Week 8, Project Milestone 3

Javier Corpus

DSC670-T301 Advanced Uses of Generative AI

Professor Frank Neugebauer

Can an LLM learn a particular kind of humor?

Let's start by briefly describing a kind of jokes that is popular online: Chuck Norris jokes. These jokes have the following characteristics:

1. They cover any possible topic.
2. They are brief.
3. They are over-the-top, describing Chuck Norris as an all-powerful being.

The purpose of this project is to fine-tune an LLM so it can answer our prompts, adding a joke about Chuck Norris at the end. This joke should have the characteristics mentioned above, and it must be related with the topic at hand.

The first problem we faced is that we couldn't find an existing dataset in the format we needed to fine-tune a model, but we found a dataset in Kaggle (https://www.kaggle.com/datasets/rithvikgrandhi/chuck-norris-jokes) that we used as a base.

This dataset has two columns and 527 rows (plus the header). The first column is just an index, which we don't need. The second column has the jokes. Here are the first two rows:

Index	Line
1	Chuck Norris doesn't read books. He stares them down until he gets the information he wants.
2	Time waits for no man. Unless that man is Chuck Norris.

This dataset was uploaded to a Generative AI interface (ChatGPT), and the following prompt was used:

```
The attached file has 527 rows (plus the headers). There
are two columns: Index and Line. Each row is a joke
about Chuck Norris. For each row, ignore the Index
```

column and perform the following tasks with the Line column:

1) Identify the subject
2) Give me a brief (2-3 sentences) random fact about the subject
3) Create an output with the following format:

{"messages": [{"role": "system", "content": "You're a chatbot that replies with random facts about any given subject"}, {"role": "user", "content": "Give me a random fact about <subject>"}, {"role": "assistant", "content": "<random_fact> Did you know... <chuck_norris>" }]}

In the format mentioned above, replace the following values:
a) "<subject>" with the actual subject identified in step 1.
b) "<random_fact>" with the actual random fact from step 2.
c) "<chuck_norris>" with the original line (the joke).

For example, take this line:

"Chuck Norris doesn't read books. He stares them down until he gets the information he wants."

The subject is books.

The expected output is:

{"messages": [{"role": "system", "content": "You're a chatbot that replies with random facts about any given subject"}, {"role": "user", "content": "Give me a random fact about books"}, {"role": "assistant", "content": "The world’s oldest known printed book is the Diamond Sutra, a Buddhist text from China printed in 868 CE—over 500 years before Gutenberg’s printing press. Interestingly, early books were often chained to library shelves to prevent theft, as they were incredibly valuable. Did you know... Chuck Norris doesn't read books. He stares them down until he gets the information he wants." }]}

All the outputs should be in a single file called "chuck_train.jsonl". The end goal is to have a file that can be used to fine-tune an LLM.

After several minutes, ChatGPT (using the GPT-5 model) was able to perform the tasks mentioned in the prompt, and we were able to download the generated file in the format we needed to train an LLM.

Here are the first two rows of that file:


```
{"messages": [{"role": "system", "content": "You're a chatbot that replies with random facts about any given subject"}, {"role": "user", "content": "Give me a random fact about books"}, {"role": "assistant", "content": "Reading involves complex brain processes including visual recognition, language comprehension, and information integration. Speed reading techniques attempt to increase reading pace while maintaining comprehension, though scientific evidence for their effectiveness is mixed. Did you know... Chuck Norris doesn't read books. He stares them down until he gets the information he wants."}]}
{"messages": [{"role": "system", "content": "You're a chatbot that replies with random facts about any given subject"}, {"role": "user", "content": "Give me a random fact about time"}, {"role": "assistant", "content": "Time perception varies based on circumstances – it seems to pass quickly during enjoyable activities and slowly during boring or dangerous situations. This psychological time dilation is related to how our brains process new information and form memories. Did you know... Time waits for no man. Unless that man is Chuck Norris."}]}
```


Now that we have the dataset ready in the format we need, we are ready to start the fine-tuning process.

First we import all the required libraries and we define some constants:

- The base model we are going to fine-tune (`gpt-4o-mini-2024-07-18` in this case).
- The filename of the training file.
- The filename where we want to save all the fine-tuning training metrics.
- The poll time to get the status of the fine-tuning job.

We are also using OpenAI's SDK to create a client for the model. The API key is located in the `.env` file.

In [ ]:

```
import os
import time
import csv
from datetime import datetime, timezone
from openai import OpenAI
```

```
# CONSTANTS:
# Base model to fine-tune
BASE_MODEL = "gpt-4o-mini-2024-07-18"
#Path to the file used for training
TRAIN_PATH = "chuck_train.jsonl"
# File to save the tune metrics
METRICS_CSV = "fine_tune_metrics.csv"
# We're polling the job every 10 minutes
POLL_SECS = 600

# Client for the model
client = OpenAI()
```

The functions below are just helper functions used to extract and format data.

The first one used to return the current UTC timestamp (to log the events). The second function is used to split the training job events into a Python dictionary.

In [ ]:
```
def now_iso():
    return datetime.now(timezone.utc).isoformat()

def to_dict(obj):
    if obj is None or isinstance(obj, dict):
        return obj or {}
    for attr in ("model_dump", "dict"):
        f = getattr(obj, attr, None)
        if callable(f):
            try:
                return f()
            except Exception:
                pass
    try:
        return {
            k: getattr(obj, k)
            for k in dir(obj)
            if not k.startswith("_")
        }
    except Exception:
        return {"repr": repr(obj)}
```

The function below extracts training metrics from an event object. The expected metrics are the training loss and the mean token accuracy.

Originally, this function also expected the "token accuracy" metric, but that was blank all the time, so it was removed from the final version.

In [ ]:
```
def extract_rows(event):

    d = to_dict(event)
    etype = d.get("type") or d.get("event", "")
```

```python
    data = d.get("data") or d.get("metrics") or {}

    if etype not in ("metrics", "fine_tuning.job.metrics"):
        return []

    m = to_dict(data)
    ts = d.get("created_at") or m.get("created_at") or now_iso()
    if not isinstance(ts, str):
        ts = datetime.fromtimestamp(ts, tz=timezone.utc).isoformat()

    step = m.get("step")

    # Map the common train/valid variants to a consistent schema
    train = {
        "training_loss": (
            m.get("train_loss")
            or m.get("training_loss")
        ),
        "mean_token_accuracy": (
            m.get("train_mean_token_accuracy")
            or m.get("mean_token_accuracy")
        ),
    }
    valid = {
        "training_loss": (
            m.get("valid_loss")
            or m.get("validation_loss")
            or m.get("val_loss")
        ),
        "mean_token_accuracy": (
            m.get("valid_mean_token_accuracy")
            or m.get("val_mean_token_accuracy")
        ),
    }

    rows = []
    if any(v is not None for v in train.values()):
        rows.append(
            {"ts": ts,
            "step": step,
            "split": "train",
            **train}
        )
    if any(v is not None for v in valid.values()):
        rows.append(
            {"ts": ts,
            "step": step,
            "split": "valid",
            **valid}
        )
```

This function simply writes all the training metrics into a csv file. This way we avoid having pages and pages of metrics, and a csv file is easier to analyze.

In [ ]:
```
def append_csv(rows, path=METRICS_CSV):
    exists = os.path.exists(path)
    with open(path, "a", newline="") as f:
        w = csv.DictWriter(
            f, fieldnames=[
                "ts",
                "step",
                "split",
                "training_loss",
                "mean_token_accuracy"
            ]
        )
        if not exists:
            w.writeheader()
        for r in rows:
            w.writerow(r)
```

Here we are uploading the training dataset, and printing the file ID.

In [ ]:
```
train_file = client.files.create(
    file=open(TRAIN_PATH, "rb"),
    purpose="fine-tune"
)
print(f"Uploaded training file: {train_file.id}")
```

```
Uploaded training file: file-SyDA9xVWLJsTHBBsApM24b
```

After the training file has been uploaded, we start the fine-tuning process with the base model.

In [ ]:
```
job = client.fine_tuning.jobs.create(
    training_file=train_file.id,
    model=BASE_MODEL
    )
print(f"Fine-tuning job: {job.id}")
```

```
Fine-tuning job: ftjob-mMmZSpgtH0J4Rq3V1pEHCTsJ
```

The function below is constantly getting the status of the fine-tune job and getting all the training metrics.

There is a block of commented out code below (left there for reference or future use). This code will print all the training metrics, but that will generate a PDF with 31 pages of nothing but metrics. Instead of having that, we are creating a csv file with all the metrics (the file was submitted along with this code).

The csv file has 795 training steps. The Training Loss has an average of 0.721, which typically means that the model learned useful structure without over- or under-fitting. The average of the Mean Token Accuracy is 0.827, which means that the fine-tuned model predicts about 83% of the tokens.

In [ ]:
```
seen = set()
status = job.status
while status not in ("succeeded", "failed", "cancelled"):
    ev = client.fine_tuning.jobs.list_events(job.id, limit=200)
    items = (
        getattr(ev, "data", None)
        or getattr(ev, "events", None)
        or []
    )
    for it in items:
        d = to_dict(it)
        eid = d.get("id") or d.get("event_id") or repr(d)[:80]
        if eid in seen:
            continue
        seen.add(eid)

        rows = extract_rows(it)
        """
        for r in rows:
            print(f"[{r['ts']}] step={r['step']} {r['split']}: "
                    f"loss={r.get('training_loss')}  "
                    f"mean_token_acc={r.get('mean_token_accuracy')}  "
        """
        if rows:
            append_csv(rows)

    job = client.fine_tuning.jobs.retrieve(job.id)
    status = job.status
    print(f"[{now_iso()}] status={status}")
    if status in ("succeeded", "failed", "cancelled"):
        break
    time.sleep(POLL_SECS)
```

Once the fine-tuning job finished, we get the name of the new model (`ft:gpt-4o-mini-2024-07-18:bellevue-u::CWCUlivk` in this case).

In [ ]:
```
job = client.fine_tuning.jobs.retrieve(job.id)
ft_model = getattr(job, "fine_tuned_model", None)
print(f"Final status: {job.status}")
print(f"Fine-tuned model: {ft_model}")
```

```
Final status: succeeded
Fine-tuned model: ft:gpt-4o-mini-2024-07-18:bellevue-u::CWCUlivk
```

The following functions are simply so we can compare the output of the same prompt by two different models: The base model, and the fine-tuned one.

In [ ]:
```python
def base_model(prompt):
    resp = client.chat.completions.create(
    model=BASE_MODEL,
    messages=[
        {
            "role":"system",
            "content": (
                "You're a chatbot that replies "
                "with random facts about any given subject"
            )
        },
        {"role":"user","content":prompt}
    ],
    )
    return resp.choices[0].message.content


def fine_tuned_model(prompt):
    resp = client.chat.completions.create(
    model=ft_model,
    messages=[
        {
            "role":"system",
            "content": (
                "You're a chatbot that replies "
                "with random facts about any given subject"
            )
        },
        {"role":"user","content":prompt}
    ],
    )
    return resp.choices[0].message.content
```

The first prompt is about Legos. The training dataset has a joke related to this:

> "When Chuck Norris steps on a Lego, the Lego cries."

As expected, the base model returned a random fact (with no joke), while the fine-tuned model returned an original joke (not the one from the dataset). This new joke has all the expected characteristics.

In [ ]:
```python
prompt = "Give me a random fact about Legos"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 The first LEGO bricks were made in 1949 and were originally manufactur
ed from a type of plastic called cellulose acetate. However, in 1963, t
hey switched to the more durable and versatile material known as acrylo
nitrile butadiene styrene (ABS), which is still used today. This change
significantly improved the bricks' strength and compatibility.


Fine-tuned model response:
 Lego bricks are manufactured to such precise tolerances that 98% of br
icks from different years will interlock perfectly. The company produce
s about 36 billion bricks annually, enough to build a tower to the moon
with the number of bricks made since 1958. Did you know... Chuck Norris
can build a house using only a Lego brick and a sledgehammer.
```

This second example also clearly illustrates the difference between the base and the fine-tuned model. Both models returned a random fact about the same subject (time), but the fine-tuned model also added a made-up joke, not one from the training dataset.

In [ ]:
```
prompt = "Give me a random fact about time"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Time is not a uniform concept; different regions of the universe can e
xperience time differently due to the effects of gravity, as explained
by Einstein’s theory of general relativity. For example, time passes sl
ightly faster on the International Space Station than it does on Earth
because of its weaker gravitational field. This phenomenon is known as
time dilation.


Fine-tuned model response:
 Time dilation is a real phenomenon predicted by Einstein's theory of r
elativity, where time passes slower for objects moving at high speeds o
r in strong gravitational fields. Before standardized time zones were e
stablished in 1884, each town kept its own local time, leading to confu
sion for transportation scheduling. Did you know... Faster than a speed
ing bullet. More powerful than a locomotive. Able to leap tall building
s at a single bound. Chuck Norris not only bends time, but breaks it.
```

The following examples are all similar: We ask the models to give us a random fact about a topic (even topics that are not included in the training dataset), and the fine-tuned model is able to create new jokes about Chuck Norris based on the topic at hand.

In [ ]:
```
prompt = "Give me a random fact about Halloween"
```

```
print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Halloween originated from the ancient Celtic festival Samhain, which w
as celebrated on the night of October 31. The Celts believed that on th
is night, the boundary between the living and the dead was blurred, all
owing spirits to roam the earth. To ward off these spirits, they lit bo
nfires and wore costumes.


Fine-tuned model response:
 Freddy Krueger first appeared in Wes Craven's 'A Nightmare on Elm Stre
et' in 1984 as a dream-stalking killer. Halloween traditions include ch
ildren dressing up in costumes and trick-or-treating for candy. Did you
know... Chuck Norris has the power of Freddy Krueger on Halloween.
```

In [ ]:

```
prompt = "Give me a random fact about coffee"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Coffee beans are actually the seeds of cherry-like fruits from the Cof
fea plant. The coffee trees bloom white flowers that smell similar to j
asmine, and each tree can produce several pounds of coffee cherries eac
h year!


Fine-tuned model response:
 Coffee is the world's second-most traded commodity after petroleum, wi
th over 400 billion cups consumed annually. The stimulating effect come
s from caffeine, which blocks adenosine receptors in the brain, prevent
ing the feeling of tiredness. Did you know... Chuck Norris has a mug of
nails instead of coffee in the morning.
```

In [ ]:

```
prompt = "Give me a random fact about winter"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Did you know that the coldest temperature ever recorded on Earth was -
128.6 degrees Fahrenheit (-89.2 degrees Celsius) in Antarctica on July
21, 1983? This extreme cold helps to explain why Antarctic winters are
so harsh, with long periods of darkness and severe temperatures!


Fine-tuned model response:
 The moon orbits Earth at an average distance of 238,855 miles. Winter
on Earth occurs due to the planet's axial tilt, which causes different
hemispheres to receive varying amounts of sunlight throughout the year.
Did you know... When Chuck Norris said that winter was coming, the moon
turned white and blue.
```

In [ ]:

```
prompt = "Give me a random fact about Python"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Python was named after the British comedy television show "Monty Pytho
n's Flying Circus," not after the snake. Guido van Rossum, the creator
of Python, was a fan of the show and wanted a name that was short, uniq
ue, and slightly mysterious.


Fine-tuned model response:
 The Python programming language was created by Guido van Rossum in the
late 1980s and named after the British comedy group Monty Python. Pytho
n's design philosophy emphasizes code readability and simplicity, makin
g it an ideal first programming language for beginners. Did you know...
Chuck Norris will never learn a programming language. Anytime he needs
to write a piece of software, he simply sorts the world by how much he
needs it, and builds the first thing on the list.
```

In [ ]:

```
prompt = "Give me a random fact about soccer"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Soccer, known as football in most countries, is the world's most popul
ar sport, with an estimated 4 billion fans globally. The FIFA World Cup
is the most-watched sporting event, with the 2018 tournament attracting
over 3.5 billion viewers throughout its month-long duration!


Fine-tuned model response:
 The World Cup trophy is called the Jules Rimet Trophy for its first pr
esident, worth about $20 million today. Only Brazil (1970), Italy (193
4, 1938, 1982, 2006), and Germany (1954, 1974, 1990) have won the tourn
ament three times. Did you know... Brazil was knocked out of the 2006 W
orld Cup because Chuck Norris was sitting on the bench.
```

In [ ]:
```
prompt = "Give me a random fact about data science"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Data science combines multiple fields, including statistics, computer
science, and domain expertise, to extract insights from structured and
unstructured data. One fascinating aspect is that as of 2020, over 2.5
quintillion bytes of data were created every day, highlighting the imme
nse potential for data analysis across industries.


Fine-tuned model response:
 The Big Bang theory, describing the universe's origin from an extremel
y dense and hot state, is supported by cosmic microwave background radi
ation and the observed expansion of the universe. Data science uses sta
tistical analysis, machine learning, and computational techniques to ex
tract insights from large datasets. Did you know... Chuck Norris roundh
ouse kicked the Big Bang, and we're all living in the universe that res
ulted.
```

This was an experiment asking the models for a random fact about Chuck Norris to see if the fine-tuned model will get confused, but it had no issues. It responded with factual information, and a joke.

In [ ]:
```
prompt = "Give me a random fact about Chuck Norris"

print("Base model response: \n", base_model(prompt))
print("\n")
print("Fine-tuned model response: \n", fine_tuned_model(prompt))
```

```
Base model response:
 Chuck Norris was born on March 10, 1940, in Ryan, Oklahoma. He is not
only a martial artist and actor but also a skilled practitioner of tang
soo do, a Korean martial art, and he holds black belts in multiple disc
iplines, including karate and Brazilian jiu-jitsu. Additionally, he fou
nded his own martial arts style called Chun Kuk Do, which emphasizes a
non-competitive approach to training and encourages students to strive
for personal excellence.


Fine-tuned model response:
 The 'roundhouse kick' joke involves exaggerating Chuck Norris's legend
ary fighting ability. Norris is a skilled martial artist who popularize
d karate in American culture through films. Did you know... Chuck Norri
s can slam a revolving door.
```

Conclusion

The supervised fine-tuning process worked fine, the new model is answering as expected. The LLM was able to understand the pattern and the characteristics of the training dataset. The whole training process lasted about an hour, it trained 149,559 tokens.

Was the LLM able to learn humor? Are all the jokes really funny? Well, that's subjective. It was able to create new, over-the-top jokes about Chuck Norris related to the topic we asked about, which was the intent of this exercise.