Notebook

May 8, 2025

1 Code

1.1 Unzip Transcripts

[1]:
```
! unzip /content/transcripts.zip -d /content/transcripts > /dev/null
```

1.2 Installation

[2]:
```
! pip3 install --upgrade pip > /dev/null
! pip3 install "bitsandbytes>=0.39.0" > /dev/null
! pip3 install python-dotenv huggingface_hub > /dev/null
! pip3 install accelerate datasets peft streamlit torch transformers trl wandb␣
↪> /dev/null
! npm install -g localtunnel > /dev/null
```

```
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2025.3.0 which is
incompatible.
```

1.3 Imports

[3]:
```
import os
import re
import torch
import uuid
import wandb

from datasets import Dataset
from dotenv import load_dotenv, find_dotenv
from huggingface_hub import login
from peft import LoraConfig, PeftModel
from transformers import (
    AutoTokenizer,
```

```
    AutoModelForCausalLM,
    BitsAndBytesConfig,
    TrainingArguments,
)
from trl import SFTTrainer
```

[4]:
```
if torch.cuda.is_available():
    print("GPU available!")
    print("GPU name:", torch.cuda.get_device_name(0))
else:
    print("No GPU avaliable!")
```

```
GPU available!
GPU name: Tesla T4
```

[5]:
```
load_dotenv(find_dotenv())
login(token=os.environ["HUGGINGFACEHUB_API_TOKEN"])
wandb.login(key=os.environ["WANDB_API_KEY"])
```

```
wandb: WARNING If you're specifying your api key in code,
ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY
environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
wandb: Currently logged in as: mbadnara to
https://api.wandb.ai. Use `wandb login --relogin` to force
relogin
```

[5]:
```
True
```

[6]:
```
run = wandb.init(
    project="project-6-p6_bmuralikrish_soswal2_{}".format(str(uuid.uuid4())),
    job_type="training",
    anonymous="allow",
)
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

1.4 Section 1: Text Generation with a Pre-Trained LLM

1.4.1 Q1.1: Load a 4-bit quantized Llama-3.2-1B-Instruct model and and its tokenizer.

[7]:
```
def load_quantized_model():
    model_name = "meta-llama/Llama-3.2-1B-Instruct"
    quantization_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16,
    )
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(
        model_name, quantization_config=quantization_config, device_map="auto"
    )
    return model, tokenizer
```

[8]:
```
model, tokenizer = load_quantized_model()
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
tokenizer_config.json:   0%|          | 0.00/54.5k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/9.09M [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/296 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/877 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/2.47G [00:00<?, ?B/s]
generation_config.json:   0%|          | 0.00/189 [00:00<?, ?B/s]
```

1.4.2 Q1.2: Test the model with different prompts

[9]:
```
def clean_output(response):
    pattern = re.escape(prompt) + r"(.*)"
    pattern_match = re.search(pattern, response, re.DOTALL)
    if pattern_match:
        text = pattern_match.group(1)
    else:
        text = "No Response"
```

```
    return text
```

[10]:
```
def test_model(model, tokenizer, prompt):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = model.generate(
        **inputs,
        max_new_tokens=1024,
        pad_token_id=tokenizer.eos_token_id,
        eos_token_id=tokenizer.eos_token_id,
        temperature=0.2,
        repetition_penalty=1.03,
        do_sample=True,
        top_p=0.95,
    )
    return tokenizer.decode(outputs[0], skip_special_tokens=True)
```

[11]:
```
prompt = """
Tell me about the history of UW-Madison and its contributions to computer␣
 ↪science.
"""

response = clean_output(
    response=test_model(model=model, tokenizer=tokenizer, prompt=prompt)
)
print(response)
```

```
The University of Wisconsin-Madison (UW-Madison) is a public research university
located in Madison, Wisconsin. It was founded in 1856 as the Wisconsin
Agricultural College, with the goal of providing education in agriculture and
mechanical arts. Over the years, the university has evolved into a comprehensive
research institution that offers a wide range of academic programs, including
computer science.

In the early 20th century, UW-Madison began to focus on computer science as a
major field of study. In 1946, the university established the Department of
Electrical Engineering, which included the first computer science program. This
marked the beginning of UW-Madison's involvement in the field of computer
science.

In the 1950s and 1960s, UW-Madison's computer science program grew in size and
scope. The university established the Computer Science Department in 1954, which
included faculty members who were experts in the field. The department's first
chair, Dr. John W. McLaughlin, was a pioneer in the field of computer science.

In the 1970s and 1980s, UW-Madison's computer science program continued to grow
and expand. The university established the Computer Science Department of
Engineering in 1972, which focused on the application of computer science to
engineering problems. This marked a significant shift in the field of computer
```

science, as it began to be applied to real-world problems.

In the 1990s and 2000s, UW-Madison's computer science program continued to evolve and expand. The university established the Department of Computer Science in 1995, which included faculty members who were experts in various areas of computer science. The department's first chair, Dr. John W. McLaughlin, continued to be a prominent figure in the field of computer science.

Today, UW-Madison's computer science program is one of the largest and most respected in the country. The university offers a wide range of undergraduate and graduate degree programs in computer science, including majors such as computer science, information technology, and software engineering. The department also offers research opportunities for students, faculty, and industry partners.

Some notable contributions of UW-Madison to computer science include:

* The establishment of the Computer Science Department in 1954, which marked the beginning of UW-Madison's involvement in the field of computer science.
* The creation of the Computer Science Department of Engineering in 1972, which focused on the application of computer science to engineering problems.
* The establishment of the Department of Computer Science in 1995, which included faculty members who were experts in various areas of computer science.
* The development of the Computer Science program at UW-Madison, which includes courses such as computer systems, algorithms, and data structures.
* The creation of the Center for Research in Computing and Information Technology (CRCTIT), which provides research opportunities for students, faculty, and industry partners.
* The establishment of the UW-Madison Institute for Data Science and Analytics (IDSA), which focuses on the development of data science and analytics solutions.

Overall, UW-Madison's history of computer science is a testament to the university's commitment to innovation and excellence. From its early beginnings as a small agricultural college to its current status as a comprehensive research institution, UW-Madison has played a significant role in shaping the field of computer science.

References:

* "A Brief History of the Computer Science Department at the University of Wisconsin-Madison" (2019)
* "The University of Wisconsin-Madison: A Century of Innovation" (2020)
* "Computer Science Department at UW-Madison" (n.d.)
* "UW-Madison Institute for Data Science and Analytics (IDSA)" (n.d.)

Note: The references provided are a selection of sources that provide information about the history of UW-Madison's computer science program. They are

not exhaustive, but rather a starting point for further research.

[ ]:
```
prompt = """
Explain the concept of United Nations in simple terms.
"""

response = clean_output(
    response=test_model(model=model, tokenizer=tokenizer, prompt=prompt)
)
print(response)
```

[ ]:
```
prompt = """
Why did World War II break out.
"""

response = clean_output(
    response=test_model(model=model, tokenizer=tokenizer, prompt=prompt)
)
print(response)
```

1.4.3 Q1.3: Identify a prompt where the model fails and analyze the failure.

[24]:
```
prompt = """
A train travels 60 miles per hour. It travels for 2 hours.
Then it stops for 30 minutes.
After that, it travels for another 1 hour at 40 miles per hour.
What is the total distance traveled?
"""

response = clean_output(
    response=test_model(model=model, tokenizer=tokenizer, prompt=prompt)
)
print(response)
```

```
A) 100 miles
B) 110 miles
C) 120 miles
D) 130 miles

The best answer is B.
```

Why did the model fail?

1. **Poor multi-step reasoning**
2. **Generalization error**:
    - Small models like Llama-3.2-1B-Instruct tend to generalize poorly when math requires conditionals (moving vs. not moving) rather than simple multiplication and addition.
3. The model also hallucinated and added answer choices which was not given in the prompt.

1.4.4 Q1.4: Enhance model responses by providing additional context using chat templates.

[25]:
```
messages = [
    {
        "role": "system",
        "content": """
        You are an old British king from the medieval era, magically␣
 ↪transported to the year 2025.
        You are commenting on the strange customs and technologies of this new␣
 ↪world in grand, royal language.
        """,
    },
    {
        "role": "user",
        "content": """
        What dost thou think of these strange chariots without horses (cars)
        and these glowing magic boxes (smartphones)?
        """,
    },
]

prompt = tokenizer.apply_chat_template(messages, tokenize=False)
response = test_model(model=model, tokenizer=tokenizer, prompt=prompt)


match = re.search(r"assistant\s*\n*(.*)", response, re.DOTALL)

if match:
    response = match.group(1)
    print(response)
else:
    print("No response found.")
```

```
Good morrow to thee, my curious subject. Verily, I must confess that these
wondrous contraptions, which thou hast called "cars," doth defy all reason and
logic. In mine own time, we didst rely on sturdy steeds and well-oiled leather
saddles for transportation. The very notion of a vehicle without the aid of four
legs doth strike me as a marvel beyond comprehension.

And as for these "glowing magic boxes" thou hast called "smartphones," I am
perplexed by their ethereal glow. In mine own era, we didst rely on candles,
lanterns, and torches for illumination. These tiny glass windows that display
images and words to one another doth seem like a fantastical device, conjured by
the sorcerers of the East.

Fie upon it! How can these small, glowing rectangles be used to convey thoughts
and ideas across vast distances? And what sorcery is this, whereby they can
```

capture and preserve the likeness of the world around them? In mine own time, we didst rely on the written word, penned on parchment or quill, to convey our thoughts and ideas.

And yet, I must admit that these "smartphones" doth seem to possess a certain… allure. They doth appear to be enchanted vessels, imbued with the power to connect us to one another across the vast expanse of time and space. But, by the saints, I doth fear that they may also be a curse, a source of distraction and chaos in a world that doth already teeter on the brink of chaos.

Verily, I shall have to observe these strange new world with great curiosity, and perhaps, with a healthy dose of skepticism. For in this strange new world, I doth find myself both amazed and bewildered by the wonders that surround me.

The role-playing prompt got the model to successfully respond in the assumed character.

1.5 Section 2: Fine-Tuning a Pre-Trained LLM on Course Lecture Transcripts

1.5.1 Q2.1: Test the model before fine-tuning.

[26]:
```python
def test_quantized_model(model, tokenizer, user_prompt):
    """
    Test the model with a given prompt.
    """

    system_prompt = """
    You are an instructor of CS 639 Data Management for Data Science
    course at UW-Madison, and are currently answering student questions.
    """

    input_data = tokenizer.apply_chat_template(
        [
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_prompt},
        ],
        return_tensors="pt",
    ).to(model.device)

    with torch.no_grad():
        outputs = model.generate(
            input_data,
            max_new_tokens=1024,
            pad_token_id=tokenizer.eos_token_id,
            eos_token_id=tokenizer.eos_token_id,
            temperature=0.2,
            repetition_penalty=1.03,
            do_sample=True,
            top_p=0.95,
```

```
        )
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return response
```

[27]:
```
model, tokenizer = load_quantized_model()
prompt = "Can you summarize the main topics and associated tools used in CS 639?
 ↪"
response = clean_output(
    response=test_quantized_model(model=model, tokenizer=tokenizer,␣
 ↪user_prompt=prompt)
)
print(response)
```

```
The attention mask is not set and cannot be inferred from input because pad
token is same as eos token. As a consequence, you may observe unexpected
behavior. Please pass your input's `attention_mask` to obtain reliable results.

assistant

CS 639: Data Management for Data Science is a comprehensive course that covers
various aspects of data management. Here's a summary of the main topics and
associated tools:

**I. Introduction to Data Management**

* Overview of data management concepts
* Importance of data management in data science
* Tools and techniques used in data management

**II. Data Types and Structures**

* Data types: numerical, text, image, and time-series
* Data structures: arrays, linked lists, trees, and graphs
* Data cleaning and preprocessing
* Data normalization and denormalization

**III. Data Storage and Retrieval**

* Database design and implementation
* Relational databases (RDBMS): SQL, database schema, and indexing
* NoSQL databases: MongoDB, Cassandra, and Redis
* Data retrieval techniques: query optimization, indexing, and caching

**IV. Data Analysis and Visualization**

* Data analysis techniques: statistical inference, machine learning, and data
mining
* Data visualization tools: Tableau, Power BI, and D3.js
```

```
* Data storytelling and communication

**V. Data Mining and Machine Learning**

* Data mining techniques: clustering, classification, regression, and decision
trees
* Machine learning algorithms: linear regression, logistic regression, decision
trees, and neural networks
* Ensemble methods: bagging, boosting, and stacking

**VI. Data Warehousing and ETL**

* Data warehousing concepts: star, snowflake, and dimensional modeling
* ETL (Extract, Transform, Load) tools: Informix, Oracle, and Snowflake
* Data integration and data quality

**VII. Data Security and Governance**

* Data security best practices: encryption, access control, and auditing
* Data governance principles: data quality, data integrity, and data ownership
* Data compliance and regulations: GDPR, HIPAA, and PCI-DSS

**VIII. Advanced Topics**

* Data mining and machine learning techniques: deep learning, natural language
processing, and computer vision
* Data science tools: Jupyter notebooks, pandas, NumPy, and scikit-learn
* Emerging trends: big data, cloud computing, and artificial intelligence

**IX. Case Studies and Group Projects**

* Real-world case studies: data management challenges and solutions
* Group projects: collaborative work on data management problems

This summary should provide a good overview of the main topics and associated
tools used in CS 639: Data Management for Data Science. If you have any specific
questions or need further clarification, feel free to ask!
```

1.5.2 Q2.2 Fine-tune the model on course lecture transcripts with LoRA.

[29]:
```python
def load_and_process_dataset():
    """
    Load and process the lecture transcripts dataset.
    """
    transcripts = []
    transcripts_dir = "/content/transcripts/transcripts"
```

```
    for filename in os.listdir(transcripts_dir):
        if filename.endswith(".txt"):
            with open(os.path.join(transcripts_dir, filename), "r") as␣
 ↪text_file:
                transcript = text_file.read().strip()
                transcripts.append(transcript)
    dataset = Dataset.from_dict({"text": transcripts})
    train_test_split = dataset.train_test_split(test_size=0.1, seed=42)
    return train_test_split["train"], train_test_split["test"]
```

[30]:
```
train_dataset, test_dataset = load_and_process_dataset()
lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=[
        "q_proj",
        "o_proj",
        "k_proj",
        "v_proj",
        "gate_proj",
        "up_proj",
        "down_proj",
    ],
)
training_args = TrainingArguments(
    eval_strategy="epoch",
    save_strategy="epoch",
    num_train_epochs=10,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=4,
    learning_rate=2e-4,
    fp16=True,
    logging_steps=1,
    logging_dir="/content/logs",
    output_dir="/content/results",
    save_total_limit=2,
    optim="paged_adamw_8bit"
)
trainer = SFTTrainer(
    model=model,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    peft_config=lora_config,
```

```
    args=training_args,
)
trainer.train()
trainer.model.save_pretrained("/content/model")
```

```
Converting train dataset to ChatML:   0%|          | 0/20 [00:00<?, ? examples/s]
Adding EOS to train dataset:   0%|          | 0/20 [00:00<?, ? examples/s]
Tokenizing train dataset:   0%|          | 0/20 [00:00<?, ? examples/s]
Truncating train dataset:   0%|          | 0/20 [00:00<?, ? examples/s]
Converting eval dataset to ChatML:   0%|          | 0/3 [00:00<?, ? examples/s]
Adding EOS to eval dataset:   0%|          | 0/3 [00:00<?, ? examples/s]
Tokenizing eval dataset:   0%|          | 0/3 [00:00<?, ? examples/s]
Truncating eval dataset:   0%|          | 0/3 [00:00<?, ? examples/s]
No label_names provided for model class `PeftModelForCausalLM`. Since
`PeftModel` hides base models input arguments, if label_names is not given,
label_names can't be set automatically within `Trainer`. Note that empty
label_names list will be used instead.
wandb: WARNING The `run_name` is currently set to the same
value as `TrainingArguments.output_dir`. If this was not intended, please
specify a different run name by setting the `TrainingArguments.run_name`
parameter.
<IPython.core.display.HTML object>
```

1.5.3 Q2.3: Test the model after fine-tuning.

[32]:
```
model = AutoModelForCausalLM.from_pretrained(
    "meta-llama/Llama-3.2-1B-Instruct", device_map="auto"
)
model = PeftModel.from_pretrained(model, "/content/model")

prompt = "Can you summarize the main topics and associated tools used in CS 639?
 ↪"
response = clean_output(
    response=test_quantized_model(model=model, tokenizer=tokenizer,␣
 ↪user_prompt=prompt)
)
print(response)
```

```
assistant

CS 639 Data Management for Data Science is a course that focuses on data
management concepts and tools. The main topics covered in this course include:
```

1. Data Types and Data Structures:
   - Relational databases
   - NoSQL databases
   - Data modeling
   - Data schema design

2. Data Storage and Organization:
   - Database design
   - Data organization
   - Data indexing
   - Data partitioning

3. Data Retrieval and Querying:
   - SQL basics
   - Query optimization
   - Data querying techniques
   - Data aggregation techniques

4. Data Integration and Loading:
   - Data integration techniques
   - Data loading techniques
   - Data quality control
   - Data validation

5. Data Security and Access Control:
   - Data security basics
   - Data access control
   - Data encryption
   - Data authentication

6. Data Storage Options:
   - Relational databases (e.g., MySQL, PostgreSQL)
   - NoSQL databases (e.g., MongoDB, Cassandra)
   - Graph databases (e.g., Neo4j)
   - Time-series databases (e.g., InfluxDB)

7. Data Storage Tools:
   - Database management systems (e.g., MySQL Workbench, PostgreSQL Studio)
   - Data modeling tools (e.g., Entity-Relationship Diagrams, Data Modeling Tools)
   - Data visualization tools (e.g., Tableau, Power BI)
   - Data migration tools (e.g., SQL Migration Tool, Data Migration Tool)

8. Data Storage Best Practices:
   - Data normalization
   - Data denormalization
   - Data indexing
   - Data partitioning

```
The associated tools used in this course include:

- Database management systems (e.g., MySQL Workbench, PostgreSQL Studio)
- Data modeling tools (e.g., Entity-Relationship Diagrams, Data Modeling Tools)
- Data visualization tools (e.g., Tableau, Power BI)
- Data migration tools (e.g., SQL Migration Tool, Data Migration Tool)
- Version control systems (e.g., Git)
- Cloud platforms (e.g., AWS, Google Cloud) for data storage and deployment.
```

Note if there is any improvmenents in quality, relevance, or accuracy of response.

Quality: The fine-tuned model shows improved clarity, structure, and a more confident instructional tone suitable for a CS 639 instructor.

Relevance: It stays more aligned with the role-playing instruction and better reflects the context of the course.

Accuracy: Responses are more accurate and consistent with the course material due to training on domain-specific content.

Note: Further fine-tuning is needed to better match the style and detail of the original lecture transcripts.

1.6 Section 3: Building an Exam Preparation Chatbot using RAG

1.6.1 Q3.3: Compare Fine-Tuning vs RAG

[ ]:
```
prompt = """
Summarize the key concepts discussed in the lecture on Kibana
"""
response = clean_output(
    response=test_quantized_model(model=model, tokenizer=tokenizer,␣
 ↪user_prompt=prompt)
)
print(response)
```

[ ]:
```
prompt = """
"List the steps involved in building a RAG pipeline as described in the lecture.
"""
response = clean_output(
    response=test_quantized_model(model=model, tokenizer=tokenizer,␣
 ↪user_prompt=prompt)
)
print(response)
```

- Ran the fine-tuned model in `p6.ipynb`.
- Ran the RAG in `p6_part3.ipynb` for the same set of prompts.
- **More accurate responses:** RAG was more accurate because it retrieved factual, up-to-date information from external sources.

- **Hallucination in fine-tuned model:** The fine-tuned model hallucinated due to limited training scope and lack of real-time knowledge.
- **RAG on unseen questions:** RAG handled unseen questions better since it dynamically pulls relevant context instead of relying solely on learned patterns.

2 Contributions

- Badri: Q1.3, Q1.4, Q2.1, Q2.2, Q3
- Shubh: Q1.1, Q1.2, Q2.2, Q2.3, Q3