" Open in Playground Here's an example input with a correct answer that also provides a bit more detail than is necessary: SYSTEM USER Question: """What event is Neil Armstrong most famous for and on what date did it occur? Assume UTC time.""" Submitted Answer: """At approximately 02:56 UTC on July 21st 1969, Neil Armstrong became the first human to set foot on the lunar surface, marking a monumental achievement in human history.""" Expert Answer: """Neil Armstrong is most famous for being the first person to walk on the moon. This historic event occurred on July 21, 1969.""" Open in Playground Other resources For more inspiration, visit the OpenAI Cookbook, which contains example code and also links to third-party resources such as: • Prompting libraries & tools • Prompting guides • Video courses • Papers on advanced prompting to improve reasoning From Production best practices This guide provides a comprehensive set of best practices to help you transition from prototype to production. Whether you are a seasoned machine learning engineer or a Stella Page 100 prototype to production. Whether you are a seasoned machine learning engineer or a recent enthusiast, this guide should provide you with the tools you need to successfully put the platform to work in a production setting: from securing access to our API to designing a robust architecture that can handle high traffic volumes. Use this guide to help develop a plan for deploying your application as smoothly and effectively as possible. Setting up your organization Once you log in to your OpenAI account, you can find your organization name and ID in your organization settings. The organization name is the label for your organization, shown in user interfaces. The organization ID is the unique identifier for your organization which can be used in API requests. Users who belong to multiple organizations can pass a header to specify which organization is used for an API request. Usage from these API requests will count against the specified organization's quota. If no header is provided, the default organization will be billed. You can change your default organization in your user settings. You can invite new members to your organization from the Team page. Members can be readers or owners. Readers can make API requests and view basic organization information, while owners can modify billing information and manage members within an organization. Managing billing limits New free trial users receive an initial credit of $5 that expires after three months. Once the credit has been used or expires, you can choose to enter billing information to continue your use of the API. If no billing information is entered, you will still have login access but will be unable to make any further API requests. Once you’ve entered your billing information, you will have an approved usage limit of $100 per month, which is set by OpenAI. Your quota limit will automatically increase as your usage on your platform increases and you move from one usage tier to another. You can review your current usage limit in the limits page in your account settings. If you’d like to be notified when your usage exceeds a certain dollar amount, you can set a notification threshold through the usage limits page. When the notification threshold is reached, the owners of the organization will receive an email notification. You can also set a monthly budget so that, once the monthly budget is reached, any subsequent API requests will be rejected. Note that these limits are best effort, and there may be 5 to 10 minutes of delay between the usage and the limits being enforced. API keys The OpenAI API uses API keys for authentication. Visit your API keys page to retrieve the API key you'll use in your requests. This is a relatively straightforward way to control access, but you must be vigilant about securing these keys. Avoid exposing the API keys in your code or in public repositories; instead, store them in a secure location. You should expose your keys to your application using environment variables or secret management service, so that you don't need to hard-code them in your codebase. Read more in our Best practices for API key safety. Staging accounts Stella Page 101 Staging accounts As you scale, you may want to create separate organizations for your staging and production environments. Please note that you can sign up using two separate email addresses like bob+prod@widgetcorp.com and bob+dev@widgetcorp.com to create two organizations. This will allow you to isolate your development and testing work so you don't accidentally disrupt your live application. You can also limit access to your production organization this way. Scaling your solution architecture When designing your application or service for production that uses our API, it's important to consider how you will scale to meet traffic demands. There are a few key areas you will need to consider regardless of the cloud service provider of your choice: • Horizontal scaling: You may want to scale your application out horizontally to accommodate requests to your application that come from multiple sources. This could involve deploying additional servers or containers to distribute the load. If you opt for this type of scaling, make sure that your architecture is designed to handle multiple nodes and that you have mechanisms in place to balance the load between them. • Vertical scaling: Another option is to scale your application up vertically, meaning you can beef up the resources available to a single node. This would involve upgrading your server's capabilities to handle the additional load. If you opt for this type of scaling, make sure your application is designed to take advantage of these additional resources. • Caching: By storing frequently accessed data, you can improve response times without needing to make repeated calls to our API. Your application will need to be designed to use cached data whenever possible and invalidate the cache when new information is added. There are a few different ways you could do this. For example, you could store data in a database, filesystem, or in-memory cache, depending on what makes the most sense for your application. • Load balancing: Finally, consider load-balancing techniques to ensure requests are distributed evenly across your available servers. This could involve using a load balancer in front of your servers or using DNS round-robin. Balancing the load will help improve performance and reduce bottlenecks. Managing rate limits When using our API, it's important to understand and plan for rate limits. Improving latencies Latency is the time it takes for a request to be processed and a response to be returned. In this section, we will discuss some factors that influence the latency of our text generation models and provide suggestions on how to reduce it. The latency of a completion request is mostly influenced by two factors: the model and the number of tokens generated. The life cycle of a completion request looks like this: Network End user to API latency Server Time to process prompt tokens Server Stella Page 102 Server Time to sample/generate tokens Network API to end user latency The bulk of the latency typically arises from the token generation step. Intuition: Prompt tokens add very little latency to completion calls. Time to generate completion tokens is much longer, as tokens are generated one at a time. Longer generation lengths will accumulate latency due to generation required for each token. Common factors affecting latency and possible mitigation techniques Now■that■we■have■looked■at■the ■basics■of■latency,■let’s■take■a■look■at■various■factors■that■ can affect latency, broadly ordered from most impactful to least impactful. Model Our API offers different models with varying levels of complexity and generality. The most capable models, such as gpt-4, can generate more complex and diverse completions, but they also take longer to process your query. Models such as gpt-3.5- turbo, can generate faster and cheaper chat completions, but they may generate results that are less accurate or relevant for your query. You can choose the model that best suits your use case and the trade-off between speed and quality. Number of completion tokens Requesting a large amount of generated tokens completions can lead to increased latencies: • Lower max tokens: for requests with a similar token generation count, those that have a lower max_tokens parameter incur less latency. • Include stop sequences: to prevent generating unneeded tokens, add a stop sequence. For example, you can use stop sequences to generate a list with a specific number of items. In this case, by using 11. as a stop sequence, you can generate a list with only 10 items, since the completion will stop when 11. is reached. Read our help article on stop sequences for more context on how you can do this. • Generate fewer completions: lower the values of n and best_of when possible where n refers to how many completions to generate for each prompt and best_of is used to represent the result with the highest log probability per token. If n and best_of both equal 1 (which is the default), the number of generated tokens will be at most, equal to max_tokens. If n (the number of completions returned) or best_of (the number of completions generated for consideration) are set to > 1, each request will create multiple outputs. Here, you can consider the number of generated tokens as [max_tokens * max (n, best_of)] Streaming Setting stream: true in a request makes the model start returning tokens as soon as they are available, instead of waiting for the full sequence of tokens to be generated. It does not change the time to get all the tokens, but it reduces the time for first token for an application where we want to show partial progress or are going to stop generations. This■can■be■a■better■user■experience■and■a■UX ■improvement■so■it’s■worth■experimenting■ with streaming. Stella Page 103 with streaming. Infrastructure Our servers are currently located in the US. While we hope to have global redundancy in the future, in the meantime you could consider locating the relevant parts of your infrastructure in the US to minimize the roundtrip time between your servers and the OpenAI servers. Batching Depending on your use case, batching may help. If you are sending multiple requests to the same endpoint, you can batch the prompts to be sent in the same request. This will reduce the number of requests you need to make. The prompt parameter can hold up to 20 unique prompts. We advise you to test out this method and see if it helps. In some cases, you may end up increasing the number of generated tokens which will slow the response time. Managing costs To monitor your costs, you can set a notification threshold in your account to receive an email alert once you pass a certain usage threshold. You can also set a monthly budget. Please be mindful of the potential for a monthly budget to cause disruptions to your application/users. Use the usage tracking dashboard to monitor your token usage during the current and past billing cycles. Text generation One of the challenges of moving your prototype into production is budgeting for the costs associated with running your application. OpenAI offers a pay-as-you-go pricing model, with prices per 1,000 tokens (roughly equal to 750 words). To estimate your costs, you will need to project the token utilization. Consider factors such as traffic levels, the frequency with which users will interact with your application, and the amount of data you will be processing. One useful framework for thinking about reducing costs is to consider costs as a function of the number of tokens and the cost per token. There are two potential avenues for reducing costs using this framework. First, you could work to reduce the cost per token by switching to smaller models for some tasks in order to reduce costs. Alternatively, you could try to reduce the number of tokens required. There are a few ways you could do this, such as by using shorter prompts, fine-tuning models, or caching common user queries so that they don't need to be processed repeatedly. You can experiment with our interactive tokenizer tool to help you estimate costs. The API■and■playground■also■returns■token■counts■as■part■of■the■response.■Once■you’ve■got■ things working with our most capable model, you can see if the other models can produce the same results with lower latency and costs. Learn more in our token usage help article. MLOps strategy As you move your prototype into production, you may want to consider developing an MLOps strategy. MLOps (machine learning operations) refers to the process of managing the end-to-end life cycle of your machine learning models, including any models you may be fine-tuning using our API. There are a number of areas to consider when designing your MLOps strategy. These include • Data and model management: managing the data used to train or fine-tune your Stella Page 104 • Data and model management: managing the data used to train or fine-tune your model and tracking versions and changes. • Model monitoring: tracking your model's performance over time and detecting any potential issues or degradation. • Model retraining: ensuring your model stays up to date with changes in data or evolving requirements and retraining or fine-tuning it as needed. • Model deployment: automating the process of deploying your model and related artifacts into production. Thinking through these aspects of your application will help ensure your model stays relevant and performs well over time. Security and compliance As you move your prototype into production, you will need to assess and address any security and compliance requirements that may apply to your application. This will involve examining the data you are handling, understanding how our API processes data, and determining what regulations you must adhere to. Our security practices and trust and compliance portal provide our most comprehensive and up-to- date documentation. For reference, here is our Privacy Policy and Terms of Use. Some common areas you'll need to consider include data storage, data transmission, and data retention. You might also need to implement data privacy protections, such as encryption or anonymization where possible. In addition, you should follow best practices for secure coding, such as input sanitization and proper error handling. Safety best practices When creating your application with our API, consider our safety best practices to ensure your application is safe and successful. These recommendations highlight the importance of testing the product extensively, being proactive about addressing potential issues, and limiting opportunities for misuse. From Safety best practices Use our free Moderation API OpenAI's Moderation API is free-to-use and can help reduce the frequency of unsafe content in your completions. Alternatively, you may wish to develop your own content filtration system tailored to your use case. Adversarial testing We■recommend■“red-teaming”■you r■application■to■ensure■it's■robust■to■adversarial■input.■ Test your product over a wide range of inputs and user behaviors, both a representative set■and■those■reflective■of■someone ■trying■to■‘break'■your■application.■Does■it■wander■off■ topic?■Can■someone■easily■redirect■the■feature■via■prompt■injections,■e.g.■“ignore■the■ previous■instructions■and■do■this■instead”? Human in the loop (HITL) Wherever possible, we recommend having a human review outputs before they are Stella Page 105 Wherever possible, we recommend having a human review outputs before they are used in practice. This is especially critical in high-stakes domains, and for code generation. Humans should be aware of the limitations of the system, and have access to any information needed to verify the outputs (for example, if the application summarizes notes, a human should have easy access to the original notes to refer back). Prompt engineering “Prompt■engineering”■can■help■constrain■the■topic■and■tone■of ■output■text.■This■reduces■ the chance of producing undesired content, even if a user tries to produce it. Providing additional context to the model (such as by giving a few high-quality examples of desired behavior prior to the new input) can make it easier to steer model outputs in desired directions. “Know your customer” (KYC) Users should generally need to register and log-in to access your service. Linking this service to an existing account, such as a Gmail, LinkedIn, or Facebook log-in, may help, though may not be appropriate for all use-cases. Requiring a credit card or ID card reduces risk further. Constrain user input and limit output tokens Limiting the amount of text a user can input into the prompt helps avoid prompt injection. Limiting the number of output tokens helps reduce the chance of misuse. Narrowing the ranges of inputs or outputs, especially drawn from trusted sources, reduces the extent of misuse possible within an application. Allowing user inputs through validated dropdown fields (e.g., a list of movies on Wikipedia) can be more secure than allowing open-ended text inputs. Returning outputs from a validated set of materials on the backend, where possible, can be safer than returning novel generated content (for instance, routing a customer query to the best-matching existing customer support article, rather than attempting to answer the query from-scratch). Allow users to report issues Users should generally have an easily-available method for reporting improper functionality or other concerns about application behavior (listed email address, ticket submission method, etc). This method should be monitored by a human and responded to as appropriate. Understand and communicate limitations From hallucinating inaccurate information, to offensive outputs, to bias, and much more, language models may not be suitable for every use case without significant modifications. Consider whether the model is fit for your purpose, and evaluate the performance of the API on a wide range of potential inputs in order to identify cases where the API's performance might drop. Consider your customer base and the range of inputs that they will be using, and ensure their expectations are calibrated appropriately. Safety and security are very important to us at OpenAI. If in the course of your development you do notice any safety or security issues with the Stella Page 106 If in the course of your development you do notice any safety or security issues with the API or anything else related to OpenAI, please submit these through our Coordinated Vulnerability Disclosure Program. End-user IDs Sending end-user IDs in your requests can be a useful tool to help OpenAI monitor and detect abuse. This allows OpenAI to provide your team with more actionable feedback in the event that we detect any policy violations in your application. The IDs should be a string that uniquely identifies each user. We recommend hashing their username or email address, in order to avoid sending us any identifying information. If you offer a preview of your product to non-logged in users, you can send a session ID instead. You can include end-user IDs in your API requests via the user parameter as follows: Example: Providing a user identifer python Copy■ 1 2 3 4 5 6 7 8 9 from openai import OpenAI client = OpenAI() response = client.completions.create(model="gpt-3.5-turbo-instruct", prompt="This is a test", max_tokens=5, user="user_123456") From Rate limits Rate limits are restrictions that our API imposes on the number of times a user or client can access our services within a specified period of time. Why do we have rate limits? Rate limits are a common practice for APIs, and they're put in place for a few different reasons: • They help protect against abuse or misuse of the API. For example, a malicious actor could flood the API with requests in an attempt to overload it or cause disruptions in service. By setting rate limits, OpenAI can prevent this kind of activity. • Rate limits help ensure that everyone has fair access to the API. If one person or organization makes an excessive number of requests, it could bog down the API for everyone else. By throttling the number of requests that a single user can make, OpenAI ensures that the most number of people have an opportunity to use the API without experiencing slowdowns. • Rate limits can help OpenAI manage the aggregate load on its infrastructure. If requests to the API increase dramatically, it could tax the servers and cause performance issues. By setting rate Stella Page 107 API increase dramatically, it could tax the servers and cause performance issues. By setting rate limits, OpenAI can help maintain a smooth and consistent experience for all users. Please work through this document in its entirety to better understand how OpenAI's rate limit system works. We include code examples and possible solutions to handle common issues. We also include details around how your rate limits are automatically increased in the usage tiers section below. How do these rate limits work? Rate limits are measured in four ways: RPM (requests per minute), RPD (requests per day), TPM (tokens per minute), and IPM (images per minute). Rate limits can be hit across any of the options depending on what occurs first. For example, you might send 20 requests with only 100 tokens to the ChatCompletions endpoint and that would fill your limit (if your RPM was 20), even if you did not send 150k tokens (if your TPM limit was 150k) within those 20 requests. Other important things worth noting: • Rate limits are imposed at the organization level, not user level. • Rate limits vary by the model being used. • Limits are also placed on the total amount an organization can spend on the API each month. These are also known as "usage limits". Usage tiers You can view the rate and usage limits for your organization under the limits section of your account settings. As your usage of the OpenAI API and your spend on our API goes up, we automatically graduate you to the next usage tier. This usually results in an increase in rate limits across most models. Organizations in higher tiers also get access to lower latency models. TIER QUALIFICATION USAGE LIMITS Free User must be in an allowed geography $100 / month Tier 1 $5 paid $100 / month Tier 2 $50 paid and 7+ days since first successful payment $500 / month Tier 3 $100 paid and 7+ days since first successful payment $1,000 / month Tier 4 $250 paid and 14+ days since first successful payment $5,000 / month Tier 5 $1,000 paid and 30+ days since first successful payment $10,000 / month Select a tier below to view a high-level summary of rate limits per model. Free Tier 1 Tier 2 Tier 3 Tier 4 Tier 5 Free tier rate limits This is a high level summary and there are per-model exceptions to these limits (e.g. some legacy models or models with larger context windows have different rate limits). To view the exact rate limits per model for your account, visit the limits section of your account settings. MODEL RPM RPD TPM gpt-4 3 200 10,000 gpt-3.5-turbo 3 200 20,000 text-embedding-ada-002 3 200 150,000 whisper-1 3 200 tts-1 3 200 dall-e-2 5 img/min dall-e-3 1 img/min Stella Page 108 dall-e-3 1 img/min Rate limits in headers In addition to seeing your rate limit on your account page, you can also view important information about your rate limits such as the remaining requests, tokens, and other metadata in the headers of the HTTP response. You can expect to see the following header fields: FIELD SAMPLE VALUE DESCRIPTION x-ratelimit-limit- requests 60 The maximum number of requests that are permitted before exhausting the rate limit. x-ratelimit-limit-tokens 150000 The maximum number of tokens that are permitted before exhausting the rate limit. x-ratelimit-remaining- requests 59 The remaining number of requests that are permitted before exhausting the rate limit. x-ratelimit-remaining- tokens 149984 The remaining number of tokens that are permitted before exhausting the rate limit. x-ratelimit-reset- requests 1s The time until the rate limit (based on requests) resets to its initial state. x-ratelimit-reset-tokens 6m0s The time until the rate limit (based on tokens) resets to its initial state. Error Mitigation What are some steps I can take to mitigate this? The OpenAI Cookbook has a Python notebook that explains how to avoid rate limit errors, as well an example Python script for staying under rate limits while batch processing API requests. You should also exercise caution when providing programmatic access, bulk processing features, and automated social media posting - consider only enabling these for trusted customers. To protect against automated and high-volume misuse, set a usage limit for individual users within a specified time frame (daily, weekly, or monthly). Consider implementing a hard cap or a manual review process for users who exceed the limit. Retrying with exponential backoff One easy way to avoid rate limit errors is to automatically retry requests with a random exponential backoff. Retrying with exponential backoff means performing a short sleep when a rate limit error is hit, then retrying the unsuccessful request. If the request is still unsuccessful, the sleep length is increased and the process is repeated. This continues until the request is successful or until a maximum number of retries is reached. This approach has many benefits: • Automatic retries means you can recover from rate limit errors without crashes or missing data • Exponential backoff means that your first retries can be tried quickly, while still benefiting from longer delays if your first few retries fail • Adding random jitter to the delay helps retries from all hitting at the same time. Note that unsuccessful requests contribute to your per-minute limit, so continuously resending a request won't work. Below are a few example solutions for Python that use exponential backoff. Stella Page

109 Below are a few example solutions for Python that use exponential backoff. Example 1: Using the Tenacity library Tenacity is an Apache 2.0 licensed general-purpose retrying library, written in Python, to simplify the task of adding retry behavior to just about anything. To add exponential backoff to your requests, you can use the tenacity.retry decorator. The below example uses the tenacity.wait_random_exponential function to add random exponential backoff to a request. Using the Tenacity library python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 from openai import OpenAI client = OpenAI() from tenacity import (retry, stop_after_attempt, wait_random_exponential,) # for exponential backoff @retry(wait=wait_random_exponential(min=1, max=60), stop=stop_after_attempt(6)) def completion_with_backoff(**kwargs): return client.completions.create(**kwargs) completion_with_backoff(model="gpt-3.5-turbo-instruct", prompt="Once upon a time,") Note that the Tenacity library is a third-party tool, and OpenAI makes no guarantees about its reliability or security. Collapse■ Example 2: Using the backoff library Another python library that provides function decorators for backoff and retry is backoff: Using the Tenacity library python Copy■ 1 2 3 4 5 6 7 8 9 10 Stella Page 110 10 import backoff import openai from openai import OpenAI client = OpenAI() @backoff.on_exception(backoff.expo, openai.RateLimitError) def completions_with_backoff(**kwargs): return client.completions.create(**kwargs) completions_with_backoff(model="gpt-3.5-turbo-instruct", prompt="Once upon a time,") Like Tenacity, the backoff library is a third-party tool, and OpenAI makes no guarantees about its reliability or security. Collapse■ Example 3: Manual backoff implementation If you don't want to use third-party libraries, you can implement your own backoff logic following this example: Using manual backoff implementation python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 Stella Page 111 42 43 44 45 46 47 48 49 50 51 52 53 54 55 # imports import random import time import openai from openai import OpenAI client = OpenAI() # define a retry decorator def retry_with_exponential_backoff(func, initial_delay: float = 1, exponential_base: float = 2, jitter: bool = True, max_retries: int = 10, errors: tuple = (openai.RateLimitError,),): """Retry a function with exponential backoff.""" def wrapper(*args, **kwargs): # Initialize variables num_retries = 0 delay = initial_delay # Loop until a successful response or max_retries is hit or an exception is raised while True: try: return func(*args, **kwargs) # Retry on specific errors except errors as e: # Increment retries num_retries += 1 # Check if max retries has been reached if num_retries > max_retries: raise Exception(f"Maximum number of retries ({max_retries}) exceeded.") # Increment the delay delay *= exponential_base * (1 + jitter * random.random()) # Sleep for the delay time.sleep(delay) # Raise exceptions for any errors not specified except Exception as e: raise e return wrapper @retry_with_exponential_backoff Stella Page 112 @retry_with_exponential_backoff def completions_with_backoff(**kwargs): return client.completions.create(**kwargs) Again, OpenAI makes no guarantees on the security or efficiency of this solution but it can be a good starting place for your own solution. Collapse■ Reduce the max_tokens to match the size of your completions Your rate limit is calculated as the maximum of max_tokens and the estimated number of tokens based on the character count of your request. Try to set the max_tokens value as close to your expected response size as possible. Batching requests The OpenAI API has separate limits for requests per minute and tokens per minute. If you're hitting the limit on requests per minute, but have available capacity on tokens per minute, you can increase your throughput by batching multiple tasks into each request. This will allow you to process more tokens per minute, especially with our smaller models. Sending in a batch of prompts works exactly the same as a normal API call, except you pass in a list of strings to the prompt parameter instead of a single string. Example without batching No batching python Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 from openai import OpenAI client = OpenAI() num_stories = 10 prompt = "Once upon a time," # serial example, with one story completion per request for _ in range(num_stories): response = client.completions.create(model="curie", prompt=prompt, max_tokens=20,) # print story print(prompt + response.choices[0].text) Collapse■ Example with batching Batching python Copy■ Stella Page 113 Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 from openai import OpenAI client = OpenAI() num_stories = 10 prompts = ["Once upon a time,"] * num_stories # batched example, with 10 story completions per request response = client.completions.create(model="curie", prompt=prompts, max_tokens=20,) # match completions to prompts by index stories = [""] * len(prompts) for choice in response.choices: stories[choice.index] = prompts[choice.index] + choice.text # print stories for story in stories: print(story) Collapse■ From Error codes This guide includes an overview on error codes you might see from both the API and our official Python library.

Each error code mentioned in the overview has a dedicated section with further guidance. API errors CODE OVERVIEW 401 - Invalid Authentication Cause: Invalid Authentication Solution: Ensure the correct API key and requesting organization are being used. Stella Page 114 requesting organization are being used. 401 - Incorrect API key provided Cause: The requesting API key is not correct. Solution: Ensure the API key used is correct, clear your browser cache, or generate a new one. 401 - You must be a member of an organization to use the API Cause: Your account is not part of an organization. Solution: Contact us to get added to a new organization or ask your organization manager to invite you to an organization. 429 - Rate limit reached for requests Cause: You are sending requests too quickly. Solution: Pace your requests. Read the Rate limit guide. 429 - You exceeded your current quota, please check your plan and billing details Cause: You have run out of credits or hit your maximum monthly spend. Solution: Buy more credits or learn how to increase your limits. 500 - The server had an error while processing your request Cause: Issue on our servers. Solution: Retry your request after a brief wait and contact us if the issue persists. Check the status page. 503 - The engine is currently overloaded, please try again later Cause: Our servers are experiencing high traffic. Solution: Please retry your requests after a brief wait. 401 - Invalid Authentication This error message indicates that your authentication credentials are invalid. This could happen for several reasons, such as: • You are using a revoked API key. • You are using a different API key than the one assigned to the requesting organization. • You are using an API key that does not have the required permissions for the endpoint you are calling. To resolve this error, please follow these steps: • Check that you are using the correct API key and organization ID in your request header. You can find your API key and organization ID in your account settings. • If you are unsure whether your API key is valid, you can generate a new one. Make sure to replace your old API key with the new one in your requests and follow our best practices guide. Collapse■ 401 - Incorrect API key provided This error message indicates that the API key you are using in your request is not correct. This could happen for several reasons, such as: • There is a typo or an extra space in your API key. • You are using an API key that belongs to a different organization. • You are using an API key that has been deleted or deactivated. • An old, revoked API key might be cached locally. To resolve this error, please follow these steps: • Try clearing your browser's cache and cookies, then try again. Stella Page 115 • Try clearing your browser's cache and cookies, then try again. • Check that you are using the correct API key in your request header. • If you are unsure whether your API key is correct, you can generate a new one. Make sure to replace your old API key in your codebase and follow our best practices guide. Collapse■ 401 - You must be a member of an organization to use the API This error message indicates that your account is not part of an organization. This could happen for several reasons, such as: • You have left or been removed from your previous organization. • Your organization has been deleted. To resolve this error, please follow these steps: • If you have left or been removed from your previous organization, you can either request a new organization or get invited to an existing one. • To request a new organization, reach out to us via help.openai.com • Existing organization owners can invite you to join their organization via the Team page. Collapse■ 429 - Rate limit reached for requests This error message indicates that you have hit your assigned rate limit for the API. This means that you have submitted too many tokens or requests in a short period of time and have exceeded the number of requests allowed. This could happen for several reasons, such as: • You are using a loop or a script that makes frequent or concurrent requests. • You are sharing your API key with other users or applications. • You are using a free plan that has a low rate limit. To resolve this error, please follow these steps: • Pace your requests and avoid making unnecessary or redundant calls. • If you are using a loop or a script, make sure to implement a backoff mechanism or a retry logic that respects the rate limit and the response headers. You can read more about our rate limiting policy and best practices in our rate limit guide. • If you are sharing your organization with other users, note that limits are applied per organization and not per user. It is worth checking on the usage of the rest of your team as this will contribute to the limit. • If you are using a free or low-tier plan, consider upgrading to a pay-as-you-go plan that offers a higher rate limit. You can compare the restrictions of each plan in our rate limit guide. Collapse■ 429 - You exceeded your current quota, please check your plan and billing details This error message indicates that you hit your monthly [usage limit](/account/limits) for the API, or for prepaid credits customers that you've consumed all your credits. You can view your maximum usage limit on the [limits page](/account/limits). This could happen for several reasons, such as: • You are using a high-volume or complex service that consumes a lot of credits or tokens. •

Your■monthly■budget■is■set■too■low■for■your■organization’s■usage. Stella Page 116 • Your■monthly■budget■is■set■too■low■for■your■organization’s■usage. To resolve this error, please follow these steps: • Check your current usage of your account, and compare that to your account's limits. • If you are on a free plan, consider upgrading to a paid plan to get higher limits. Collapse■ 503 - The engine is currently overloaded, please try again later This error message indicates that our servers are experiencing high traffic and are unable to process your request at the moment. This could happen for several reasons, such as: • There is a sudden spike or surge in demand for our services. • There is scheduled or unscheduled maintenance or update on our servers. • There is an unexpected or unavoidable outage or incident on our servers. To resolve this error, please follow these steps: • Retry your request after a brief wait. We recommend using an exponential backoff strategy or a retry logic that respects the response headers and the rate limit. You can read more about our rate limit best practices. • Check our status page for any updates or announcements regarding our services and servers. • If you are still getting this error after a reasonable amount of time, please contact us for further assistance. We apologize for any inconvenience and appreciate your patience and understanding. Collapse■ Python library error types TYPE OVERVIEW APIError Cause: Issue on our side. Solution: Retry your request after a brief wait and contact us if the issue persists. Timeout Cause: Request timed out. Solution: Retry your request after a brief wait and contact us if the issue persists. RateLimitE rror Cause: You have hit your assigned rate limit. Solution: Pace your requests. Read more in our Rate limit guide. APIConne ctionError Cause: Issue connecting to our services. Solution: Check your network settings, proxy configuration, SSL certificates, or firewall rules. InvalidReq uestError Cause: Your request was malformed or missing some required parameters, such as a token or an input. Solution: The error message should advise you on the specific error made. Check the documentation for the specific API method you are calling and make sure you are sending valid and complete parameters. You may also need to check the encoding, format, or size of your request data. Authentica tionError Cause: Your API key or token was invalid, expired, or revoked. Solution: Check your API key or token and make sure it is correct and active. You may need to generate a new one from your account dashboard. Stella Page 117 dashboard. ServiceUn availableEr ror Cause: Issue on our servers. Solution: Retry your request after a brief wait and contact us if the issue persists. Check the status page. APIError An `APIError` indicates that something went wrong on our side when processing your request. This could be due to a temporary error, a bug, or a system outage. We apologize for any inconvenience and we are working hard to resolve any issues as soon as possible. You can check our system status page for more information. If you encounter an APIError, please try the following steps: • Wait a few seconds and retry your request. Sometimes, the issue may be resolved quickly and your request may succeed on the second attempt. • Check our status page for any ongoing incidents or maintenance that may affect our services. If there is an active incident, please follow the updates and wait until it is resolved before retrying your request. • If the issue persists, check out our Persistent errors next steps section. Our support team will investigate the issue and get back to you as soon as possible. Note that our support queue times may be long due to high demand. You can also post in our Community Forum but be sure to omit any sensitive information. Collapse■ Timeout A `Timeout` error indicates that your request took too long to complete and our server closed the connection. This could be due to a network issue, a heavy load on our services, or a complex request that requires more processing time. If you encounter a Timeout error, please try the following steps: • Wait a few seconds and retry your request. Sometimes, the network congestion or the load on our services may be reduced and your request may succeed on the second attempt. • Check your network settings and make sure you have a stable and fast internet connection. You may need to switch to a different network, use a wired connection, or reduce the number of devices or applications using your bandwidth. • If the issue persists, check out our persistent errors next steps section. Collapse■ RateLimitError A `RateLimitError` indicates that you have hit your assigned rate limit. This means that you have sent too many tokens or requests in a given period of time, and our services have temporarily blocked you from sending more. We impose rate limits to ensure fair and efficient use of our resources and to prevent abuse or overload of our services. If you encounter a RateLimitError, please try the following steps: • Send fewer tokens or requests or slow down. You may need to reduce the frequency or volume of your requests, batch your tokens, or implement exponential backoff. You can read our Rate limit guide for more details. • Wait until your rate limit resets (one minute) and retry your request. The error message should give you a sense of your usage rate and permitted usage. Stella Page 118 • You can also check your API usage statistics from your account dashboard. Collapse■ APIConnectionError An `APIConnectionError` indicates that your request could not reach our servers or establish a secure connection. This could be due to a network issue, a proxy configuration, an SSL certificate, or a firewall rule. If you encounter an APIConnectionError, please try the following steps: • Check your network settings and make sure you have a stable and fast internet connection. You may need to switch to a different network, use a wired connection, or reduce the number of devices or applications using your bandwidth. • Check your proxy configuration and make sure it is compatible with our services. You may need to update your proxy settings, use a different proxy, or bypass the proxy altogether. • Check your SSL certificates and make sure they are valid and up-to-date. You may need to install or renew your certificates, use a different certificate authority, or disable SSL verification. • Check your firewall rules and make sure they are not blocking or filtering our services. You may need to modify your firewall settings. • If appropriate, check that your container has the correct permissions to send and receive traffic. • If the issue persists, check out our persistent errors next steps section. Collapse■ InvalidRequestError An InvalidRequestError indicates that your request was malformed or missing some required parameters, such as a token or an input. This could be due to a typo, a formatting error, or a logic error in your code. If you encounter an InvalidRequestError, please try the following steps: • Read the error message carefully and identify the specific error made. The error message should advise you on what parameter was invalid or missing, and what value or format was expected. • Check the API Reference for the specific API method you were calling and make sure you are sending valid and complete parameters. You may need to review the parameter names, types, values, and formats, and ensure they match the documentation. • Check the encoding, format, or size of your request data and make sure they are compatible with our services. You may need to encode your data in UTF-8, format your data in JSON, or compress your data if it is too large. • Test your request using a tool like Postman or curl and make sure it works as expected. You may need to debug your code and fix any errors or inconsistencies in your request logic. • If the issue persists, check out our persistent errors next steps section. Collapse■ AuthenticationError An `AuthenticationError` indicates that your API key or token was invalid, expired, or revoked. This could be due to a typo, a formatting error, or a security breach. If you encounter an AuthenticationError, please try the following steps: Stella Page 119 • Check your API key or token and make sure it is correct and active. You may need to generate a new key from the API Key dashboard, ensure there are no extra spaces or characters, or use a different key or token if you have multiple ones. • Ensure that you have followed the correct formatting. Collapse■ ServiceUnavailableError A `ServiceUnavailableError` indicates that our servers are temporarily unable to handle your request. This could be due to a planned or unplanned maintenance, a system upgrade, or a server failure. These errors can also be returned during periods of high traffic. We apologize for any inconvenience and we are working hard to restore our services as soon as possible. If you encounter a ServiceUnavailableError, please try the following steps: • Wait a few minutes and retry your request. Sometimes, the issue may be resolved quickly and your request may succeed on the next attempt. • Check our status page for any ongoing incidents or maintenance that may affect our services. If there is an active incident, please follow the updates and wait until it is resolved before retrying your request. • If the issue persists, check out our persistent errors next steps section. Collapse■ Persistent errors If the issue persists, contact our support team via chat and provide them with the following information: • The model you were using • The error message and code you received • The request data and headers you sent • The timestamp and timezone of your request • Any other relevant details that may help us diagnose the issue Our support team will investigate the issue and get back to you as soon as possible. Note that our support queue times may be long due to high demand. You can also post in our Community Forum but be sure to omit any sensitive information. Handling errors We advise you to programmatically handle errors returned by the API. To do so, you may want to use a code snippet like below: 1 2 3 4 5 6 7 8 9 10 11 Stella Page 120 11 12 13 14 15 16 17 18 19 20 21 22 import openai from openai import OpenAI client = OpenAI() try: #Make your OpenAI API request here response = client.completions.create(prompt="Hello world", model="gpt-3.5-turbo-instruct") except openai.APIError as e: #Handle API error here, e.g. retry or log print(f"OpenAI API returned an API Error: {e}") pass except openai.APIConnectionError as e: #Handle connection error here print(f"Failed to connect to OpenAI API: {e}") pass except openai.RateLimitError as e: #Handle rate limit error (we recommend using exponential backoff) print(f"OpenAI API request exceeded rate limit: {e}") pass From Libraries Python library We provide a Python library, which you can install as follows: $ pip install openai Once installed, you can use the bindings and your secret key to run the following: 1 2 3 Stella Page 121 3 4 5 6 7 8 9 10 from openai import OpenAI client = OpenAI(# Defaults to os.environ.get("OPENAI_API_KEY") # Otherwise use: api_key="Your_API_Key",) chat_completion = client.chat.completions.create(model="gpt-3.5-turbo", messages=[{"role": "user", "content": "Hello world"}]) The bindings also will install a command-line utility you can use as follows: $ openai api chat_completions.create -m gpt-3.5-turbo -g user "Hello world" Node.js library We also have a Node.js library, which you can install by running the following command in your Node.js project directory: $ npm install openai Once installed, you can use the library and your secret key to run the following: 1 2 3 4 5 6 7 8 9 10 import OpenAI from "openai"; const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY, }); const chatCompletion = await openai.chat.completions.create({ messages: [{ role: "user", content: "Say this is a test" }], model: "gpt-3.5-turbo", }); Azure OpenAI libraries Microsoft's Azure team maintains libraries that are compatible with both the OpenAI API Stella Page 122 Microsoft's Azure team maintains libraries that are compatible with both the OpenAI API and Azure OpenAI services. Read the library documentation below to learn how you can use them with the OpenAI API. • Azure OpenAI client library for .NET • Azure OpenAI client library for JavaScript • Azure OpenAI client library for Java • Azure OpenAI client library for Go Community libraries The libraries below are built and maintained by the broader developer community. If you'd like to add a new library here, please follow the instructions in our help center article on adding community libraries. You can also watch our OpenAPI specification repository on GitHub to get timely updates on when we make changes to our API. Please note that OpenAI does not verify the correctness or security of these projects. Use them at your own risk! C# / .NET • Betalgo.OpenAI by Betalgo • OpenAI-API-dotnet by OkGoDoIt • OpenAI-DotNet by RageAgainstThePixel C++ • liboai by D7EAD Clojure • openai-clojure by wkok Crystal • openai-crystal by sferik Dart/Flutter • openai by anasfik Delphi • DelphiOpenAI by HemulGM Elixir • openai.ex by mgallo Go • go-gpt3 by sashabaranov Java • openai-java by Theo Kanning Julia • OpenAI.jl by rory-linehan Kotlin • openai-kotlin by Mouaad Aallam Node.js Stella Page 123 Node.js • openai-api by Njerschow • openai-api-node by erlapso • gpt-x by ceifa • gpt3 by poteat • gpts by thencc • @dalenguyen/openai by dalenguyen • tectalic/openai by tectalic PHP • orhanerday/open-ai by orhanerday • tectalic/openai by tectalic • openai-php clinet by openai-php Python • chronology by OthersideAI R • rgpt3 by ben-aaron188 Ruby • openai by nileshtrivedi • ruby-openai by alexrudall Rust • async-openai by 64bit • fieri by lbkolev Scala • openai-scala-client by cequence-io Swift • OpenAIKit by dylanshine • OpenAI by MacPaw Unity • OpenAi-Api-Unity by hexthedev • com.openai.unity by RageAgainstThePixel Unreal Engine • OpenAI-Api-Unreal by KellanM From Deprecations Overview As we launch safer and more capable models, we regularly retire older models. Software relying on OpenAI models may need occasional updates to keep working. Impacted customers will always be notified by email and in our documentation along with blog posts for larger changes. Stella Page 124 with blog posts for larger changes. This page lists all API deprecations, along with recommended replacements. Incremental model updates As announced in March 2023, we regularly release new versions of gpt-4 and gpt-3.5- turbo. Each model version is dated with an -MMDD suffix; e.g., gpt-4-0613. The undated model name, e.g., gpt-4, will typically point to the latest version (e.g. gpt-4 points to gpt-4-0613). Users of undated model names will be notified by email typically 2 weeks before any change takes place. After a new version is launched, older versions will typically be deprecated 3 months later. Migrating to replacements Once a model is deprecated, be sure to migrate all usage to a suitable replacement before the shutdown date. Requests to models past the shutdown date will fail. To help measure the performance of replacement models on your tasks, we have open- sourced Evals, a Python framework for evaluating language models. If new models perform worse on your tasks, let us know by by submitting a pull request to our Evals repo with examples of the task. Deprecation history All deprecations are listed below, with the most recent announcements at the top. 2023-11-06: Chat model updates On November 6th, 2023, we announced the release of an updated GPT-3.5-Turbo model (which now comes by default with 16k context) along with deprecation of gpt-3.5- turbo-0613 and gpt-3.5-turbo-16k-0613. SHUTDOWN DATE LEGACY MODEL LEGACY MODEL PRICE RECOMMENDED REPLACEMENT 2024-06-13 gpt-3.5- turbo-0613 $0.0015 / 1K input tokens + $0.0020 / 1K output tokens gpt-3.5-turbo-1106 2024-06-13 gpt-3.5- turbo-16k-0613 $0.0030 / 1K input tokens + $0.0040 / 1K output tokens gpt-3.5-turbo-1106 Fine-tuned models created from these base models are not effected by this deprecation, but you will no longer be able to create new fine-tuned versions with these models. 2023-08-22: Fine-tunes endpoint On August 22nd, 2023, we announced the new fine-tuning API (/v1/fine_tuning/jobs) and that the original /v1/fine-tunes API along with legacy models (including those fine-tuned with the /v1/fine-tunes API) will be shut down on January 04, 2024. Fine-tunes endpoint SHUTDOWN DATE SYSTEM RECOMMENDED REPLACEMENT Stella Page 125 SHUTDOWN DATE SYSTEM RECOMMENDED REPLACEMENT 2024-01-04 /v1/fine-tunes /v1/fine_tuning/jobs 2023-07-06: GPT and embeddings On July 06, 2023, we announced the upcoming retirements of older GPT-3 and GPT-3.5 models served via the completions endpoint. We also announced the upcoming retirement of our first-generation text embedding models. They will be shut down on January 04, 2024. InstructGPT models SHUTDOWN DATE LEGACY MODEL LEGACY MODEL PRICE RECOMMENDED REPLACEMENT 2024-01-04 text-ada-001 $0.0004 / 1K tokens gpt-3.5-turbo-instruct 2024-01-04 text-babbage-001 $0.0005 / 1K tokens gpt-3.5-turbo-instruct 2024-01-04 text-curie-001 $0.0020 / 1K tokens gpt-3.5-turbo-instruct 2024-01-04 text-davinci-001 $0.0200 / 1K tokens gpt-3.5-turbo-instruct 2024-01-04 text-davinci-002 $0.0200 / 1K tokens gpt-3.5-turbo-instruct 2024-01-04 text-davinci-003 $0.0200 / 1K tokens gpt-3.5-turbo-instruct Pricing for the replacement gpt-3.5-turbo-instruct model can be found on the pricing page. Base GPT models SHUTDOWN DATE LEGACY MODEL LEGACY MODEL PRICE RECOMMENDED REPLACEMENT 2024-01-04 ada $0.0004 / 1K tokens babbage-002 2024-01-04 babbage $0.0005 / 1K tokens babbage-002 2024-01-04 curie $0.0020 / 1K tokens davinci-002 2024-01-04 davinci $0.0200 / 1K tokens davinci-002 2024-01-04 code-davinci-002 free to researchers gpt-3.5-turbo-base Pricing for the replacement babbage-002 and davinci-002 models can be found on the pricing page. Edit models & endpoint SHUTDOWN DATE MODEL / SYSTEM RECOMMENDED REPLACEMENT 2024-01-04 text-davinci-edit-001 gpt-4 2024-01-04 code-davinci-edit-001 gpt-4 2024-01-04 /v1/edits /v1/chat/completions Fine-tuning GPT models SHUTDOWN DATE LEGACY MODEL TRAINING PRICE USAGE PRICE RECOMMENDED REPLACEMENT 2024-01-04 ada $0.0004 / 1K tokens $0.0016 / 1K tokens babbage-002 2024-01-04 babbage $0.0006 / 1K tokens $0.0024 / 1K tokens babbage-002 2024-01-04 curie $0.003 / 1K tokens $0.012 / 1K tokens davinci-002 2024-01-04 davinci $0.03 / 1K tokens $0.12 / 1K tokens davinci-002, gpt-3.5- turbo, gpt-4 Stella Page 126 tokens tokens turbo, gpt-4 First-generation text embedding models SHUTDOWN DATE LEGACY MODEL LEGACY MODEL PRICE RECOMMENDED REPLACEMENT 2024-01-04 text-similarity-ada-001 $0.004 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-ada-doc-001 $0.004 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-ada-query-001 $0.004 / 1K tokens text-embedding- ada-002 2024-01-04 code-search-ada-code-001 $0.004 / 1K tokens text-embedding- ada-002 2024-01-04 code-search-ada-text-001 $0.004 / 1K tokens text-embedding- ada-002 2024-01-04 text-similarity-babbage-001 $0.005 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-babbage- doc-001 $0.005 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-babbage- query-001 $0.005 / 1K tokens text-embedding- ada-002 2024-01-04 code-search-babbage- code-001 $0.005 / 1K tokens text-embedding- ada-002 2024-01-04 code-search-babbage- text-001 $0.005 / 1K tokens text-embedding- ada-002 2024-01-04 text-similarity-curie-001 $0.020 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-curie-doc-001 $0.020 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-curie-query-001 $0.020 / 1K tokens text-embedding- ada-002 2024-01-04 text-similarity-davinci-001 $0.200 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-davinci-doc-001 $0.200 / 1K tokens text-embedding- ada-002 2024-01-04 text-search-davinci- query-001 $0.200 / 1K tokens text-embedding- ada-002 2023-06-13: Updated chat models On June 13, 2023, we announced new chat model versions in the Function calling and other API updates blog post. The three original versions will be retired in June 2024 at the earliest. SHUTDOWN DATE LEGACY MODEL LEGACY MODEL PRICE RECOMMENDED REPLACEMENT at earliest 2024-06-13 gpt-3.5- turbo-0301 $0.0015 / 1K input tokens + $0.0020 / 1K output tokens gpt-3.5-turbo-0613 at earliest gpt-4-0314 $0.03 / 1K input tokens + $0.06 / gpt-4-0613 Stella Page 127 at earliest 2024-06-13 gpt-4-0314 $0.03 / 1K input tokens + $0.06 / 1K output tokens gpt-4-0613 at earliest 2024-06-13 gpt-4-32k-03 14 $0.06 / 1K input tokens + $0.12 / 1K output tokens gpt-4-32k-0613 2023-03-20: Codex models SHUTDOWN DATE LEGACY MODEL RECOMMENDED REPLACEMENT 2023-03-23 code-davinci-002 gpt-4 or researcher access program 2023-03-23 code-davinci-001 gpt-4 or researcher access program 2023-03-23 code-cushman-002 gpt-4 or researcher access program 2023-03-23 code-cushman-001 gpt-4 or researcher access program 2022-06-03: Legacy endpoints SHUTDOWN DATE SYSTEM RECOMMENDED REPLACEMENT 2022-12-03 /v1/engines /v1/models 2022-12-03 /v1/search View transition guide 2022-12-03 /v1/classifications View transition guide 2022-12-03 /v1/answers View transition guide From Chat Plugins Beta Learn how to build a plugin that allows ChatGPT to intelligently call your API. GPTs and custom Actions are here! We’re■rolling■out■custom■versions■of■ChatGPT■that■you■can■create■for■a■specific■ purpose—called GPTs. GPTs are a new way for anyone to create a tailored version of ChatGPT to be more helpful in their daily life, at specific tasks, at work, or at home—and then share that creation with others. We are excited to announce Actions, which build on plugins. Actions take many of the core ideas of plugins while also introducing many new features builders have been asking for. Introduction OpenAI plugins connect ChatGPT to third-party applications. These plugins enable ChatGPT to interact with APIs defined by developers, enhancing ChatGPT's capabilities and allowing it to perform a wide range of actions. Plugins enable ChatGPT to do things like: • Retrieve real-time information; e.g., sports scores, stock prices, the latest news, etc. • Retrieve knowledge-base information; e.g., company docs, personal notes, etc. • Assist users with actions; e.g., booking a flight, ordering food, etc. If you want to have an example running as you read through the documentation and learn more about plugins, you can begin with our plugin quickstart repo. Stella Page 128 Plugin developers expose one or more API endpoints, accompanied by a standardized manifest file and an OpenAPI specification. These define the plugin's functionality, allowing ChatGPT to consume the files and make calls to the developer-defined APIs. The AI model acts as an intelligent API caller. Given an API spec and a natural- language description of when to use the API, the model proactively calls the API to perform actions. For instance, if a user asks, "Where should I stay in Paris for a couple nights?", the model may choose to call a hotel reservation plugin API, receive the API response, and generate a user-facing answer combining the API data and its natural language capabilities. Over time, we anticipate the system will evolve to accommodate more advanced use cases. Plugin flow To build a plugin, it is important to understand the end-to-end flow. 1. Create a manifest file and host it at yourdomain.com/.well-known/ai-plugin.json ■ The file includes metadata about your plugin (name, logo, etc.), details about authentication required (type of auth, OAuth URLs, etc.), and an OpenAPI spec for the endpoints you want to expose. ■ The model will see the OpenAPI description fields, which can be used to provide a natural language description for the different fields. ■ We suggest exposing only 1-2 endpoints in the beginning with a minimum number of parameters to minimize the length of the text. The plugin description, API requests, and API responses are all inserted into the conversation with ChatGPT. This counts against the context limit of the model. 2. Register your plugin in the ChatGPT UI ■ Select■the■plugin■model■from■the■top■drop■down,■then■select■“Plugins”,■“Plugin■ Store”,■and■finally■“Develop■your■own■plugin”. ■ If authentication is required, provide an OAuth 2 client_id and client_secret or an API key. 3. Users activate your plugin ■ Users must manually activate your plugin in the ChatGPT UI. (ChatGPT will not use your plugin by default.) ■ You will be able to share your plugin with 100 additional users (only other developers can install unverified plugins). ■ If OAuth is required, users will be redirected via OAuth to your plugin to sign in. 4. Users begin a conversation ■ OpenAI will inject a compact description of your plugin in a message to ChatGPT, invisible to end users. This will include the plugin description, endpoints, and examples. ■ When a user asks a relevant question, the model may choose to invoke an API call from your plugin if it seems relevant; for POST requests, we require that developers build a user confirmation flow to avoid destruction actions. ■ The model will incorporate the API call results into its response to the user. ■ The model might include links returned from the API calls in its response. These will be displayed as rich previews (using the OpenGraph protocol, Stella Page 129 ■ These will be displayed as rich previews (using the OpenGraph protocol, where we pull the site_name, title, description, image, and url fields). ■ The model can also format data from your API in markdown and the ChatGPT UI will render the markdown automatically. Currently,■we■will■be ■sending■the■user’s■country■and■state■in■the■Plugin■conversation■ header (if you are in California for example, it would look like {"openai-subdivision-1-iso- code": "US-CA"}. This is useful for shopping, restaurants, weather, and more. You can read more in our developer terms of use. Next steps Now that you know the basics of plugins, you might want to: • Get started building a plugin • Explore example plugins • Find the right plugin authentication schema • Read about important steps for productionizing your plugin • Learn about the plugin review process • Familiarize yourself with the plugin policies From Actions in GPTs GPTs and custom Actions are here! We’re■rolling■out■custom■versions■of■ChatGPT■that■you■can■create■for■a■specific■ purpose—called GPTs. GPTs are a new way for anyone to create a tailored version of ChatGPT to be more helpful in their daily life, at specific tasks, at work, or at home—and then share that creation with others. We are excited to announce Actions, which build on plugins. Actions take many of the core ideas of plugins while also introducing many new features builders have been asking for. What is a GPT? GPTs provide the ability to deeply customize ChatGPT with all new capabilities. GPTs also lower the barrier for builders. You can read more in the GPT launch blog post. What is an action? In addition to using our built-in capabilities, you can also define custom actions by making one or more APIs available to the GPT. Like plugins, actions allow GPTs to integrate external data or interact with the real-world. Connect GPTs to databases, plug them into emails, or make them your shopping assistant. For example, you could integrate■a■travel■listings■database,■connect■a■user’s■email■inbox,■or■facilitate■e-commerce orders. The design of actions builds upon insights from our plugins beta, granting developers greater control over the model and how their APIs are called. Migrating from the plugins beta is easy with the ability to use your existing plugin manifest to define actions for your GPT. Create an Action Stella Page 130 Create an Action To create an Action, you can define an OpenAPI specification similarly to that of a plugin with a few changes listed below. If you have a plugin today, creating a GPT with an action should only take a few minutes. You can start by creating a GPT in the ChatGPT UI and then connect it to your existing plugin OpenAPI reference. From the GPT editor: • Select "Configure" • "Add Action" • Fill in your OpenAPI spec or paste in a URL where it is hosted (you can use an existing plugin URL) Actions vs Plugins Like ChatGPT plugins, Actions allow you to connect a GPT to a custom API. There are a few noticeable differences between Actions and plugins which you can see mentioned below. Functions Endpoints defined in the OpenAPI specification are now called "functions". There is no difference in how these are defined. Hosted OpenAPI specification With Actions, OpenAI now hosts the OpenAPI specification for your API. This means you no longer need to host your own OpenAPI specification. You can import and existing OpenAPI specification or create a new one from scratch using the UI in the GPT creator. Consequential flag In the OpenAPI specification, you can now set certain endpoints as "consequential" as shown below: 1 2 3 4 5 6 get: operationId: blah x-openai-isConsequential: false post: operationId: blah2 x-openai-isConsequential: true • If the x-openai-isConsequential field is true, we treat the operation as "must always prompt the user for confirmation before running" and don't show an "always allow" button (both are new features of GPTs designed to give users more control). • If the x-openai-isConsequential field is false, we show the "always allow button". • If the field isn't present, we default all GET operations to false and all other Stella Page 131 • If the field isn't present, we default all GET operations to false and all other operations to true Multiple authentication schemas Actions now support multiple authentication schemas which can be set on a per- endpoint basis. This means you can have some endpoints that require authentication and some that don't. This can be set as a components -> securityschemes -> object in the OpenAPI spec, and on each operation in the spec there will be a security object. If no security object is specified in the operation, we consider it unauthed or noauth. Updated store process The GPT marketplace will supersede the plugin store. As the GPT marketplace rolls out, we will have more to share. From Introduction You can interact with the API through HTTP requests from any language, via our official Python bindings, our official Node.js library, or a community-maintained library. To install the official Python bindings, run the following command: pip install openai To install the official Node.js library, run the following command in your Node.js project directory: npm install openai@^4.0.0 Authentication The OpenAI API uses API keys for authentication. Visit your API Keys page to retrieve the API key you'll use in your requests. Remember that your API key is a secret! Do not share it with others or expose it in any client-side code (browsers, apps). Production requests must be routed through your own backend server where your API key can be securely loaded from an environment variable or key management service. All API requests should include your API key in an Authorization HTTP header as follows: Authorization: Bearer OPENAI_API_KEY Organization (optional) For users who belong to multiple organizations, you can pass a header to specify which organization is used for an API request. Usage from these API requests will count as usage for the specified organization. Example curl command: Stella Page 132 1 2 3 curl https://api.openai.com/v1/models \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Organization: org-Q13fm7txVz9t6LI45P8FoXBp" Example with the openai Python package: 1 2 3 4 5 6 from openai import OpenAI client = OpenAI(organization='org-Q13fm7txVz9t6LI45P8FoXBp',) client.models.list() Example with the openai Node.js package: 1 2 3 4 5 6 7 import { Configuration, OpenAIApi } from "openai"; const configuration = new Configuration({ organization: "org-Q13fm7txVz9t6LI45P8FoXBp", apiKey: process.env.OPENAI_API_KEY, }); const openai = new OpenAIApi(configuration); const response = await openai.listEngines(); Organization IDs can be found on your Organization settings page. Making requests You can paste the command below into your terminal to run your first API request. Make sure to replace $OPENAI_API_KEY with your secret API key. 1 2 3 4 5 6 7 8 curl https://api.openai.com/v1/chat/completions \ -H "Content-Type: application/json" \ Stella Page 133 -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "model": "gpt-3.5-turbo", "messages": [{"role": "user", "content": "Say this is a test!"}], "temperature": 0.7 }' This request queries the gpt-3.5-turbo model (which under the hood points to the latest gpt-3.5-turbo model variant) to complete the text starting with a prompt of "Say this is a test". You should get a response back that resembles the following: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 { "id": "chatcmpl-abc123", "object": "chat.completion", "created": 1677858242, "model": "gpt-3.5-turbo-1006", "usage": { "prompt_tokens": 13, "completion_tokens": 7, "total_tokens": 20 }, "choices": [{ "message": { "role": "assistant", "content": "\n\nThis is a test!" }, "finish_reason": "stop", "index": 0 }] } Now that you've generated your first chat completion, let's break down the response object. We can see the finish_reason is stop which means the API returned the full chat completion generated by the model without running into any limits. In the choices list, Stella Page 134 completion generated by the model without running into any limits. In the choices list, we only generated a single message but you can set the n parameter to generate multiple messages choices. Audio Learn how to turn audio into text or text into audio. Related guide: Speech to text Create speech POST https://api.openai.com/v1/audio/speech Generates audio from the input text. Request body model string Required One of the available TTS models: tts-1 or tts-1-hd input string Required The text to generate audio for. The maximum length is 4096 characters. voice string Required The voice to use when generating the audio. Supported voices are alloy, echo, fable, onyx, nova, and shimmer. response_format string Optional Defaults to mp3 The format to audio in. Supported formats are mp3, opus, aac, and flac. speed number Optional Defaults to 1 The speed of the generated audio. Select a value from 0.25 to 4.0. 1.0 is the default. Returns The audio file content. Example request curl Copy■ 1 2 3 4 5 6 7 8 9 curl https://api.openai.com/v1/audio/speech \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "Content-Type: application/json" \ -d '{ "model": "tts-1", "input": "The quick brown fox jumped over the lazy dog.", "voice": "alloy" }' \ --output speech.mp3 Create transcription Stella Page 135 Create transcription POST https://api.openai.com/v1/audio/transcriptions Transcribes audio into the input language. Request body file file Required The audio file object (not file name) to transcribe, in one of these formats: flac, mp3, mp4, mpeg, mpga, m4a, ogg, wav, or webm. model string Required ID of the model to use. Only whisper-1 is currently available. language string Optional The language of the input audio. Supplying the input language in ISO-639-1 format will improve accuracy and latency. prompt string Optional An optional text to guide the model's style or continue a previous audio segment. The prompt should match the audio language. response_format string Optional Defaults to json The format of the transcript output, in one of these options: json, text, srt, verbose_json, or vtt. temperature number Optional Defaults to 0 The sampling temperature, between 0 and 1. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. If set to 0, the model will use log probability to automatically increase the temperature until certain thresholds are hit. Returns The transcribed text. Example request curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/audio/transcriptions \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "Content-Type: multipart/form-data" \ -F file="@/path/to/file/audio.mp3" \ -F model="whisper-1" Response Copy■ 1 2 3 { "text": "Imagine the wildest idea that you've ever had, and you're curious about how it might scale to something that's a 100, a 1,000 times bigger. This is a place where you can get to do that." } Create translation Stella Page 136 Create translation POST https://api.openai.com/v1/audio/translations Translates audio into English. Request body file file Required The audio file object (not file name) translate, in one of these formats: flac, mp3, mp4, mpeg, mpga, m4a, ogg, wav, or webm. model string Required ID of the model to use. Only whisper-1 is currently available. prompt string Optional An optional text to guide the model's style or continue a previous audio segment. The prompt should be in English. response_format string Optional Defaults to json The format of the transcript output, in one of these options: json, text, srt, verbose_json, or vtt. temperature number Optional Defaults to 0 The sampling temperature, between 0 and 1. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. If set to 0, the model will use log probability to automatically increase the temperature until certain thresholds are hit. Returns The translated text. Example request curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/audio/translations \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "Content-Type: multipart/form-data" \ -F file="@/path/to/file/german.m4a" \ -F model="whisper-1" Response Copy■ 1 2 3 { "text": "Hello, my name is Wolfgang and I come from Germany. Where are you heading today?" } Chat Given a list of messages comprising a conversation, the model will return a response. Related guide: Chat Completions The chat completion object Stella Page 137 The chat completion object Represents a chat completion response returned by model, based on the provided input. id string A unique identifier for the chat completion. choices array A list of chat completion choices. Can be more than one if n is greater than 1. Show properties created integer The Unix timestamp (in seconds) of when the chat completion was created. model string The model used for the chat completion. system_fingerprint string This fingerprint represents the backend configuration that the model runs with. Can be used in conjunction with the seed request parameter to understand when backend changes have been made that might impact determinism. object string The object type, which is always chat.completion. usage object Usage statistics for the completion request. Show properties The chat completion object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "id": "chatcmpl-123", "object": "chat.completion", "created": 1677652288, "model": "gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices": [{ "index": 0, "message": { "role": "assistant", "content": "\n\nHello there, how may I assist you today?", }, Stella Page 138 }, "finish_reason": "stop" }], "usage": { "prompt_tokens": 9, "completion_tokens": 12, "total_tokens": 21 } } The chat completion chunk object Represents a streamed chunk of a chat completion response returned by model, based on the provided input. id string A unique identifier for the chat completion. Each chunk has the same ID. choices array A list of chat completion choices. Can be more than one if n is greater than 1. Show properties created integer The Unix timestamp (in seconds) of when the chat completion was created. Each chunk has the same timestamp. model string The model to generate the completion. object string The object type, which is always chat.completion.chunk. The chat completion chunk object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 {"id":"chatcmpl-123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices":[{"index":0,"delta":{"role":"assistant","content":""},"finish_reason":null}]} {"id":"chatcmpl-123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices":[{"index":0,"delta":{"content":"Hello"},"finish_reason":null}]} {"id":"chatcmpl-123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices":[{"index":0,"delta":{"content":"!"},"finish_reason":null}]} .... {"id":"chatcmpl-123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices":[{"index":0,"delta":{"content":" today"},"finish_reason":null}]} {"id":"chatcmpl-123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices":[{"index":0,"delta":{"content":"?"},"finish_reason":null}]} {"id":"chatcmpl-123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices":[{"index":0,"delta":{},"finish_reason":"stop"}]} Create chat completion Stella Page 139 Create chat completion POST https://api.openai.com/v1/chat/completions Creates a model response for the given chat conversation. Request body messages array Required A list of messages comprising the conversation so far. Example Python code. Show possible types model string Required ID of the model to use. See the model endpoint compatibility table for details on which models work with the Chat API. frequency_penalty number or null Optional Defaults to 0 Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim. See more information about frequency and presence penalties. logit_bias map Optional Defaults to null Modify the likelihood of specified tokens appearing in the completion. Accepts a JSON object that maps tokens (specified by their token ID in the tokenizer) to an associated bias value from -100 to 100. Mathematically, the bias is added to the logits generated by the model prior to sampling. The exact effect will vary per model, but values between -1 and 1 should decrease or increase likelihood of selection; values like -100 or 100 should result in a ban or exclusive selection of the relevant token. max_tokens integer or null Optional Defaults to inf The maximum number of tokens to generate in the chat completion. The total length of input tokens and generated tokens is limited by the model's context length. Example Python code for counting tokens. n integer or null Optional Defaults to 1 How many chat completion choices to generate for each input message. presence_penalty number or null Optional Defaults to 0 Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics. See more information about frequency and presence penalties. response_format object Optional An object specifying the format that the model must output. Setting to { type: "json_object" } enables JSON mode, which guarantees the message the model generates is valid JSON. Important: when using JSON mode you must still instruct the model to produce JSON yourself via some conversation message, for example via your system message. If you don't do this, the model may generate an unending stream of whitespace until the generation reaches the token limit, which Stella Page 140 may generate an unending stream of whitespace until the generation reaches the token limit, which may take a lot of time and give the appearance of a "stuck" request. Also note that the message content may be partial (i.e. cut off) if finish_reason="length", which indicates the generation exceeded max_tokens or the conversation exceeded the max context length. Show properties seed integer or null Optional This feature is in Beta. If specified, our system will make a best effort to sample deterministically, such that repeated requests with the same seed and parameters should return the same result. Determinism is not guaranteed, and you should refer to the system_fingerprint response parameter to monitor changes in the backend. stop string / array / null Optional Defaults to null Up to 4 sequences where the API will stop generating further tokens. stream boolean or null Optional Defaults to false If set, partial message deltas will be sent, like in ChatGPT. Tokens will be sent as data-only server- sent events as they become available, with the stream terminated by a data: [DONE] message. Example Python code. temperature number or null Optional Defaults to 1 What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. We generally recommend altering this or top_p but not both. top_p number or null Optional Defaults to 1 An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered. We generally recommend altering this or temperature but not both. tools array Optional A list of tools the model may call. Currently, only functions are supported as a tool. Use this to provide a list of functions the model may generate JSON inputs for. Show properties tool_choice string or object Optional Controls which (if any) function is called by the model. none means the model will not call a function and instead generates a message. auto means the model can pick between generating a message or calling a function. Specifying a particular function via {"type: "function", "function": {"name": "my_function"}} forces the model to call that function. none is the default when no functions are present. auto is the default if functions are present. Show possible types user string Optional A unique identifier representing your end-user, which can help OpenAI to monitor and detect abuse. Learn more. function_call Deprecated string or object Stella Page 141 string or object Optional Deprecated in favor of tool_choice. Controls which (if any) function is called by the model. none means the model will not call a function and instead generates a message. auto means the model can pick between generating a message or calling a function. Specifying a particular function via {"name": "my_function"} forces the model to call that function. none is the default when no functions are present. `auto`` is the default if functions are present. Show possible types functions Deprecated array Optional Deprecated in favor of tools. A list of functions the model may generate JSON inputs for. Show properties Returns Returns a chat completion object, or a streamed sequence of chat completion chunk objects if the request is streamed. DEFAULTIMAGE INPUTSTREAMINGFUNCTION CALLING Example request gpt-3.5-turbo curl Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 curl https://api.openai.com/v1/chat/completions \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "model": "gpt-3.5-turbo", "messages": [{ "role": "system", "content": "You are a helpful assistant." }, { "role": "user", "content": "Hello!" }] }' Response Copy■ 1 2 3 Stella Page 142 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "id": "chatcmpl-123", "object": "chat.completion", "created": 1677652288, "model": "gpt-3.5-turbo-0613", "system_fingerprint": "fp_44709d6fcb", "choices": [{ "index": 0, "message": { "role": "assistant", "content": "\n\nHello there, how may I assist you today?", }, "finish_reason":

"stop" }], "usage": { "prompt_tokens": 9, "completion_tokens": 12, "total_tokens": 21 } } Completions Legacy Given a prompt, the model will return one or more predicted completions, and can also return the probabilities of alternative tokens at each position. We recommend most users use our Chat Completions API. Learn more Related guide: Legacy Completions The completion object Legacy Represents a completion response from the API. Note: both the streamed and non-streamed response objects share the same shape (unlike the chat endpoint). id string A unique identifier for the completion. choices array The list of completion choices the model generated for the input prompt. Show properties created integer The Unix timestamp (in seconds) of when the completion was created. model string The model used for completion. Stella Page 143 The model used for completion. system_fingerprint string This fingerprint represents the backend configuration that the model runs with. Can be used in conjunction with the seed request parameter to understand when backend changes have been made that might impact determinism. object string The object type, which is always "text_completion" usage object Usage statistics for the completion request. Show properties The completion object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 { "id": "cmpl-uqkvlQyYK7bGYrRHQ0eXlWi7", "object": "text_completion", "created": 1589478378, "model": "gpt-3.5-turbo", "choices": [{ "text": "\n\nThis is indeed a test", "index": 0, "logprobs": null, "finish_reason": "length" }], "usage": { "prompt_tokens": 5, "completion_tokens": 7, "total_tokens": 12 } } Create completion Legacy POST https://api.openai.com/v1/completions Creates a completion for the provided prompt and parameters. Request body model string Required ID of the model to use. You can use the List models API to see all of your available models, or see our Model overview for descriptions of them. Stella Page 144 our Model overview for descriptions of them. prompt string or array Required The prompt(s) to generate completions for, encoded as a string, array of strings, array of tokens, or array of token arrays. Note that <|endoftext|> is the document separator that the model sees during training, so if a prompt is not specified the model will generate as if from the beginning of a new document. best_of integer or null Optional Defaults to 1 Generates best_of completions server-side and returns the "best" (the one with the highest log probability per token). Results cannot be streamed. When used with n, best_of controls the number of candidate completions and n specifies how many to return – best_of must be greater than n. Note: Because this parameter generates many completions, it can quickly consume your token quota. Use carefully and ensure that you have reasonable settings for max_tokens and stop. echo boolean or null Optional Defaults to false Echo back the prompt in addition to the completion frequency_penalty number or null Optional Defaults to 0 Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim. See more information about frequency and presence penalties. logit_bias map Optional Defaults to null Modify the likelihood of specified tokens appearing in the completion. Accepts a JSON object that maps tokens (specified by their token ID in the GPT tokenizer) to an associated bias value from -100 to 100. You can use this tokenizer tool (which works for both GPT-2 and GPT-3) to convert text to token IDs. Mathematically, the bias is added to the logits generated by the model prior to sampling. The exact effect will vary per model, but values between -1 and 1 should decrease or increase likelihood of selection; values like -100 or 100 should result in a ban or exclusive selection of the relevant token. As an example, you can pass {"50256": -100} to prevent the <|endoftext|> token from being generated. logprobs integer or null Optional Defaults to null Include the log probabilities on the logprobs most likely tokens, as well the chosen tokens. For example, if logprobs is 5, the API will return a list of the 5 most likely tokens. The API will always return the logprob of the sampled token, so there may be up to logprobs+1 elements in the response. The maximum value for logprobs is 5. max_tokens integer or null Optional Defaults to 16 The maximum number of tokens to generate in the completion. The token count of your prompt plus max_tokens cannot exceed the model's context length. Example Python code for counting tokens. n Stella Page 145 n integer or null Optional Defaults to 1 How many completions to generate for each prompt. Note: Because this parameter generates many completions, it can quickly consume your token quota. Use carefully and ensure that you have reasonable settings for max_tokens and stop. presence_penalty number or null Optional Defaults to 0 Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics. See more information about frequency and presence penalties. seed integer or null Optional If specified, our system will make a best effort to sample deterministically, such that repeated requests with the same seed and parameters should return the same result. Determinism is not guaranteed, and you should refer to the system_fingerprint response parameter to monitor changes in the backend. stop string / array / null Optional Defaults to null Up to 4 sequences where the API will stop generating further tokens. The returned text will not contain the stop sequence. stream boolean or null Optional Defaults to false Whether to stream back partial progress. If set, tokens will be sent as data-only server-sent events as they become available, with the stream terminated by a data: [DONE] message. Example Python code. suffix string or null Optional Defaults to null The suffix that comes after a completion of inserted text. temperature number or null Optional Defaults to 1 What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. We generally recommend altering this or top_p but not both. top_p number or null Optional Defaults to 1 An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered. We generally recommend altering this or temperature but not both. user string Optional A unique identifier representing your end-user, which can help OpenAI to monitor and detect abuse. Learn more. Returns Stella Page 146 Returns Returns a completion object, or a sequence of completion objects if the request is streamed. NO STREAMINGSTREAMING Example request gpt-3.5-turbo-instruct curl Copy■ 1 2 3 4 5 6 7 8 9 curl https://api.openai.com/v1/completions \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "model": "gpt-3.5-turbo-instruct", "prompt": "Say this is a test", "max_tokens": 7, "temperature": 0 }' Response gpt-3.5-turbo-instruct Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "id": "cmpl-uqkvlQyYK7bGYrRHQ0eXlWi7", "object": "text_completion", "created": 1589478378, "model": "gpt-3.5-turbo-instruct", "system_fingerprint": "fp_44709d6fcb", "choices": [{ "text": "\n\nThis is indeed a test", "index": 0, "logprobs": null, "finish_reason": "length" }], "usage": { Stella Page 147 "usage": { "prompt_tokens": 5, "completion_tokens": 7, "total_tokens": 12 } } Embeddings Get a vector representation of a given input that can be easily consumed by machine learning models and algorithms. Related guide: Embeddings The embedding object Represents an embedding vector returned by embedding endpoint. index integer The index of the embedding in the list of embeddings. embedding array The embedding vector, which is a list of floats. The length of vector depends on the model as listed in the embedding guide. object string The object type, which is always "embedding". The embedding object Copy■ 1 2 3 4 5 6 7 8 9 10 { "object": "embedding", "embedding": [0.0023064255, -0.009327292, ... (1536 floats total for ada-002) -0.0028842222,], "index": 0 } Create embeddings POST https://api.openai.com/v1/embeddings Creates an embedding vector representing the input text. Request body input string or array Required Input text to embed, encoded as a string or array of tokens. To embed multiple inputs in a single request, pass an array of strings or array of token arrays. The input must not exceed the max input tokens for the model (8192 tokens for text-embedding-ada-002) and cannot be an empty string. Example Python code for counting tokens. model string Stella Page 148 string Required ID of the model to use. You can use the List models API to see all of your available models, or see our Model overview for descriptions of them. encoding_format string Optional Defaults to float The format to return the embeddings in. Can be either float or base64. user string Optional A unique identifier representing your end-user, which can help OpenAI to monitor and detect abuse. Learn more. Returns A list of embedding objects. Example request curl Copy■ 1 2 3 4 5 6 7 8 curl https://api.openai.com/v1/embeddings \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "Content-Type: application/json" \ -d '{ "input": "The food was delicious and the waiter...", "model": "text-embedding-ada-002", "encoding_format": "float" }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "object": "list", "data": [{ "object": "embedding", "embedding": [Stella Page 149 "embedding": [0.0023064255, -0.009327292, ... (1536 floats total for ada-002) -0.0028842222,], "index": 0 }], "model": "text-embedding-ada-002", "usage": { "prompt_tokens": 8, "total_tokens": 8 } } Fine-tuning Manage fine-tuning jobs to tailor a model to your specific training data. Related guide: Fine-tune models The fine-tuning job object The fine_tuning.job object represents a fine-tuning job that has been created through the API. id string The object identifier, which can be referenced in the API endpoints. created_at integer The Unix timestamp (in seconds) for when the fine-tuning job was created. error object or null For fine-tuning jobs that have failed, this will contain more information on the cause of the failure. Show properties fine_tuned_model string or null The name of the fine-tuned model that is being created. The value will be null if the fine-tuning job is still running. finished_at integer or null The Unix timestamp (in seconds) for when the fine-tuning job was finished. The value will be null if the fine-tuning job is still running. hyperparameters object The hyperparameters used for the fine-tuning job. See the fine-tuning guide for more details. Show properties model string The base model that is being fine-tuned. object string The object type, which is always "fine_tuning.job". organization_id string The organization that owns the fine-tuning job. result_files array The compiled results file ID(s) for the fine-tuning job. You can retrieve the results with the Files API. status string The current status of the fine-tuning job, which can be either validating_files, queued, running, succeeded, failed, or cancelled. trained_tokens Stella Page 150 trained_tokens integer or null The total number of billable tokens processed by this fine-tuning job. The value will be null if the fine- tuning job is still running. training_file string The file ID used for training. You can retrieve the training data with the Files API. validation_file string or null The file ID used for validation. You can retrieve the validation results with the Files API. The fine-tuning job object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 { "object": "fine_tuning.job", "id": "ftjob-abc123", "model": "davinci-002", "created_at": 1692661014, "finished_at": 1692661190, "fine_tuned_model": "ft:davinci-002:my-org:custom_suffix:7q8mpxmy", "organization_id": "org-123", "result_files": ["file-abc123"], "status": "succeeded", "validation_file": null, "training_file": "file-abc123", "hyperparameters": { "n_epochs": 4, }, "trained_tokens": 5768 } Create fine-tuning job POST https://api.openai.com/v1/fine_tuning/jobs Creates a job that fine-tunes a specified model from a given dataset. Response includes details of the enqueued job including job status and the name of the fine-tuned models once complete. Learn more about fine-tuning Request body model string Required The name of the model to fine-tune. You can select one of the supported models. training_file Stella Page 151 training_file string Required The ID of an uploaded file that contains training data. See upload file for how to upload a file. Your dataset must be formatted as a JSONL file. Additionally, you must upload your file with the purpose fine-tune. See the fine-tuning guide for more details. hyperparameters object Optional The hyperparameters used for the fine-tuning job. Show properties suffix string or null Optional Defaults to null A string of up to 18 characters that will be added to your fine-tuned model name. For example, a suffix of "custom-model-name" would produce a model name like ft:gpt-3.5- turbo:openai:custom-model-name:7p4lURel. validation_file string or null Optional The ID of an uploaded file that contains validation data. If you provide this file, the data is used to generate validation metrics periodically during fine-tuning. These metrics can be viewed in the fine-tuning results file. The same data should not be present in both train and validation files. Your dataset must be formatted as a JSONL file. You must upload your file with the purpose fine- tune. See the fine-tuning guide for more details. Returns A fine-tuning.job object. NO HYPERPARAMETERSHYPERPARAMETERSVALIDATION FILE Example request curl Copy■ 1 2 3 4 5 6 7 curl https://api.openai.com/v1/fine_tuning/jobs \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "training_file": "file-BK7bzQj3FfZFXr7DbL6xJwfo", "model": "gpt-3.5-turbo" }' Response Copy■ 1 2 3 4 5 6 Stella Page 152 6 7 8 9 10 11 12 { "object": "fine_tuning.job", "id": "ftjob-abc123", "model": "gpt-3.5-turbo-0613", "created_at": 1614807352, "fine_tuned_model": null, "organization_id": "org-123", "result_files": [], "status": "queued", "validation_file": null, "training_file": "file-abc123", } List fine-tuning jobs GET https://api.openai.com/v1/fine_tuning/jobs List your organization's fine-tuning jobs Query parameters after string Optional Identifier for the last job from the previous pagination request. limit integer Optional Defaults to 20 Number of fine-tuning jobs to retrieve. Returns A list of paginated fine-tuning job objects. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/fine_tuning/jobs?limit=2 \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 { "object": "list", Stella Page 153 "object": "list", "data": [{ "object": "fine_tuning.job.event", "id": "ft-event-TjX0lMfOniCZX64t9PUQT5hn", "created_at": 1689813489, "level": "warn", "message": "Fine tuning process stopping due to job cancellation", "data": null, "type": "message" }, { ... }, { ... }], "has_more": true } Retrieve fine-tuning job GET https://api.openai.com/v1/fine_tuning/jobs/{fine_tuning_job_id} Get info about a fine-tuning job. Learn more about fine-tuning Path parameters fine_tuning_job_id string Required The ID of the fine-tuning job. Returns The fine-tuning object with the given ID. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/fine_tuning/jobs/ft-AF1WoRqd3aJAHsqc9NY7iL8F \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 { "object": "fine_tuning.job", "id": "ftjob-abc123", "model": "davinci-002", "created_at": 1692661014, "finished_at": 1692661190, "fine_tuned_model": "ft:davinci-002:my-org:custom_suffix:7q8mpxmy", Stella Page 154 "fine_tuned_model":

"ft:davinci-002:my-org:custom_suffix:7q8mpxmy", "organization_id": "org-123", "result_files": ["file-abc123"], "status": "succeeded", "validation_file": null, "training_file": "file-abc123", "hyperparameters": { "n_epochs": 4, }, "trained_tokens": 5768 } Cancel fine-tuning POST https://api.openai.com/v1/fine_tuning/jobs/{fine_tuning_job_id}/cancel Immediately cancel a fine-tune job. Path parameters fine_tuning_job_id string Required The ID of the fine-tuning job to cancel. Returns The cancelled fine-tuning object. Example request curl Copy■ 1 2 curl -X POST https://api.openai.com/v1/fine_tuning/jobs/ftjob-abc123/cancel \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 { "object": "fine_tuning.job", "id": "ftjob-abc123", "model": "gpt-3.5-turbo-0613", "created_at": 1689376978, "fine_tuned_model": null, "organization_id": "org-123", "result_files": [], "hyperparameters": { "n_epochs": "auto" }, "status": "cancelled", "validation_file": "file-abc123", "training_file": "file-abc123" Stella Page 155 "training_file": "file-abc123" } The fine-tuning job event object Fine-tuning job event object id string created_at integer level string message string object string The fine-tuning job event object Copy■ 1 2 3 4 5 6 7 { "object": "fine_tuning.job.event", "id": "ftevent-abc123" "created_at": 1677610602, "level": "info", "message": "Created fine-tuning job" } List fine-tuning events GET https://api.openai.com/v1/fine_tuning/jobs/{fine_tuning_job_id}/events Get status updates for a fine-tuning job. Path parameters fine_tuning_job_id string Required The ID of the fine-tuning job to get events for. Query parameters after string Optional Identifier for the last event from the previous pagination request. limit integer Optional Defaults to 20 Number of events to retrieve. Returns A list of fine-tuning event objects. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/fine_tuning/jobs/ftjob-abc123/events \ -H "Authorization: Bearer $OPENAI_API_KEY" Stella Page 156 -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 { "object": "list", "data": [{ "object": "fine_tuning.job.event", "id": "ft-event-ddTJfwuMVpfLXseO0Am0Gqjm", "created_at": 1692407401, "level": "info", "message": "Fine tuning job successfully completed", "data": null, "type": "message" }, { "object": "fine_tuning.job.event", "id": "ft-event-tyiGuB72evQncpH87xe505Sv", "created_at": 1692407400, "level": "info", "message": "New fine-tuned model created: ft:gpt-3.5-turbo:openai::7p4lURel", "data": null, "type": "message" }], "has_more": true } Files Files are used to upload documents that can be used with features like Assistants and Fine-tuning. The File object The File object represents a document that has been uploaded to OpenAI. id string The file identifier, which can be referenced in the API endpoints. bytes integer The size of the file, in bytes. Stella Page 157 The size of the file, in bytes. created_at integer The Unix timestamp (in seconds) for when the file was created. filename string The name of the file. object string The object type, which is always file. purpose string The intended purpose of the file. Supported values are fine-tune, fine-tune-results, assistants, and assistants_output. status Deprecated string Deprecated. The current status of the file, which can be either uploaded, processed, or error. status_details Deprecated string Deprecated. For details on why a fine-tuning training file failed validation, see the error field on fine_tuning.job. The File object Copy■ 1 2 3 4 5 6 7 8 { "id": "file-BK7bzQj3FfZFXr7DbL6xJwfo", "object": "file", "bytes": 120000, "created_at": 1677610602, "filename": "salesOverview.pdf", "purpose": "assistants", } List files GET https://api.openai.com/v1/files Returns a list of files that belong to the user's organization. Query parameters purpose string Optional Only return files with the given purpose. Returns A list of File objects. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/files \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Stella Page 158 Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 { "data": [{ "id": "file-abc123", "object": "file", "bytes": 175, "created_at": 1613677385, "filename": "salesOverview.pdf", "purpose": "assistants", }, { "id": "file-abc123", "object": "file", "bytes": 140, "created_at": 1613779121, "filename": "puppy.jsonl", "purpose": "fine-tune", }], "object": "list" } Upload file POST https://api.openai.com/v1/files Upload a file that can be used across various endpoints/features. The size of all the files uploaded by one organization can be up to 100 GB. The size of individual files for can be a maximum of 512MB. See the Assistants Tools guide to learn more about the types of files supported. The Fine-tuning API only supports .jsonl files. Please contact us if you need to increase these storage limits. Request body file string Required The File object (not file name) to be uploaded. purpose string Required The intended purpose of the uploaded file. Use "fine-tune" for Fine-tuning and "assistants" for Assistants and Messages. This allows us to Stella Page 159 Use "fine-tune" for Fine-tuning and "assistants" for Assistants and Messages. This allows us to validate the format of the uploaded file is correct for fine-tuning. Returns The uploaded File object. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/files \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -F purpose="fine-tune" \ -F file="@mydata.jsonl" Response Copy■ 1 2 3 4 5 6 7 8 { "id": "file-BK7bzQj3FfZFXr7DbL6xJwfo", "object": "file", "bytes": 120000, "created_at": 1677610602, "filename": "mydata.jsonl", "purpose": "fine-tune", } Delete file DELETE https://api.openai.com/v1/files/{file_id} Delete a file. Path parameters file_id string Required The ID of the file to use for this request. Returns Deletion status. Example request curl Copy■ 1 2 3 curl https://api.openai.com/v1/files/file-abc123 \ -X DELETE \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 Stella Page 160 3 4 5 { "id": "file-abc123", "object": "file", "deleted": true } Retrieve file GET https://api.openai.com/v1/files/{file_id} Returns information about a specific file. Path parameters file_id string Required The ID of the file to use for this request. Returns The File object matching the specified ID. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/files/file-BK7bzQj3FfZFXr7DbL6xJwfo \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 { "id": "file-BK7bzQj3FfZFXr7DbL6xJwfo", "object": "file", "bytes": 120000, "created_at": 1677610602, "filename": "mydata.jsonl", "purpose": "fine-tune", } Retrieve file content GET https://api.openai.com/v1/files/{file_id}/content Returns the contents of the specified file. Path parameters file_id string Required The ID of the file to use for this request. Returns The file content. Example request curl Copy■ Stella Page 161 Copy■ 1 2 curl https://api.openai.com/v1/files/file-BK7bzQj3FfZFXr7DbL6xJwfo/content \ -H "Authorization: Bearer $OPENAI_API_KEY" > file.jsonl Images Given a prompt and/or an input image, the model will generate a new image. Related guide: Image generation The image object Represents the url or the content of an image generated by the OpenAI API. b64_json string The base64-encoded JSON of the generated image, if response_format is b64_json. url string The URL of the generated image, if response_format is url (default). revised_prompt string The prompt that was used to generate the image, if there was any revision to the prompt. The image object Copy■ 1 2 3 4 { "url": "...", "revised_prompt": "..." } Create image POST https://api.openai.com/v1/images/generations Creates an image given a prompt. Request body prompt string Required A text description of the desired image(s). The maximum length is 1000 characters for dall-e-2 and 4000 characters for dall-e-3. model string Optional Defaults to dall-e-2 The model to use for image generation. n integer or null Optional Defaults to 1 The number of images to generate. Must be between 1 and 10. For dall-e-3, only n=1 is supported. quality string Optional Defaults to standard The quality of the image that will be generated. hd creates images with finer details and greater consistency across the image. This param is only supported for dall-e-3. response_format string or null Optional Stella Page 162 Optional Defaults to url The format in which the generated images are returned. Must be one of url or b64_json. size string or null Optional Defaults to 1024x1024 The size of the generated images. Must be one of 256x256, 512x512, or 1024x1024 for dall-e-2. Must be one of 1024x1024, 1792x1024, or 1024x1792 for dall-e-3 models. style string or null Optional Defaults to vivid The style of the generated images. Must be one of vivid or natural. Vivid causes the model to lean towards generating hyper-real and dramatic images. Natural causes the model to produce more natural, less hyper-real looking images. This param is only supported for dall-e-3. user string Optional A unique identifier representing your end-user, which can help OpenAI to monitor and detect abuse. Learn more. Returns Returns a list of image objects. Example request curl Copy■ 1 2 3 4 5 6 7 8 9 curl https://api.openai.com/v1/images/generations \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "model": "dall-e-3", "prompt": "A cute baby sea otter", "n": 1, "size": "1024x1024" }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 { "created": 1589478378, "data": [{ "url": "https://..." }, { Stella Page 163 { "url": "https://..." }] } Create image edit POST https://api.openai.com/v1/images/edits Creates an edited or extended image given an original image and a prompt. Request body image string Required The image to edit. Must be a valid PNG file, less than 4MB, and square. If mask is not provided, image must have transparency, which will be used as the mask. prompt string Required A text description of the desired image(s). The maximum length is 1000 characters. mask string Optional An additional image whose fully transparent areas (e.g. where alpha is zero) indicate where image should be edited. Must be a valid PNG file, less than 4MB, and have the same dimensions as image. model string Optional Defaults to dall-e-2 The model to use for image generation. Only dall-e-2 is supported at this time. n integer or null Optional Defaults to 1 The number of images to generate. Must be between 1 and 10. size string or null Optional Defaults to 1024x1024 The size of the generated images. Must be one of 256x256, 512x512, or 1024x1024. response_format string or null Optional Defaults to url The format in which the generated images are returned. Must be one of url or b64_json. user string Optional A unique identifier representing your end-user, which can help OpenAI to monitor and detect abuse. Learn more. Returns Returns a list of image objects. Example request curl Copy■ 1 2 3 4 5 6 Stella Page 164 6 7 curl https://api.openai.com/v1/images/edits \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -F image="@otter.png" \ -F mask="@mask.png" \ -F prompt="A cute baby sea otter wearing a beret" \ -F n=2 \ -F size="1024x1024" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 { "created": 1589478378, "data": [{ "url": "https://..." }, { "url": "https://..." }] } Create image variation POST https://api.openai.com/v1/images/variations Creates a variation of a given image. Request body image string Required The image to use as the basis for the variation(s). Must be a valid PNG file, less than 4MB, and square. model string Optional Defaults to dall-e-2 The model to use for image generation. Only dall-e-2 is supported at this time. n integer or null Optional Defaults to 1 The number of images to generate. Must be between 1 and 10. For dall-e-3, only n=1 is supported. response_format string or null Optional Defaults to url The format in which the generated images are returned. Must be one of url or b64_json. size string or null Optional Defaults to 1024x1024 The size of the generated images. Must be one of 256x256, 512x512, or 1024x1024. user Stella Page 165 user string Optional A unique identifier representing your end-user, which can help OpenAI to monitor and detect abuse. Learn more. Returns Returns a list of image objects. Example request curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/images/variations \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -F image="@otter.png" \ -F n=2 \ -F size="1024x1024" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 { "created": 1589478378, "data": [{ "url": "https://..." }, { "url": "https://..." }] } Models List and describe the various models available in the API. You can refer to the Models documentation to understand what models are available and the differences between them. The model object Describes an OpenAI model offering that can be used with the API. id string The model identifier, which can be referenced in the API endpoints. created integer The Unix timestamp (in seconds) when the model was created. object string Stella Page 166 string The object type, which is always "model". owned_by string The organization that owns the model. The model object Copy■ 1 2 3 4 5 6 { "id": "davinci", "object": "model", "created": 1686935002, "owned_by": "openai" } List models GET https://api.openai.com/v1/models Lists the currently available models, and provides basic information about each one such as the owner and availability. Returns A list of model objects. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/models \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 { "object": "list", "data": [Stella Page 167 "data": [{ "id": "model-id-0", "object": "model", "created": 1686935002, "owned_by": "organization-owner" }, { "id": "model-id-1", "object": "model", "created": 1686935002, "owned_by": "organization-owner", }, { "id": "model-id-2", "object": "model", "created": 1686935002, "owned_by": "openai" },], "object": "list" } Retrieve model GET https://api.openai.com/v1/models/{model} Retrieves a model instance, providing basic information about the model such as the owner and permissioning. Path parameters model string Required The ID of the model to use for this request Returns The model object matching the specified ID. Example request gpt-3.5-turbo-instruct curl Copy■ 1 2 curl https://api.openai.com/v1/models/gpt-3.5-turbo-instruct \ -H "Authorization: Bearer $OPENAI_API_KEY" Response gpt-3.5-turbo-instruct Copy■ 1 2 3 4 5 6 { "id": "gpt-3.5-turbo-instruct", "object": "model", "created": 1686935002, "owned_by": "openai" } Delete fine-tune model DELETE https://api.openai.com/v1/models/{model} Delete a fine-tuned model. You must have the Owner role in your organization to delete Stella Page 168 Delete a fine-tuned model. You must have the Owner role in your organization to delete a model. Path parameters model string Required The model to delete Returns Deletion status. Example request curl Copy■ 1 2 3 curl https://api.openai.com/v1/models/ft:gpt-3.5-turbo:acemeco:suffix:abc123 \ -X DELETE \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 { "id": "ft:gpt-3.5-turbo:acemeco:suffix:abc123", "object": "model", "deleted": true } Moderations Given a input text, outputs if the model classifies it as violating OpenAI's content policy. Related guide: Moderations The moderation object Represents policy compliance report by OpenAI's content moderation model against a given input. id string The unique identifier for the moderation request. model string The model used to generate the moderation results. results array A list of moderation objects. Show properties The moderation object Copy■ 1 2 3 4 5 6 7 8 Stella Page 169 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 { "id": "modr-XXXXX", "model": "text-moderation-005", "results": [{ "flagged": true, "categories": { "sexual": false, "hate": false, "harassment": false, "self-harm": false, "sexual/minors": false, "hate/threatening": false, "violence/graphic": false, "self-harm/intent": false, "self-harm/instructions": false, "harassment/threatening": true, "violence": true, }, "category_scores": { "sexual": 1.2282071e-06, "hate": 0.010696256, "harassment": 0.29842457, "self-harm": 1.5236925e-08, "sexual/minors": 5.7246268e-08, "hate/threatening": 0.0060676364, "violence/graphic": 4.435014e-06, "self-harm/intent": 8.098441e-10, "self-harm/instructions": 2.8498655e-11, "harassment/threatening": 0.63055265, "violence": 0.99011886, } }] } Create moderation POST https://api.openai.com/v1/moderations Stella Page 170 POST https://api.openai.com/v1/moderations Classifies if text violates OpenAI's Content Policy Request body input string or array Required The input text to classify model string Optional Defaults to text-moderation-latest Two content moderations models are available: text-moderation-stable and text-moderation-latest. The default is text-moderation-latest which will be automatically upgraded over time. This ensures you are always using our most accurate model. If you use text-moderation-stable, we will provide advanced notice before updating the model. Accuracy of text-moderation-stable may be slightly lower than for text-moderation-latest. Returns A moderation object. Example request curl Copy■ 1 2 3 4 5 6 curl https://api.openai.com/v1/moderations \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "input": "I want to kill them." }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 Stella Page 171 26 27 28 29 30 31 32 33 34 35 { "id": "modr-XXXXX", "model": "text-moderation-005", "results": [{ "flagged": true, "categories": { "sexual": false, "hate": false, "harassment": false, "self-harm": false, "sexual/minors": false, "hate/threatening": false, "violence/graphic": false, "self-harm/intent": false, "self-harm/instructions": false, "harassment/threatening": true, "violence": true, }, "category_scores": { "sexual": 1.2282071e-06, "hate": 0.010696256, "harassment": 0.29842457, "self-harm": 1.5236925e-08, "sexual/minors": 5.7246268e-08, "hate/threatening": 0.0060676364, "violence/graphic": 4.435014e-06, "self-harm/intent": 8.098441e-10, "self-harm/instructions": 2.8498655e-11, "harassment/threatening": 0.63055265, "violence": 0.99011886, } }] } Assistants Beta Build assistants that can call models and use tools to perform tasks. Get started with the Assistants API The assistant object Beta Represents an assistant that can call the model and use tools. id string The identifier, which can be referenced in API endpoints. object string The object type, which is always assistant. created_at integer The Unix timestamp (in seconds) for when the assistant was created. Stella Page 172 The Unix timestamp (in seconds) for when the assistant was created. name string or null The name of the assistant. The maximum length is 256 characters. description string or null The description of the assistant. The maximum length is 512 characters. model string ID of the model to use. You can use the List models API to see all of your available models, or see our Model overview for descriptions of them. instructions string or null The system instructions that the assistant uses. The maximum length is 32768 characters. tools array A list of tool enabled on the assistant. There can be a maximum of 128 tools per assistant. Tools can be of types code_interpreter, retrieval, or function. Show possible types file_ids array A list of file IDs attached to this assistant. There can be a maximum of 20 files attached to the assistant. Files are ordered by their creation date in ascending order. metadata map Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. The assistant object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 { "id": "asst_abc123", "object": "assistant", "created_at": 1698984975, "name": "Math Tutor", "description": null, "model": "gpt-4", "instructions": "You are a personal math tutor. When asked a question, write and run Python code to answer the question.", "tools": [{ "type": "code_interpreter" }], "file_ids": [], "metadata": {} } Create assistant Stella Page 173 Create assistant Beta POST https://api.openai.com/v1/assistants Create an assistant with a model and instructions. Request body model Required ID of the model to use. You can use the List models API to see all of your available models, or see our Model overview for descriptions of them. name string or null Optional The name of the assistant. The maximum length is 256 characters. description string or null Optional The description of the assistant. The maximum length is 512 characters. instructions string or null Optional The system instructions that the assistant uses. The maximum length is 32768 characters. tools array Optional Defaults to [] A list of tool enabled on the assistant. There can be a maximum of 128 tools per assistant. Tools can be of types code_interpreter, retrieval, or function. Show possible types file_ids array Optional Defaults to [] A list of file IDs attached to this assistant. There can be a maximum of 20 files attached to the assistant. Files are ordered by their creation date in ascending order. metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns An assistant object. CODE INTERPRETERFILES Example request curl Copy■ 1 2 3 4 5 6 7 8 9 10 curl "https://api.openai.com/v1/assistants" \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" \ Stella Page 174 -H "OpenAI-Beta: assistants=v1" \ -d '{ "instructions": "You are a personal math tutor. When asked a question, write and run Python code to answer the question.", "name": "Math Tutor" "tools": [{"type": "code_interpreter"}], "model": "gpt-4" }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 { "id": "asst_abc123", "object": "assistant", "created_at": 1698984975, "name": "Math Tutor", "description": null, "model": "gpt-4", "instructions": "You are a personal math tutor. When asked a question, write and run Python code to answer the question.", "tools": [{ "type": "code_interpreter" }], "file_ids": [], "metadata": {} } Retrieve assistant Beta GET https://api.openai.com/v1/assistants/{assistant_id} Retrieves an assistant. Path parameters assistant_id string Required The ID of the assistant to retrieve. Returns The assistant object matching the specified ID. Example request curl Copy■ 1 2 3 4 Stella Page 175 4 curl https://api.openai.com/v1/assistants/asst_abc123 \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 { "id": "asst_abc123", "object": "assistant", "created_at": 1699009709, "name": "HR Helper", "description": null, "model": "gpt-4", "instructions": "You are an HR bot, and you have access to files to answer employee questions about company policies.", "tools": [{ "type": "retrieval" }], "file_ids": ["file-abc123"], "metadata": {} } Modify assistant Beta POST https://api.openai.com/v1/assistants/{assistant_id} Modifies an assistant. Path parameters assistant_id string Required The ID of the assistant to modify. Request body model Optional ID of the model to use. You can use the List models API to see all of your available models, or see our Model overview for descriptions of them. name string or null Optional The name of the assistant. The maximum length is 256 characters. Stella Page 176 The name of the assistant. The maximum length is 256 characters. description string or null Optional The description of the assistant. The maximum length is 512 characters. instructions string or null Optional The system instructions that the assistant uses. The maximum length is 32768 characters. tools array Optional Defaults to [] A list of tool enabled on the assistant. There can be a maximum of 128 tools per assistant. Tools can be of types code_interpreter, retrieval, or function. Show possible types file_ids array Optional Defaults to [] A list of File IDs attached to this assistant. There can be a maximum of 20 files attached to the assistant. Files are ordered by their creation date in ascending order. If a file was previosuly attached to the list but does not show up in the list, it will be deleted from the assistant. metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns The modified assistant object. Example request curl Copy■ 1 2 3 4 5 6 7 8 9 10 curl https://api.openai.com/v1/assistants/asst_abc123 \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" \ -d '{ "instructions": "You are an HR bot, and you have access to files to answer employee questions about company policies. Always response with info from either of the files.", "tools": [{"type": "retrieval"}], "model": "gpt-4", "file_ids": ["file-abc123", "file-abc456"] }' Response Copy■ 1 2 3 4 5 Stella Page 177 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 { "id": "asst_abc123", "object": "assistant", "created_at": 1699009709, "name": "HR Helper", "description": null, "model": "gpt-4", "instructions": "You are an HR bot, and you have access to files to answer employee questions about company policies. Always response with info from either of the files.", "tools": [{ "type": "retrieval" }], "file_ids": ["file-abc123", "file-abc456"], "metadata": {} } Delete assistant Beta DELETE https://api.openai.com/v1/assistants/{assistant_id} Delete an assistant. Path parameters assistant_id string Required The ID of the assistant to delete. Returns Deletion status Example request curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/assistants/asst_abc123 \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" \ -X DELETE Response Copy■ Stella Page 178 Copy■ 1 2 3 4 5 { "id": "asst_abc123", "object": "assistant.deleted", "deleted": true } List assistants Beta GET https://api.openai.com/v1/assistants Returns a list of assistants. Query parameters limit integer Optional Defaults to 20 A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 20. order string Optional Defaults to desc Sort order by the created_at timestamp of the objects. asc for ascending order and desc for descending order. after string Optional A cursor for use in pagination. after is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include after=obj_foo in order to fetch the next page of the list. before string Optional A cursor for use in pagination. before is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include before=obj_foo in order to fetch the previous page of the list. Returns A list of assistant objects. Example request curl Copy■ 1 2 3 4 curl "https://api.openai.com/v1/assistants?order=desc&limit;=20" \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" Response Copy■ 1 2 3 4 5 Stella Page 179 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 { "object": "list", "data": [{ "id": "asst_abc123", "object": "assistant", "created_at": 1698982736, "name": "Coding Tutor", "description": null, "model": "gpt-4", "instructions": "You are a helpful assistant designed to make me better at coding!", "tools": [], "file_ids": [], "metadata": {} }, { "id": "asst_abc456", "object": "assistant", "created_at": 1698982718, "name": "My Assistant", "description": null, "model": "gpt-4", "instructions": "You are a helpful assistant designed to make me better at coding!", "tools": [], "file_ids": [], "metadata": {} }, Stella Page 180 }, { "id": "asst_abc789", "object": "assistant", "created_at": 1698982643, "name": null, "description": null, "model": "gpt-4", "instructions": null, "tools": [], "file_ids": [], "metadata": {} }], "first_id": "asst_abc123", "last_id": "asst_abc789", "has_more": false } The assistant file object Beta A list of Files attached to an assistant. id string The identifier, which can be referenced in API endpoints. object string The object type, which is always assistant.file. created_at integer The Unix timestamp (in seconds) for when the assistant file was created. assistant_id string The assistant ID that the file is attached to. The assistant file object Copy■ 1 2 3 4 5 6 { "id": "file-wB6RM6wHdA49HfS2DJ9fEyrH", "object": "assistant.file", "created_at": 1699055364, "assistant_id": "asst_FBOFvAOHhwEWMghbMGseaPGQ" } Create assistant file Beta POST https://api.openai.com/v1/assistants/{assistant_id}/files Create an assistant file by attaching a File to an assistant. Path parameters assistant_id string Required The ID of the assistant for which to create a File. Request body file_id string Required Stella Page 181 Required A File ID (with purpose="assistants") that the assistant should use. Useful for tools like retrieval and code_interpreter that can access files. Returns An assistant file object. Example request curl Copy■ 1 2 3 4 5 6 7 curl https://api.openai.com/v1/assistants/asst_FBOFvAOHhwEWMghbMGseaPGQ/files \ -H 'Authorization: Bearer $OPENAI_API_KEY"' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' \ -d '{ "file_id": "file-wB6RM6wHdA49HfS2DJ9fEyrH" }' Response Copy■ 1 2 3 4 5 6 { "id": "file-wB6RM6wHdA49HfS2DJ9fEyrH", "object": "assistant.file", "created_at": 1699055364, "assistant_id": "asst_FBOFvAOHhwEWMghbMGseaPGQ" } Retrieve assistant file Beta GET https://api.openai.com/v1/assistants/{assistant_id}/files/{file_id} Retrieves an AssistantFile. Path parameters assistant_id string Required The ID of the assistant who the file belongs to. file_id string Required The ID of the file we're getting. Returns The assistant file object matching the specified ID. Example request curl Copy■ 1 2 3 4 Stella Page 182 4 curl https://api.openai.com/v1/assistants/asst_FBOFvAOHhwEWMghbMGseaPGQ/files/file-wB6RM6wHdA49HfS2DJ9fEyrH \ -H 'Authorization: Bearer $OPENAI_API_KEY"' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 { "id": "file-wB6RM6wHdA49HfS2DJ9fEyrH", "object": "assistant.file", "created_at": 1699055364, "assistant_id": "asst_FBOFvAOHhwEWMghbMGseaPGQ" } Delete assistant file Beta DELETE https://api.openai.com/v1/assistants/{assistant_id}/files/{file_id} Delete an assistant file. Path parameters assistant_id string Required The ID of the assistant that the file belongs to. file_id string Required The ID of the file to delete. Returns Deletion status Example request curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/assistants/asst_DUGk5I7sK0FpKeijvrO30z9J/files/file-9F1ex49ipEnK zyLUNnCA0 Yzx \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' \ -X DELETE Response Copy■ 1 2 3 4 5 { id: "file-BK7bzQj3FfZFXr7DbL6xJwfo", object: "assistant.file.deleted", Stella Page 183 object: "assistant.file.deleted", deleted: true } List assistant files Beta GET https://api.openai.com/v1/assistants/{assistant_id}/files Returns a list of assistant files. Path parameters assistant_id string Required The ID of the assistant the file belongs to. Query parameters limit integer Optional Defaults to 20 A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 20. order string Optional Defaults to desc Sort order by the created_at timestamp of the objects. asc for ascending order and desc for descending order. after string Optional A cursor for use in pagination. after is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include after=obj_foo in order to fetch the next page of the list. before string Optional A cursor for use in pagination. before is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include before=obj_foo in order to fetch the previous page of the list. Returns A list of assistant file objects. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/assistants/asst_DUGk5I7sK0FpKeijvrO30z9J/files \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 7 Stella Page 184 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "object": "list", "data": [{ "id": "file-dEWwUbt2UGHp3v0e0DpCzemP", "object": "assistant.file", "created_at": 1699060412, "assistant_id": "asst_DUGk5I7sK0FpKeijvrO30z9J" }, { "id": "file-9F1ex49ipEnKzyLUNnCA0Yzx", "object": "assistant.file", "created_at": 1699060412, "assistant_id": "asst_DUGk5I7sK0FpKeijvrO30z9J" }], "first_id": "file-dEWwUbt2UGHp3v0e0DpCzemP", "last_id": "file-9F1ex49ipEnKzyLUNnCA0Yzx", "has_more": false } Threads Beta Create threads that assistants can interact with. Related guide: Assistants The thread object Beta Represents a thread that contains messages. id string The identifier, which can be referenced in API endpoints. object string The object type, which is always thread. created_at integer The Unix timestamp (in seconds) for when the thread was created. metadata map Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. The thread object Copy■ 1 2 3 Stella Page 185 3 4 5 6 { "id": "thread_abc123", "object": "thread", "created_at": 1698107661, "metadata": {} } Create thread Beta POST https://api.openai.com/v1/threads Create a thread. Request body messages array Optional A list of messages to start the thread with. Show properties metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns A thread object. EMPTYMESSAGES Example request curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/threads \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" \ -d '' Response Copy■ 1 2 3 4 5 6 { "id": "thread_abc123", "object": "thread", "created_at": 1699012949, "metadata": {} } Retrieve thread Beta GET https://api.openai.com/v1/threads/{thread_id} Retrieves a thread. Stella Page 186 Retrieves a thread. Path parameters thread_id string Required The ID of the thread to retrieve. Returns The thread object matching the specified ID. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_abc123 \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" Response Copy■ 1 2 3 4 5 6 { "id": "thread_abc123", "object": "thread", "created_at": 1699014083, "metadata": {} } Modify thread Beta POST https://api.openai.com/v1/threads/{thread_id} Modifies a thread. Path parameters thread_id string Required The ID of the thread to modify. Only the metadata can be modified. Request body metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns The modified thread object matching the specified ID. Example request curl Copy■ 1 Stella Page 187 1 2 3 4 5 6 7 8 9 10 curl https://api.openai.com/v1/threads/thread_abc123 \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" \ -d '{ "metadata": { "modified": "true", "user": "abc123" } }' Response Copy■ 1 2 3 4 5 6 7 8 9 { "id": "thread_abc123", "object": "thread", "created_at": 1699014083, "metadata": { "modified": "true", "user": "abc123" } } Delete thread Beta DELETE https://api.openai.com/v1/threads/{thread_id} Delete a thread. Path parameters thread_id string Required The ID of the thread to delete. Returns Deletion status Example request curl Copy■ 1 2 3 4 5 curl https://api.openai.com/v1/threads/thread_abc123 \ -H "Content-Type: application/json" \ Stella Page 188 -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" \ -X DELETE

Response Copy■ 1 2 3 4 5 { "id": "thread_abc123", "object": "thread.deleted", "deleted": true } Messages Beta Create messages within threads Related guide: Assistants The message object Beta Represents a message within a thread. id string The identifier, which can be referenced in API endpoints. object string The object type, which is always thread.message. created_at integer The Unix timestamp (in seconds) for when the message was created. thread_id string The thread ID that this message belongs to. role string The entity that produced the message. One of user or assistant. content array The content of the message in array of text and/or images. Show possible types assistant_id string or null If applicable, the ID of the assistant that authored this message. run_id string or null If applicable, the ID of the run associated with the authoring of this message. file_ids array A list of file IDs that the assistant should use. Useful for tools like retrieval and code_interpreter that can access files. A maximum of 10 files can be attached to a message. metadata map Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. The message object Copy■ Stella Page 189 Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "id": "msg_dKYDWyQvtjDBi3tudL1yWKDa", "object": "thread.message", "created_at": 1698983503, "thread_id": "thread_RGUhOuO9b2nrktrmsQ2uSR6I", "role": "assistant", "content": [{ "type": "text", "text": { "value": "Hi! How can I help you today?", "annotations": [] } }], "file_ids": [], "assistant_id": "asst_ToSF7Gb04YMj8AMMm50ZLLtY", "run_id": "run_BjylUJgDqYK9bOhy4yjAiMrn", "metadata": {} } Create message Beta POST https://api.openai.com/v1/threads/{thread_id}/messages Create a message. Path parameters thread_id string Required The ID of the thread to create a message for. Request body role string Required The role of the entity that is creating the message. Currently only user is supported. content string Required The content of the message. file_ids array Optional Defaults to [] Stella Page 190 Defaults to [] A list of File IDs that the message should use. There can be a maximum of 10 files attached to a message. Useful for tools like retrieval and code_interpreter that can access and use files. metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns A message object. Example request curl Copy■ 1 2 3 4 5 6 7 8 curl https://api.openai.com/v1/threads/thread_abc123/messages \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" \ -d '{ "role": "user", "content": "How does AI work? Explain it in simple terms." }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "id": "msg_abc123", "object": "thread.message", "created_at": 1699017614, "thread_id": "thread_abc123", "role": "user", "content": [{ "type": "text", "text": { "value": "How does AI work? Explain it in simple terms.", Stella Page 191 "value": "How does AI work? Explain it in simple terms.", "annotations": [] } }], "file_ids": [], "assistant_id": null, "run_id": null, "metadata": {} } Retrieve message Beta GET https://api.openai.com/v1/threads/{thread_id}/messages/{message_id} Retrieve a message. Path parameters thread_id string Required The ID of the thread to which this message belongs. message_id string Required The ID of the message to retrieve. Returns The message object matching the specified ID. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_abc123/messages/msg_abc123 \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "id": "msg_abc123", "object": "thread.message", Stella Page 192 "object": "thread.message", "created_at": 1699017614, "thread_id": "thread_abc123", "role": "user", "content": [{ "type": "text", "text": { "value": "How does AI work? Explain it in simple terms.", "annotations": [] } }], "file_ids": [], "assistant_id": null, "run_id": null, "metadata": {} } Modify message Beta POST https://api.openai.com/v1/threads/{thread_id}/messages/{message_id} Modifies a message. Path parameters thread_id string Required The ID of the thread to which this message belongs. message_id string Required The ID of the message to modify. Request body metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns The modified message object. Example request curl Copy■ 1 2 3 4 5 6 7 8 9 10 curl https://api.openai.com/v1/threads/thread_abc123/messages/msg_abc123 \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta:

assistants=v1" \ -d '{ "metadata": { "modified": "true", Stella Page 193 "modified": "true", "user": "abc123" } }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 { "id": "msg_abc123", "object": "thread.message", "created_at": 1699017614, "thread_id": "thread_abc123", "role": "user", "content": [{ "type": "text", "text": { "value": "How does AI work? Explain it in simple terms.", "annotations": [] } }], "file_ids": [], "assistant_id": null, "run_id": null, "metadata": { "modified": "true", "user": "abc123" } } List messages Beta GET https://api.openai.com/v1/threads/{thread_id}/messages Returns a list of messages for a given thread. Path parameters thread_id string Required The ID of the thread the messages belong to. Query parameters limit Stella Page 194 limit integer Optional Defaults to 20 A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 20. order string Optional Defaults to desc Sort order by the created_at timestamp of the objects. asc for ascending order and desc for descending order. after string Optional A cursor for use in pagination. after is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include after=obj_foo in order to fetch the next page of the list. before string Optional A cursor for use in pagination. before is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include before=obj_foo in order to fetch the previous page of the list. Returns A list of message objects. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_abc123/messages \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -H "OpenAI-Beta: assistants=v1" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 Stella Page 195 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 { "object": "list", "data": [{ "id": "msg_abc123", "object": "thread.message", "created_at": 1699016383, "thread_id": "thread_abc123", "role": "user", "content": [{ "type": "text", "text": { "value": "How does AI work? Explain it in simple terms.", "annotations": [] } }], "file_ids": [], "assistant_id": null, "run_id": null, "metadata": {} }, { "id": "msg_abc456", "object": "thread.message", "created_at": 1699016383, "thread_id": "thread_abc123", "role": "user", "content": [{ "type": "text", "text": { "value": "Hello, what is AI?", "annotations": [] } }], "file_ids": [Stella Page 196 "file_ids": ["file-abc123"], "assistant_id": null, "run_id": null, "metadata": {} }], "first_id": "msg_abc123", "last_id": "msg_abc456", "has_more": false } The message file object Beta A list of files attached to a message. id string The identifier, which can be referenced in API endpoints. object string The object type, which is always thread.message.file. created_at integer The Unix timestamp (in seconds) for when the message file was created. message_id string The ID of the message that the File is attached to. The message file object Copy■ 1 2 3 4 5 6 7 { "id": "file-BK7bzQj3FfZFXr7DbL6xJwfo", "object": "thread.message.file", "created_at": 1698107661, "message_id": "message_QLoItBbqwyAJEzlTy4y9kOMM", "file_id": "file-BK7bzQj3FfZFXr7DbL6xJwfo" } Retrieve message file Beta GET https://api.openai.com/v1/threads/{thread_id}/messages/{message_id}/files/{file_id} Retrieves a message file. Path parameters thread_id string Required The ID of the thread to which the message and File belong. message_id string Required The ID of the message the file belongs to. file_id string Required The ID of the file being retrieved. Stella Page 197 The ID of the file being retrieved. Returns The message file object. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_RGUhOuO9b2nrktrmsQ2uSR6I/messages/msg_q3XhbGmMzsqEFa81gMLBDAVU/files/file-dEWwUbt2UGHp3v0e0DpCzemP \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 { "id": "file-dEWwUbt2UGHp3v0e0DpCzemP", "object": "thread.message.file", "created_at": 1699061776, "message_id": "msg_q3XhbGmMzsqEFa81gMLBDAVU" } List message files Beta GET https://api.openai.com/v1/threads/{thread_id}/messages/{message_id}/files Returns a list of message files. Path parameters thread_id string Required The ID of the thread that the message and files belong to. message_id string Required The ID of the message that the files belongs to. Query parameters limit integer Optional Defaults to 20 A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 20. order string Optional Defaults to desc Sort order by the created_at timestamp of the objects. asc for ascending order and desc for descending order. after string Optional Stella Page 198 Optional A cursor for use in pagination. after is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include after=obj_foo in order to fetch the next page of the list. before string Optional A cursor for use in pagination. before is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include before=obj_foo in order to fetch the previous page of the list. Returns A list of message file objects. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_RGUhOuO9b2nrktrmsQ2uSR6I/messages/msg_q3XhbGmMzsqE Fa81gMLBDAVU/files \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 { "object": "list", "data": [{ "id": "file-dEWwUbt2UGHp3v0e0DpCzemP", "object": "thread.message.file", "created_at": 1699061776, "message_id": "msg_q3XhbGmMzsqEFa81gMLBDAVU" }, { "id": "file-dEWwUbt2UGHp3v0e0DpCzemP", "object": "thread.message.file", "created_at": 1699061776, "message_id": "msg_q3XhbGmMzsqEFa81gMLBDAVU" }], Stella Page 199], "first_id": "file-dEWwUbt2UGHp3v0e0DpCzemP", "last_id": "file-dEWwUbt2UGHp3v0e0DpCzemP", "has_more": false } Runs Beta Represents an execution run on a thread. Related guide: Assistants The run object Beta Represents an execution run on a thread. id string The identifier, which can be referenced in API endpoints. object string The object type, which is always assistant.run. created_at integer The Unix timestamp (in seconds) for when the run was created. thread_id string The ID of the thread that was executed on as a part of this run. assistant_id string The ID of the assistant used for execution of this run. status string The status of the run, which can be either queued, in_progress, requires_action, cancelling, cancelled, failed, completed, or expired. required_action object or null Details on the action required to continue the run. Will be null if no action is required. Show properties last_error object or null The last error associated with this run. Will be null if there are no errors. Show properties expires_at integer The Unix timestamp (in seconds) for when the run will expire. started_at integer or null The Unix timestamp (in seconds) for when the run was started. cancelled_at integer or null The Unix timestamp (in seconds) for when the run was cancelled. failed_at integer or null The Unix timestamp (in seconds) for when the run failed. completed_at integer or null The Unix timestamp (in seconds) for when the run was completed. model string The model that the assistant used for this run. instructions string The instructions that the assistant used for this run. Stella Page 200 The instructions that the assistant used for this run. tools array The list of tools that the assistant used for this run. Show possible types file_ids array The list of File IDs the assistant used for this run. metadata map Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. The run object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 { "id": "run_example123", "object": "thread.run", "created_at": 1698107661, "assistant_id": "asst_gZ1aOomboBuYWPcXJx4vAYB0", "thread_id": "thread_adOpf7Jbb5Abymz0QbwxAh3c", "status": "completed", "started_at": 1699073476, "expires_at": null, "cancelled_at": null, "failed_at": null, "completed_at": 1699073498, "last_error": null, "model": "gpt-4", "instructions": null, "tools": [{"type": "retrieval"}, {"type": "code_interpreter"}], "file_ids": [], "metadata": {} } Create run Beta POST https://api.openai.com/v1/threads/{thread_id}/runs Create a run. Path parameters thread_id string Required The ID of the thread to run. Request body Stella Page 201 Request body assistant_id string Required The ID of the assistant to use to execute this run. model string or null Optional The ID of the Model to be used to execute this run. If a value is provided here, it will override the model associated with the assistant. If not, the model associated with the assistant will be used. instructions string or null Optional Override the default system message of the assistant. This is useful for modifying the behavior on a per-run basis. tools array or null Optional Override the tools the assistant can use for this run. This is useful for modifying the behavior on a per-run basis. Show possible types metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns A run object. Example request curl Copy■ 1 2 3 4 5 6 7 curl https://api.openai.com/v1/threads/thread_BDDwIqM4KgHibXX3mqmN3Lgs/runs \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' \ -d '{ "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ" }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 Stella Page 202 13 14 15 16 17 18 19 20 21 22 23 24 25 26 { "id": "run_UWvV94U0FQYiT2rlbBrdEVmC", "object": "thread.run", "created_at": 1699063290, "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id":

"thread_BDDwIqM4KgHibXX3mqmN3Lgs", "status": "queued", "started_at": 1699063290, "expires_at": null, "cancelled_at": null, "failed_at": null, "completed_at": 1699063291, "last_error": null, "model": "gpt-4", "instructions": null, "tools": [{ "type": "code_interpreter" }], "file_ids": ["file-9F1ex49ipEnKzyLUNnCA0Yzx", "file-dEWwUbt2UGHp3v0e0DpCzemP"], "metadata": {} } Retrieve run Beta GET https://api.openai.com/v1/threads/{thread_id}/runs/{run_id} Retrieves a run. Path parameters thread_id string Required The ID of the thread that was run. run_id string Required The ID of the run to retrieve. Returns The run object matching the specified ID. Example request curl Copy■ 1 2 3 Stella Page 203 3 curl https://api.openai.com/v1/threads/thread_BDDwIqM4KgHibXX3mqmN3Lgs/runs/run_5pyUEwhaPk11vCKiDneUWXXY \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 { "id": "run_5pyUEwhaPk11vCKiDneUWXXY", "object": "thread.run", "created_at": 1699075072, "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id": "thread_BDDwIqM4KgHibXX3mqmN3Lgs", "status": "completed", "started_at": 1699075072, "expires_at": null, "cancelled_at": null, "failed_at": null, "completed_at": 1699075073, "last_error": null, "model": "gpt-3.5-turbo", "instructions": null, "tools": [{ "type": "code_interpreter" }], "file_ids": ["file-9F1ex49ipEnKzyLUNnCA0Yzx", "file-dEWwUbt2UGHp3v0e0DpCzemP"], "metadata": {} } Modify run Beta POST https://api.openai.com/v1/threads/{thread_id}/runs/{run_id} Modifies a run. Path parameters Stella Page 204 Path parameters thread_id string Required The ID of the thread that was run. run_id string Required The ID of the run to modify. Request body metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns The modified run object matching the specified ID. Example request curl Copy■ 1 2 3 4 5 6 7 8 9 curl https://api.openai.com/v1/threads/thread_BDDwIqM4KgHibXX3mqmN3Lgs/runs/run_5pyUEwhaPk11vCKiDneUWXXY \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' \ -d '{ "metadata": { "user_id": "user_zmVY6FvuBDDwIqM4KgH" } }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 Stella Page 205 19 20 21 22 23 24 25 26 27 28 { "id": "run_5pyUEwhaPk11vCKiDneUWXXY", "object": "thread.run", "created_at": 1699075072, "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id": "thread_BDDwIqM4KgHibXX3mqmN3Lgs", "status": "completed", "started_at": 1699075072, "expires_at": null, "cancelled_at": null, "failed_at": null, "completed_at": 1699075073, "last_error": null, "model": "gpt-3.5-turbo", "instructions": null, "tools": [{ "type": "code_interpreter" }], "file_ids": ["file-9F1ex49ipEnKzyLUNnCA0Yzx", "file-dEWwUbt2UGHp3v0e0DpCzemP"], "metadata": { "user_id": "user_zmVY6FvuBDDwIqM4KgH" } } List runs Beta GET https://api.openai.com/v1/threads/{thread_id}/runs Returns a list of runs belonging to a thread. Path parameters thread_id string Required The ID of the thread the run belongs to. Query parameters limit integer Optional Defaults to 20 A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 20. order string Optional Defaults to desc Sort order by the created_at timestamp of the objects. asc for ascending order and desc for descending order. after string Optional Stella Page 206 Optional A cursor for use in pagination. after is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include after=obj_foo in order to fetch the next page of the list. before string Optional A cursor for use in pagination. before is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include before=obj_foo in order to fetch the previous page of the list. Returns A list of run objects. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_BDDwIqM4KgHibXX3mqmN3Lgs/runs \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 Stella Page 207 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 { "object": "list", "data": [{ "id": "run_5pyUEwhaPk11vCKiDneUWXXY", "object": "thread.run", "created_at": 1699075072, "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id": "thread_BDDwIqM4KgHibXX3mqmN3Lgs", "status": "completed", "started_at": 1699075072, "expires_at": null, "cancelled_at": null, "failed_at": null, "completed_at": 1699075073, "last_error": null, "model": "gpt-3.5-turbo", "instructions": null, "tools": [{ "type": "code_interpreter" }

], "file_ids": ["file-9F1ex49ipEnKzyLUNnCA0Yzx", "file-dEWwUbt2UGHp3v0e0DpCzemP"], "metadata": {} }, { "id": "run_UWvV94U0FQYiT2rlbBrdEVmC", "object": "thread.run", "created_at": 1699063290, "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id": "thread_BDDwIqM4KgHibXX3mqmN3Lgs", "status": "completed", "started_at": 1699063290, "expires_at": null, "cancelled_at": null, "failed_at": null, "completed_at": 1699063291, "last_error": null, "model": "gpt-3.5-turbo", "instructions": null, Stella Page 208 "instructions": null, "tools": [{ "type": "code_interpreter" }], "file_ids": ["file-9F1ex49ipEnKzyLUNnCA0Yzx", "file-dEWwUbt2UGHp3v0e0DpCzemP"], "metadata": {} }], "first_id": "run_5pyUEwhaPk11vCKiDneUWXXY", "last_id": "run_UWvV94U0FQYiT2rlbBrdEVmC", "has_more": false } Submit tool outputs to run Beta POST https://api.openai.com/v1/threads/{thread_id}/runs/{run_id}/submit_tool_outputs When a run has the status: "requires_action" and required_action.type is submit_tool_outputs, this endpoint can be used to submit the outputs from the tool calls once they're all completed. All outputs must be submitted in a single request. Path parameters thread_id string Required The ID of the thread to which this run belongs. run_id string Required The ID of the run that requires the tool output submission. Request body tool_outputs array Required A list of tools for which the outputs are being submitted. Show properties Returns The modified run object matching the specified ID. Example request curl Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 curl https://api.openai.com/v1/threads/thread_EdR8UvCDJ035LFEJZMt3AxCd/runs/run_PHLyHQYIQn4F7JrSXs lEYWwh/submit_tool_outputs \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ Stella Page 209 -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' \ -d '{ "tool_outputs": [{ "tool_call_id": "call_MbELIQcB72cq35Yzo2MRw5qs", "output": "28C" }] }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 { "id": "run_PHLyHQYIQn4F7JrSXslEYWwh", "object": "thread.run", "created_at": 1699075592, "assistant_id": "asst_IgmpQTah3ZfPHCVZjTqAY8Kv", "thread_id": "thread_EdR8UvCDJ035LFEJZMt3AxCd", "status": "queued", "started_at": 1699075592, Stella Page 210 "started_at": 1699075592, "expires_at": 1699076192, "cancelled_at": null, "failed_at": null, "completed_at": null, "last_error": null, "model": "gpt-4", "instructions": "You tell the weather.", "tools": [{ "type": "function", "function": { "name": "get_weather", "description": "Determine weather in my location", "parameters": { "type": "object", "properties": { "location": { "type": "string", "description": "The city and state e.g. San Francisco, CA" }, "unit": { "type": "string", "enum": ["c", "f"] } }, "required": ["location"] } } }], "file_ids": [], "metadata": {} } Cancel a run Beta POST https://api.openai.com/v1/threads/{thread_id}/runs/{run_id}/cancel Cancels a run that is in_progress. Path parameters thread_id string Required The ID of the thread to which this run belongs. run_id string Required The ID of the run to cancel. Returns The modified run object matching the specified ID. Example request curl Copy■ 1 2 3 4 Stella Page 211 4 curl https://api.openai.com/v1/threads/thread_1cjnJPXj8MFiqTx58jU9TivC/runs/run_BeRGmpGt2wb1VI22ZRniOkrR/cancel \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'OpenAI-Beta: assistants=v1' \ -X POST Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 { "id": "run_BeRGmpGt2wb1VI22ZRniOkrR", "object": "thread.run", "created_at": 1699076126, "assistant_id": "asst_IgmpQTah3ZfPHCVZjTqAY8Kv", "thread_id": "thread_1cjnJPXj8MFiqTx58jU9TivC", "status": "cancelling", "started_at": 1699076126, "expires_at": 1699076726, "cancelled_at": null, "failed_at": null, "completed_at": null, "last_error": null, "model": "gpt-4", "instructions": "You summarize books.", "tools": [{ "type": "retrieval" }], "file_ids": [], "metadata": {} } Create thread and run Beta POST https://api.openai.com/v1/threads/runs Create a thread and run it in one request. Request body assistant_id string Required The ID of the assistant to use to execute this run. thread Stella Page 212 thread object Optional Show properties model string or null Optional The ID of the Model to be used to execute this run. If a value is provided here, it will override the model associated with the assistant. If not, the model associated with the assistant will be used. instructions string or null Optional Override the default system message of the assistant. This is useful for modifying the behavior on a per-run basis. tools array or null Optional Override the tools the assistant can use for this run. This is useful for modifying the behavior on a per-run basis. metadata map Optional Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. Returns A run object. Example request curl Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 curl https://api.openai.com/v1/threads/runs \ -H 'Authorization:

Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' \ -d '{ "assistant_id": "asst_IgmpQTah3ZfPHCVZjTqAY8Kv", "thread": { "messages": [{"role": "user", "content": "Explain deep learning to a 5 year old."}] } }' Response Copy■ 1 2 3 4 5 6 Stella Page 213 6 7 8 9 10 11 12 13 14 15 16 17 18 19 { "id": "run_3Qudf05GGhCleEg9ggwfJQih", "object": "thread.run", "created_at": 1699076792, "assistant_id": "asst_IgmpQTah3ZfPHCVZjTqAY8Kv", "thread_id": "thread_Ec3eKZcWI00WDZRC7FZci8hP", "status": "queued", "started_at": null, "expires_at": 1699077392, "cancelled_at": null, "failed_at": null, "completed_at": null, "last_error": null, "model": "gpt-4", "instructions": "You are a helpful assistant.", "tools": [], "file_ids": [], "metadata": {} } The run step object Beta Represents a step in execution of a run. id string The identifier of the run step, which can be referenced in API endpoints. object string The object type, which is always `assistant.run.step``. created_at integer The Unix timestamp (in seconds) for when the run step was created. assistant_id string The ID of the assistant associated with the run step. thread_id string The ID of the thread that was run. run_id string The ID of the run that this run step is a part of. type string The type of run step, which can be either message_creation or tool_calls. status string The status of the run, which can be either in_progress, cancelled, failed, completed, or expired. step_details object The details of the run step. Stella Page 214 The details of the run step. Show possible types last_error object or null The last error associated with this run step. Will be null if there are no errors. Show properties expired_at integer or null The Unix timestamp (in seconds) for when the run step expired. A step is considered expired if the parent run is expired. cancelled_at integer or null The Unix timestamp (in seconds) for when the run step was cancelled. failed_at integer or null The Unix timestamp (in seconds) for when the run step failed. completed_at integer or null The Unix timestamp (in seconds) for when the run step completed. metadata map Set of 16 key-value pairs that can be attached to an object. This can be useful for storing additional information about the object in a structured format. Keys can be a maximum of 64 characters long and values can be a maxium of 512 characters long. The run step object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 { "id": "step_QyjyrsVsysd7F4K894BZHG97", "object": "thread.run.step", "created_at": 1699063291, "run_id": "run_UWvV94U0FQYiT2rlbBrdEVmC", "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id": "thread_BDDwIqM4KgHibXX3mqmN3Lgs", "type": "message_creation", "status": "completed", "cancelled_at": null, "completed_at": 1699063291, "expired_at": null, "failed_at": null, "last_error": null, "step_details": { "type": "message_creation", "message_creation": { "message_id": "msg_6YmiCRmMbbE6FALYNePPHqwm" Stella Page 215 "message_id": "msg_6YmiCRmMbbE6FALYNePPHqwm" } } } Retrieve run step Beta GET https://api.openai.com/v1/threads/{thread_id}/runs/{run_id}/steps/{step_id} Retrieves a run step. Path parameters thread_id string Required The ID of the thread to which the run and run step belongs. run_id string Required The ID of the run to which the run step belongs. step_id string Required The ID of the run step to retrieve. Returns The run step object matching the specified ID. Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_BDDwIqM4KgHibXX3mqmN3Lgs/runs/run_UWvV94U0FQYiT2rlbBrdEVmC/steps/step_QyjyrsVsysd7F4K894BZHG97 \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 { "id": "step_QyjyrsVsysd7F4K894BZHG97", Stella Page 216 "id": "step_QyjyrsVsysd7F4K894BZHG97", "object": "thread.run.step", "created_at": 1699063291, "run_id": "run_UWvV94U0FQYiT2rlbBrdEVmC", "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id": "thread_BDDwIqM4KgHibXX3mqmN3Lgs", "type": "message_creation", "status": "completed", "cancelled_at": null, "completed_at": 1699063291, "expired_at": null, "failed_at": null, "last_error": null, "step_details": { "type": "message_creation", "message_creation": { "message_id": "msg_6YmiCRmMbbE6FALYNePPHqwm" } } } List run steps Beta GET https://api.openai.com/v1/threads/{thread_id}/runs/{run_id}/steps Returns a list of run steps belonging to a run. Path parameters thread_id string Required The ID of the thread the run and run steps belong to. run_id string Required The ID of the run the run steps belong to. Query parameters limit integer Optional Defaults to 20 A limit on the number of objects to be returned. Limit can range between 1 and 100, and the default is 20. order string Optional Defaults to desc Sort order by the created_at timestamp of the objects. asc for ascending order and desc for descending order. after string Optional A cursor for use in pagination. after is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include after=obj_foo in order to fetch the next page of the list. before string Optional A cursor for use in pagination. before is an object ID that defines your place in the list. For instance, if you make a list request and receive 100 objects, ending with obj_foo, your subsequent call can include before=obj_foo in order to fetch the previous page of the list. Returns A list of run step objects. Stella Page 217 Example request curl Copy■ 1 2 3 4 curl https://api.openai.com/v1/threads/thread_BDDwIqM4KgHibXX3mqmN3Lgs/runs/run_UWvV94U0FQYiT2rlb BrdEVmC/steps \ -H 'Authorization: Bearer $OPENAI_API_KEY' \ -H 'Content-Type: application/json' \ -H 'OpenAI-Beta: assistants=v1' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 { "object": "list", "data": [{ "id": "step_QyjyrsVsysd7F4K894BZHG97", "object": "thread.run.step", "created_at": 1699063291, "run_id": "run_UWvV94U0FQYiT2rlbBrdEVmC", "assistant_id": "asst_nGl00s4xa9zmVY6Fvuvz9wwQ", "thread_id": "thread_BDDwIqM4KgHibXX3mqmN3Lgs", "type": "message_creation", "status": "completed", "cancelled_at": null, "completed_at": 1699063291, "expired_at": null, "failed_at": null, "last_error": null, "step_details": { "type": "message_creation", "message_creation": { "message_id": "msg_6YmiCRmMbbE6FALYNePPHqwm" Stella Page 218 "message_id": "msg_6YmiCRmMbbE6FALYNePPHqwm" } } }], "first_id": "step_QyjyrsVsysd7F4K894BZHG97", "last_id": "step_QyjyrsVsysd7F4K894BZHG97", "has_more": false } Fine-tunes Deprecated Manage legacy fine-tuning jobs to tailor a model to your specific training data. We recommend transitioning to the updating fine-tuning API The fine-tune object Deprecated The FineTune object represents a legacy fine-tune job that has been created through the API. id string The object identifier, which can be referenced in the API endpoints. created_at integer The Unix timestamp (in seconds) for when the fine-tuning job was created. events array The list of events that have been observed in the lifecycle of the FineTune job. Show properties fine_tuned_model string or null The name of the fine-tuned model that is being created. hyperparams object The hyperparameters used for the fine-tuning job. See the fine-tuning guide for more details. Show properties model string The base model that is being fine-tuned. object string The object type, which is always "fine-tune". organization_id string The organization that owns the fine-tuning job. result_files array The compiled results files for the fine-tuning job. status string The current status of the fine-tuning job, which can be either created, running, succeeded, failed, or cancelled. training_files array The list of files used for training. updated_at integer The Unix timestamp (in seconds) for when the fine-tuning job was last updated. validation_files array The list of files used for validation. The fine-tune object Stella Page 219 The fine-tune object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 Stella Page 220 64 65 66 67 68 69 { "id": "ft-AF1WoRqd3aJAHsqc9NY7iL8F", "object": "fine-tune", "model": "curie", "created_at": 1614807352, "events": [{ "object": "fine-tune-event", "created_at": 1614807352, "level": "info", "message": "Job enqueued. Waiting for jobs ahead to complete. Queue number: 0." }, { "object": "fine-tune-event", "created_at": 1614807356, "level": "info", "message": "Job started." }, { "object": "fine-tune-event", "created_at": 1614807861, "level": "info", "message": "Uploaded snapshot: curie:ft-acmeco-2021-03-03-21-44-20." }, { "object": "fine-tune-event", "created_at": 1614807864, "level": "info", "message": "Uploaded result files: file-abc123." }, { "object": "fine-tune-event", "created_at": 1614807864, "level": "info", "message": "Job succeeded." }], "fine_tuned_model": "curie:ft-acmeco-2021-03-03-21-44-20", "hyperparams": { "batch_size": 4, "learning_rate_multiplier": 0.1, "n_epochs": 4, "prompt_loss_weight": 0.1, }, "organization_id": "org-123", "result_files": [{ "id": "file-abc123", "object": "file", "bytes": 81509, "created_at": 1614807863, "filename": "compiled_results.csv", "purpose": "fine-tune-results" }], "status": "succeeded", "validation_files": [], "training_files": [{ "id": "file-abc123", "object": "file", Stella Page 221 "object": "file", "bytes": 1547276, "created_at": 1610062281, "filename": "my-data-train.jsonl", "purpose": "fine-tune" }], "updated_at": 1614807865, } Create fine-tune Deprecated POST https://api.openai.com/v1/fine-tunes Creates a job that fine-tunes a specified model from a given dataset. Response includes details of the enqueued job including job status and the name of the fine-tuned models once complete. Learn more about fine-tuning Request body training_file string Required The ID of an uploaded file that contains training data. See upload file for how to upload a file. Your dataset must be formatted as a JSONL file, where each training example is a JSON object with the keys "prompt" and "completion". Additionally, you must upload your file with the purpose fine- tune. See the fine-tuning guide for more details. batch_size integer or null Optional Defaults to null The batch size to use for training. The batch size is the number of training examples used to train a single forward and backward pass. By default, the batch size will be dynamically configured to be ~0.2% of the number of examples in the training set, capped at 256 - in general, we've found that larger batch sizes tend to work better for larger datasets. classification_betas array or null Optional Defaults to null If this is provided, we calculate F-beta scores at the specified beta values. The F-beta score is a generalization of F-1 score. This is only used for binary classification. With a beta of 1 (i.e. the F-1 score), precision and recall are given the same weight. A larger beta score puts more weight on recall and less on precision. A smaller beta score puts more weight on precision and less on recall. classification_n_classes integer or null Optional Defaults to null The number of classes in a classification task. This parameter is required for multiclass classification. classification_positive_class string or null Optional Defaults to null The positive class in binary classification. This parameter is needed to generate precision, recall, and F1 metrics when doing binary classification. compute_classification_metrics Stella Page 222 compute_classification_metrics boolean or null Optional Defaults to false If set, we calculate classification-specific metrics such as accuracy and F-1 score using the validation set at the end of every epoch. These metrics can be viewed in the results file. In order to compute classification metrics, you must provide a validation_file. Additionally, you must specify classification_n_classes for multiclass classification or classification_positive_class for binary classification. hyperparameters object Optional The hyperparameters used for the fine-tuning job. Show properties learning_rate_multiplier number or null Optional Defaults to null The learning rate multiplier to use for training. The fine-tuning learning rate is the original learning rate used for pretraining multiplied by this value. By default, the learning rate multiplier is the 0.05, 0.1, or 0.2 depending on final batch_size (larger learning rates tend to perform better with larger batch sizes). We recommend experimenting with values in the range 0.02 to 0.2 to see what produces the best results. model string Optional Defaults to curie The name of the base model to fine-tune. You can select one of "ada", "babbage", "curie", "davinci", or a fine-tuned model created after 2022-04-21 and before 2023-08-22. To learn more about these models, see the Models documentation. prompt_loss_weight number or null Optional Defaults to 0.01 The weight to use for loss on the prompt tokens. This controls how much the model tries to learn to generate the prompt (as compared to the completion which always has a weight of 1.0), and can add a stabilizing effect to training when completions are short. If prompts are extremely long (relative to completions), it may make sense to reduce this weight so as to avoid over-prioritizing learning the prompt. suffix string or null Optional Defaults to null A string of up to 40 characters that will be added to your fine-tuned model name. For example, a suffix of "custom-model-name" would produce a model name like ada:ft-your- org:custom-model-name-2022-02-15-04-21-04. validation_file string or null Optional The ID of an uploaded file that contains validation data. If you provide this file, the data is used to generate validation metrics periodically during fine-tuning. These metrics can be viewed in the fine-tuning results file. Your train and validation data should be mutually exclusive. Your dataset must be formatted as a JSONL file, where each validation example is a JSON object with the keys "prompt" and "completion". Additionally, you must upload your file with the purpose fine-tune. See the fine-tuning guide for more details. Returns A fine-tune object. Stella Page 223 A fine-tune object. Example request curl Copy■ 1 2 3 4 5 6 curl https://api.openai.com/v1/fine-tunes \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "training_file": "file-abc123" }' Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 { "id": "ft-AF1WoRqd3aJAHsqc9NY7iL8F", "object": "fine-tune", "model": "curie", "created_at": 1614807352, "events": [{ "object": "fine-tune-event", "created_at": 1614807352, "level": "info", Stella Page 224 "level": "info", "message": "Job enqueued. Waiting for jobs ahead to complete. Queue number: 0." }], "fine_tuned_model": null, "hyperparams": { "batch_size": 4, "learning_rate_multiplier": 0.1, "n_epochs": 4, "prompt_loss_weight": 0.1, }, "organization_id": "org-123", "result_files": [], "status": "pending", "validation_files": [], "training_files": [{ "id": "file-abc123", "object": "file", "bytes": 1547276, "created_at": 1610062281, "filename": "my-data-train.jsonl", "purpose": "fine-tune-results" }], "updated_at": 1614807352, } List fine-tunes Deprecated GET https://api.openai.com/v1/fine-tunes List your organization's fine-tuning jobs Returns A list of fine-tune objects. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/fine-tunes \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 Stella Page 225 20 21 { "object": "list", "data": [{ "id": "ft-AF1WoRqd3aJAHsqc9NY7iL8F", "object": "fine-tune", "model": "curie", "created_at":

1614807352, "fine_tuned_model": null, "hyperparams": { ... }, "organization_id": "org-123", "result_files": [], "status": "pending", "validation_files": [], "training_files": [{ ... }], "updated_at": 1614807352, }, { ... }, { ... }] } Retrieve fine-tune Deprecated GET https://api.openai.com/v1/fine-tunes/{fine_tune_id} Gets info about the fine-tune job. Learn more about fine-tuning Path parameters fine_tune_id string Required The ID of the fine-tune job Returns The fine-tune object with the given ID. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/fine-tunes/ft-AF1WoRqd3aJAHsqc9NY7iL8F \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 Stella Page 226 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 { "id": "ft-AF1WoRqd3aJAHsqc9NY7iL8F", "object": "fine-tune", "model": "curie", "created_at": 1614807352, "events": [{ "object": "fine-tune-event", "created_at": 1614807352, "level": "info", "message": "Job enqueued. Waiting for jobs ahead to complete. Queue number: 0." }, { "object": "fine-tune-event", Stella Page 227 "object": "fine-tune-event", "created_at": 1614807356, "level": "info", "message": "Job started." }, { "object": "fine-tune-event", "created_at": 1614807861, "level": "info", "message": "Uploaded snapshot: curie:ft-acmeco-2021-03-03-21-44-20." }, { "object": "fine-tune-event", "created_at": 1614807864, "level": "info", "message": "Uploaded result files: file-abc123." }, { "object": "fine-tune-event", "created_at": 1614807864, "level": "info", "message": "Job succeeded." }], "fine_tuned_model": "curie:ft-acmeco-2021-03-03-21-44-20", "hyperparams": { "batch_size": 4, "learning_rate_multiplier": 0.1, "n_epochs": 4, "prompt_loss_weight": 0.1, }, "organization_id": "org-123", "result_files": [{ "id": "file-abc123", "object": "file", "bytes": 81509, "created_at": 1614807863, "filename": "compiled_results.csv", "purpose": "fine-tune-results" }], "status": "succeeded", "validation_files": [], "training_files": [{ "id": "file-abc123", "object": "file", "bytes": 1547276, "created_at": 1610062281, "filename": "my-data-train.jsonl", "purpose": "fine-tune" }], "updated_at": 1614807865, } Cancel fine-tune Deprecated POST https://api.openai.com/v1/fine-tunes/{fine_tune_id}/cancel Immediately cancel a fine-tune job. Path parameters fine_tune_id string Required Stella Page 228 Required The ID of the fine-tune job to cancel Returns The cancelled fine-tune object. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/fine-tunes/ft-AF1WoRqd3aJAHsqc9NY7iL8F/cancel \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 { "id": "ft-xhrpBbvVUzYGo8oUO1FY4nI7", "object": "fine-tune", "model": "curie", "created_at": 1614807770, "events": [{ ... }], "fine_tuned_model": null, "hyperparams": { ... }, "organization_id": "org-123", "result_files": [], "status": "cancelled", "validation_files": [], "training_files": [{ "id": "file-abc123", "object": "file", "bytes": 1547276, "created_at": 1610062281, "filename": "my-data-train.jsonl", "purpose": "fine-tune" }], "updated_at": 1614807789, } The fine-tune event object Stella Page 229 The fine-tune event object Deprecated Fine-tune event object created_at integer level string message string object string The fine-tune event object Copy■ 1 2 3 4 5 6 { "object": "fine-tune-event", "created_at": 1677610602, "level": "info", "message": "Created fine-tune job" } List fine-tune events Deprecated GET https://api.openai.com/v1/fine-tunes/{fine_tune_id}/events Get fine-grained status updates for a fine-tune job. Path parameters fine_tune_id string Required The ID of the fine-tune job to get events for. Query parameters stream boolean Optional Defaults to false Whether to stream events for the fine-tune job. If set to true, events will be sent as data-only server- sent events as they become available. The stream will terminate with a data: [DONE] message when the job is finished (succeeded, cancelled, or failed). If set to false, only events generated so far will be returned. Returns A list of fine-tune event objects. Example request curl Copy■ 1 2 curl https://api.openai.com/v1/fine-tunes/ft-AF1WoRqd3aJAHsqc9NY7iL8F/events \ -H "Authorization: Bearer $OPENAI_API_KEY" Response Copy■ 1 2 Stella Page 230 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 { "object": "list", "data": [{ "object": "fine-tune-event", "created_at": 1614807352, "level": "info", "message": "Job enqueued. Waiting for jobs ahead to complete. Queue number: 0." }, { "object": "fine-tune-event", "created_at": 1614807356, "level": "info", "message": "Job started." }, { "object": "fine-tune-event", "created_at": 1614807861, "level": "info", "message": "Uploaded snapshot: curie:ft-acmeco-2021-03-03-21-44-20." }, { "object": "fine-tune-event", "created_at": 1614807864, "level": "info", "message": "Uploaded result files: file-abc123" }, { "object": "fine-tune-event", "created_at": 1614807864, "level": "info", "message": "Job succeeded." } Stella Page 231 }] } Edits Deprecated Given a prompt and an instruction, the model will return an edited version of the prompt. The edit object Deprecated choices array A list of edit choices. Can be more than one if n is greater than 1. Show properties object string The object type, which is always edit. created integer The Unix timestamp (in seconds) of when the edit was created. usage object Usage statistics for the completion request. Show properties The edit object Copy■ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 { "object": "edit", "created": 1589478378, "choices": [{ "text": "What day of the week is it?", "index": 0, }], "usage": { "prompt_tokens": 25, "completion_tokens": 32, "total_tokens": 57 } } Create edit Deprecated POST https://api.openai.com/v1/edits Creates a new edit for the provided input, instruction, and parameters. Request body instruction string Stella Page 232 string Required The instruction that tells the model how to edit the prompt. model string Required ID of the model to use. You can use the text-davinci-edit-001 or code-davinci-edit-001 model with this endpoint. input string or null Optional Defaults to '' The input text to use as a starting point for the edit. n integer or null Optional Defaults to 1 How many edits to generate for the input and instruction. temperature number or null Optional Defaults to 1 What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. We generally recommend altering this or top_p but not both. top_p number or null Optional Defaults to 1 An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered. We generally recommend altering this or temperature but not both. Returns Returns an edit object. Example request text-davinci-edit-001 curl Copy■ 1 2 3 4 5 6 7 8 curl https://api.openai.com/v1/edits \ -H "Content-Type: application/json" \ -H "Authorization: Bearer $OPENAI_API_KEY" \ -d '{ "model": "text-davinci-edit-001", "input": "What day of the wek is it?", "instruction": "Fix the spelling mistakes" }' Response Copy■ 1 2 3 4 Stella Page 233 4 5 6 7 8 9 10 11 12 13 14 15 { "object": "edit", "created": 1589478378, "choices": [{ "text": "What day of the week is it?", "index": 0, }], "usage": { "prompt_tokens": 25, "completion_tokens": 32, "total_tokens": 57 } } From Stella Page 234

Fine-tuning

This section would contain the formatted fine-tuning information from the API documentation.