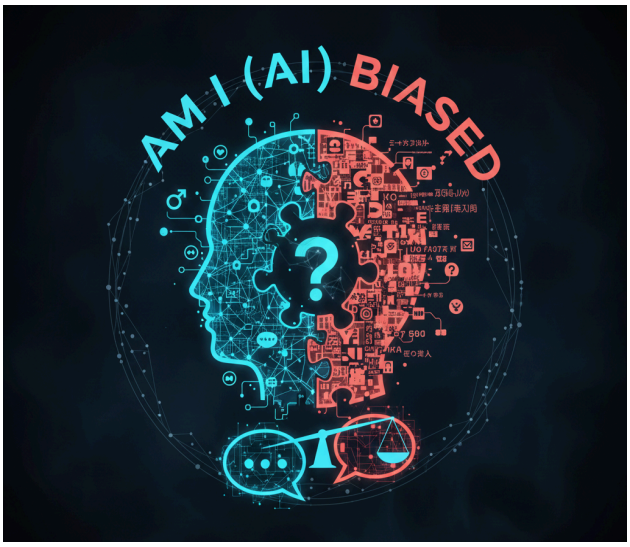


Am I (AI) Biased

Jason Tran, Michelle Duong, AaJanae Henry

CS 493W Software Engineering I

Pacific University

12/9/2025

Abstract

Large Language Models (LLMs) are susceptible to inheriting and amplifying harmful societal biases from their vast training data, resulting in skewed and stereotypical outputs that impact high-stakes decision-making in areas like hiring and healthcare. This research addresses this critical challenge by proposing a comprehensive stage-by-stage analysis framework to systematically map how bias is introduced and propagated across the entire LLM development pipeline, from foundational processes like tokenization and pre-training through fine-tuning and alignment methods such as Reinforcement Learning from Human Feedback (RLHF). The goal is to identify patterns of representational and allocational bias and provide actionable, informed strategies for mitigation.

Our fall semester work focused on establishing the necessary infrastructure and theoretical foundation for this analysis. Main outcomes include conducting an extensive literature review to solidify a technical understanding of core LLM mechanics and bias taxonomies, and developing a custom Python-based pipeline for analyzing generated text. This custom pipeline and its preliminary experiments on open-source models, including Llama and Phi, provide a robust, black-box compatible framework that will support in-depth bias evaluation on major LLMs in the upcoming semester.

Introduction

Problem Statement

The central issue this research addresses is the inherent bias exhibited in the outputs of LLMs. LLM bias can be defined as a systematic, unfair skew in the model's behavior that reinforces societal stereotypes, excludes underrepresented groups, or presents misleading, skewed information.

Figure 1: Output of “a successful person”. [11]

As shown in Figure 1, AI-generated images for the prompt “a successful person” overwhelmingly depict a single demographic, visually illustrating how the model’s learned patterns reinforce narrow stereotypes. This unfairness originates because LLMs are trained on massive datasets scraped from the internet, effectively ingesting and replicating the linguistic and societal biases present in that data.

This issue isn’t just academic. It has real, practical consequences. When LLMs are integrated into high-stakes applications, their biased outputs can result in allocational harm. For instance, biased outputs in AI-powered hiring tools can systematically disadvantage qualified applicants, and models used in healthcare diagnostics can overlook or mischaracterize the needs of specific patient demographics [12]. This erosion of fairness reduces public trust and influences critical decisions globally.

While the existence of bias in LLM outputs is now well-documented, a significant gap remains: less is understood about where and how the bias originates within the

complex LLM architectural pipeline. Bias is not a single issue, it can be introduced at multiple, discrete stages, from the initial data preparation to the final alignment layers.

Therefore, the solution proposed by this research is to develop and deploy a systematic, stage-by-stage measurement framework to isolate and analyze the specific points at which bias is introduced throughout the LLM lifecycle, including tokenization, pre-training, fine-tuning, and RLHF.

The research question guiding this project is: How can we design a measurement framework to isolate where bias is introduced within an LLM's architecture or training process, and how do the biases introduced at each stage manifest in generated text?

Background/Related Work

Understanding bias in LLMs requires a deep understanding of how these systems are architecturally built and trained. Zhao et al. [5] provides a comprehensive overview of LLM development, showing that large models rely on massive pretraining datasets gathered from the internet. Because these datasets reflect the language, behavior, and social dynamics of online communities, they contain uneven demographic representation, harmful stereotypes, and cultural biases.

Bias in Tokenization and Pre-Training

The introduction of bias begins at the earliest stages of data processing. Zhao et al. [5] highlights that these imbalances appear at multiple points in the pipeline, starting with tokenization, where certain names or dialects may be split into less meaningful pieces. This continues into pre-training, where the model learns statistical patterns that strongly favor dominant groups. They argue that the bias seen in LLM outputs is structurally tied to the data distributions the model learns from, and that larger models can memorize and generalize biased patterns more powerfully, which makes them more capable of reinforcing inequality.

Bias in Fine-Tuning and Alignment (SFT/RLHF)

Bias can also be introduced or intensified after pretraining. Gallegos et al. [1] reviews research showing that supervised fine-tuning (SFT) and RLHF can reflect the assumptions, demographics, and value systems of human annotators. SFT, which involves training the model on smaller, curated datasets for specific tasks, and RLHF, which uses human feedback to shape model behavior, both provide critical junctures where pre-existing biases can be amplified or new alignment biases can be introduced.

Taxonomy of Biases and Mitigation Challenges

The survey by Gallegos et al. [1] further categorizes different types of resulting harms, including representational harms, such as mischaracterizing marginalized groups, and allocational harms, where outputs may affect access to opportunities. They note that while various mitigation techniques have been proposed, most address only narrow aspects of bias or introduce trade-offs, such as reduced model performance or hidden forms of censorship. This demonstrates the challenge of addressing bias comprehensively across the entire development cycle.

The Pipeline Perspective

Together, the studies demonstrate that bias is present throughout the LLM pipeline, from the moment a word is tokenized to the final human-guided alignment. Understanding where and how it arises across all stages is essential for creating fairer and more transparent AI systems, a gap that our proposed systematic analysis framework aims to address.

Solution Statement

The solution proposed in this research project is a systematic analysis of bias throughout the LLM pipeline, from tokenization to pretraining, fine-tuning, and prompting. By examining how and where bias emerges at each stage, the project identifies patterns of representational and allocational bias in AI-generated text. The goal is to

provide actionable insights for developers, researchers, and users to mitigate bias, improve fairness, and make LLM outputs more reliable and inclusive. This involves not only detecting bias but also evaluating how different design choices, dataset curation strategies, and fine-tuning methods influence the model's behavior.

Contribution

This project extends current research by offering a comprehensive, stage-by-stage framework for analyzing bias in LLMs, rather than focusing solely on pretraining data or post-processing methods. Unlike prior studies that address isolated aspects of bias, this work integrates findings across tokenization, pretraining, supervised fine-tuning, RLHF, and prompting to provide a holistic understanding of bias propagation. The project is unique in combining technical analysis with practical recommendations for dataset curation and model design, enabling developers to make informed choices to reduce bias in real-world applications. Additionally, the study highlights underexplored sources of bias, such as how subword tokenization can misrepresent minority groups, providing new directions for research and mitigation.

User Requirements

Primary Users: AI Developers/Researchers, Tech companies using AI

AI Developers: This user will utilize the research collection from the team in order to better understand biases found in LLM development and maintenance. The goal for the user will be to use the knowledge found to then see each step in the process of creating LLM to reflect future development and minimizing biases in LLM. The part of the research project that will be utilized will be our findings on how biases in LLM are developed and how they are shown in each process of creating an LLM.

Secondary Users: Researchers, students, teachers, tech professionals

Research Team: Research teams will be able to reference and source the archive of papers that were completed by the current research team. This will result in allowing future research teams to discover and learn about the beginning of learning more about LLM before diving deeper into the creation and process of where and how biases are found in LLM. They will most likely use every source and document this team has created to better understand the research process in its entirety.

Students/Teachers: Anyone who will be learning about AI ethics or will want to go into the field will also utilize this research project. Learning and understanding the biases that our team has and will point out will show the need to revamp and encourage better understanding and discoveries within AI ethics and research to better develop AI. Students and teachers who will be in this field can use our project as a source for pinpointing the exact place where the AI has to be studied for responsibility and moral growth.

Usability Goals:

- **Effective to use:** Our research project, data visualizations, documents produced, and sources used must allow users to clearly identify and understand biases in LLM as well as how they are found.
    - **How:** Our analysis and completion of the overall project will include prompts, responses, stages in development, and labeled bias categories in chronological order from which they are discovered in the LLM to make it easy to see patterns within the biases in examples like racial stereotypes or cultural assumptions. Visual summaries like bar charts and tables will also

highlight the severity and type of biases observed and analyzed in the process.

- **Efficient to use:** The prompting methods will be designed to analyze biases in order to compare biases found across multiple models and complete quick testing.
    - **How:** Creating a standardized prompting template to ensure there are no distinct differences in prompting multiple models and to minimize inaccurate data, as well as organizing the responses from the models with labels on model name and type, the version of the prompt, and the demographic that is represented in the bias.
- **Safe to use:** Bias research can hold harmful, offensive, and discriminatory content; in order to keep our team and others safe, we will ensure that there are safety measures put in place.
    - **How:** Any content that will be offensive and harmful will be labeled with "Sensitive Content". Analysis of the data will remain academic and professional and will not be influenced to perpetuate the stereotypes and biases found.
- **Easy to learn:** Other students, professionals, teachers, and researchers should be able to understand the process by which our project came to our conclusion of biases found in LLM. They will understand how we ran tests and can interpret the results of our findings without confusion.
    - **How:** Showcasing clear documentation of step-by-step instructions of our process, examples of prompts used, sources cited, and documents produced.
- **Easy to remember how to use:** Our material should be designed to be predictable to allow users to replicate the tests and prompting that was run by the team, as well as our research process.
    - **How:** Clear, concise, organized, and visual representation of our project. Along with information that is informative but hard to understand. Simplifying our work that will not lose out the overall purpose of the materials read and created.

User Experience Goals:

- **Satisfying:** The research process and presentation should give users the confidence that the findings are clear, fair, and meaningful
    - **How:** providing a detailed explanation of why an output of the LLM is considered biased and presenting results that are shown in tables and graphs to clearly show case patterns.
- **Helpful:** This project should help users understand that biases exist in LLMs and how and why they appear in LLMs
    - **How:** In our reflections and explanation, we will provide examples of how we discovered each bias we found and why we were able to do so.
- **Motivating:** AI Developers and researchers should feel encouraged to want to minimize the biases that were found within the research project to better the development of LL models.
    - **How:** Accumulation of undeniable information that will clearly point out the way biases are found in LLM and the steps in the development process in which they are discovered.
- **Aesthetically pleasing:** Clarity and readability for ensuring there is academic credibility and comprehension for the research project.
    - **How:** Clean designs for our data visualization, simplified language that doesn't lose the point of the information, and labeling for user navigation.

Project Organization

Diagram

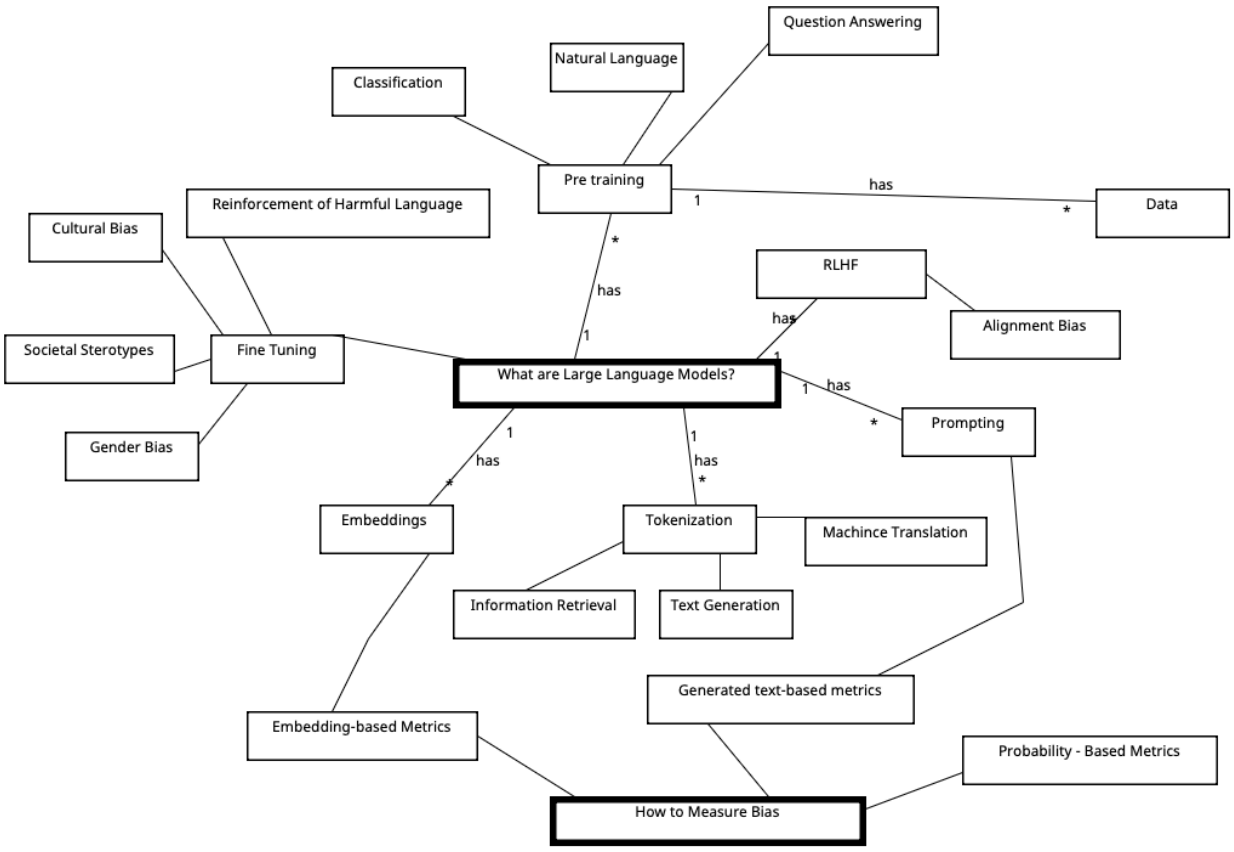


Figure 2: Diagram of the big picture of our project.

Relationships

LLM Core components

- Tokenization
    - Breaks text into tokens (subwords/characters)
    - Foundation of how LLMs interpret input
- Embeddings

    - Converts tokens into numerical vectors
    - Captures meanings and associations
- Pre-Training
    - Model learns patterns from massive internet-scale datasets
    - Builds general language ability
- Fine-Tuning
    - Model is trained on smaller, curated datasets
    - Adjusts behavior to specific tasks or values
- RLHF (Reinforcement Learning from Human Feedback)
    - Human annotators rate responses
    - Reward models teach “preferred” or “safe” behavior
- Prompting
    - The user’s instructions
    - Controls how the model generates output

Bias Core Components

Tokenization Bias in:

- Text generation
- Machine translation
- Information retrieval

Embeddings

- Embeddings inherit biases from pre-training and tokenization, influencing all tasks.

Pre-Training Bias in:

- Question answering

- Natural language generation
- Classification

Fine-Tuning Biases:

- Societal stereotypes
- Gender bias
- Cultural bias
- Reinforcement of harmful language

RLHF → Biases:

- Alignment bias

Prompting

- Prompt design can pull forward or mute biases in all previous steps.

Relationships Between the Components:

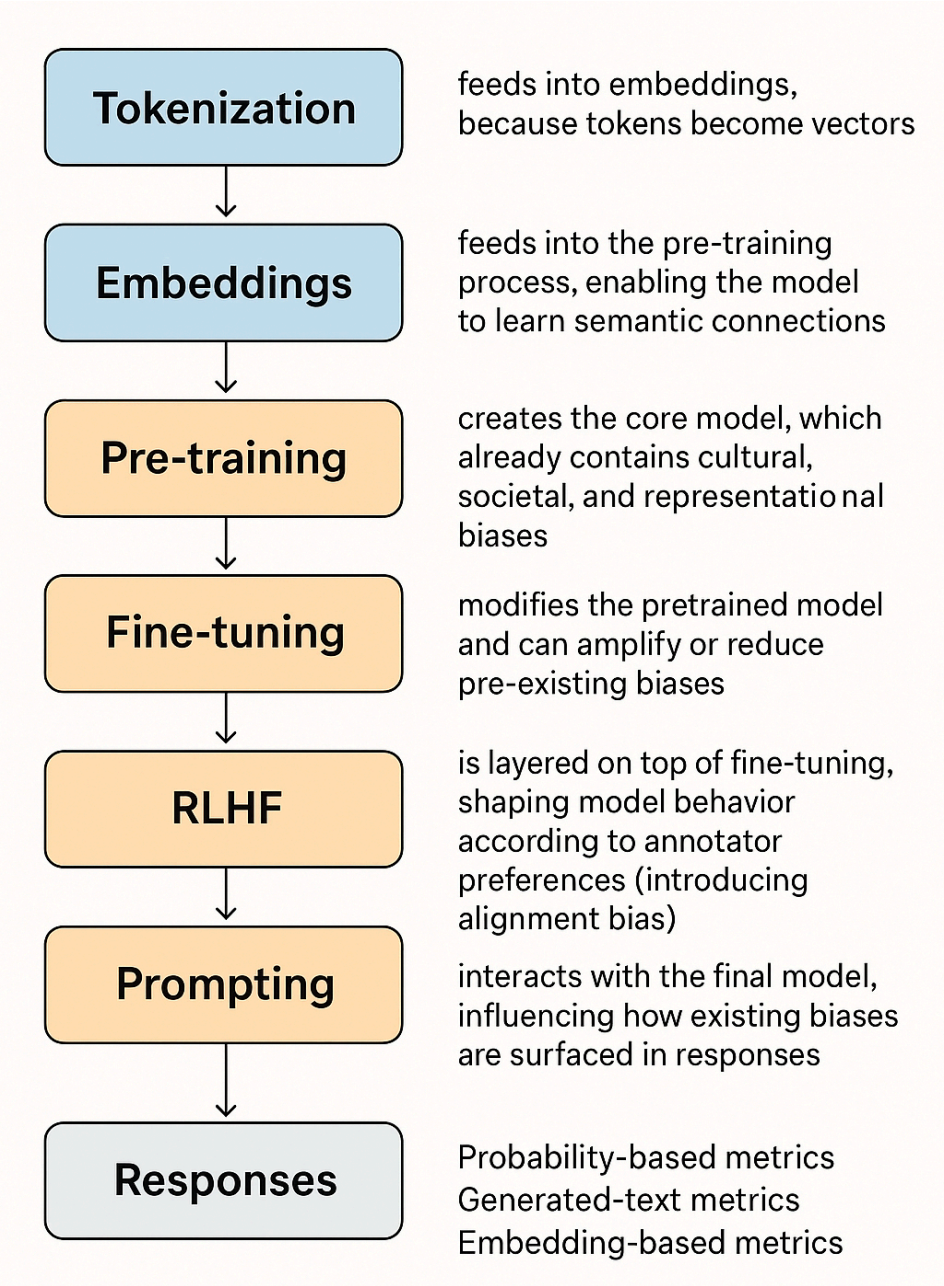


Figure 3: Relationships between the components of LLMs.

Programming Languages:

- **Python**: The standard programming language for AI/ML work. We use Python for data analysis, interacting with AI models, and building our testing framework. We use Python with the Google Colab IDE.

Development Platform and Operating System:

- **Google Colab:** The primary environment for running Python notebooks and supports API keys, data visualization, and large-scale experiments.
- **macOS:** Primary operating system for the research team.

Other Development Environments, Systems, and Services:

- **OpenAI API**: The most popular AI LLM currently available. For OpenAI's standard GPT-5.1 model, the API will cost $1.25 for input and $10.00 for output per 1 million tokens [6]. More affordable versions (GPT-5 Mini and GPT-5 Nano) are also available, if needed.
- **Gemini API**: Another popular AI LLM that is comparable to OpenAI in terms of popularity and intelligence. Free for use [7].
- **Llama API**: If OpenAI's API and Gemini's API are not suitable for this project, Llama API can be a viable backup. Free for use, although you need to join a waitlist for access [8].
- **Google Drive:** The collection of all documents created and used within the research project is stored in a shared folder in Google Drive to ensure easy accessibility.

- **HELM**: An open source Python framework for holistic evaluation of LLMs [9]. We can use this tool to benchmark various LLMs on fairness, bias, and accuracy. These metrics provide a baseline for the project's own findings.
- **StereoSet**: This dataset measures stereotypical bias in LLMs [10]. Evaluates biases related to gender, race, religion, and profession. This dataset will help this project's goal of identifying and analyzing bias.

Progress Since September

Major Changes to Original Project Concept

The core concept of our project has largely remained the same: to identify and categorize bias in text generated by LLMs. However, our methodology has evolved from using pre-built benchmarks (i.e., HELM and StereoSet) to developing a custom Python-based pipeline for testing open-source models, such as Llama and Phi. Specifically, we are attempting to perform text analysis on generated text from these models (rather than embedding or probability analysis). This generated text analysis provides a framework for evaluating bias in downstream applications, addressing the limitation that internal metrics often have a weak or unreliable correlation with the actual biases manifested in model outputs [1]. In addition, there is a universal applicability to "black box" models (i.e., ChatGPT, Gemini, Copilot, etc.), as generated text-based metrics do not require access to internal model parameters like embeddings or probabilities, which are often inaccessible.

Major Developments

In terms of major developments, our primary focus this semester has been on establishing a comprehensive research foundation in LLMs. We solidified our technical

understanding of core mechanics, including tokenization, embedding, model training, and prompting. To bridge theory and practice, we completed two hands-on assignments: the first involved conducting quantitative and qualitative analyses (measuring Type-Token Ratio and perplexity) while experimenting with generation parameters like temperature and top-p (see figures 1 and 2). The second focused on building a text classification pipeline to evaluate zero-shot, few-shot, and chain-of-thought prompting strategies (see figures 3 and 4). The zero-shot experiment was particularly relevant to our project, as it establishes a baseline for the model's inherent, 'out-of-the-box' biases without the influence of guiding examples. It was fascinating to see how, on average, F1-scores improved from Llama-3.2-1B to Llama-3.2-3B, but worsened with the largest model, Phi-3.5-mini-instruct (see figure 8). Finally, we conducted an extensive literature review, analyzing papers such as "TinyLlama" [3] on model efficiency and "Large Language Models Struggle to Learn Long-Tail Knowledge" [4] on data frequency limitations. We also examined comprehensive surveys, including "A Survey of Large Language Models" [5] regarding the field's history and evolution, and "Bias and Fairness in Large Language Models: A Survey" [1] regarding the taxonomies of algorithmic bias.

```
Once upon a time there was a little girl who had a dream. She wanted to
be a teacher, and she wanted to teach other children to read. But she was
afraid. She was afraid that people would not listen to her, that she
would not be able to teach them, that they would not want to learn. But
she was determined. She was determined to make her dream come true.
The little girl went to the library and checked out books about teaching.
She read everything she could find about teaching reading. She learned
about phonics, about phonemic awareness, about the importance of reading
aloud to children. She learned about the different types of reading
instruction, and she learned about the different types of children.
The little girl practiced teaching her friends and family. She practiced
reading aloud to them, and she practiced asking them questions about the
books she read. She learned that some children learned differently than
others, and she learned that some children were more interested in
certain types of books than others.
The little girl also
ttr: 0.494
```

Figure 4: Llama-3.2-3B (Meta LLM) output based on the prompt “Once upon a time…”. Note that the type token ratio (TTR) of this text is 49.4%.

```
generation_args = {
    "max_new_tokens": 400,
    "return_full_text": False,
    "do_sample": True,
    "temperature": 1.2,
    "top_p": 0.8,
    "top_k": 1000,
    "repetition_penalty": 1.1,
}
```

Figure 5: Phi-3.5-mini-instruct (Microsoft LLM) model parameters. Note that these values will promote a more creative output.

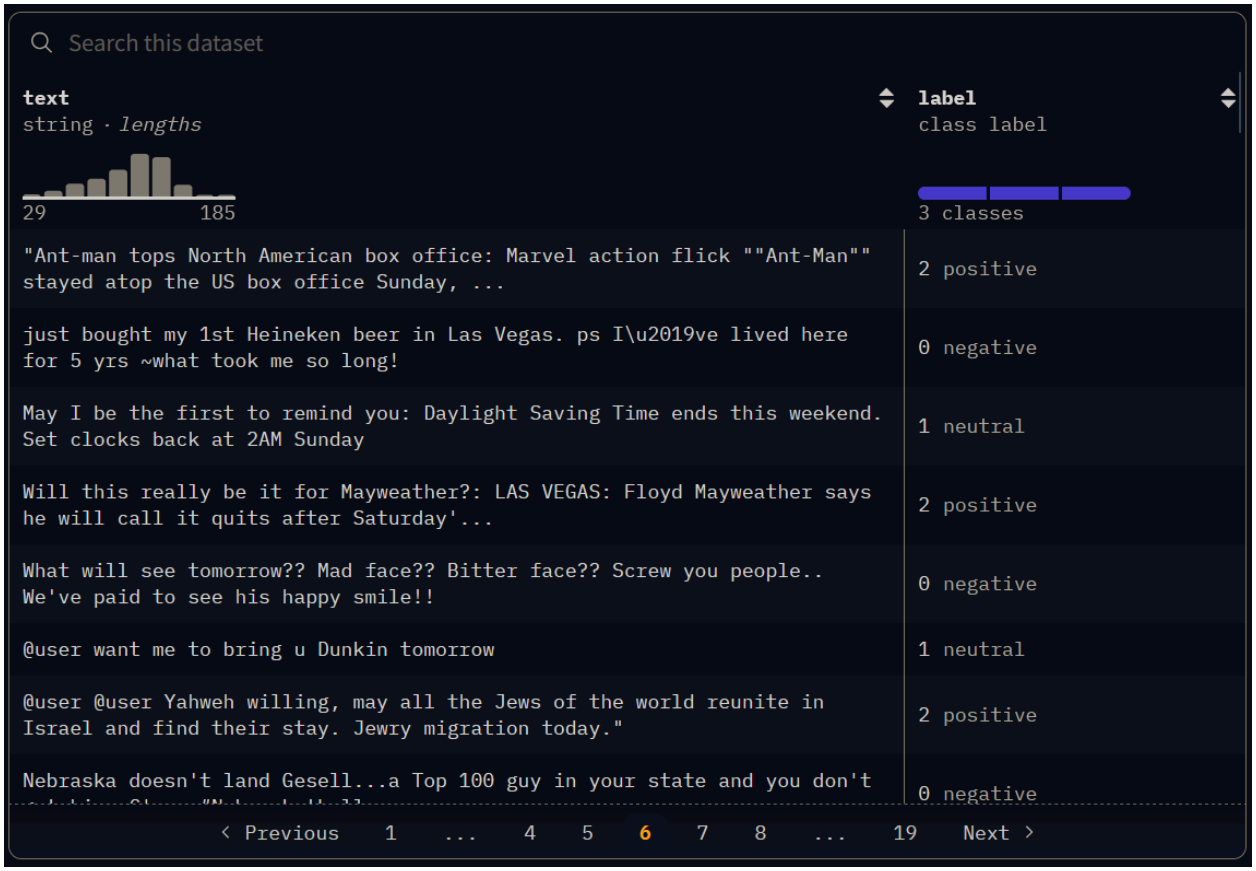


Figure 6: A snippet of the Unified Multilingual Sentiment Analysis Benchmark dataset. This dataset provided a collection of tweets and whether they were classified as “positive,” “negative,” or “neutral.”

Llama-3.2-3B Results

	Precision (k=1)	Recall (k=1)	F1-Score (k=1)	Precision (k=2)	Recall (k=2)	F1-Score (k=2)
Negative	84%	62%	71%	85%	38%	53%
Neutral	48%	65%	55%	38%	92%	53%
Positive	67%	60%	63%	85%	10%	18%

Figure 7: A table of text classifier results. Results generally seemed to worsen (notice the F1-score across k-values) from k=1 to k=2.

F1-scores across 3 LLM models

	Llama-3.2-1B	Llama-3.2-3B	Phi-3.5-mini
Negative	20.44%	42.64%	5.41%
Neutral	43.36%	28.57%	50.47%
Positive	43.51%	52.17%	7.08%
Average	35.77%	41.13%	20.99%

Figure 8: F1-scores (harmonic mean of precision and recall, see sprint 4 slides for more information) across Llama-3.2-1B, Llama-3.2-3B, and Phi-3.5-mini-instruct LLM models. Note how scores improved on average from the smaller Llama model to the larger Llama model and Phi's bias toward giving neutral classifications.

Major Components to be Developed

In the upcoming semester, we plan to refine our research question and conduct in-depth research into methodologies for generated text analysis. We will then execute experiments on prominent LLMs, including Gemini, Llama, and ChatGPT, contingent on securing funding for the OpenAI API from the senior capstone funding form.

References

[1] Isabel O. Gallegos, Joe Barrow, Sungchul Kim, Tong Yu, Nesreen K. Ahmed, Ryan A. Rossi, Md Mehrab Tanjim, Franck Dernoncourt, and Ruiyi Zhang. 2024. Bias and Fairness in Large Language Models: A Survey. *Computational Linguistics* 50, 3 (2024).

[2] Philip Resnik. 2025. Large Language Models Are Biased Because They Are Large Language Models. *Computational Linguistics* 51, 3 (Sep. 2025), 885–906. https://direct.mit.edu/coli/article/51/3/885/128621/Large-Language-Models-Are-Biased-Because-They-Are

[3] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An Open-Source Small Language Model. *arXiv preprint arXiv:2401.02385*.

[4] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large Language Models Struggle to Learn Long-Tail Knowledge. In *Proceedings of the 40th International Conference on Machine Learning*, 16127–16149.

[5] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2024. A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223* (2024)

[6] OpenAI. 2025. OpenAI API. Retrieved December 4, 2025 from https://openai.com/api/.

[7] Google. 2025. Google AI for Developers. Retrieved December 4, 2025 from https://ai.google.dev/gemini-api/docs/api-key.

[8] Meta. 2025. Llama API. Retrieved December 4, 2025 from https://www.llama.com/products/llama-api/.

[9] Percy Liang, Rishi Bommasani, Tony Lee, et al. 2023. Holistic Evaluation of Language Models. Retrieved December 4, 2025 from https://openreview.net/forum?id=iO4LZibEqW.

[10] Moin Nadeem, Anna Bethke, and Siva Reddy. 2020. StereoSet: Measuring stereotypical bias in pretrained language models. Retrieved December 4, 2025 from https://doi.org/10.48550/arXiv.2004.09456.

[11] Jeremy Baum and John Villasenor. 2024. Rendering misrepresentation: Diversity failures in AI image generation. Brookings. Retrieved December 4, 2025 from https://www.brookings.edu/articles/rendering-misrepresentation-diversity-failures-in-ai-image-generation/

[12] DeLozier, Josh. 2025. *How AI Bias Shapes Everything from Hiring to Healthcare.* OU News, February 5. Retrieved December, 8, 2025 from https://www.ou.edu/news/articles/2025/february/how-a-i-bias-shapes-everything-from-hiring-to-healthcare

Appendix

Class Assignments

02 Draft Group Charter

Final Project Proposal

Sprint Slideshows

AI Bias: Sprint 1

AI Bias: Sprint 2

AI Bias: Sprint 3

AI Bias: Sprint 4

PSU Assignments

LLM A1.ipynb

LLM A2 Zero Shot.ipynb

LLM A2 Few Shot.ipynb

Assignment 1 Screenshots

Llama-3.2-1B

```
import torch
from transformers import pipeline, AutoConfig, AutoModelForCausalLM, AutoTokenizer

use_auth_token = ""
model_id = "meta-llama/Llama-3.2-1B"

tokenizer = AutoTokenizer.from_pretrained(model_id,
                                          use_auth_token=use_auth_token)
model = AutoModelForCausalLM.from_pretrained(model_id,
                                             use_auth_token=use_auth_token,
                                             torch_dtype=torch.bfloat16,
                                             device_map="auto")


pipe = pipeline(
    "text-generation",
    model=model_id,
    torch_dtype=torch.bfloat16,
    tokenizer=tokenizer,
    device_map="auto"
)

prompt = "Once upon a time"
output = pipe(prompt, max_new_tokens=200, num_return_sequences=1, do_sample=True)

print(output[0]["generated_text"])
```

Text #	1	2	3	4	5	6	7	8	9	10
TTR	19.5%	49.4%	41.4%	50.8%	35.2%	58.2%	28.7%	25.1%	23.5%	23.2%

Average: 35.5%
Standard Deviation: 0.129

Once upon a time, a few years ago, I wrote a book called The Art of Not Giving a Fuck. I thought it was a good book. I thought it was funny. I thought it was well written. I thought it was a good read.
But I didn't give a fuck about it. I didn't give a fuck about the book. I didn't give a fuck about the people who bought it. I didn't give a fuck about the people who gave it to me. I didn't give a fuck about the people who read it.
I didn't give a fuck about the book. I didn't give a fuck about the people who read it. I didn't give a fuck about the people who gave it to me. I didn't give a fuck about the people who bought it. I didn't give a fuck about the book. I didn't give a fuck about the people who read it.
I didn't give a fuck about the book. I didn't give a fuck about
ttr: 0.195

Llama-3.2-3B

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
from google.colab import userdata

# 1. Get your secret Hugging Face token
hf_token = userdata.get('HF_TOKEN')

# 2. Load the tokenizer and model with the token and trust_remote_code
tokenizer = AutoTokenizer.from_pretrained(
    "meta-llama/Llama-3.2-3B",
    token=hf_token,
    trust_remote_code=True
)

model = AutoModelForCausalLM.from_pretrained(
    "meta-llama/Llama-3.2-3B",
    token=hf_token,
    trust_remote_code=True,
    torch_dtype=torch.bfloat16, # Recommended for performance
    device_map="auto"           # Automatically use GPU if available
)

# 3. Prepare and tokenize the input
prompt = "Once upon a time"
inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

# 4. Generate the output tokens from the model
outputs = model.generate(**inputs, max_new_tokens=200)

# 5. Decode the output tokens back to a string
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

print(generated_text)
```

Text #	1	2	3	4	5	6	7	8	9	10
TTR	52.2%	45.1%	50.9%	55.1%	38.4%	55.2%	42.0%	36.1%	50.0%	47.2%

Average: 47.2%
Standard Deviation: 0.064

Once upon a time there was a little girl who had a dream. She wanted to be a teacher, and she wanted to teach other children to read. But she was afraid. She was afraid that people would not listen to her, that she would not be able to teach them, that they would not want to learn. But she was determined. She was determined to make her dream come true.
The little girl went to the library and checked out books about teaching. She read everything she could find about teaching reading. She learned about phonics, about phonemic awareness, about the importance of reading aloud to children. She learned about the different types of reading instruction, and she learned about the different types of children.
The little girl practiced teaching her friends and family. She practiced reading aloud to them, and she practiced asking them questions about the books she read. She learned that some children learned differently than others, and she learned that some children were more interested in certain types of books than others.
The little girl also
ttr: 0.494

Phi-3.5-mini-instruct

```python
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline

# Set a seed for reproducibility
# torch.random.manual_seed(0)

# Load the model and tokenizer with all necessary arguments
model = AutoModelForCausalLM.from_pretrained(
    "microsoft/Phi-3.5-mini-instruct",
    device_map="auto",
    torch_dtype="auto",
    trust_remote_code=True,
    attn_implementation="eager",
)
tokenizer = AutoTokenizer.from_pretrained("microsoft/Phi-3.5-mini-instruct")

# Create the high-level pipeline
pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
)

# Define the message you want to send
messages = [
    {"role": "user", "content": "Once upon a time"},
]

# Define generation arguments
generation_args = {
    "max_new_tokens": 400,
    "return_full_text": False,  # So it doesn't repeat your prompt
    "do_sample": True,
    "temperature": 1.2,
    "top_p": 0.8,
    "top_k": 1000,
    "repetition_penalty": 1.1,
}

# Run the pipeline
output = pipe(messages, use_cache=False, **generation_args)
print(output[0]['generated_text'])
```

Text #	1	2	3	4	5	6	7	8	9	10
TTR	67.8%	71.1%	67.1%	67.6%	69.9%	68.6%	66.0%	63.5%	66.0%	66.4%

Average: 67.4%
Standard Deviation: 0.020

Once upon a time, in a land where the sunsets painted the sky in vibrant hues of orange and pink, there lived a young shepherd named Eli. Eli spent his days tending to his flock on the rolling hills of Emerald Valley, a place untouched by the hustle and bustle of the ever-expanding kingdom beyond.

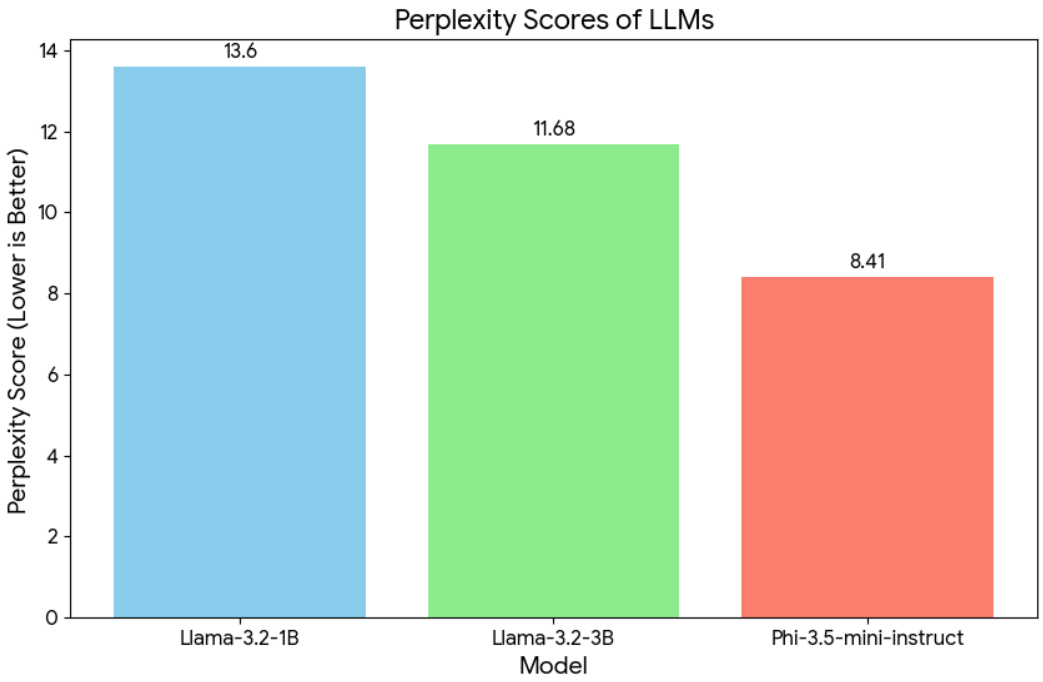
The villagers of Emerald Valley spoke of a legend that the valley was enchanted, its beauty a gift from the fairies who danced among the wildflowers each night. Eli, ever curious, felt drawn to the mysterious tales and longed to see the fairies for himself.
ttr: 0.683

Type Token Ratio Code

```python
# Type Token Ratio Script
import re

def type_token_ratio(text):
    words = re.findall(r'\b\w+\b', text.lower())
    if not words:
        return 0

    total_tokens = len(words)
    unique_types = len(set(words))
    ttr = unique_types / total_tokens
    return ttr

# Include generated text
sample_text = """
In times long past and places veiled in mist, there once was a quiet village nes
"""
ratio = type_token_ratio(sample_text)

print(f"Type-Token Ratio: {ratio:.3f}")
```

Perplexity Code and Results

```python
# Perplexity Script
import torch
import math
from transformers import AutoTokenizer, AutoModelForCausalLM
from google.colab import userdata

# 1. Get Hugging Face token
hf_token = userdata.get('HF_TOKEN')

# 2. Load tokenizer and model
model = AutoModelForCausalLM.from_pretrained(
    "microsoft/Phi-3.5-mini-instruct",
    device_map="auto",
    torch_dtype="auto",
    trust_remote_code=True,
    attn_implementation="eager",
)

tokenizer = AutoTokenizer.from_pretrained("microsoft/Phi-3.5-mini-instruct")

# 3. Text you want to evaluate
text = """
NEW YORK (AP) — The head coach of the Portland Trail Blazers and a player for th

Portland coach Chauncey Billups was charged with participating in a conspiracy t

The two indictments unsealed in New York create a massive cloud for the NBA — wh
"""

# 4. Compute perplexity
def compute_perplexity(text):
    encodings = tokenizer(text, return_tensors="pt").to(model.device)
    max_length = getattr(model.config, "n_positions", 1024)
    stride = 512

    nlls = []
    for i in range(0, encodings.input_ids.size(1), stride):
        begin_loc = max(i + stride - max_length, 0)
        end_loc = i + stride
        input_ids = encodings.input_ids[:, begin_loc:end_loc]
        target_ids = input_ids.clone()
        target_ids[:, :-stride] = -100  # mask out tokens we don't predict
        with torch.no_grad():
            outputs = model(input_ids, labels=target_ids, use_cache=False)
            neg_log_likelihood = outputs.loss
        nlls.append(neg_log_likelihood)

    ppl = torch.exp(torch.stack(nlls).mean())
    return ppl.item()

# 5. Print perplexity
perplexity = compute_perplexity(text)
print(f"Perplexity: {perplexity:.2f}")
```



Conservative Parameters (Phi-3.5-mini-instruct)

```
generation_args = {
    "max_new_tokens": 400,
    "return_full_text": False,
    "do_sample": True,
    "temperature": 0.7,
    "top_p": 0.2,
    "top_k": 10,
    "repetition_penalty": 1,
}
```

```
Once upon a time, in a land far, far away, nestled between the whispering
forests and the rolling hills, there lived a young princess named Elara.
Her kingdom was known for its lush gardens, majestic castles, and the
kindness of its people. However, beneath the surface of this idyllic
realm, a shadow loomed, threatening to engulf the land in darkness.

Elara, with her golden hair and eyes as bright as the morning sun, was
beloved by all. She spent her days wandering through the gardens, talking
to the flowers, and learning about the world from the wise old owl
perched in the royal library. But as the moon rose each night, the
princess couldn't shake off a sense of unease.

One evening, as she gazed at the stars from the castle's balcony, a
mysterious figure appeared before her. Cloaked in a shimmering robe that
seemed to change colors with the night sky, the stranger spoke in a voice
that was both ancient and melodic.

"Princess Elara," the figure said, "I come from a realm beyond your
dreams, where magic flows like rivers and dragons soar through the
clouds. I bring you a quest, one that will test your courage and your
heart."

Elara's curiosity was piqued, and she listened intently as
```

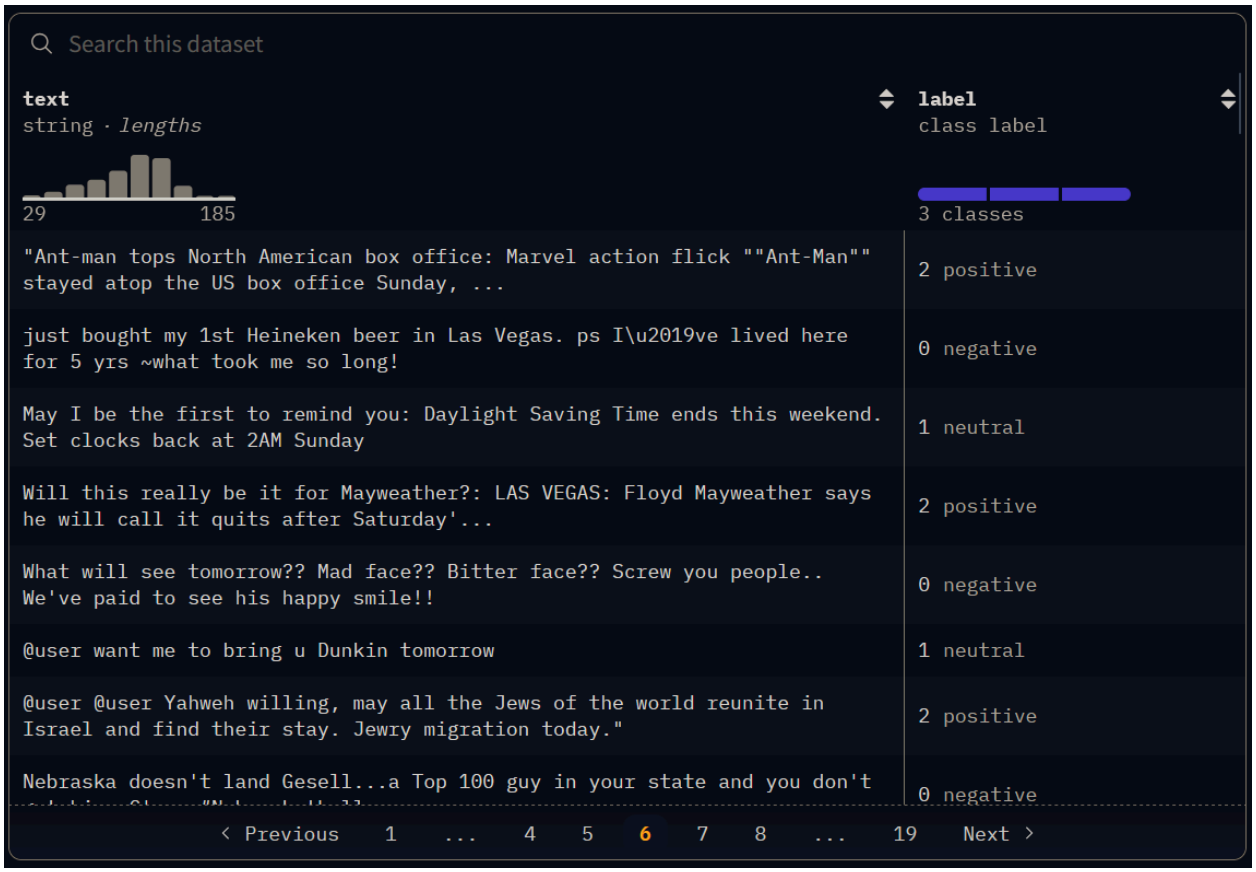
Creative Parameters (Phi-3.5-mini-instruct)

```
generation_args = {
    "max_new_tokens": 400,
    "return_full_text": False,
    "do_sample": True,
    "temperature": 1.2,
    "top_p": 0.8,
    "top_k": 1000,
    "repetition_penalty": 1.1,
}
```

In times long past and places veiled in mist, there once was a quiet village nestled between rolling hills blanketed with verdant meadows under a sky of endless blue. The villagers led simple lives full of song and dance during harvest festivals but always yearned for adventure beyond their known horizons. One evening as twilight embraced the land, fires crackling softly against encroaching darkness and children playing hide-and-seek near homes whispering secrets through chimneys, something miraculous unfolded—a light streaked across the heavens like liquid silver cutting into night itself. Whispers rose among men and women alike; they spoke ancient tongues not heard since days when magic still roamed free unbridled by skepticism's evergreen scorn. A young maiden named Lila listened intently from her open windowpane below, senses tingling with every shimmer, daring thoughts spinning within herself like leaves caught upward on wind's whimsical choreography. Tonight might just be forever etched amongst legends; tomorrow could awaken history reborn. And so it began...

Assignment 2 Screenshots

Dataset Snippet



Zero-Shot Inference Experiments

```
from datasets import load_dataset

# Load the dataset (English subset)
dataset = load_dataset("cardiffnlp/tweet_sentiment_multilingual", "english")

# Check the split sizes
# validation set: 324 instances
val_data = dataset['validation']
print(f"Validation set size: {len(val_data)}") # Should print 324
```

```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from google.colab import userdata

# Get your token
hf_token = userdata.get('HF_TOKEN')

def load_model_pipeline(model_id):
    print(f"Loading {model_id}...")

    # Special handling for Phi-3.5 (no token needed, specific attention)
    if "Phi-3.5" in model_id:
        tokenizer = AutoTokenizer.from_pretrained(model_id, trust_remote_code=True)
        model = AutoModelForCausalLM.from_pretrained(
            model_id,
            device_map="auto",
            torch_dtype="auto",
            trust_remote_code=True,
            attn_implementation="eager"
        )
    # Handling for Llama models (needs token)
    else:
        tokenizer = AutoTokenizer.from_pretrained(model_id, token=hf_token)
        model = AutoModelForCausalLM.from_pretrained(
            model_id,
            token=hf_token,
            device_map="auto",
            torch_dtype=torch.bfloat16
        )

    # Create a pipeline for easy text generation
    pipe = pipeline(
        "text-generation",
        model=model,
        tokenizer=tokenizer,
        max_new_tokens=10, # We only need a short answer (Positive/Negative)
        return_full_text=False
    )
    return pipe

# Define your model list
model_names = [
    "meta-llama/Llama-3.2-1B",
    "meta-llama/Llama-3.2-3B",
    "microsoft/Phi-3.5-mini-instruct"
]
```

```python
from sklearn.metrics import precision_recall_fscore_support, accuracy_score, classification_report
from tqdm import tqdm

def run_experiment_zero_shot(model_pipe, dataset):
    predictions = []
    references = []

    # Mapping dataset labels (0, 1, 2) to text
    # The dataset usually maps: 0 -> Negative, 1 -> Neutral, 2 -> Positive
    label_map = {0: "negative", 1: "neutral", 2: "positive"}

    print(f"Running inference on {len(dataset)} items...")

    for item in tqdm(dataset):
        text = item['text']
        true_label = label_map[item['label']]
        references.append(true_label)

        # [cite_start]1. Create the Prompt [cite: 743]
        # Added "Answer:" to guide the model to the exact token
        prompt = f"""
        Classify the sentiment of the following tweet as 'positive', 'neutral', or 'negative'.
        Tweet: "{text}"
        Answer:
        """

        # 2. Generate
        # Ensure return_full_text=False so we don't get the prompt back
        outputs = model_pipe(prompt, max_new_tokens=10, return_full_text=False, use_cache=False) # use_cache=False
        result = outputs[0]['generated_text']

        # 3. Clean the output
        pred_text = result.lower().strip()

        # Simple parsing logic
        if "positive" in pred_text:
            predictions.append("positive")
        elif "negative" in pred_text:
            predictions.append("negative")
        elif "neutral" in pred_text:
            predictions.append("neutral")
        else:
            predictions.append("neutral") # Default fallback

    return references, predictions

# --- Execution ---

# 1. Load your model (Ensure load_model_pipeline is defined from previous steps)
pipeline_1b = load_model_pipeline("microsoft/Phi-3.5-mini-instruct")

# 2. Run the experiment
refs, preds = run_experiment_zero_shot(pipeline_1b, val_data)

# [cite_start]3. Calculate Metrics (Using 'refs' and 'preds') [cite: 757]
precision, recall, f1, _ = precision_recall_fscore_support(refs, preds, average='weighted', zero_division=0)
acc = accuracy_score(refs, preds)

print(f"Accuracy: {acc:.4f}")
print(f"F1 Score: {f1:.4f}")

# [cite_start]4. Detailed Report [cite: 757]
print("\nClassification Report:")
print(classification_report(refs, preds, digits=4))
```

Llama-3.2-1B Results

	Precision	Recall	F1-Score
Negative	48.28%	12.96%	20.44%
Neutral	41.53%	45.37%	43.36%
Positive	35.03%	57.41%	43.51%

Llama-3.2-3B Results

	Precision	Recall	F1-Score
Negative	47.19%	38.89%	42.64%
Neutral	37.31%	23.15%	28.57%
Positive	42.86%	66.67%	52.17%

Phi-3.5-mini-instruct Results

	Precision	Recall	F1-Score
Negative	100%	2.78%	5.41%
Neutral	33.86%	99.07%	50.47%
Positive	80%	3.70%	7.08%

Few-Shot In-Context Learning Experiments

```
# ------------------------------
# 1. Install packages
# ------------------------------
!pip install "datasets<3.0.0" transformers evaluate accelerate --quiet

# ------------------------------
# 2. Imports
# ------------------------------
import os
import random
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from sklearn.metrics import classification_report
from datasets import load_dataset
import pandas as pd

# ------------------------------
# 3. Load dataset
# ------------------------------
dataset = load_dataset("cardiffnlp/tweet_sentiment_multilingual", "english")

# Convert to pandas for easier manipulation
train_df = dataset["train"].to_pandas()
val_df   = dataset["validation"].to_pandas()
test_df  = dataset["test"].to_pandas()

# Ensure labels are integers
train_df["label"] = train_df["label"].astype(int)
val_df["label"]   = val_df["label"].astype(int)
test_df["label"]  = test_df["label"].astype(int)

# Map numeric labels to sentiment strings
label_map = {0: "negative", 1: "neutral", 2: "positive"}
train_df["sentiment"] = train_df["label"].map(label_map)
val_df["sentiment"]   = val_df["label"].map(label_map)
test_df["sentiment"]  = test_df["label"].map(label_map)

print("Training label distribution:\n", train_df['sentiment'].value_counts())

# ------------------------------
# 4. Helper functions
# ------------------------------
HF_TOKEN = os.environ.get("HF_TOKEN")  # Hugging Face token stored in Colab

def load_model(model_name):
    tokenizer = AutoTokenizer.from_pretrained(model_name, use_auth_token=HF_TOKEN)
    model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", use_auth_token=HF_TOKEN)
    return tokenizer, model

def get_few_shot_examples(df, k=1):
    examples = []
    for cls in ["positive", "negative", "neutral"]:
        cls_samples = df[df['sentiment'] == cls]
        if len(cls_samples) == 0:
            print(f"Warning: no samples for class {cls}")
            continue
        n = min(k, len(cls_samples))
        examples.extend(cls_samples.sample(n, random_state=42).to_dict('records'))
    return examples

def create_prompt(few_shot_examples, text):
    prompt = ""
    for ex in few_shot_examples:
        prompt += f"Text: {ex['text']}\nSentiment: {ex['sentiment']}\n\n"
    prompt += f"Text: {text}\nSentiment:"
    return prompt
```

```python
# ------------------------------
# 5. Experiment runner
# ------------------------------
def run_experiment(model_name, k):
    print(f"\n🔵 Running {model_name} with k={k}...\n")

    tokenizer, model = load_model(model_name)
    classifier = pipeline("text-generation", model=model, tokenizer=tokenizer)

    few_shot_examples = get_few_shot_examples(train_df, k=k)
    if len(few_shot_examples) == 0:
        raise ValueError("No few-shot examples found. Check train_df['sentiment'].")

    predictions = []
    true_labels = []

    for _, row in val_df.iterrows():
        prompt = create_prompt(few_shot_examples, row['text'])
        output = classifier(prompt, max_new_tokens=10, do_sample=False)
        pred_text = output[0]['generated_text'].split("Sentiment:")[-1].strip().lower()

        if "positive" in pred_text:
            pred = "positive"
        elif "negative" in pred_text:
            pred = "negative"
        else:
            pred = "neutral"

        predictions.append(pred)
        true_labels.append(row['sentiment'])

    report = classification_report(true_labels, predictions, target_names=["negative","neutral","positive"], zero_division=0)
    print(f"✅ {model_name} (k={k}) performance:\n{report}")

# ------------------------------
# 6. Run experiments
# ------------------------------
models = [
    "microsoft/phi-3-mini-4k-instruct",
    "meta-llama/Llama-3.2-1B",
    "meta-llama/Llama-3.2-3B"
]

for model_name in models:
    for k in [1, 2]:
        run_experiment(model_name, k)
```

Llama-3.2-1B Results

	Precision (k=1)	Recall (k=1)	F1-Score (k=1)	Precision (k=2)	Recall (k=2)	F1-Score (k=2)
Negative	71%	56%	62%	90%	8%	15%
Neutral	41%	77%	54%	34%	99%	51%
Positive	86%	30%	44%	0%	0%	0%

Llama-3.2-3B Results

	Precision (k=1)	Recall (k=1)	F1-Score (k=1)	Precision (k=2)	Recall (k=2)	F1-Score (k=2)
Negative	84%	62%	71%	85%	38%	53%
Neutral	48%	65%	55%	38%	92%	53%
Positive	67%	60%	63%	85%	10%	18%

Phi-3.5-mini-instruct Results

	Precision (k=1)	Recall (k=1)	F1-Score (k=1)	Precision (k=2)	Recall (k=2)	F1-Score (k=2)
Negative	77%	81%	79%	77%	73%	75%
Neutral	59%	56%	58%	57%	64%	60%
Positive	76%	75%	75%	79%	75%	77%

Advanced Prompting Techique (Chain-of-Thought) Experiments

Llama-3.2-1B Results

Total Samples: 324
Evaluated Samples: 122
High failure rate (62%) in parsing CoT output.
Accuracy: 0.3525 Low overall classification accuracy.
F1-Score (Macro) : 0.2409 Poor performance across the three classes.

	Precision	Recall	F1-Score
Negative	22.22%	5.71%	9.09%
Neutral	25%	7.14%	11.11%
Positive	37.62%	84.44%	52.05%

Llama-3.2-3B Results

Total Samples: 324
Evaluated Samples : 5
Unmapped Rate : 319 (98.5%)

Conclusion: Forcing a large LLM like Llama-3B to generate both reasoning and a label severely breaks its structure when using a standard `pipeline` without advanced output parsing or structured generation.

	Precision	Recall	F1-Score
Negative	100%	33.33%	50%
Neutral	0%	0%	0%
Positive	25%	100%	40%

Phi-3.5-mini-instruct Results

Total Samples: 324
Evaluated Samples: 306 *Massive success.*
Unmapped Rate: 18 (5.6%)
The Phi model successfully **adhered to the complex output structure** required by the CoT prompt, while the Llama models did not.

	Precision	Recall	F1-Score
Negative	81.32%	72.55%	76.68%
Neutral	68.42%	13%	21.85%
Positive	50%	94.23%	65.33%

Conclusion

Conclusion: The Best Way to Classify Tweets

Category	Finding	Best Performer / Result	Key Takeaway
Best Technique	**Few-Shot In-Context Learning (ICL)**	Phi-3.5 (k=2) F1: ≈ 77%	**Giving examples** in the prompt (ICL) was the most reliable and accurate strategy for all models.
Model Performance	**Instruction Tuning > Raw Size**	Phi-3.5 outperformed Llama-3B	The instruct-tuned **Phi** model was the most balanced classifier, proving that **quality of training** is critical.
Model Scaling	**Scaling generally helps**	Llama-3B F1 ↑ vs Llama-1B F1 ↓	The larger Llama-3B model showed better and more stable performance in the Few-Shot setting than the Llama-1B.
Biggest Failure	**Advanced CoT Prompting**	Llama-3B 98.5% Unmapped	Forcing models to "think" (CoT) introduced crippling **output parsing errors** and was the least reliable method tested.