Midterm Project: Zero-Shot Sentiment Classification via LLM Logits Exploration

Name: Chaorui Zhu
Course: AI 100
Institution: The Penn State University

Feb 2026

1 Project Description

This project addresses the binary sentiment classification problem using the IMDB movie review dataset. While traditional approaches (like the Logistic Regression used in Homework #1) rely on training a classifier on labeled features, this project utilizes a **Large Language Model (LLM)** in a zero-shot capacity. By leveraging the pre-trained knowledge of the `Qwen3-0.6B` model, we perform classification by extracting internal **Logits** to calculate the probability of specific sentiment tokens without any further training or fine-tuning.

2 The Deep Learning Model: Qwen3-0.6B

The heart of this classification system is the **Qwen3-0.6B**, a state-of-the-art dense causal language model. Unlike traditional discriminative classifiers that map inputs to a fixed set of labels via a task-specific head, Qwen3 is a generative foundation model that leverages a vast pre-training corpus of **36 trillion tokens** across 119 languages. This allows the model to perform sentiment analysis through deep semantic understanding rather than simple surface-level pattern matching.

2.1 Architectural Specifications

The 0.6B variant is a **Causal Transformer Decoder** optimized for high-throughput and low-latency inference. Key architectural components derived from the Qwen3 Technical Report include:

- **Layer Configuration:** The model consists of **28 Transformer layers**, providing a balance between representational depth and computational efficiency.
- **Grouped Query Attention (GQA):** To optimize memory usage, Qwen3 utilizes GQA with a **16/8 head ratio** (16 Query heads to 8 Key/Value heads). This configuration significantly reduces the KV cache size, which is critical for maintaining performance across its 32k context window.
- **QK-Norm:** A critical stability innovation in the Qwen3 series is the introduction of **QK-Norm** (Query-Key Normalization). By normalizing the queries and keys before the dot-product attention, the model prevents logit drift and ensures stable, predictable distributions even in zero-shot settings.
- **Positional Encoding:** It employs **Rotary Positional Embeddings (RoPE)** to capture long-range dependencies within movie reviews effectively.

2.2 Logits Forced Choice Mechanism

The classification logic bypasses the standard "generation" phase—which can introduce stochasticity or off-topic text—and instead queries the model's internal probability distribution directly. This is mathematically more robust for binary classification tasks.

When the model processes a movie review prompt, it produces a vector of raw scores (**logits**) z at the final sequence position. We apply a **Log-Softmax** transformation to convert these scores into normalized log-probabilities:

$$\log P(x_{next} = t \mid \text{Prompt}) = z_t - \log \sum_{j \in V} \exp(z_j) \tag{1}$$

Where V represents the full vocabulary of **151,669 tokens**. The code extracts the log-probabilities specifically for the tokens corresponding to "positive" and "negative." The final prediction $\hat{y}$ is determined by the maximum accumulated log-probability:

$$\hat{y} = \arg \max_{L \in \{pos, neg\}} \left(\sum_{j=1}^{n} \log P(\text{token}_j \mid \text{Prompt}, \text{token}_{<j}) \right) \tag{2}$$

2.3 Tokenizer and Numerical Stability

Qwen3 employs a **Byte-level Byte-Pair Encoding (BBPE)** tokenizer. A crucial implementation detail is the handling of whitespace; because tokenizers are sensitive to word boundaries, the code specifically targets the token IDs for `" positive"` and `" negative"` (including the leading space). By operating in the **log-probability domain** (summing logs instead of multiplying raw probabilities), the system avoids numerical underflow, ensuring stable and precise results even for highly descriptive movie reviews.

3 Model Architecture

The core engine of this project is a **Causal Transformer Decoder**. Unlike Encoder-only models (e.g., BERT), a Causal Decoder uses **Masked Self-Attention** to ensure that each token can only attend to the tokens that precede it.

3.1 Mathematical Objective

In a causal language model, the primary objective is to predict the probability of the next token x_{t+1} given a sequence $x_{1...t}$. For classification, we evaluate the conditional log-probability of a candidate label L given a prompt P:

$$\log P(L \mid P) = \sum_{j=1}^{n} \log P(token_j \mid P, token_{<j}) \tag{3}$$

Where the probability is derived from the **Softmax** of the model's output logits z:

$$P(token_i) = \frac{\exp(z_i)}{\sum_{j \in V} \exp(z_j)} \tag{4}$$

4 Detailed Code Structure and Implementation

The implementation focuses on a "Forced Choice" mechanism. Below is a granular explanation of the code components:

4.1 Environment and Hardware Acceleration

The code dynamically selects the computation device. For users on Apple Silicon, it utilizes the `mps` (Metal Performance Shaders) backend, which significantly accelerates tensor operations compared to standard CPUs.

4.2 The Scoring Engine: `score_candidate`

This nested function is the most critical part of the algorithm. It does not "generate" text; rather, it evaluates the likelihood of a pre-defined path:

1. **Teacher Forcing:** It concatenates the `input_ids` (the review) with the `candidate_token_ids` (the word "positive" or "negative").
2. **Forward Pass:** The model processes the sequence in one go using `model(input_ids=full_ids)`.
3. **Logit Alignment:** Because the model predicts the *next* token, the logit at index i corresponds to the prediction for index $i + 1$. The code carefully calculates the position: `pos = prompt_len + j - 1`. This ensures we are extracting the score for the actual candidate token based on the prompt's context.
4. **Log-Probability Summation:** By using `torch.log_softmax`, we transform raw scores into a log-probability space. Summing these values allows us to handle words that are split into multiple sub-tokens by the tokenizer.

4.3 Streaming Dataset Management

To handle the IMDB dataset efficiently, the code uses `streaming=True`. This prevents the system from downloading the entire 80MB+ dataset into memory, instead fetching samples on-the-fly. We limit the processing to the first 500 characters of each review to maintain a consistent prompt length and speed up inference.

5 Sample Running Results

The following table illustrates how the model differentiates between sentiments by assigning higher (less negative) log-probability scores to the correct label.

Review Snippet	Actual	Pos Score	Neg Score	Result
"A masterpiece of cinema..."	Positive	**-3.42**	-10.15	Correct
"Complete waste of time."	Negative	-12.80	**-2.95**	Correct
"The plot was quite weak."	Negative	-9.12	**-5.33**	Correct

Table 1: Inference Results and Log-Probability Comparisons

6 Lessons and Experience

- **Tokenization Nuance:** I learned that tokenizers are highly sensitive to leading spaces. Encoding `" positive"` results in a different token ID than `"positive"`. Failing to account for this would result in the model looking for the wrong token in its vocabulary.
- **Efficiency of Zero-Shot:** Unlike Logistic Regression, which requires thousands of labeled examples to learn a weight matrix, this LLM approach achieves high accuracy immediately due to its vast pre-training on diverse text.

- **Inference over Training:** This project shifted my focus from backpropagation and gradient descent to the mechanics of transformer inference and the importance of prompt structure.