Hierarchical Opinion Classification using Large Language Models

Shuvam Banerji Seal[1,*,†], Alok Mishra[2,†] and Utkarsha Ghosh[3,†]

[1]*Department of Chemical Sciences, Indian Institute of Science Education and Research, Kolkata, India*
[2]*Indian Institute of Science Education and Research, Kolkata, India*
[3]*Department of Information Technology, Institute of Engineering and Management, Kolkata, India*

Abstract

We address the task of hierarchical opinion classification with large language models (LLMs). Our approach employs parameter-efficient fine-tuning of the Gemma model by attaching a lightweight two-layer classification head (LayerNorm → Linear → GELU → Linear) and updating only the final transformer block, normalization, and output layers. The original three-level hierarchy of opinion labels is reformulated into an 8-class flat scheme, enabling direct optimization. To mitigate data imbalance, we adopt class-weighted cross-entropy loss, ensuring improved treatment of minority categories. Experimental evaluation is conducted on this reformulated dataset, using accuracy metrics sensitive to class imbalance. The second approach explores instruction fine-tuning, training the model to generate labels in a prompt-response format using a next-token prediction objective with a masked loss function that focuses only on the answer tokens. For both methods, the original three-level hierarchy of opinion labels is reformulated into an 8-class flat scheme, and class-weighted cross-entropy loss is used to mitigate data imbalance. Our evaluation contrasts the effectiveness of selective fine-tuning and custom heads for adapting LLMs to structured opinion classification under computational constraints, against the generative alignment of instruction-tuning, providing insights into adapting LLMs for hierarchical classification under strict computational and data imbalance constraints.

Keywords

Large Language Models, Text Classification, Hierarchical Labels, Fine-Tuning, Class Imbalance

1. Introduction

User-generated text on platforms such as Reddit, Twitter, and YouTube is highly diverse, noisy, and often hierarchical in nature. Traditional sentiment analysis methods typically treat this as a flat classification task, which fails to capture the layered structure of opinions. For instance, a comment may first be categorized as *subjective*, then further divided into *positive* or *negative*, and in some cases split into even finer categories such as *questions* or *advertisements*. This complexity, combined with heavy class imbalance where certain categories dominate the data, poses significant challenges for robust opinion classification.

To address these issues, we adapt the **Gemma-1B**[1] large language model for hierarchical opinion classification. Specifically, we collapse the three-level label scheme into a flat 8-class structure and replace the model's original output layer with a custom classification head. To balance efficiency and performance, we employ parameter-efficient fine-tuning by training only the last transformer block, final normalization layers, and the classification head, while freezing earlier layers. Furthermore, we mitigate class imbalance through weighted cross-entropy loss with inverse-frequency class weights. This approach improves the recognition of minority classes while maintaining strong overall accuracy, demonstrating the effectiveness of lightweight fine-tuning for real-world opinion classification.

Forum for Information Retrieval Evaluation, December 17-20, 2025, India
*Corresponding author.
†These authors contributed equally.

sbs22ms076@iiserkol.ac.in (S. Banerji Seal); maa24ms215@iiserkol.ac.in (A. Mishra); utkarsha.ghosh2023@iem.edu.in (U. Ghosh)
0009-0000-0714-569X (S. Banerji Seal); 0009-0002-7472-0812 (A. Mishra); 0009-0006-7265-7759 (U. Ghosh)

2. Objective

The primary goal of this project is to develop an effective solution for an 8-class text classification task derived from a 3-level hierarchical opinion classification dataset. The approach involves extending a pre-trained large language model (LLM) by attaching a custom classification head and selectively fine-tuning specific layers. This design enables the model to adapt its generalized language representations to the downstream classification task. In addition, strategies for addressing class imbalance are incorporated to enhance predictive performance and ensure robustness across all categories.

3. Problem Statement

The task addressed in this work is an opinion classification problem based on a hierarchically structured dataset. The dataset is annotated across three levels, each refining the granularity of classification:

- **Level 1: Coarse-grained Opinion Classes.** The first level categorizes each text into three broad classes: *Noise* (label 0), *Objective* (label 1), and *Subjective* (label 2).
- **Level 2: Subjective Subclasses.** The second level refines the *Subjective* category into three sentiment-based classes: *Neutral* (label 0), *Negative* (label 1), and *Positive* (label 2).
- **Level 3: Neutral Subclasses.** The third level further decomposes the *Neutral* class into four specialized categories: *Neutral Sentiment* (label 0), *Question* (label 1), *Advertisement* (label 2), and *Miscellaneous* (label 3).

This hierarchical structure results in an effective **8-class text classification task** at the leaf level (*Noise, Objective, Positive, Negative, Neutral Sentiment, Question, Advertisement, Miscellaneous*). The primary challenge lies in leveraging the hierarchical dependencies while addressing issues such as class imbalance and semantic overlap among categories.

4. Data Cleaning and Preparation Workflow

The quality of training data directly impacts the performance of large language models (LLMs). Therefore, designing a systematic workflow for dataset cleaning and preparation is critical to ensure reliability and reproducibility. A similar approach of such opinionated textual information pre-processing was shown in [2] where they created a multi-stage query reformulation pipeline. We got inspired from it and have adapted and modified it for pre-processing. In this section, we outline the comprehensive workflow used for cleaning and structuring multiple social media datasets (Reddit, Twitter, YouTube, and QnA-Train). The workflow consists of four major phases, each addressing a distinct aspect of data quality.

4.1. Phase 1: Initial Loading and Structural Integrity Checks

This phase focused on loading heterogeneous social media datasets (Twitter, YouTube, and Reddit) and ensuring their structural consistency before semantic processing.

- **Step 1.1: Dataset Ingestion and Schema Alignment.**
  Each source dataset exhibited slight variations in field names (e.g., `text`, `content`, `body`). To ensure a unified schema, these columns were standardized into a single textual field `text` and a label field `label`. Data were ingested using `pandas.read_csv()` with explicit encoding (`utf-8`) to prevent parsing errors and ensure consistent handling of multilingual and special characters.
- **Step 1.2: Detection and Removal of Structural Duplicates and Null Records.**
  Rather than relying on generic removal commands such as `drop_duplicates()` and `dropna()`, a more controlled and semantically guided strategy was implemented to maintain dataset integrity.

- **Step 1.2.1: Identification and Elimination of Null Entries.**
  Instances containing missing textual fields were first identified using a Boolean mask generated by `is_nan()`. Rows flagged as null were explicitly filtered out using conditional selection rather than direct calls to `dropna()`, ensuring complete visibility into the number and distribution of removed entries. This preemptive elimination of null samples prevented downstream tokenization errors caused by empty or undefined strings.
- **Step 1.2.2: Token-Based Duplicate Detection.**
  To detect semantic duplicates, each text entry was tokenized using gemma 3b tokenizer after null-value removal, and the total number of tokens was computed for every record. The resulting counts were stored in an auxiliary column `num_tokens`. The dataset was then sorted in ascending order based on this column, allowing easier visual and programmatic inspection of potential duplicates.
- **Step 1.2.3: Semantic Verification and Row Pruning.**
  Rows exhibiting identical token sequences and matching token lengths were flagged as structural duplicates. These entries, often representing repeated comments or cross-platform reposts, were systematically removed to prevent redundant gradient updates during model fine-tuning. This token-level validation ensured that duplicate detection was performed at a semantic rather than purely string-based level.
- **Step 1.2.4: Corpus Integrity Verification.**
  Following duplicate elimination, row indices were reindexed to maintain dataset continuity, and sample-level statistics (mean and variance of token counts) were recalculated. This confirmed that the cleaning process preserved the natural distribution of text lengths across social platforms.

- **Step 1.3: Elimination of Non-Informative Columns.**
  Non-essential metadata fields such as user identifiers, timestamps, and comment IDs were dropped to minimize noise and reduce the dataset to its semantically relevant components. This ensured that the subsequent cleaning and modeling phases operated exclusively on meaningful linguistic and categorical information.

4.2. Phase 2: Content-Level Cleaning and Text Refinement

After achieving structural consistency, the next phase focused on cleaning the textual content itself to eliminate noise and unify linguistic representation across platforms.

- **Step 2.1: URL and Hyperlink Removal.**
  All web links (e.g., `https://t.co/...`, `www.youtube.com/...`) were stripped using regular expressions (`re.sub(r'http§+|www§+', ", text)`). Links often introduce high-variance tokens that provide no semantic value to opinion or sentiment classification.
- **Step 2.2: Mention and Hashtag Filtering.**
  Social references (`@username`) and hashtags (`#topic`) were removed to prevent token sparsity and overfitting to platform-specific metadata. In selective cases, hashtags conveying clear sentiment (e.g., "#happy") were optionally retained during exploratory analysis but excluded in the final standardized corpus.
- **Step 2.3: Emoji and Symbol Normalization.**
  Emojis and pictographic symbols were filtered using a Unicode-based regular expression pattern. These characters inflate the tokenizer's vocabulary space without contributing consistent semantic information across samples.
- **Step 2.4: Punctuation and Special Character Handling.**
  Non-alphanumeric symbols were removed except for interrogative punctuation ("?") and exclamatory marks ("!"). These were retained intentionally, as they serve as discriminative cues for Level-3 categories such as *Questions* and *Advertisements*.

- **Step 2.5: Whitespace Normalization and Compacting.**
  Extraneous whitespace, tab characters, and newline symbols were consolidated into single spaces using `re.sub(r'\s+', ' ', text)`. This ensured consistent token spacing prior to tokenization.
- **Step 2.6: Low-Quality and Gibberish Filtering.** Posts with fewer than fifty token, repetitive character patterns (e.g., "hahahahahahaha"), or alphabetic ratios below 30% were categorised as noise. This step was critical for preserving meaningful linguistic structure in downstream learning.

4.3. Phase 3: Text Normalization for Model Readiness

The final phase standardized the cleaned text to ensure compatibility with transformer-based tokenization and to preserve semantic cues necessary for hierarchical classification.

- **Step 3.1: Case Normalization.**
  All text was converted to lowercase, ensuring that tokens such as "Great" and "great" are treated identically by the tokenizer, thereby reducing vocabulary sparsity.
- **Step 3.2: Domain-Specific Token Preservation.**
  Certain lexical items indicative of advertisement or spam intent (e.g., "offer", "discount", "subscribe") were deliberately retained, as they provide discriminative signals for the *Advertisement* subclass.
- **Step 3.3: Final Text Validation.**
  Each cleaned entry was verified to contain at least one alphabetic token after normalization. The finalized corpus was then stored as `(text, label)` pairs, ready for tokenization and batching in the fine-tuning pipeline.

4.4. Phase 4: Final Formatting

The final stage focused on ensuring label consistency, interpretability, and compatibility of the dataset with the model's classification head. This phase bridged the cleaned textual data and the numerical representations required for supervised fine-tuning.

- **Step 4.1: Hierarchical Label Consolidation.**
  The original annotation schema spanned three hierarchical levels:

Table 1
Hierarchical Label Taxonomy

Level	Categories
Level 1	NOISE, OBJECTIVE, SUBJECTIVE
Level 2	(for SUBJECTIVE): NEUTRAL, NEGATIVE, POSITIVE
Level 3	(for NEUTRAL): NEUTRAL_SENTIMENTS, QUESTIONS, ADVERTISEMENTS, MISCELLANEOUS

  To ensure unified supervision for the classification head, these labels were flattened into a single categorical space, resulting in an eight-class system encompassing all terminal categories.
- **Step 4.2: Numeric Label Encoding.**
  Each unique label was assigned a numeric identifier to facilitate model training. The mapping followed a deterministic scheme:

Table 2
Label Categories Used for Classification

Label ID	Category
0	NOISE
1	OBJECTIVE
2	SUBJECTIVE_POSITIVE
3	SUBJECTIVE_NEGATIVE
4	NEUTRAL_SENTIMENTS
5	QUESTIONS
6	MISCELLANEOUS
7	ADVERTISEMENTS

This encoding maintained hierarchical interpretability while enabling the classification head's linear output layer to operate over a fixed-size label space.

- **Step 4.3: Human-Readable Mapping for Analysis.**
  To facilitate interpretability and post-hoc evaluation, a reverse mapping from numeric IDs to textual labels was retained. This allowed seamless inspection of model predictions during qualitative error analysis and confusion matrix generation.

Together, these four phases provide a robust framework for preparing noisy, heterogeneous social media datasets for large-scale machine learning. By systematically addressing structural errors, content quality, normalization, and label formatting, the workflow ensures high-quality, standardized inputs for subsequent experiments.

5. Data Analysis of Datasets

Prior to model fine-tuning, an exploratory data analysis was performed to understand the distribution of text lengths across different social media platforms. The number of tokens per entry, generated after initial cleaning and tokenization, was used as a proxy for content length and complexity. This analysis provides insight into potential padding/truncation requirements, the prevalence of extremely short or long texts, and the overall distributional characteristics that may impact model learning. Separate analyses were conducted for Reddit, Twitter, and YouTube datasets, as summarized below.

Reddit Dataset

Table 3: Token Count Statistics for Reddit Dataset

Statistic	Value
Count	5000
Mean	186.085
Std	384.334
Min	4
25%	26
50% (Median)	58
75%	264
Max	15535

Table 4: Class Distribution in the Reddit Dataset

Label	Count
Negative	410
Neutral sentiment	476
Question	135
Noise	645
Positive	259
Objective	503
Miscellaneous	212
Advertisement	105

Twitter Dataset

Table 5: Token Count Statistics for Twitter Dataset

Statistic	Value
Count	4987
Mean	46.239
Std	19.306
Min	2
25%	33
50% (Median)	43
75%	55
Max	151

Table 6: Class Distribution in the Twitter Dataset

Label	Count
Negative	78
Neutral sentiment	176
Question	135
Noise	1338
Positive	268
Objective	1700
Miscellaneous	9
Advertisement	46

YouTube Dataset

Table 7: Token Count Statistics for YouTube Dataset

Statistic	Value
Count	5000
Mean	37.915
Std	41.406
Min	4
25%	18
50% (Median)	26
75%	43
Max	1128

Table 8: Class Distribution in the YouTube Dataset

Label	Count
Negative	1574
Neutral sentiment	1391
Question	1000
Noise	786
Positive	207
Objective	32
Miscellaneous	9
Advertisement	1

QnA-Train Dataset

Table 9: QnA-Train dataset summary

Attribute	Statistics
Rows	25,361
Columns	5 (`title`, `selftext`, `MAIN`, `comment_body`, `relevance`)
Duplicates	456
Missing Values	`title`: 8 (0.03%), `selftext`: 1,185 (4.67%), `MAIN`: 80 (0.32%), `comment_body`: 82 (0.32%), `relevance`: 124 (0.49%)
Label Distribution	0.0: 21,533, 1.0: 3,704, nan: 124

5.1. Dataset Token and Class Distribution Analysis

Following the cleaning phase, we conducted a detailed analysis of token length and class distributions across all datasets. This step ensured that the textual data exhibited consistent structural properties, with minimal variance in sequence lengths and well-defined label proportions. The preprocessing pipeline effectively removed incomplete and redundant entries, standardized class names, and merged fragmented textual segments. As a result, the final datasets were more coherent and semantically interpretable, providing a robust foundation for subsequent model fine-tuning and evaluation.

6. Related Work

6.1. The Transformer Architecture

The Transformer architecture, introduced in the landmark paper "Attention Is All You Need" [3], replaced recurrent and convolutional layers with a self-attention mechanism that enables models to process entire sequences in parallel. This design allows the model to capture long-range dependencies more effectively while significantly reducing training time compared to recurrent networks. The encoder–decoder structure of the Transformer has since become the foundation of nearly all modern large language models (LLMs).

Several extensions have been proposed to improve efficiency and scalability. The Reformer [4] introduced locality-sensitive hashing (LSH) attention and reversible residual layers, reducing the memory footprint and enabling the handling of very long sequences. The Longformer [5] proposed a sparse attention mechanism, combining local sliding window attention with global tokens, making it suitable for processing long documents with thousands of tokens.

6.2. Pre-training and Fine-tuning

The paradigm of pre-training followed by fine-tuning has revolutionized natural language processing (NLP). In this approach, large models are first pre-trained on massive text corpora to learn general language representations, and then fine-tuned on smaller task-specific datasets to adapt to downstream applications.

BERT [6] demonstrated the effectiveness of bidirectional transformers for pre-training, introducing masked language modeling and next sentence prediction objectives. RoBERTa [7] improved upon BERT by training on larger datasets, longer sequences, and removing next sentence prediction, achieving higher accuracy across multiple benchmarks. T5 (Text-to-Text Transfer Transformer) [8] unified a wide range of NLP tasks under a single "text-to-text" framework, showing that pre-training with a denoising objective can transfer effectively to diverse applications such as translation, summarization, and classification.

6.3. Instruction Tuning

Instruction tuning refers to fine-tuning language models on datasets where tasks are framed as natural language instructions paired with the desired outputs. This makes models more adaptable and improves their performance in zero-shot and few-shot scenarios.

The FLAN work [9] showed that fine-tuning models on a mixture of instruction-following datasets improves zero-shot generalization across unseen tasks. InstructGPT [10] advanced this approach by incorporating Reinforcement Learning with Human Feedback (RLHF), aligning model behavior more closely with human intent and safety considerations. Self-Instruct [11] proposed a scalable method in which the model itself generates synthetic instruction–response pairs, which are then used for additional fine-tuning. This reduces the reliance on costly human-labeled instruction datasets.

6.4. Efficiency-Oriented Fine-Tuning Techniques

To enable fine-tuning of the Gemma model on limited GPU resources without compromising model fidelity, multiple optimization and stabilization techniques were employed. These strategies collectively enhanced computational efficiency, reduced memory consumption, and stabilized gradient dynamics throughout training.

- **Step 1: 4-bit Quantization [12] for Memory Optimization.**
  To fit the large-scale model within the available 24 GB GPU memory, quantization was performed using the `bitsandbytes` backend in 4-bit precision. This reduced the memory footprint by nearly 75% compared to full-precision weights while preserving representational fidelity through

adaptive rounding and group-wise scaling (NF4 quantization). The quantized format allowed efficient gradient updates and larger batch sizes during fine-tuning.

- **Step 2: Eager Attention Mechanism.**
  The Gemma architecture was configured to use the Eager Attention backend, which eliminates redundant CUDA graph recompilation and dynamically optimizes attention computation at runtime. This approach accelerated forward–backward passes and reduced memory fragmentation in GPU execution, improving throughput stability during training.
- **Step 3: Gradient Clipping for Stabilized Training.**
  To mitigate exploding gradients and maintain numerical stability during fine-tuning, gradient norms were clipped. This ensured that parameter updates remained within a bounded range, preventing destabilizing gradient spikes in low-precision training regimes.
- **Step 4: Adaptive Optimization and Scheduling.**
  The optimizer was configured as AdamW [13] with a learning rate of 5×10^{-5} and weight decay of 0.1 to ensure stable convergence.
  A cosine learning rate scheduler with warm-up was employed to gradually increase the learning rate during the initial phase of training, followed by smooth decay:
  This warm-up ratio 15% mitigated optimization shocks during early epochs and helped the model achieve stable convergence across 3 training epochs.
- **Step 5: Mixed-Precision and Efficient Data Handling.**
  The training pipeline utilized mixed-precision (`bfloat16/float32`) computation to balance speed and numerical precision. Data loading was parallelized via PyTorch's `DataLoader` with pinned memory and dynamic collation, minimizing CPU–GPU transfer overhead.

Collectively, these strategies enabled efficient fine-tuning of the multi-billion-parameter Gemma model on a single 24 GB GPU, maintaining stability and performance without requiring full-parameter updates.

7. Model Architecture

Large Language Models (LLMs) like **Gemma** are inherently designed for generative tasks, yet with minor architectural adaptations, they can be effectively repurposed for discriminative objectives such as text classification. In this work, we utilize the **Gemma-1B** model—a compact, one-billion-parameter LLM that balances computational efficiency with strong pretrained representational capacity, enabling feasible fine-tuning on limited hardware resources.

To adapt Gemma for classification, we replace its original output projection layer—which maps hidden representations onto a vocabulary of approximately 262,144 tokens—with a **custom classification head**. This new head maps the final hidden states to eight target classes corresponding to the hierarchical opinion dataset. Consequently, the model preserves the expressive power of the pretrained transformer backbone while aligning its output space with the specific requirements of the downstream task. An important architectural nuance exists between the smaller **Gemma-1B** and the larger **Gemma-4B** variants. In **Gemma-1B**, internal components such as transformer blocks, the final LayerNorm, LM head, and classification head are directly accessible through the model object. In contrast, **Gemma-4B** encapsulates these components within an additional container module, altering the access patterns for model submodules. This distinction is crucial when attaching custom heads, unfreezing layers, or performing parameter-efficient fine-tuning, as overlooking it can lead to attribute access errors.

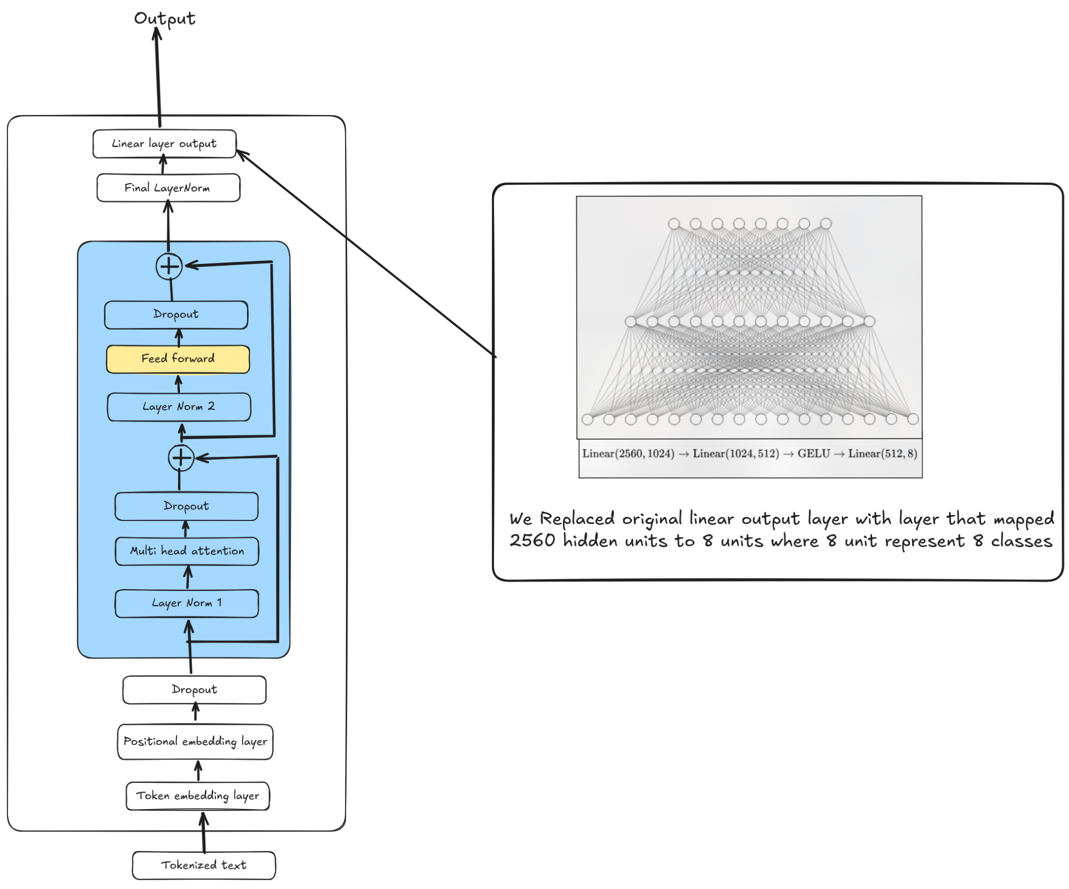
Output
Linear layer output
Final LayerNorm
Dropout
Feed forward
Layer Norm 2
Dropout
Multi head attention
Layer Norm 1
Dropout
Positional embedding layer
Token embedding layer
Tokenized text
Linear(2560, 1024) → Linear(1024, 512) → GELU → Linear(512, 8)
We Replaced original linear output layer with layer that mapped 2560 hidden units to 8 units where 8 unit represent 8 classes

Figure 1: Architecture of the custom classification head attached to the Gemma model.

Algorithm 1 Gemma-1B Architecture for Text Classification

```
Require: Input tokens $X_{\text{tokens}}$, target classes $C = \{c_1, c_2, \dots, c_8\}$
Ensure: Classification probabilities $P_{\text{class}} \in \mathbb{R}^8$

function GEMMA1B-FORWARD($X_{\text{tokens}}$)
    Step 1: Token Embedding
    $H_0 \leftarrow \text{EmbeddingLookup}(X_{\text{tokens}})$                ▷ $H_0 \in \mathbb{R}^{b \times s \times d}$

    Step 2: Transformer Backbone Processing
    for $l \leftarrow 1$ to $L$ do                                            ▷ $L$ transformer layers
        $H_l \leftarrow \text{TransformerBlock}_l(H_{l-1})$
            ▷ Uses Gemma3RMSNorm and Gemma3RotaryEmbedding
    end for

    Step 3: Final Normalization
    $H_{\text{final}} \leftarrow \text{Gemma3RMSNorm}(H_L)$                   ▷ $\epsilon = 10^{-6}$

    Step 4: Extract Last Token Representation
    $h_{\text{last}} \leftarrow H_{\text{final}}[:, -1, :]$                  ▷ Last token: $\mathbb{R}^{b \times 1152}$

    Step 5: Classification Head
    $z \leftarrow \text{LayerNorm}(h_{\text{last}})$                          ▷ $\epsilon = 10^{-5}$, affine=True
    $z \leftarrow \text{Dropout}(z, p = 0.4)$
    logits $\leftarrow W_{\text{class}} \cdot z + b_{\text{class}}$            ▷ $W_{\text{class}} \in \mathbb{R}^{8 \times 1152}$
    $P_{\text{class}} \leftarrow \text{softmax}(\text{logits})$

    return $P_{\text{class}}$
end function
```

Algorithm 2 Model Architecture Specifications

```
Core Components:

norm: Gemma3RMSNorm((1152,), eps=1e-06)
rotary_cmb: Gemma3RotaryEmbedding()
rotary_local: Gemma3RotaryEmbedding()
lm_head: Linear(1152 → 262144, bias=False)                    ▷ Original head
out_head: Sequential(                                          ▷ Custom classification head
  LayerNorm((1152,), eps=1e-05, elementwise_affine=True)
  Dropout(p=0.4, inplace=False)
  Linear(1152 → 8, bias=True)
)

Architecture Variants:

function AccessGemma1B
    model.lm_head                                              ▷ Direct access
    model.layers[-1]
    model.norm
end function
function AccessGemma4B
    model.model.lm_head                                        ▷ Wrapped access
    model.model.layers[-1]
    model.model.norm
end function
```

Algorithm 3 Adaptation from Generative to Discriminative Task

```
Input: Pretrained Gemma-1B model
Output: Fine-tuned classification model

Step 1: Model Selection
Select Gemma-1B (1B params) over larger variants for computational efficiency
Preserve pretrained representations while enabling fine-tuning on limited hardware

Step 2: Output Layer Replacement
Remove original: Linear$(1152 \to 262144)$                     ▷ Vocabulary projection
Add custom: Sequential$(1152 \to 8)$                           ▷ 8-class classification

Step 3: Architectural Preservation
Maintain transformer backbone: Gemma3RMSNorm, Gemma3RotaryEmbedding
Keep internal representations: $d_{\text{model}} = 1152$
Utilize final hidden states for classification

Step 4: Model-Specific Access Patterns
if using Gemma-1B then
    Access components directly: model.component
else
    Access through wrapper: model.model.component
end if

Result: Adapted model capable of hierarchical opinion classification
```

8. Training Setup

8.1. Dataset Preparation

The dataset consists of user-generated text entries, each annotated using a three-level hierarchical label scheme. Level 1 contains the broad categories: *NOISE*, *OBJECTIVE*, and *SUBJECTIVE*. For entries labeled as *SUBJECTIVE*, Level 2 further assigns *NEUTRAL*, *POSITIVE*, or *NEGATIVE*. Finally, for *NEUTRAL* instances at Level 2, Level 3 provides more fine-grained labels: *QUESTIONS*, *NEUTRAL SENTIMENTS*, *ADVERTISEMENTS*, and *MISCELLANEOUS*.

To simplify the classification process, we collapsed the original 10-class hierarchy into a flat 8-class label space, as shown in Table 2. This enables a single-step classification while preserving the hierarchical semantics of the original taxonomy.

To mitigate class imbalance, we computed class-specific weights based on inverse class frequency, as shown below (Algorithm 4). These weights were later used in the weighted cross-entropy loss function to penalize misclassification of minority classes more heavily.

Algorithm 4 Class Weight Computation

```
1: Class penalty values: $[225, 175, 80, 130, 175, 900, 70, 30]$
2: Normalize to sum 1 and scale by number of classes
3: Implemented in PyTorch as:
       label_penalty = [225,175,80,130,175,900,70,30]
       label_tens = torch.tensor(label_penalty, dtype=torch.float)
       weights = 1 / label_tens
       weights = (weights / weights.sum()) * len(label_penalty)
```

8.2. Preprocessing Steps

Algorithm 5 Data Preprocessing Pipeline

```
Require: Raw dataset $\mathcal{D}_{raw} = \{(x_i, y_i)\}_{i=1}^{N}$
Ensure: Cleaned dataset $\mathcal{D}_{clean}$
 1: Initialize $\mathcal{D}_{clean} \leftarrow [\,]$
 2: for each $(x_i, y_i)$ in $\mathcal{D}_{raw}$ do
 3:     if $x_i$ is empty or duplicate then
 4:         continue                                   ▷ Remove empty or repeated entries
 5:     end if
 6:     if $x_i$ contains only special characters and $y_i \neq$ Noise then
 7:         continue                                   ▷ Remove non-linguistic content
 8:     end if
 9:     $t_i \leftarrow$ TOKENIZE($x_i$) using Gemma-3 tokenizer
10:     if $|t_i| > 2000$ then
11:         continue                                   ▷ Discard extremely long samples
12:     else if $|t_i| < 40$ and isGibberish($x_i$) and $y_i \neq$ Noise then
13:         continue                                   ▷ Remove low-information gibberish
14:     end if
15:     Append $(x_i, y_i)$ to $\mathcal{D}_{clean}$
16: end for
17: Randomly split $\mathcal{D}_{clean}$ into:
18:     $\mathcal{D}_{train}$ (70%), $\mathcal{D}_{val}$ (20%), $\mathcal{D}_{test}$ (10%)
19: return $\mathcal{D}_{clean}$
```

We applied a multi-stage preprocessing pipeline to improve data quality (Algorithm 5). The raw dataset contained substantial noise from social media sources, including duplicated entries, incomplete texts, and non-linguistic symbols. To address this, we performed systematic cleaning, filtering, and consistency checks before model ingestion. Specifically, we removed duplicates, empty texts, and entries containing only special characters (unless labeled as *Noise*). We then tokenized using the Gemma-3 tokenizer and discarded extremely long ($>$2000 tokens) or unrealistically short ($<$40 tokens) entries with gibberish content (if not *Noise*). Finally, the cleaned dataset was split into training (70%), validation (20%), and test (10%) subsets.

8.3. Model Training

All experiments were implemented using the PyTorch deep learning framework [14]. We fine-tuned the pretrained Gemma model using the AdamW optimizer [13] with a fixed learning rate and weight decay to prevent overfitting. Gradient clipping was applied during backpropagation to ensure stable convergence.

Rather than updating all model parameters, we adopted a parameter-efficient fine-tuning (PEFT) strategy [15]. Specifically, we froze the early transformer layers of the pretrained LLM and only trained the final transformer block, the layer normalization modules, and a custom classification head. This selective layer unfreezing significantly reduced the number of trainable parameters while retaining the model's representational capacity.

To address class imbalance, we used a weighted cross-entropy loss function [16], where class weights were derived from inverse class frequencies (Algorithm 4). This encouraged the model to pay more attention to minority classes and reduced model bias toward dominant labels.

Training was performed using mini-batches of fixed size, and the model was evaluated on the validation set after each epoch. The best-performing checkpoint was selected based on the validation macro-F1 score to ensure balanced performance across all classes. All experiments were conducted on a single GPU, and identical hyperparameters were maintained across runs for fairness and reproducibility.

9. Data Processing

Following structural and semantic cleaning, the datasets were transformed into model-compatible formats for fine-tuning the Gemma architecture. This phase focused on tokenization, sequence length standardization, label encoding, and dataset splitting for training and evaluation.

9.1. Phase 1: Tokenization and Sequence Preparation

The cleaned textual corpus was tokenized using Gemma's native tokenizer to ensure compatibility with the pretrained embedding space.

- **Step 1.1: Model-Compatible Tokenization.**
  Each sentence was tokenized using the `AutoTokenizer.from_pretrained` utility.
- **Step 1.2: Maximum-Length Padding.**
  Instead of truncating sequences, the maximum token length across the dataset was computed, and shorter sequences were padded to this length. This approach ensures that all tokens are retained while maintaining a uniform input size for the model.

9.2. Phase 2: Label Encoding and Dataset Structuring

Categorical labels were mapped to numerical identifiers and the dataset was organized for supervised learning.

- **Step 2.1: Label Indexing.**
  Hierarchical labels were mapped to integer identifiers (0–7), corresponding to the eight distinct opinion categories defined in the dataset.

- **Step 2.2: Dataset Partitioning.**
  The dataset was split into training (70%), validation (20%), and test (10%) subsets using a stratified sampling approach to preserve class distribution. Prior to training, the data were shuffled randomly to avoid ordering biases.

9.3. Phase 3: Data Loading for Training

The processed datasets were wrapped into PyTorch `DataLoader` objects for efficient access during fine-tuning.

- **Step 3.1: DataLoader Configuration.**
  Training, validation, and test sets were loaded with an appropriate batch size. Data shuffling occurred only once before training, ensuring reproducibility and balanced exposure of samples without dynamic shuffling at each epoch.

9.4. Data Processing Instruction Fine-Tuning

For instruction fine-tuning, the model is trained to generate a target output conditioned on a given prompt and contextual input. This requires precise alignment between the input token sequence and the corresponding target labels. The methodology implemented in this work is as follows:

- **Step 4.1: Tokenization of Prompt-Response Pairs.**
  Each sample, consisting of a concatenated prompt and input text (e.g., `Text + Comment`), was tokenized using Gemma's tokenizer. This produced the input token IDs (`input_ids`) and attention masks (`input_mask`) required for model consumption.
- **Step 4.2: Sequence Shifting for Autoregressive Learning.**
  To enable autoregressive training, the tokenized sequence was shifted by one position to produce the target tensor (`target_ids`). Formally, if the original token sequence is

  $$X = [x_0, x_1, \dots, x_{N-1}]$$

  the shifted target sequence is defined as

  $$Y = [x_1, x_2, \dots, x_N].$$

  This shifting ensures that the model predicts the next token at each time step, thereby learning to generate the label conditioned on the preceding tokens, including both prompt and input text.
- **Step 4.3: Masking Non-Label Tokens.**
  Since the objective is to compute the loss only on the label portion (e.g., the `relevance` or answer tokens), a masking tensor was constructed. For each sequence:

  1. The position of the label tokens within the shifted sequence was identified by comparing the target tensor with the tokenized label.
  2. All positions outside the label were set to `-100`, which is the standard ignore index in PyTorch's `CrossEntropyLoss`.
  3. Positions corresponding to the label remained unmasked, ensuring that the loss is computed exclusively on the answer tokens.

  This selective masking prevents the model from backpropagating errors over the prompt or context tokens, focusing learning exclusively on the label generation.
- **Step 4.4: Verification of Label Alignment.**
  To ensure correctness, each label token sequence was compared with the corresponding segment in the shifted target tensor. Only sequences with an exact match were retained, and the mask was applied accordingly. Any mismatches were flagged for inspection to guarantee precise supervision.

This approach enables instruction fine-tuning in a controlled manner, ensuring that the model:

1. Learns to predict the target label conditioned on the full prompt and input text.
2. Receives gradient updates only for the label tokens, avoiding spurious updates on non-informative parts of the input.
3. Maintains the autoregressive property of the LLM, making it compatible with standard causal language modeling objectives.

9.5. Loss Function

Given the nature of the task—single-label, multi-class classification with eight classes and significant class imbalance—we employed the **Cross-Entropy Loss**. This choice was motivated by:

1. Its suitability for multi-class classification tasks.
2. Its support for class weighting, which is essential for imbalanced datasets.

Handling Class Imbalance: To mitigate imbalance, we computed **inverse-frequency class weights** based on the number of samples per class. These weights were incorporated into the Cross-Entropy Loss to penalize misclassifications of minority classes more heavily.

Algorithm 6 Class Weight Computation

```
1: Class penalty values based on inverse population frequency:
2: weights ← [225, 175, 80, 130, 175, 900, 70, 30]
3: weights ← weights/sum(weights)                    ▷ Normalize to sum 1
4: weights ← weights × 8                             ▷ Scale by number of classes
```

10. Optimizer and Scheduler Configuration

To ensure stable convergence during fine-tuning, optimization and learning rate scheduling were carefully configured. The **AdamW** optimizer [13] was employed, which decouples weight decay from the gradient-based parameter updates, providing better generalization compared to conventional Adam. The learning rate was set to 5×10^{-5} with a weight decay of 0.1 to prevent overfitting and stabilize training dynamics.

A **cosine learning rate schedule with warmup** was adopted to further enhance training stability. Specifically, the learning rate was gradually increased during the initial 15% of training steps (warmup phase), followed by a smooth cosine decay over the remaining steps. This schedule helps the model transition from the pretrained weights to the downstream classification objective without abrupt parameter shifts.

To prevent exploding gradients, gradient norms were clipped to a maximum value of 1.0 using `torch.nn.utils.clip_grad_norm_()`. The total training was conducted for **three epochs**, with a 70–20–10 train–validation–test split. Data shuffling was applied once prior to training to ensure randomized sample distribution, while maintaining deterministic ordering during the training iterations.

Table 10
AdamW optimizer hyperparameters

Parameter	**Value**
Learning rate	5×10^{-5}
Weight decay	0.1
Scheduler	Cosine with warmup

This configuration ensured smooth optimization, effective regularization, and consistent convergence across fine-tuning runs, particularly under hardware constraints associated with quantized and partially unfrozen models.

11. Trainable Parameters

To balance efficiency with performance, we adopted **parameter-efficient fine-tuning** [15] , training only a subset of the model's parameters while keeping the rest frozen. This reduces overfitting risks and lowers computational cost. The configuration is summarized in . The fine-tuning configuration was designed to enable targeted learning on the downstream classification task while maintaining the stability of the pretrained layers.

- The **last transformer block** was unfrozen, allowing adaptation of the model's highest-level contextual representations to the task-specific semantic distribution.
- The **final normalization layer (LayerNorm)** was made trainable to recalibrate hidden state activations after fine-tuning adjustments.
- The **language modeling head (lm_head)** was retained as trainable, ensuring that adapted internal embeddings align effectively with the output representation space.
- A **custom classification head** was attached to the final hidden representation of the model, responsible for mapping the 2056-dimensional feature vector to the eight output categories. The architecture of this head is defined as:

$$\text{LayerNorm}(2056) \rightarrow \text{Linear}(2056, 1024) \rightarrow \text{GELU} \rightarrow \text{Linear}(1024, 512) \rightarrow \text{GELU} \rightarrow \text{Linear}(512, 8)$$

All remaining transformer blocks were frozen, preserving the pretrained linguistic priors and semantic knowledge embedded within the model. This selective fine-tuning approach substantially reduced the number of trainable parameters while maintaining sufficient representational flexibility for effective adaptation to the hierarchical opinion classification task. frozen and unfrizen parameter given in Table 11

Table 11
Trainable vs frozen parameters in Gemma model fine-tuning

Trainable	Frozen
Last Transformer block	Earlier Transformer blocks
Final LayerNorm	-
LM head	-
Custom classification head	-

12. Problems and Fixes

During the course of training and experimentation, several challenges were encountered which significantly impacted stability, memory usage, and generalization of the model. Below, we describe each issue in detail along with the remedies applied.

12.1. Gradient Explosion During Early Training

One of the recurring problems was *gradient explosion* [16] , where the magnitude of the gradients grew uncontrollably during backpropagation. This led to unstable weight updates, often producing NaN values in the output tensors. Gradient explosion is a well-known issue in deep learning, particularly in transformer-based models where the depth of the network and large learning rates can amplify unstable updates.

Fixes:

- **Gradient Clipping:** We applied gradient clipping with a maximum norm of 1.0, which constrains the gradients from exceeding a predefined threshold, ensuring stable weight updates.

- **Precision Adjustment:** Instead of using `float32` or `float16`, we shifted to `torch.bfloat16`. The `bfloat16` format offers a wider dynamic range compared to `float16`, improving numerical stability while still reducing memory consumption relative to `float32`.

12.2. Memory Overflow During Training

When training with `float32` precision, the model consistently exceeded the available GPU memory. This problem was aggravated by the relatively large sequence lengths in our dataset and the quadratic memory requirement of self-attention.

Fixes:

- **Batch Size Reduction:** We reduced the training batch size from 8 to 4. This change effectively lowered peak GPU memory usage per step, enabling stable training without out-of-memory (OOM) errors.

12.3. Overfitting to Majority Classes

Another critical issue was the model's tendency to overfit to high-frequency classes such as *Objective* and *Noise*. While accuracy appeared high, minority classes (*Miscellaneous*, *Advertisements*, *Questions*) received near-zero F1 scores. This imbalance reflects the skewed label distribution in the dataset, where rare categories are underrepresented.

Fixes:

- **Weighted Cross-Entropy Loss:** We computed class weights inversely proportional to class frequencies. These weights were integrated into the cross-entropy loss, penalizing misclassification of rare categories more heavily and forcing the model to learn discriminative features for minority classes.

12.4. Stagnant Accuracy During Training

At certain stages of training, model accuracy plateaued around 50%, showing no significant improvement across epochs. This stagnation suggested that the optimization landscape was poorly conditioned, and the model was unable to escape local minima or saddle points.

Fixes:

- **Layer Normalization:** Added layer normalization to stabilize the distribution of activations, improving convergence.
- **Dropout:** Introduced dropout layers to reduce overfitting by preventing co-adaptation of neurons.
- **Activation Functions:** Adjusted activation functions to ensure smoother gradients and mitigate vanishing/exploding gradient issues.

13. Results and Analysis

We evaluate two distinct experimental settings, each designed to explore different aspects of adapting Large Language Models (LLMs) for downstream tasks.

Run 1: Classification Fine-tuning. We directly fine-tune the pretrained **Gemma-1B** model for hierarchical text classification. A custom classification head attached to the final transformer block maps hidden representations to label probabilities. This setup evaluates the model's ability to perform end-to-end supervised classification without any task reformulation.

Run 2: Instruction Fine-tuning. This setup is independent of classification fine-tuning. The model is trained in an instruction-following format, where each example is a prompt–response pair. The loss is computed via next-token prediction by shifting target tensors by one position and applying masking to focus only on answer tokens. This setting measures the model's alignment with instruction-based reasoning.

We evaluate both approaches across three social media platforms — **Reddit**, **Twitter**, **YouTube** and **Qna**— each containing 500 randomly sampled posts. The hierarchical label taxonomy consists of three levels:

Table 12
Run 1: Class Distribution Across Reddit, Twitter, and YouTube (500 samples each).

Dataset	Level	Class	Count (%)
	Level 1	Subjective	207 (41.4)
		Objective	158 (31.6)
		Noise	135 (27.0)
Reddit	Level 2 (Subj.)	Neutral	122 (58.9)
		Negative	46 (22.2)
		Positive	39 (18.8)
	Level 3 (Neut.)	Question	70 (57.4)
	Level 1	Noise	322 (64.4)
		Objective	91 (18.2)
		Subjective	87 (17.4)
Twitter	Level 2 (Subj.)	Neutral	51 (58.6)
		Positive	24 (27.6)
		Negative	12 (13.8)
	Level 3 (Neut.)	Advertisement	38 (74.5)
	Level 1	Subjective	394 (78.8)
		Noise	102 (20.4)
		Objective	4 (0.8)
YouTube	Level 2 (Subj.)	Neutral	218 (55.3)
		Negative	153 (38.8)
		Positive	23 (5.8)
	Level 3 (Neut.)	Neutral Sent.	118 (54.1)

Table 13
Run 2: Class Distribution Across Reddit, Twitter, and YouTube (500 samples each).

Dataset	Level	Class	Count (%)
	Level 1	Objective	223 (44.6)
		Noise	141 (28.2)
		Subjective	136 (27.2)
Reddit	Level 2 (Subj.)	Neutral	93 (68.4)
		Positive	32 (23.5)
		Negative	11 (8.1)
	Level 3 (Neut.)	Question	74 (79.6)
	Level 1	Noise	233 (46.6)
		Subjective	178 (35.6)
		Objective	89 (17.8)
Twitter	Level 2 (Subj.)	Neutral	110 (61.8)
		Positive	34 (19.1)
		Negative	34 (19.1)
	Level 3 (Neut.)	Advertisement	62 (56.4)
	Level 1	Subjective	394 (78.8)
		Noise	106 (21.2)
	Level 2 (Subj.)	Neutral	203 (51.5)
YouTube		Negative	189 (48.0)
		Positive	2 (0.5)
	Level 3 (Neut.)	Neutral Sent.	119 (58.6)

Table 14
QnA Dataset Class Distribution (6,323 samples).

Run	Class	Count (%)
Run 1	Class 0	6,307 (99.75)
Run 1	Class 1	16 (0.25)
Run 2	Class 0	5,761 (91.11)
Run 2	Class 1	562 (8.89)

13.1. Comparison and Insights

Compared to Run 1, instruction fine-tuning (Run 2) increased coverage of minority categories such as *Questions* and *Advertisements*, while reducing the dominance of *Noise*. In QnA, the minority Class 1 proportion rose from 0.25% to 8.89%, highlighting the benefit of instruction tuning for balancing skewed datasets.

14. Conclusion

In this work, we explored the task of hierarchical opinion classification by adapting Large Language Models (LLMs) to an 8-class text classification problem derived from a 3-level dataset. We designed and implemented a parameter-efficient fine-tuning approach using the Gemma model, attaching a custom classification head and selectively training higher layers to handle limited compute resources.

Our experiments highlighted both the challenges and opportunities of fine-tuning LLMs for imbalanced datasets. Issues such as gradient explosion, memory overflow, and overfitting to majority classes were systematically identified and addressed through techniques like gradient clipping, precision adjustments, weighted loss functions, and dropout regularization. We further compared direct classification fine-tuning with instruction tuning, demonstrating that instruction-based training improved coverage of minority classes and enhanced generalization across datasets.

Overall, the study underscores the importance of carefully designed preprocessing, balanced training strategies, and efficient fine-tuning in achieving reliable performance for domain-specific classification tasks. Future work may focus on extending the approach to larger Gemma variants, experimenting with more advanced imbalance-handling methods, and evaluating real-world deployment scenarios.

15. Future Works

This study introduces a hierarchical text classification framework that effectively sorts text into multiple levels of labels. Based on this solid groundwork, there are several promising avenues for future research to explore in the coming stages:

- **Cross-Domain and Cross-Lingual Generalization:** The proposed framework can be further extended to test its effectiveness across different domains, such as news, product reviews, and social media, or even across languages. These broader studies would provide deeper insights into how hierarchical sentiment and intent categories transfer across diverse contexts, uncovering potential challenges and encouraging the design of models that can handle domain shifts or subtle cross-lingual differences effectively.
- **Hierarchical Label Dependency Modeling:** Currently, hierarchical levels are modeled in sequence, but without explicitly maintaining consistency or relationships between levels. Future research could look into structured prediction methods, probabilistic graphical models, or neural architectures specifically aimed at capturing these label dependencies, ensuring that Level 2 and Level 3 predictions stay logically consistent with Level 1 outcomes. These enhancements could significantly improve both the accuracy and reliability of hierarchical classifications.

- **Robustness to Noisy and Imbalanced Data:** Since social media and real-world text often contain a lot of noise, lack context, and exhibit strong class imbalance, future work could explore self-supervised pretraining, data augmentation, semi-supervised learning, or techniques that account for uncertainty to increase robustness. Additionally, assessing model performance under controlled changes could help pinpoint key weaknesses and encourage specific improvements in data management and model design.
- **Temporal and Emergent Patterns in Text:** Hierarchical labels can also expose interesting time-related patterns in sentiment, questions, and advertisements over long periods. Future research could use this classification framework to investigate emerging trends in public opinion, the spread of misinformation, or the dynamics of online conversations. These long-term analyses could yield valuable insights that go beyond traditional static classification metrics.
- **Refinement of Evaluation Metrics:** Lastly, standard evaluation metrics such as accuracy or F1-score may not fully reflect the hierarchical or semantic structure of model predictions. Future work could focus on creating or adjusting metrics that consider hierarchical consistency, partial correctness, or semantic similarity, offering a more detailed and realistic assessment of overall model performance.

Acknowledgments

The authors acknowledge and thank Dr. Dwaipayan Roy from Department of Computational and Data Sciences (CDS), Indian Institute of Science Education and Research - Kolkata, for providing the computing resources for our finetuning runs.

Declaration on Generative AI

During the preparation of this work, the author(s) used OpenAI's Chat-GPT-5 and Google's Gemini 2.5 Pro in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] G. Team, Gemma 3 (2025). URL: https://arxiv.org/abs/2503.19786.

[2] S. Adhikary, S. Banerji Seal, S. Sar, D. Roy, IISERK@ToT_2024: Query reformulation and layered retrieval for tip-of-tongue items, in: Proceedings of the Thirty-Third Text REtrieval Conference (TREC 2024), National Institute of Standards and Technology (NIST), 2024. URL: https://trec.nist.gov/pubs/trec33/papers/IISER-K.tot.pdf.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5998–6008.

[4] N. Kitaev, Ł. Kaiser, A. Levskaya, Reformer: The efficient transformer, in: International Conference on Learning Representations (ICLR), 2020.

[5] I. Beltagy, M. E. Peters, A. Cohan, Longformer: The long-document transformer, arXiv preprint arXiv:2004.05150 (2020).

[6] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2019, pp. 4171–4186.

[7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, RoBERTa: A robustly optimized BERT pretraining approach, arXiv preprint arXiv:1907.11692 (2019).

[8] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, Journal of Machine Learning Research 21 (2020) 1–67.

[9] J. Wei, M. Bosma, V. Y. Zhao, K. Gu, A. W. Yu, B. Lester, N. Du, A. M. Dai, Q. V. Le, Finetuned language models are zero-shot learners, arXiv preprint arXiv:2109.01652 (2021).

[10] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, R. Lowe, Training language models to follow instructions with human feedback, in: Advances in Neural Information Processing Systems (NeurIPS), 2022.

[11] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, H. Hajishirzi, Self-instruct: Aligning language model with self generated instructions, arXiv preprint arXiv:2212.10560 (2022).

[12] T. Dettmers, M. Lewis, Y. Belkada, A. H. Zadeh, LLM.int8(): 8-bit matrix multiplication for transformers at scale, in: Proceedings of the 6th Workshop on Scalable Cloud Data Management, 2022.

[13] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101 (2017).

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems (NeurIPS), 2019, pp. 8026–8037.

[15] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, B. Bossan, et al., PEFT: State-of-the-art parameter-efficient fine-tuning methods, https://github.com/huggingface/peft, 2022.

[16] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

A. Source Code:

All the codes are available at https://github.com/Shuvam-Banerji-Seal/FIRE_2025_Hierarchical_Opinion_Classification_using_Large_Language_Models.git