Classifying COVID-19 Tweets Using Transformer Models and Compression Techniques

Gal Gussarsky Adi Shalit

August 2025

Abstract

The COVID-19 pandemic triggered a massive surge in online discourse, particularly on social media platforms such as Twitter. Classifying these posts into sentiment categories provides valuable insights for public opinion analysis, crisis communication, and policymaking. In this work, we study sentiment classification on the COVID-19 NLP dataset from Kaggle, which contains over 45k labeled tweets spanning five sentiment classes. We combine rigorous preprocessing and translation steps with fine tuning of transformer based models (RoBERTa-base and DeBERTa-v3-base) under two training frameworks: a custom PyTorch loop and the Hugging Face Trainer API. Hyperparameters were optimized via Optuna, and training dynamics monitored with Weights & Biases. We further explored unsupervised ensembles for robustness and investigated model compression strategies quantization, pruning, and knowledge distillation to balance predictive performance with size and latency. Our findings show that DeBERTa-v3 consistently outperforms RoBERTa, that ensembles deliver modest but reliable gains, and that knowledge distillation yields the most practical efficiency performance trade-off compared to quantization and pruning. This pipeline demonstrates both methodological rigor and deployment-aware evaluation for real-world NLP applications.

You can find the full project repository here:
`GitHub Repository - adv-dl-project_Group_W`

1 Introduction

The COVID-19 pandemic has highlighted the crucial role of social media as a real-time channel for information exchange, opinion sharing, and public sentiment. Twitter, in particular, became a central space where individuals expressed their views, fears, and expectations regarding the evolving crisis. Analyzing such discourse through sentiment classification not only provides insights into collective attitudes but also supports governments, organizations, and researchers in understanding the dynamics of public response.

The task, however, is far from trivial. Tweets are short, noisy, and often multilingual, with informal grammar, abbreviations, and hashtags. These challenges demand sophisticated natural language processing (NLP) methods that go beyond traditional bag-of-words or shallow models. Transformer-based architectures such as RoBERTa and DeBERTa have emerged as state-of-the-art for text classification, leveraging large-scale pretraining and contextual embeddings. At the same time, practical deployment requires addressing efficiency concerns, as transformer models are computationally expensive.

In this project, we tackle sentiment classification on the Kaggle COVID-19 NLP dataset (45k labeled tweets across five sentiment categories). We adopt a comprehensive pipeline covering three stages. First, we perform exploratory data analysis (EDA) and preprocessing, including cleaning, translation of non-English samples, and careful treatment of usernames and hashtags. Second, we fine-tune RoBERTa-base and DeBERTa-v3-base under two training paradigms: a flexible custom PyTorch loop and the Hugging Face Trainer API. Hyperparameter search with Optuna and monitoring via Weights & Biases guide our optimization. Third, we evaluate ensemble methods and apply model compression techniques—quantization, pruning, and knowledge distillation—to study the trade-off between accuracy, efficiency, and deployability.

Our contributions are threefold: (i) a systematic evaluation of transformer fine-tuning frameworks on COVID-19 tweet sentiment classification, (ii) an analysis of ensemble and compression techniques in balancing performance and efficiency, and (iii) a comparison of RoBERTa versus DeBERTa under identical experimental settings. Together, these results provide both methodological insights and practical lessons for sentiment classification in resource-constrained real-world applications.

2 Exploratory Data Analysis

The dataset used in this project is the *COVID-19 NLP Text Classification* dataset from Kaggle, which contains 41,157 training tweets and 3,798 test tweets. Each record includes six fields: `UserName`, `ScreenName`, `Location`, `TweetAt`, `OriginalTweet`, and `Sentiment`. The target label `Sentiment` has five categories: *Extremely Negative*, *Negative*, *Neutral*, *Positive*, and *Extremely Positive*.

After inspection, we found that `UserName` and `ScreenName` are unique identifiers with no predictive value, while `Location` suffers from a high proportion of missing entries (21% in train, 7% in test). Therefore, our analysis and modeling focus exclusively on `OriginalTweet`.

2.1 Data Characteristics

We first examined the lengths of tweets to ensure the dataset is consistent and easy to handle. Figures 1 and 2 show the distribution of tweet lengths in the train and test sets. Most tweets range between 100–250 characters, with a sharp peak near 260, and no tweets exceed 350 characters. The similarity between train and test confirms consistent sampling.

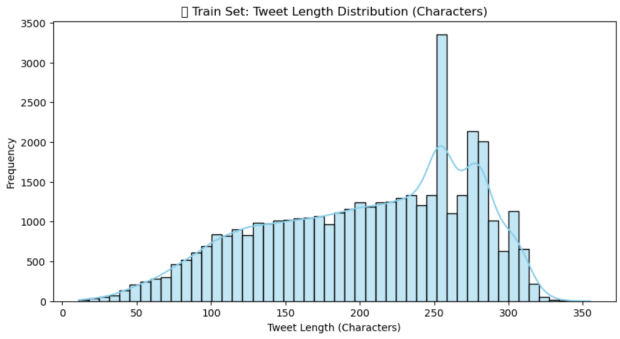

Figure 1: Tweet length distribution (train set).

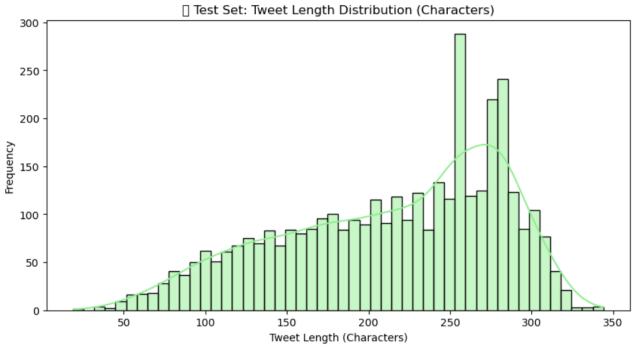

Figure 2: Tweet length distribution (test set).

Next, we looked at the label distribution to verify balance between sentiment classes. The dataset is reasonably balanced across the five categories, ensuring that models are not severely biased toward a dominant class (see Figure 3). We also explored whether simple signals in the raw text already provide predictive information. On the one hand, Figure 4 shows that tweet length varies systematically by sentiment, with *Extremely Negative* and *Extremely Positive* tweets tending to be longer than Neutral ones. On the other hand, Figure 5 illustrates that word usage also differs clearly between classes: terms such as *terroristic*, *ruin*, and *fury* are strongly associated with the *Extremely Negative* sentiment. Together, these observations confirm that the `OriginalTweet` column contains clear sentiment-related signals, even before applying advanced NLP models.

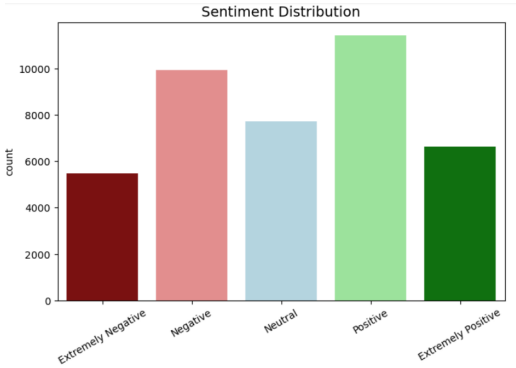

Figure 3: Distribution of sentiment labels.

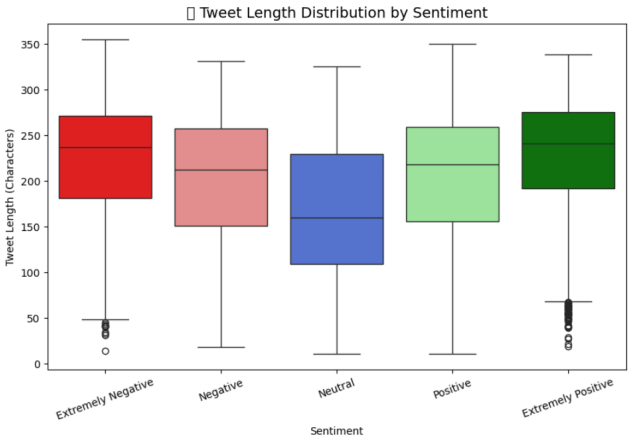

Figure 4: Tweet length distribution by sentiment.

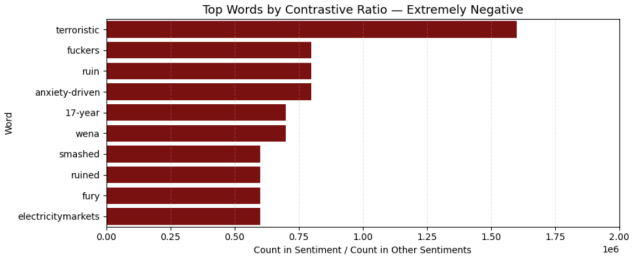

Figure 5: Top words distinctive to the *Extremely Negative* sentiment.

In addition, we examined the occurrence of usernames within tweets. Many usernames repeat frequently and are semantically meaningful, often referring to political leaders (`@realDonaldTrump`, `@BorisJohnson`), organizations (`@WHO`, `@CDCgov`), or retailers (`@Tesco`, `@Amazon`). This indicates that usernames themselves may carry sentiment-relevant signals, motivating our decision to retain them during preprocessing rather than discarding them as noise. We also observed that the dataset contains some non-English or partially non-English tweets. Although these represent only a small fraction of the data and do not affect the overall class balance, they could introduce noise if left unaddressed. This observation motivates additional preprocessing steps to ensure language consistency, which are described in the next section.

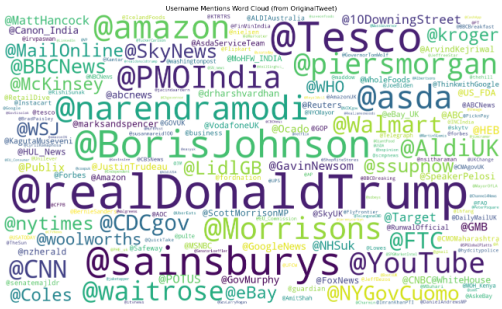

Figure 6: Frequent usernames cloud.

3 Data Processing and Translation

First, we removed URLs, HTML artifacts, and excessive whitespace from the `OriginalTweet` field. Hashtags were stripped of the '#' symbol while keeping the word itself, so that topical information was preserved rather than discarded. Usernames, on the other hand, were retained because they act as unique identifiers (`@realDonaldTrump`, `@Amazon`) that carry distinct semantic signals. Unlike hashtags, which often overlap in meaning, each username refers to a specific entity and can therefore be treated as a meaningful token.

Next, we addressed language consistency. A small fraction of tweets were non-English or partially non-English. While these did not significantly affect dataset balance, they could introduce noise if ignored. We therefore ran automatic language detection (`langid`), estimating the probability of English versus non-English text. Tweets confidently detected as non-English were translated into English using the `facebook/m2m100_418M` multilingual model. In rare cases where translation was

not feasible, only a few isolated samples from the training set were removed. This ensured that both train and test sets were linguistically consistent.

We also considered using the `Location` field as an additional token, but it is incomplete (21% missing in train, 7% in test) and inconsistent, mixing countries, states, cities, and ambiguous phrases. This heterogeneity and sparsity would likely add noise rather than signal, so we excluded it.

4 Model Training

4.1 Training Frameworks and Model Architectures

We trained both RoBERTa and DeBERTa models using two complementary frameworks: a full PyTorch loop (Exercise 4) and the HF Trainer API (Exercise 5). For each model, hyperparameter tuning was performed with Optuna, while training dynamics were monitored via Weights & Biases (W&B). The overall strategy was iterative: starting from broad search ranges, narrowing them based on validation trends, and settling on final configurations for full training.

The two training procedures differed mainly in flexibility and automation. The **full PyTorch loop** allowed fine-grained control over optimization, gradient clipping, learning-rate scheduling, and custom logging, at the cost of more boilerplate code. In contrast, the **HF Trainer API** provided a higher-level abstraction that integrated checkpointing, early stopping, and mixed-precision training more cleanly, while still allowing partial freezing of the encoder and unfreezing of the last k layers.

Regarding architectures, we selected **RoBERTa-base** as a strong and widely-used transformer baseline for sentence-level classification, and **DeBERTa-v3-base** as a more recent architecture incorporating disentangled attention and enhanced position encoding. RoBERTa (Liu et al., 2019) is a variant of BERT trained with larger batch sizes, longer sequences, and dynamic masking. DeBERTa (He et al., 2021) improves upon BERT and RoBERTa by disentangling content and position representations, which has been shown to enhance contextual understanding.

This setup allowed us to compare not only two training frameworks but also two transformer families with complementary modeling philosophies. Both models were trained in a five-class sentiment classification setting with a fixed label mapping, ensuring direct comparability of results.

4.2 Exercise 4: Full PyTorch Loop

4.2.1 RoBERTa (Full PyTorch Loop, Exercise 4)

The first round of Optuna tuning explored a wide space: learning rate $[10^{-6}, 5{\times}10^{-5}]$ (log-uniform), weight decay $[10^{-6}, 10^{-1}]$, patience $\{1, \ldots, 4\}$, epochs $\{4, \ldots, 12\}$, and number of unfrozen layers $\{1, \ldots, 6\}$. Twelve trials were run, with performance tracked in W&B.

Key observations. From these trials, two main patterns emerged:

Smaller learning rates were consistently more stable, while larger ones often diverged.
Validation F1 improved as more encoder layers were unfrozen.

A second Optuna study refined the search to: learning rate $[10^{-6}, 7{\times}10^{-5}]$, unfrozen layers $\{4, 5, 6\}$, batch size $\{4, 8, 16, 32, 64\}$, with epochs fixed to 12 and patience to 4. This confirmed the earlier trends and additionally showed that smaller batch sizes improved generalization.

The final model used: **learning rate** 4.3×10^{-5}**, weight decay** 6.3×10^{-6}**, batch size 4, 12 epochs, patience 4, and 6 unfrozen layers**. Within this range, the best results came from combining deeper fine-tuning with small batches. Since our tuning only allowed up to six unfrozen layers, these findings further motivate exploring fine-tuning of even more layers in subsequent setups.

4.2.2 DeBERTa (Full PyTorch Loop, Exercise 4)

For mDeBERTa-v3-base, the initial Optuna study explored: learning rate $[10^{-6}, 5 \times 10^{-5}]$, weight decay $[10^{-6}, 10^{-1}]$, batch size $\{4, 8, 16, 32, 64\}$, and unfreezing of the last $\{8, \ldots, 12\}$ layers, with epochs fixed to 12 and patience set to 4. Twenty trials were conducted.

Key observations (DeBERTa, Full PyTorch Loop):

Learning rate	10^{-5}–10^{-4} stable; larger diverged.
Weight decay	Values $\leq 10^{-4}$ outperformed higher ones.
Unfreezing	Best with 9–12 layers.
Batch size	8–16 most stable; larger showed no clear benefit.

The best configuration was: **learning rate** 3.5×10^{-5}**, weight decay** 9.4×10^{-5}**, batch size 8, 12 unfrozen layers, 12 epochs, and patience 4**, achieving a **validation macro F1 of 0.870**.

4.3 Exercise 5: HF Trainer API

4.3.1 RoBERTa (HF Trainer API, Exercise 5)

For Exercise 5 we used the HF Trainer API with Optuna and W&B logging. Compared to the full PyTorch loop, this framework allowed a cleaner integration of early stopping, checkpointing, and evaluation while keeping the same principle of freezing the encoder and unfreezing the last k layers. The initial Optuna study explored: learning rate $[10^{-6}, 3\times10^{-5}]$ (log-uniform), weight decay $[10^{-6}, 10^{-2}]$, batch size $\{4, 8, 16, 32\}$, and unfreezing of the last $\{5, \ldots, 12\}$ layers. Twelve trials were conducted, each training up to 12 epochs with early stopping after 4 stagnant validation evaluations.

The three best configurations were:

Set 1	learning rate 2.89×10^{-5}, weight decay 9.8×10^{-3}, batch size 16, 12 unfrozen layers.
Set 2	learning rate 2.96×10^{-5}, weight decay 8.8×10^{-3}, batch size 16, 12 unfrozen layers.
Set 3	learning rate 2.12×10^{-5}, weight decay 1.3×10^{-3}, batch size 16, 10 unfrozen layers.

Key observations (RoBERTa, HF Trainer API):

Learning rate	Near the upper bound ($\approx 3 \times 10^{-5}$) sometimes diverged; stable and performant runs clustered in the 2×10^{-5}–3×10^{-5} range.
Unfreezing	Deeper fine-tuning (10–12 layers unfrozen) consistently outperformed shallow updates.
Weight decay	High weight decay ($> 10^{-2}$) degraded performance; effective values clustered between 10^{-3} and 10^{-2}.

The chosen model was *set 2*, which achieved a macro F1 score of **0.84** on the held-out test set.

4.3.2 DeBERTa (HF Trainer API, Exercise 5)

For Exercise 5 we trained `microsoft/deberta-v3-base` using the HF `Trainer` API with Optuna and W&B logging. Two Optuna studies were conducted:

Study 1 (10 trials)	LR $[10^{-5}, 10^{-4}]$, WD $[10^{-6}, 10^{-4}]$, batch size $\{4, 8, 16, 32\}$, unfreezing $\{8, \ldots, 12\}$ layers.
Study 2 (15 trials)	LR $[7 \times 10^{-5}, 5 \times 10^{-4}]$, WD $[10^{-5}, 10^{-4}]$, batch size $\{8, 16, 32, 64, 128\}$, unfreezing $\{8, \ldots, 12\}$ layers.

The best configurations were:

Study 1, Trial 2	LR 9×10^{-5}, WD 2×10^{-6}, batch size 32, 12 unfrozen layers $\rightarrow$ macro F1 $= 0.852$.
Study 1, Trial 5	LR 9.5×10^{-5}, WD 2×10^{-5}, batch size 16, 11 unfrozen layers $\rightarrow$ macro F1 $= 0.852$.
Study 2, Trial 12	LR 1.0×10^{-4}, WD 5×10^{-5}, batch size 16, 12 unfrozen layers $\rightarrow$ macro F1 $= 0.859$.

Key observations (DeBERTa, HF Trainer API)

Unfreezing	Strongest results with 10–12 layers unfrozen.
Learning rate	Optimal between 8×10^{-5} and 1.0×10^{-4}; larger ($\geq 2 \times 10^{-4}$) collapsed.
Batch size	16 or 32 yielded most stable performance.

The final model (from Study 2, Trial 12) used **learning rate** 1.0×10^{-4}**, weight decay** 5×10^{-5}**, batch size 16, 12 unfrozen layers, and 15 epochs**, achieving a **validation macro F1 of 0.8673** and a **test macro F1 of 0.8476**.

5 Results

We summarize the best configurations and test performance for both RoBERTa-base and DeBERTa-v3-base, each trained under two frameworks (HF Trainer API / Exercise 5 and full PyTorch loop / Exercise 4).

Key observations

Model	Framework	Macro F1	ExtNeg	Neg	Neu	Pos	ExtPos
RoBERTa	HF Trainer API	0.78	0.80	0.75	0.79	0.74	0.82
RoBERTa	Full PyTorch Loop	0.84	0.85	0.82	0.86	0.82	0.85
DeBERTa	HF Trainer API	0.87	0.89	0.87	0.86	0.84	0.88
DeBERTa	Full PyTorch Loop	0.85	0.87	0.83	0.86	0.81	0.87

Table 1: Test set macro F1 and per-class F1 scores for both RoBERTa-base and DeBERTa-v3-base across both frameworks.

	Pred ExtNeg	Pred Neg	Pred Neu	Pred Pos	Pred ExtPos
True ExtNeg	538	50	2	2	0
True Neg	104	873	19	41	4
True Neu	5	64	506	42	2
True Pos	4	64	32	733	114
True ExtPos	1	1	0	44	553

Table 2: Confusion matrix for DeBERTa (HF Trainer API). Most errors occur between adjacent sentiment classes.

DeBERTa outperformed RoBERTa across both frameworks	Best performance from DeBERTa (HF Trainer API), macro F1 = 0.87.
Full PyTorch loop helped RoBERTa more	Gave a significant boost to RoBERTa; less impact on DeBERTa.
Extreme sentiment classes easier to classify	Stronger F1 scores for *extremely negative* and *extremely positive*.
Middle sentiment classes are hardest	Lower F1 for *negative* and *positive* classes across all models.
Errors mostly between adjacent classes	Confusion matrix shows predictions often close in sentiment intensity.

6 Model Compression

Motivation. Fine-tuned transformers achieve strong performance but remain heavy for deployment. Large parameter counts create latency and memory bottlenecks. Compression methods aim to reduce size and speed up inference while retaining accuracy.

6.1 Quantization

Quantization reduces weights and activations from FP32 to INT8, lowering memory use and often speeding up inference with limited accuracy loss. We applied post-training dynamic quantization to all models.

For RoBERTa (PyTorch loop and HF Trainer), performance dropped only slightly (macro F1: 0.779→0.754; 0.840→0.829), while size nearly halved (475→231 MB) and inference improved 1.36–1.46×. This illustrates the usual accuracy–efficiency trade-off, with RoBERTa robust to reduced precision.

DeBERTa, however, collapsed under full INT8 quantization. Selective quantization (keeping attention layers in FP32) preserved accuracy (macro F1 unchanged at 0.869/0.848) but increased size (524→804 MB; 531→838 MB) and slowed inference (0.70×). Mixed precision overhead canceled expected gains, showing DeBERTa's sensitivity can reverse quantization benefits.

6.2 Unstructured Pruning

Unstructured pruning zeros individual weights, yielding sparse matrices. We applied global L1 pruning (10–89%) and measured accuracy, F1, latency, and size. At 10–20% pruning, RoBERTa stayed near 0.77 macro-F1, and DeBERTa above 0.86. Beyond 30–40%, performance collapsed (F1 $<$0.1 at 60–70%).

Pruning did not reduce latency (RoBERTa $\sim$1.5–2.0 ms/sample; DeBERTa $\sim$2.5 ms/sample) and even increased storage (RoBERTa $\approx$800 MB, DeBERTa $\approx$1390 MB) due to mask overhead. Thus, mild pruning can remove some weights safely, but offers no practical speed/size benefit, while heavy pruning destroys accuracy.

6.3 Knowledge Distillation

Knowledge Distillation (KD) trains a smaller student under a larger teacher. Our best RoBERTa served as teacher, with a reduced RoBERTa student. Training combined cross-entropy with distillation loss, tuning $\alpha \in \{0.2, 0.5, 0.7\}$ and $T \in \{2, 3, 4, 5\}$ via Optuna. Runs used frozen encoders with selective unfreezing, learning rates 10^{-6}–3×10^{-5}, weight decay 10^{-6}–10^{-2}, batch sizes $\{4, 8, 16, 32\}$, and early stopping.

KD consistently outperformed training from scratch, especially at $\alpha = 0.5$, $T = 3$–4, retaining much of the teacher's accuracy while reducing size and latency. **In contrast to quantization and pruning, KD required full retraining but yielded more stable efficiency–performance trade-offs, making it the most practical method.**

6.4 Results

Table 3 summarizes compression outcomes for all models, reporting Macro-F1, size, and absolute latency (ms/sample). Relative speedups are shown in parentheses.

Model	Method	Macro F1	Model Size (MB)	Latency (ms/sample)
RoBERTa (Full PyTorch Loop)	Baseline	0.779	475.6	30.35
	Quantization	0.754	230.9	20.78 (1.46× faster)
	Pruning (20%)	0.767	801.9	30.10 (≈1.0×)
	KD (DistilRoBERTa)	**0.815**	313.3	8.74 (3.5× faster)
RoBERTa (HF Trainer API)	Baseline	0.840	498.7	30.35
	Quantization	0.829	230.9	22.33 (1.36× faster)
	Pruning (10%)	0.837	840.8	30.20 (≈1.0×)
	KD (DistilRoBERTa)	0.771	313.3	8.74 (3.5× faster)
DeBERTa (Full PyTorch Loop)	Baseline	0.869	1063.7	28.50
	Quantization (selective)	0.869	804.0	40.70 (0.70× slower)
	Pruning (20%)	0.862	1390.0	28.90 (≈1.0×)
	KD (DeBERTa-v3-small)	0.852	541.3	1.96 (14.5× faster)
DeBERTa (HF Trainer API)	Baseline	0.848	1063.7	28.50
	Quantization (selective)	0.848	838.0	40.70 (0.70× slower)
	Pruning (10%)	0.837	1063.7	28.60 (≈1.0×)
	KD (DeBERTa-v3-small)	0.851	541.3	1.41 (20.2× faster)

Table 3: Compression results with absolute inference latency (ms/sample). Relative factors in parentheses compare against each model's baseline.

Summary. Overall, **quantization** worked reliably for RoBERTa, offering substantial size and speed reductions with only minor accuracy loss, but it failed for DeBERTa where architectural sensitivity required selective strategies that erased any efficiency gains. **Pruning**, despite preserving accuracy under light sparsity, gave no practical improvements in speed and even inflated storage due to mask overhead, making it ineffective in our setting. In contrast, **knowledge distillation consistently delivered the best balance between efficiency and performance**, producing smaller, faster student models that retained much of the teacher's accuracy and offered a genuinely deployable compression pathway.

7 Conclusions

This work presented a comprehensive study of transformer-based sentiment classification for COVID-19 tweets. By combining preprocessing, translation, model fine-tuning, and ensemble evaluation, we established strong baselines using RoBERTa and DeBERTa across two training frameworks. Our experiments showed that DeBERTa-v3 consistently outperformed RoBERTa, especially under the Hugging Face Trainer API, where it reached a macro F1 score of 0.87. Ensembles offered modest yet reliable gains in robustness, with errors primarily confined to adjacent sentiment classes.

Beyond accuracy, we addressed real-world deployment by benchmarking model compression strategies. Quantization proved effective for RoBERTa but failed for DeBERTa, where selective approaches canceled efficiency gains. Pruning preserved accuracy under light sparsity but provided no practical improvements in latency or size. In contrast, knowledge distillation emerged as the most successful technique, yielding smaller, faster student models that retained much of their teacher's performance and enabled up to 20× inference speedups.

Overall, our results highlight that transformer-based models, when carefully fine-tuned and distilled, can deliver both accuracy and efficiency for real-world sentiment analysis tasks. Future work could extend this pipeline to cross-lingual sentiment classification, explore domain adaptation to other health-related datasets, and refine compression methods tailored to architectures such as DeBERTa. This study demonstrates the importance of not only maximizing predictive accuracy but also designing deployable NLP solutions that balance performance, efficiency, and robustness.

Note on Extra Material

The following section is provided as supplementary analysis. It was not possible to include it within the six-page limit of the main report. Readers may consider it optional: if accepted permits, it offers additional insights, but it is not part of the required submission.

8 Extra: Enhancing Predictions with Ensembles

Motivation. While single models such as DeBERTa (full PyTorch loop) reached strong performance, they still exhibit individual error patterns. Ensembles can mitigate this by averaging out model-specific mistakes, leading to more stable and robust predictions.

Methods. We tested six ensemble strategies, each combining the outputs of several models in a slightly different way:
paragraphMethods. We tested six ensemble strategies, each combining the outputs of several models in a slightly different way:

Results. Table 4 lists the six unsupervised ensemble strategies. Table 5 summarizes their macro F1 performance. Immediate takeaways are distilled right below in Table 6. The best single model, DeBERTa (full PyTorch loop), achieved a macro F1 of 0.8662. All ensemble methods slightly surpassed this baseline, with the strongest performance (0.8749) obtained by class-wise disagreement with confidence.

Method	Description
1	Simple average of logits: Average class scores (logits) across models; pick class with highest mean score.
2	Strict majority voting (hard): Each model votes for a class; the most frequent class wins.
3	Agreement-weighted voting (hard): Models agreeing more with others receive higher voting weights.
4	Agreement-weighted voting (logits): Like (3), but models contribute full score vectors, scaled by weight.
5	Class-wise disagreement (logits): Weights based on per-class consistency; combine weighted logits.
6	Class-wise disagreement + confidence: Extends (5) by incorporating model confidence into sample-level weights.

Table 4: Summary of ensemble strategies. All methods are unsupervised and use only model outputs. Exact computation details of each method are implemented and documented in the accompanying notebooks.

Method	Description	With RoBERTa (full PyTorch loop)	Without RoBERTa (full PyTorch loop)
Best Single Model	DeBERTa (full PyTorch loop) alone	**0.8662**	–
1	Simple average of logits	0.8678	**0.8734**
2	Strict majority voting (hard)	0.8633	0.8712
3	Agreement-weighted voting (hard)	0.8715	0.8718
4	Agreement-weighted voting (logits)	0.8726	0.8731
5	Class-wise disagreement (logits)	0.8723	0.8734
6	Class-wise disagreement + confidence	0.8723	**0.8749**

Table 5: Comparison of ensemble strategies. All ensembles outperform the DeBERTa (full PyTorch loop) baseline, though improvements are modest.

Key observations appear immediately below.

Key observations
• All ensemble methods yielded a performance boost over the strongest single model. • Excluding the weaker RoBERTa (full PyTorch loop) consistently improved ensemble results. • The best overall macro F1 (0.8749) was achieved by class-wise disagreement with confidence. This method combines two important elements: (a) class-wise weighting, which identifies which models tend to perform better on which classes, and (b) per-sample confidence, which emphasizes predictions that are more certain. In practice, this uneven weighting reduces the influence of weaker models and amplifies stronger ones, both overall and on specific classes. It thus makes use of *all available information* (logits, disagreements, and confidence) in an unsupervised way, which explains why it produced the best results. • **Despite these improvements, the overall gain was smaller than expected.** We believe this is because the different models share similar error patterns, being strong and weak on the same classes. With a more diverse pool of models, ensembles would likely be more effective and yield larger gains.

Table 6: Key observations for the ensemble comparison in Table 5.