Sentiment Classification with Transformer Models: A Comparative Analysis of nanoGPT and GPT-2 Fine-Tuning

Turkan Simge Ispak
Middle East Technical University
turkanispak@gmail.com

***Abstract*—In this study, transformer-based models were used to classify sentiment in customer service dialogues. Both a nanoGPT model trained from scratch and a fine-tuned GPT-2 model were explored. The models underwent progressively increasing architectural modifications, starting with prompt engineering to generate sentiment labels (positive, negative, or neutral), and progressing to the use of dedicated classification heads. Metrics such as accuracy, F1-score, and confusion matrices were analyzed. The fine-tuned GPT-2 model outperformed the scratch-trained model, validating the effectiveness of leveraging pre-trained weights for low-resource classification tasks.**

I. Introduction

The objective of this assignment is to apply transformer architectures to sentiment classification in customer support conversations. The study focused on adapting nanoGPT, a GPT-like transformer, and comparing its performance with a fine-tuned GPT-2 model. Given the experimental nature of this work, any improvement over naive performance was considered acceptable. The sample codebase and training framework provided in the assignment repository served as the starting point. Since these models are inherently generative, and keeping the fine-tuning task in mind, classification had to be approached using various inference and minor architectural changes to maintain compatibility with existing training pipelines and hyperparameter settings.

The three approaches include first formatting the dataset and introducing configuration-level changes such as adjusting the block_size parameter to accommodate longer conversation inputs to be compatible with the baseline sample code which allowed experiments to begin without altering model architecture or core training code.

Next, inference-time "tricks" were applied. Trick 1 restricted the model's output to valid sentiment words by scoring only a subset of next-token logits. Trick 2 used prompt engineering to steer generation behavior by asking the model to output a one-word sentiment judgment. These tricks were tested on both pretrained GPT-2 and nanoGPT variants, using identical evaluation scripts.

Finally, architectural changes were introduced. A classification head was added to the model, and the training script was updated to compute loss based on class labels rather than predicted tokens. This approach, while closer to a conventional classification setup, resulted in longer training times and degraded performance in the nanoGPT model, which appeared to collapse under this training regime. It also wasn't able to fully align with the task of finetuning GPT-2 with pre-trained weights.

II. Dataset

The dataset was acquired from the course GitHub repository and consists of customer service dialogues labeled as *positive*, *negative*, or *neutral*. Initial exploratory data analysis (EDA) revealed a class imbalance, particularly with fewer *positive* samples where Only 1.75% of samples were positive as seen in Table I. To mitigate this, oversampling techniques were employed, balancing the class distribution to enhance model generalization. Feedbacks are lengthy (300–400 words), especially negative ones which align with real-world behaviour, necessitating truncation to meet model token limits. Common terms analysis per sentiment are "sorry", "refund" in negative and "thank", "great" in positive, which were used an experiment with Trick 1 to decrease the signal-to-noise ratio of the data. A custom preprocessing script was developed, converting CSV datasets into plain-text files, formatted specifically to match GPT-2's expected input for fine-tuning. Additionally, binary files compatible with the nanoGPT training pipeline were generated using tiktoken tokenizer to convert to token IDs. Both sets of files utilized GPT-2's Byte-Pair Encoding (BPE) tokenizer, ensuring consistent vocabulary and tokenization strategies across training and evaluation stages.

TABLE I: Dataset statistics per sentiment class

Sentiment	Proportion (%)	Avg. Char Length	Avg. Word Count
Neutral	55.88	1940.83	337.78
Negative	42.37	2395.45	417.67
Positive	1.75	1702.71	294.00

After the preprocessing step class balancing yield was 542 samples per class (neutral, negative, positive) with truncated texts of 1024 characters. Train/Val split distribution has 1463 training samples and 163 validation samples where each sample has the "conversation + SENTIMENT: label".

III. Modeling

Initially, the nanoGPT architecture was employed framing sentiment classification purely as a prompt-based text generation task and trained from scratch, 6 layers, 6 attention heads, and an embedding size of 384 were chosen, along with

a context block size of 1024 tokens, dropout of 0.2, learning rate of 1×10^{-3}, and batch size of 8. The GPT-2 fine-tuning leveraged larger-scale pretrained weights with a block size of 1024, batch size of 12 (with gradient accumulation).

Subsequently, to explicitly optimize for classification accuracy, a linear classification head was added and loss function was changed, transforming the task into a supervised classification problem. The decision to introduce only a classifier head was intentional, aiming task-specific adaptation with the preservation of pretrained GPT-2 weights.

IV. Evaluation

Training was monitored using Weights & Biases (WANDB), with visualizations of loss curves to detect signs of overfitting or instability. Generated sample dialogue completions were reviewed qualitatively to assure the integrity of model training, complementing quantitative metrics.

The evaluation phase involved generating sentiment predictions and comparing these against ground truth using three key metrics: accuracy, F1-score, and confusion matrices. Accuracy provided a general overview of model correctness, while the F1-score offered critical insights due to its sensitivity in imbalanced class distributions. Confusion matrices further clarified model performance by identifying specific misclassifications across sentiment categories.

V. Results

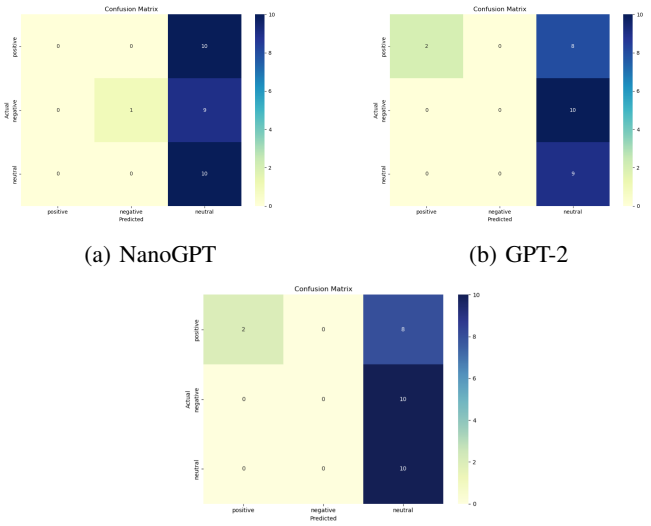
Evaluation was conducted using 30 test samples. The fine-tuned GPT-2 model showed the strongest performance when combined with Trick 1, reaching an accuracy of 40% and a macro-F1 of 0.32 and outperformed nanoGPT as seen in as seen in Table II. In contrast, nanoGPT model, despite learning syntactic patterns, struggled with label generation and heavily biased predictions toward the *neutral* class as seen in Figure 1. It performed best under the minimal configuration-only setup, where block size was increased and no additional classification head or prompting trick was used reaching an accuracy of 36.67%. Under more complex setups, nanoGPT's performance degraded, often predicting a single class or collapsing entirely.

Trick 2, led to degenerate predictions across both models. Trick 1 improved the class distribution in GPT-2 predictions but worsened performance in nanoGPT. The classification-head version of the model did not outperform the simpler alternatives and introduced higher training cost with little gain.

In short, models could consistently outperform a trivial baseline (e.g., predicting a single dominant class), but gains were largely dependent on preserving alignment with the original training objective.

TABLE II: Performance Comparison of Models

Metric	**nanoGPT**	**GPT-2**	**GPT-2 (Modified)**
Accuracy	36.67%	37.93%	40.00%
Macro F1	0.2315	0.2778	0.3205
Micro F1	0.3667	0.3793	0.4000
Precision	0.4483	0.4444	0.4487
Recall	0.3667	0.4000	0.4000



(a) NanoGPT (b) GPT-2

(c) GPT-2 w/Trick 1

Fig. 1: Confusion matrices for each model. Pretrained GPT-2 models outperform NanoGPT, especially when prompting is combined with output constraints.

VI. Discussion

The results support the expectation that fine-tuning a large pre-trained model is more effective than training from scratch, especially with limited data. While both models generated sentiment labels when prompted, fine-tuned GPT-2 model produced better distribution across the classes.

Several factors likely contributed to the overall low performance. The models were not explicitly trained for classification tasks. Decoder-only architectures are optimized for sequence continuation, not structured label prediction. Long conversations may have caused relevant sentiment cues to be truncated. Training signals were weak due to indirect supervision via next-token prediction. Some classes (e.g., negative) may rely on subtle linguistic cues that are hard to capture. Prompting and scoring tricks improved results slightly, but could not overcome fundamental limitations of the architecture.

VII. Conclusion

The experiment validated that transfer learning significantly improves transformer-based classification tasks under data-constrained scenarios. Sentiment classification using decoder-only language models presents unique challenges due to the generative nature of the architecture. However, by leveraging prompt formatting, strategic configuration changes, and inference-time tricks, reasonable performance was achieved without requiring extensive architectural redesign. Notably, Trick 1 improved zero-shot performance for pretrained GPT-2, while nanoGPT benefited most from configuration-only tuning. Introducing a classification head offered no consistent advantage and often reduced performance, suggesting that minimal, model-aligned modifications yield better results for this task.