



Chris Stanley

December 21, 2022 · 8 min read



Prompt engineering davinci-003 on our own docs for automated support (Part I)



Chris Stanley 4:07 PM

@tech-support - how can patterns help me?



tech-support APP 4:07 PM

Answer: Patterns can help you uncover valuable insights from your data, allowing you to make informed decisions that can help you achieve your goals. Our platform provides you with the tools and resources to analyze, visualize, and explore your data, allowing you to gain a deeper understanding of trends and patterns in your data. We also offer a range of professional services to help you get the most out of your data, from consulting and data science to custom development and training.

One problem we're working on at Patterns is how to scale our technical support. We have technical documentation, Slack channels, emails, and support tickets that all provide a way for us to interface with our customers. Like many folks, we've been playing around

with the power and potential of new Large Language Models like ChatGPT, so we decided to see if we could help tackle our support problem with an LLM support bot.

We came in with somewhat low expectations -- we know these models are prone to common failure modes you'd expect from a next-token optimizer -- so we were shocked when we saw the end result. Read on to learn how we did it and our experience building with LLMs. By the end of the post, we'll have built a full app that you can clone and customize to have your own version of [our bot](#) if you'd like.

Game plan

Our plan was to build an automated slack bot that would respond to tech support questions with knowledge from our technical documentation, community slack channels, and previous support tickets.

Normally with LLMs you put all the necessary context in the prompt. We had far too much content to fit in the 2048 token limit for the normal prompt though. Luckily, many LLM providers like OpenAI provide a "fine-tuning"

api where you can submit labeled example completions to fine-tune your own version of their LLM.

This fine-tuning workflow seemed like a good fit for our problem -- it could take our rich text corpus, fine-tune a model, and then provide completions via a Slack bot.

To serve this experience as a [Patterns](#) app end-to-end we need to:

1. Generate training data from our text corpus
2. Fine-tune our model (OpenAI's GPT-3 davinci-003 engine)
3. Serve the fine-tuned model as a Slack bot

Let's build it!

Prerequisites

- OpenAI account, API key
- A text corpus to train your bot, e.g. technical documentation or something similar

Generating training data

Immediately we ran into a problem -- to fine-tune an OpenAI model requires a specific format of prompt-completion pairs:

```
{"prompt": "<prompt text>", "completion": "<ideal  
generated text>"}  
{"prompt": "<prompt text>", "completion": "<ideal  
generated text>"}  
{"prompt": "<prompt text>", "completion": "<ideal  
generated text>"}
```

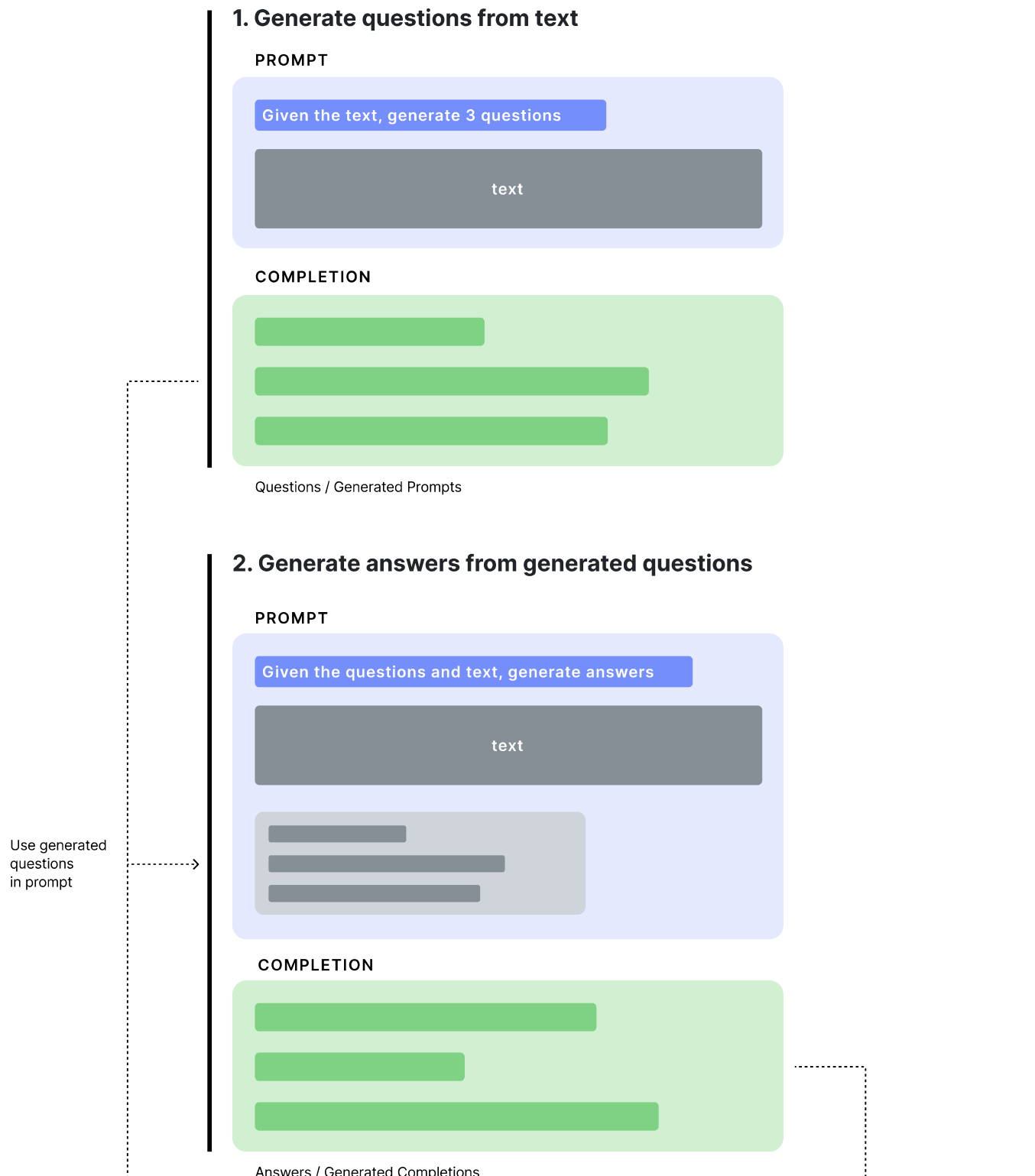
Our free-form text corpus wouldn't work out of the box. We could manually generate the training data, but that's a lot of work. So we remembered the **Golden Rule of LLMs: If it at first you don't succeed, send it through again with a new prompt.**

So instead of manually labeling we come up with a series of prompts to automatically generate our labeled pairs from our corpus:

1. Feed GPT a chunk of our corpus and ask it to generate three relevant questions for the chunk
2. Feed those questions back in with the chunk and ask GPT to specifically answer the generated

question using the chunk

3. Take this output and pair with the original generated question for a labeled training example to feed to the fine-tuner



Use generated questions as prompts in training data

3. Prepare training data for fine-tuning

TRAINING DATA

```
{ "prompt": "<questions>", "completion": "<answers>" }  
{ "prompt": "<questions>", "completion": "<answers>" }  
{ "prompt": "<questions>", "completion": "<answers>" }  
...
```

Use generated answers as questions in training data

Import training data into Patterns

There are many ways to import our documentation into Patterns. To keep it simple for our example, we loaded ours into an [Airtable base](#) that we imported via the standard Airtable component in Patterns. An example of the technical documentation text contained:

Webhooks and Streaming

Patterns has native support for stream processing. A common automation is to have a webhook ingesting real-time events from some external system, processing these events with a series of python nodes, and then sending a response, either to a communication channel like Slack or Discord, or to trigger an action in another external system.

Generate questions to use as prompts from our docs

With our text data in Patterns, we wrote a prompt to tease out three questions that would act as our prompts in training data. From the marketplace we cloned the OpenAI Completions component, added the following prompt, and ran it to generate completions (an example is shown in green below).

```
You are reviewing the documentation for a data
company called Patterns. Write
three questions based on the documentation below.
```

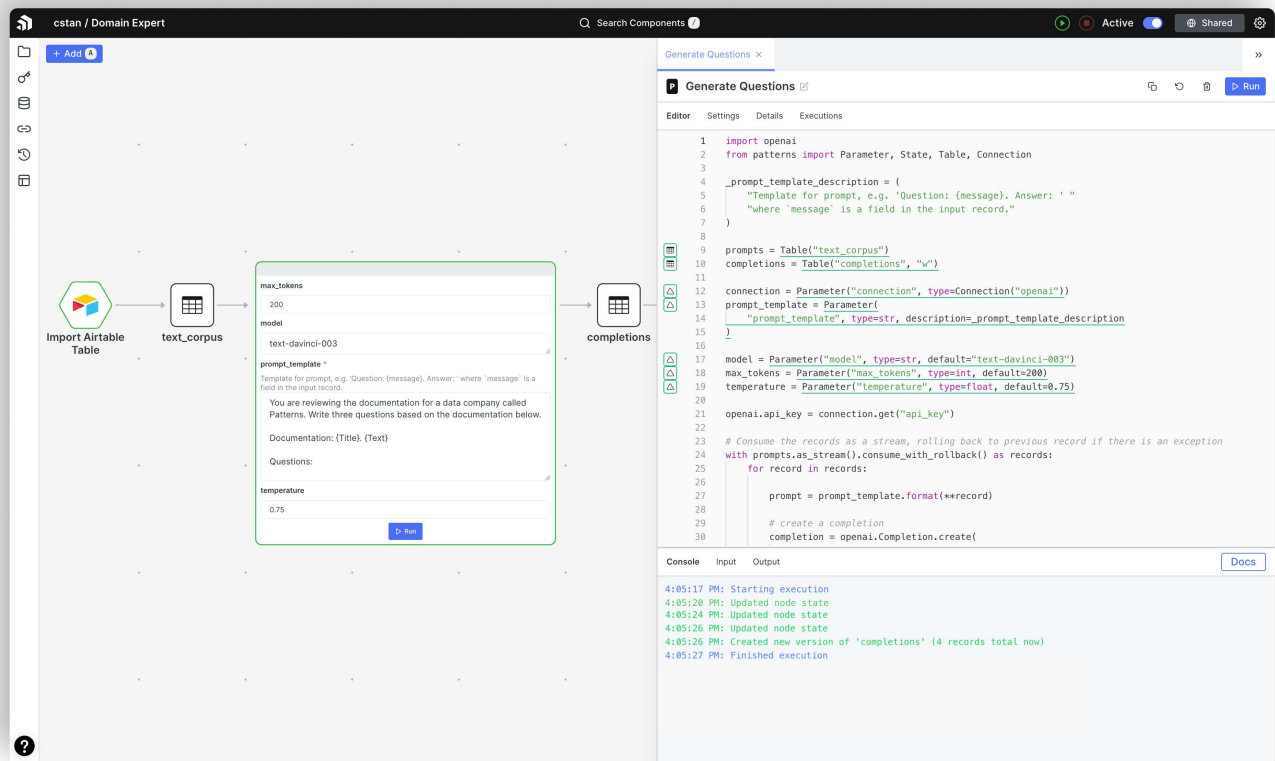
```
Documentation:
```

```
Patterns has native support for stream processing.
A common automation is to have
a webhook ingesting real-time events from some
external system, processing these
events with a series of python nodes, and then
sending a response, either to a
communication channel like Slack or Discord, or to
trigger an action in another
external system.
```

```
Questions:
```

1. What types of real-time events are supported by Patterns webhooks?
2. How are python nodes used to process events?
3. How are responses sent to external systems?

This is what it looks like in the UI.



Generate answers from the questions in the prior step

With the three generated questions / prompts that are relevant to our task, we next generated answers for these using the documentation text. Again, we cloned

the OpenAI Completions component and edited the prompt:

You are a smart and witty technical support agent for a data company called Patterns.
Please generate answers to the following questions given the documentation.

Documentation:

Patterns has native support for stream processing. A common automation is to have a webhook ingesting real-time events from some external system, processing these events with a series of python nodes, and then sending a response, either to a communication channel like Slack or Discord, or to trigger an action in another external system.

Questions:

1. What types of real-time events are supported by Patterns webhooks?
2. How are python nodes used to process events?
3. How are responses sent to external systems?

Answers:

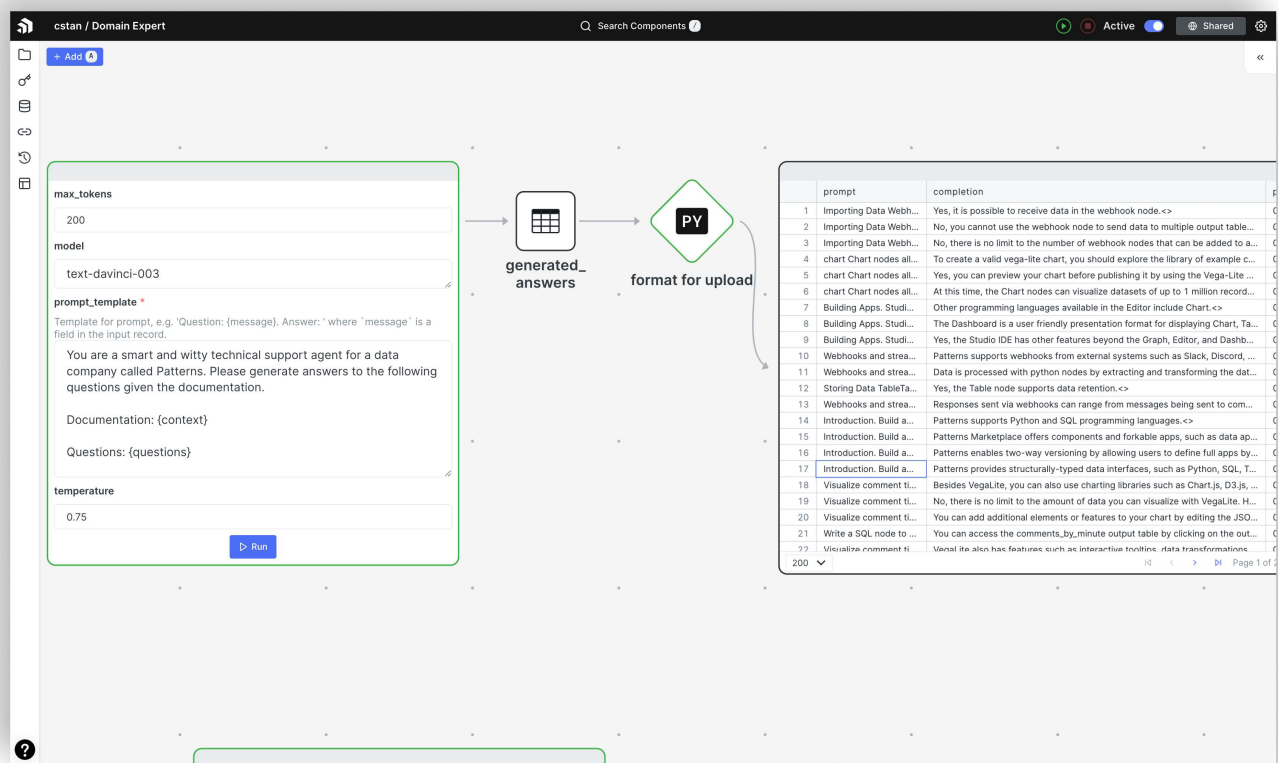
1. Patterns webhooks support a wide range of real-time events, including webhooks for Slack, Discord,

and other external systems.

2. Python nodes are used to process the real-time events received via webhooks and transform them into the desired output format.

3. Responses can be sent to external systems through a communication channel like Slack or Discord, or by triggering an action in the external system.

Next we prepped the data and formatted it for fine-tuning. There are many tricks here and OpenAI has a [CLI data preparation](#) tool to ensure the structure of your data prior to upload.



Upload training data to OpenAI, and start the fine-tuning job

This resulted in prompt/completion pairs that we could structure and upload to OpenAI for fine-tuning. This involved first uploading our completions, which took a few seconds, and then kicking off the fine-tuning job on OpenAI.

Depending on the size of training data, and OpenAI's resource availability, fine-tuning can take anywhere from a few minutes to a few hours, for that reason we kept a state object in our fine-tuning script and check for updates until it's finished.

When the fine-tuning job completes successfully, we write out the model to another table named

`finetuned_models`.

Configure fine-tuned model as a Slack Bot

With the fine-tuned model ready to go on OpenAI's servers, we could tackle the next step -- building a

Slackbot to receive and respond to customer's questions.

To configure a Slack bot in Patterns that uses GPT-3 we set up the following flow:

1. A webhook configured to receive messages from a Slack channel
2. A Python node to detect when the bot is mentioned
3. An OpenAI node, parameterized with the `model_name` from the fine-tuning process, that provides a completion to any prompt
4. A Python node that serves completion back to the Slack channel

To do this, we replicated [Engineering Advice GPT-3 Slack Bot](#) into the same App and followed the configuration in that link.

Learnings and next steps

Overall, we were impressed with the bot's ability to generate answers to relevant questions that it's trained on. However, a lot of the time the bot just makes stuff up; this occurs for both prompts that do and don't have

relevant samples in the training data. Making stuff up and being confidently wrong are well known side-effects of LLMs and there are many techniques to change this behavior.



(totally made up URL, not even our domain)

In Part II of this post (coming soon), we will explore several avenues for improvement, including expanding the corpus via automated ingestion, using [embeddings](#) to encode documentation semantics to provide more robust links directly to documentation, and labeling some negative examples of when the bot should or should not respond.

If you have any questions about setting up a use case similar to this, please email me at chris@patterns.app or create a Slack channel from your dashboard and message us there.

You can check out all the code for this bot [here](#). Clone and customize it with a free Patterns account. To

interact with the bot directly, after signing up create a shared Slack channel with our team.