GPT-4o Machine Learning Fine-Tuning: training data formatting

Overview

I made a fine-tuned machine learning model to accurately answer customer live chats as well as a trained human with up-to-date company policy and product information that was performant enough to converse with real customers in real time. I did this work over the summer of 2024. I chose to fine-tune GPT-4o because it was the best model that was released at the time while also showing sufficient capability to meet the project requirements. In this section I've described the data processing stage of the project where I turned unstructured data into machine-readable semi-structured data.

The problem

Historical customer-employee live chats were stored in a Gmail account on a mail server within a custom domain the company owned. Each email contained one full live chat conversation within the body of the email with irregular formatting including date and other metadata information surrounding the main live chat conversation. There were 5,000+ live chat conversations to be retrieved.

Reviewing the transcripts, I identified three distinct chat formats, each corresponding to a different date range.

Much of the data was over the Covid period, meaning many chats were obsolete or contained out-of-date company policy. Some chats were very short (only one conversation turn) or were malformed. Employee names and customer names needed to be anonymised.

OpenAI required training data to be structured in a .JSONL file structure.

What was achieved?

- Built a Python data pipeline for GPT-4o fine-tuning, parsing three historical livechat formats into a structured dataset of 5,000+ conversations and 74,000+ customer–staff messages.
- Implemented Google Workspace mailbox access through Google Cloud service-account authentication, domain-wide delegation and read-only Gmail OAuth scopes, with paginated email extraction.
- Developed programmatic quality filters to remove poor training data, including unanswered and short conversations, excluding 318 of 1,828 records in a verified filtering stage, alongside keyword, date-range and placeholder transformations.

- Produced a 765-conversation JSONL training dataset containing 8,774 dialogue messages, with speaker normalisation, conversation reconstruction and custom format-validation utilities.

Technical implementation

This technical implementation will discuss the Google Cloud authentication, Gmail live chat retrieval and parsing, and the first stage of processing to get the three distinct storage formats into one definable block in the same structure. Further technical details will be discussed in other posts for this project.

I used Google's Python API libraries to connect to the company's Google Workspace mailbox. Authentication used a Google Cloud service account with domain-wide delegation, allowing the application to act on behalf of the specified mailbox. I requested the Gmail.readonly OAuth scope because the pipeline only needed to retrieve messages.

The script identified the Gmail label containing the live chat transcripts and retrieved the associated message IDs. I implemented pagination using nextPageToken, requesting up to 500 message IDs per page and continuing until all pages had been retrieved. Each message was then fetched in full, and its base64-encoded body was decoded, handling both plain-text and HTML parts. This was an automation script that did not hit the rate limits of the Gmail API yet ran for 1-1.5 hours. I believe that the network Wi-Fi speed played a large role in slowing down the retrieval of these emails.

A single full customer-employee transcript was held within a single email body; once I had retrieved the transcripts, I wrote three parsers in Python, one for each of the live chat conversation storage formats. I kept local backup copies of the stored emails as I passed them through each stage of preprocessing.

Each format used different combinations of timestamps, speaker labels and separators, including square brackets, vertical bars and lines of dashes. I used these patterns to locate the conversation within each email and separate the message text from surrounding metadata. The parsers also had to recognise automated "System assistant" messages and chat-platform footers, such as "Powered by LiveZilla", so these could be excluded from the conversation. Individual replies could span several lines, meaning the parsing logic needed to distinguish a line break within a reply from the start of a new message. Depending on the format, I identified employees using either conventions in the original speaker labels or a list of known staff names. Each parser produced a common intermediate format, preserving the order of the messages and replacing the original speaker labels with [Customer] or [Employee] tags. This allowed conversations from all three source formats to pass through the same subsequent filtering and training-data formatting stages without training the AI on the names of employees or customers.

BEFORE / EMAIL BODY

Timestamped messages and system events

Department: Sales
Message:

[17-06-2024 13:44:47] [Visitor] Hi, with the deposit for the silver sample, is it refunded when the sample is returned to you? Thanks, Alex

[17-06-2024 14:22:00] [System assistant] Morgan Lee, Assistant Manager has accepted the chat!

[17-06-2024 14:22:24] [Morgan Lee, Assistant Manager] Hello my name is Morgan

[17-06-2024 14:22:47] [Morgan Lee, Assistant Manager] Yes once the sample is returned to us we will refund you

[17-06-2024 14:23:15] [System assistant] Morgan Lee, Assistant Manager has closed the chat!

AFTER / INTERMEDIATE TEXT

Three messages in a common structure

```
[Customer]Hi, with the deposit for the silver
sample, is it refunded when the sample is returned
to you? Thanks, Alex
[Employee]Hello my name is Morgan
[Employee]Yes once the sample is returned to us we
will refund you
```

The two employee replies remain separate at this stage.

Conversation dates are retained in the surrounding file separators, which are outside this excerpt.

Original email

Hello,

agent@company.example (agent@company.example) has closed a chat

Name: Visitor

Email:

Phone:

Department: Sales

Country: United Kingdom

City:

IP: [redacted]

Created: 13:44:47

User left: -

Waited: 37 m. 13 s.

Chat duration: 47 s.

Message:

[17-06-2024 13:44:47] [Visitor] Hi, with the deposit for the silver sample, is it refunded when the sample is returned to you? Thanks, Alex

[17-06-2024 14:22:00] [System assistant] Morgan Lee, Assistant Manager has accepted the chat!

[17-06-2024 14:22:24] [Morgan Lee, Assistant Manager] Hello my name is Morgan

[17-06-2024 14:22:47] [Morgan Lee, Assistant Manager] Yes once the sample is returned to us we will refund you

[17-06-2024 14:23:15] [System assistant] Morgan Lee, Assistant Manager has closed the chat!

Additional data, if any:

URL of page from which user has send request:

https://chat.company.example/[redacted]

Survey:

-

Live Chat Automatic Email

After parsing

[Customer]Hi, with the deposit for the silver sample, is it refunded when the sample is returned to you? Thanks, Alex
[Employee]Hello my name is Morgan
[Employee]Yes once the sample is returned to us we will refund you

----------17-06-2024----------