Hybrid BERTweet with Averaged GloVe Fusion for Smiley-Based Tweet Sentiment Classification

Alireza Abdollahpourrostam
EPFL
Lausanne, Switzerland
SCIPER: 380830

Alireza Sakhaeirad
EPFL
Lausanne, Switzerland
SCIPER: 373331

Moein Nasiri
EPFL
Lausanne, Switzerland
SCIPER: 390183

Abstract

We study the EPFL tweet sentiment task where supervision is derived from removed emoticons (smiley and sad face). Given only the remaining tweet text, the goal is to predict whether a held-out tweet is positive or negative. We follow a simple-to-complex progression: hashed n-gram and embedding-bag baselines, pooled in-domain GloVe embeddings, and finally **pretraining and fine-tuning** with the tweet-pretrained transformer **BERTweet**. We additionally evaluate an optional fusion module that combines averaged GloVe features with transformer representations. We select hyperparameters via local validation and report leaderboard results for the best configuration.

1 Introduction

We address tweet sentiment classification in the EPFL ML text challenge. The dataset provides weak supervision from removed emoticons: tweets in `train_pos*.txt` originally contained a positive smiley, and tweets in `train_neg*.txt` a negative one. Given only the remaining, whitespace-tokenized text, we predict sentiment for 10,000 unlabeled tweets in `test_data.txt`. This setting is challenging due to informal language, creative spelling, hashtags, and heavy use of URLs and mentions.

Our report follows an incremental modeling approach **from simple to more complex**. We begin with strong, fast baselines based on feature hashing and embedding-bag pooling to establish a reliable reference point. Next, we add in-domain GloVe embeddings trained on tweet co-occurrences and evaluate pooled embedding classifiers. Finally, we move to **pretraining and fine-tuning** by fine-tuning a tweet-pretrained transformer (BERTweet) for binary classification, and we optionally explore a hybrid model that fuses the fine-tuned transformer representation with a pooled GloVe vector to improve robustness on short, noisy tweets.

Results. Our main findings and contributions are:

- **Strong simple baselines:** hashed n-gram and embedding-bag models provide fast, reproducible reference points for iteration; we further strengthen them with **hashed TF-IDF (BoW + Char)** and **threshold tuning** (see subsection C.2).
- **In-domain embeddings:** we train GloVe on the tweet corpus and show how pooled static embeddings improve over sparse features.
- **Pretraining + fine-tuning (best model):** fine-tuning tweet-pretrained BERTweet yields our strongest model family under a controlled validation protocol; our best submission combines **full fine-tuning** with **AdEMAMix + SWA** (details in Appendix D, implementation in Appendix C).
- **Hybrid fusion:** we test whether combining averaged GloVe features with transformer representations improves robustness.

2 Related Work

2.1 From sparse features to distributional representations

Early sentiment systems often relied on sparse lexical features (e.g., unigrams/bigrams). Feature hashing (Weinberger et al., 2009) provides a simple, memory-efficient way to map such features into a fixed-dimensional space, and remains a strong baseline when paired with linear models.

Distributional word representations such as GloVe (Pennington et al., 2014) encode semantic and syntactic regularities and can improve generalization over sparse counts. A common approach for short texts is to represent an input by pooling (e.g., averaging) its word vectors and training a linear or shallow neural classifier on top.

2.2 Transformers and tweet-specific pretraining

Transformers (Vaswani et al., 2017) and large-scale pretraining (e.g., BERT (Devlin et al., 2019)) substantially improved text classification by learning contextual representations. For tweets, domain mismatch matters: BERTweet (Nguyen et al., 2020) is pretrained specifically on English Twitter data, making it a strong choice for sentiment tasks with informal language and platform-specific tokens.

2.3 Efficient adaptation and training stabilization

When full fine-tuning is expensive, parameter-efficient adaptation methods like LoRA (Hu et al., 2021) update low-rank adapters instead of all weights. Separately, stochastic weight averaging (SWA) (Izmailov et al., 2018) can improve generalization by averaging weights along the training trajectory. Our codebase includes optional support for both ideas in the transformer pipeline.

3 Dataset

3.1 Files and formats

All scripts expect the dataset under `data/twitter-datasets/` by default. This can be overridden via `-data-dir`. Training data is provided in two files, one tweet per line: `train_pos.txt` / `train_neg.txt` (smaller subset) and `train_pos_full.txt` / `train_neg_full.txt` (full set enabled via `-use-full`). The test set `test_data.txt` contains 10,000 unlabeled tweets, one per line as `<Id>,<tweet>`.

3.2 Tokenization and preprocessing

Tweets are already whitespace-tokenized in the provided files (tokens separated by single spaces, with emoticons removed). For the transformer pipeline, we optionally apply lightweight normalization before tokenization: HTML unescaping, removal of zero-width characters and leading `RT`, normalization of mentions to `@user` and URLs to `http`, hashtag splitting, and normalization of repeated characters/punctuation; optional emoji demojization and outlier filters are available via flags.

3.3 Labels and submission mapping

The provided labels are weak: the positive/negative split is derived from removed emoticons. Specifically, tweets in `train_pos.txt` / `train_pos_full.txt` originally contained `:)`, while `train_neg.txt` / `train_neg_full.txt` contained `:(`. Internally, our training code uses binary labels $\{0, 1\}$ (negative/positive). Submissions to AIcrowd require `Id,Prediction` with predictions mapped to $\{-1, +1\}$, where negative mapped to -1 and positive mapped to $+1$.

4 Experiments

4.1 Evaluation protocol

We estimate generalization using a local stratified train/validation split (via `train_test_split`) with validation fraction `-val-size`. All hyperparameters are selected based on validation performance rather than the online leaderboard.

4.2 Baselines

Provided baseline. The task description provides a single simple baseline: represent a tweet by **averaging word vectors** (GloVe-style embeddings) and train a **linear classifier** (e.g., logistic regression or SVM) on top. This establishes a minimal, interpretable reference point.

Our baseline extensions. Starting from this baseline, we implemented and evaluated several stronger yet still inexpensive variants. In particular, our contributions on the baseline side are:

- **Hashing-trick bag-of-words with unigrams+bigrams:** a memory-efficient sparse baseline where token (and bigram) features are mapped into a fixed-dimensional space.
- **GPU-friendly EmbeddingBag implementation:** both the hashed model and the embedding-averaging baseline are implemented with `EmbeddingBag`, enabling fast training on large data.
- **Optional small MLP head and trainable embeddings:** for the embedding baseline, we can keep a linear head or add a small hidden layer, and optionally finetune the embedding table.
- **Validation-driven tuning:** we consistently use a local stratified split and tune hyperparameters (including the decision threshold) on validation rather than on the leaderboard.

Hashed n-gram classifier. We implement a hashing-trick model (Weinberger et al., 2009) where unigrams and optionally bigrams (`-ngram-max`) are mapped into a fixed feature space (`-num-features`). A linear classifier (logistic regression via `EmbeddingBag` accumulation) is trained with mini-batch optimization.

GPU-friendly implementation. Our hashing-trick baseline is implemented in a **GPU-friendly** way using PyTorch `EmbeddingBag` to perform batched sum-pooling over hashed feature indices, which makes training scalable on the full dataset (details in subsection C.3).

Sparse hashed TF-IDF features (BoW + Char). To strengthen the linear baseline while keeping memory bounded, we also evaluate **hashed TF-IDF** features: we map word or character n-grams into a fixed-dimensional space using the hashing trick, obtain sparse count vectors, and then apply a TF-IDF reweighting with smoothed IDF and ℓ_2 normalization. This combines the robustness of TF-IDF weighting with the scalability of feature hashing, and captures both lexical patterns (word n-grams) and stylistic/orthographic patterns (character n-grams).

Embedding-bag pooling. Using a vocabulary `vocab.pkl` aligned with an embedding matrix `embeddings.npy`, we represent each tweet by mean pooling over token embeddings (an `EmbeddingBag` model). The classifier is either linear or a small MLP (`-hidden-dim`), with dropout (`-embedding-dropout`). Embeddings can be frozen or finetuned (`-embedding-trainable`).

Threshold tuning. Several of our linear models output a real-valued score (e.g., linear regression predictions or a linear SVM decision function). Instead of using a fixed default threshold, we optionally **tune the decision threshold** on the training split to maximize accuracy: we sort the scores and evaluate candidate cut points efficiently, then apply the resulting threshold to validation/test predictions. This is especially helpful when score calibration differs across feature sets (details in subsection C.2).

4.3 GloVe embeddings

We train in-domain GloVe vectors (Pennington et al., 2014) from tweet co-occurrences. The pipeline builds a vocabulary (frequency cutoff) and a sparse co-occurrence matrix, then optimizes GloVe with SGD updates as implemented in `glove_solution.py`. The resulting matrix `embeddings.npy` is used either for pooled baselines or as auxiliary features in the hybrid transformer model.

4.4 Transformer fine-tuning and hybrid fusion

Our main model fine-tunes a transformer for binary classification. We transitioned from classical linear baselines to transformers for two reasons: (i) tweets are short and context-dependent (negation, sarcasm, polysemy), and (ii) token-level interactions are important for sentiment cues beyond bag-of-words features.

Why BERT, then BERTweet. General-purpose BERT (Devlin et al., 2019) provides strong contextual representations, but tweets exhibit substantial domain shift (hashtags, mentions, informal spelling, emoji, and platform-specific tokenization). We therefore fine-tune **BERTweet** (Nguyen et al., 2020), which is pretrained specifically on English Twitter data, improving robustness to tweet-style language.

Training setups. The training script supports (i) freezing the encoder and training only the classification head, (ii) optional LoRA adapters (Hu et al., 2021) for parameter-efficient tuning, and (iii) full fine-tuning of all weights. For hybrid fusion, we compute averaged GloVe embeddings per tweet and concatenate a projected version of this vector with the transformer pooled output before classification; a scale parameter can be estimated automatically from a sample to match feature magnitudes (additional implementation/model details in Appendix C).

Optimizer choice and SWA. For full fine-tuning, optimization stability matters because the dataset is large and weakly supervised (labels inferred from emoticons), which can increase gradient noise. We experimented with **AdEMAMix** (Pagliardini et al., 2024) as an alternative to AdamW (Loshchilov and Hutter, 2019); it mixes two exponential moving averages and can yield more stable progress in practice. We further used **stochastic weight averaging (SWA)** (Izmailov et al., 2018) to average weights along the later training trajectory, which often improves generalization by converging to flatter solutions. We provide additional intuition and

illustrative figures in Appendix D.

4.5 Implementation and reproducibility

All training entrypoints write the submission CSV with header `Id,Prediction`. Example commands (including the hybrid BERTweet + embedding fusion configuration) are provided in `run_distilbert.sh`. Appendix A documents file formats and the submission mapping.

5 Results

5.1 Baselines

Experimental setup. We report baseline performance on a local **stratified train/validation split** of the provided training data. Hyperparameters are selected using the validation set (not the online leaderboard). Unless stated otherwise, we use accuracy as the primary metric.

Provided baseline and our extensions. The task handout proposes a single baseline: **average word embeddings** for each tweet and train a **linear classifier**. We reproduce this baseline and then extend it with several improvements. In particular, our baseline-side contributions are **(i) sparse n-gram features with TF-IDF weighting (BoW + Char)**, **(ii) hashing trick for memory-efficient high-dimensional features**, and **(iii) validation-driven tuning of hyperparameters and decision threshold**.

Logistic regression. Our first linear baseline is logistic regression: a linear decision function $s(x) = w^\top x + b$ trained by minimizing the **logistic (cross-entropy) loss**. We apply it to (i) dense pooled GloVe features (the provided baseline setting) and (ii) a high-dimensional sparse representation built from hashed TF-IDF features over word and character n-grams (our extension). At inference time, predictions are produced by thresholding the sigmoid probability (we optionally tune the threshold τ on the validation split).

Linear SVM. As a complementary linear classifier, we train a **linear SVM** with hinge loss. Compared to logistic regression, the SVM margin objective can be more robust in very high-dimensional sparse feature spaces; we select its regularization strength on the validation split.

Table 1 summarizes the effect of progressively stronger feature representations. Dense pooled GloVe features alone perform close to chance, whereas adding sparse TF-IDF n-gram features (BoW + Char) yields a large improvement, confirming that short phrases and surface patterns are highly predictive for emoticon-supervised sentiment.

5.2 Main results (BERTweet)

Building on the above baselines, our strongest models fine-tune **BERTweet** on the tweet sentiment task. We experimented with three configurations that trade off compute and flexibility:

- **Head-only tuning (frozen encoder):** freeze the transformer encoder and train only the classification head (optionally with the averaged-GloVe fusion features).
- **Parameter-efficient tuning (LoRA):** insert low-rank adapters in attention/FFN modules and train only these adapters plus the head.
- **Full fine-tuning:** update all transformer weights end-to-end; this is the most compute-intensive but yielded the best performance in our runs.

Table 2 reports our main submission results for BERTweet full fine-tuning under different optimizer/SWA choices (optimizer details in Appendix D).

6 Discussion

6.1 What worked and why

Two patterns were consistent across our experiments. First, sparse lexical features matter: adding hashed word/character n-grams on top of pooled embeddings produced a large jump in validation accuracy (Table 1). This suggests that the label signal for emoticon-removed sentiment is often expressed via short phrases, negations, and stylistic markers that are well captured by n-gram features.

Second, strong in-domain pretraining matters: fine-tuning BERTweet (tweet-pretrained) substantially improved over linear baselines. The optional hybrid fusion with averaged GloVe features can be viewed as a low-cost way to inject a complementary "bag-of-words" signal into the transformer head; empirically, it was most useful in early experiments and low-epoch regimes, where the transformer may underfit without enough updates.

6.2 Optimizer choice and SWA

Beyond model architecture, we observed that optimization details had a measurable impact in full fine-tuning. Using AdEMAMix (Pagliardini et al.,

Table 1: **Sentiment Classification Performance.** Comparison of dense embeddings (GloVe) versus combined sparse features (GloVe + BoW + Char). Note the significant accuracy jump when including n-gram features.

Feature Set	Model	Dims (D)	Time (s)	Thresh. (τ)	Train Acc.	Val. Acc.
GloVe	Linear Reg. (OLS)	20	0.6	0.437	0.622	0.621
	Linear SVM†	20	135.8	-0.196	0.621	0.621
GloVe + BoW + Char	Linear Reg. (SGD)	393,236	20.8	0.512	0.839	0.836
	Linear SVM	393,236	1408.7	0.118	**0.879**	**0.867**

Table 2: **Main submission results (BERTweet full fine-tuning).** Accuracy on our AIcrowd submission for different optimization setups.

Model	Optimizer	SWA	Train Acc. (full)	Submission Acc.
BERTweet (full FT)	AdamW (Loshchilov and Hutter, 2019)	–	88.10%	87.3%
BERTweet (full FT)	AdEMAMix (Pagliardini et al., 2024)	–	90.07%	89.7%
BERTweet (full FT)	AdEMAMix (Pagliardini et al., 2024)	✓	**91.12%**	**90.7%**

2024) improved over AdamW (Loshchilov and Hutter, 2019) in our runs, and adding Stochastic Weight Averaging (SWA) (Izmailov et al., 2018) further improved the final submission accuracy (from 89.7% to 90.7% in our setting; see Table 2). A plausible explanation is that SWA smooths the end-of-training trajectory and yields a flatter solution, which is particularly beneficial when fine-tuning large transformers on noisy, weakly supervised data (additional intuition in Appendix D).

6.3 What didn't work

We explored a tighter coupling between static and contextual representations by **dynamically fusing norm-adapted GloVe features with the transformer [CLS] embedding**. Concretely, we tried to rescale the averaged GloVe vector to better match the [CLS] representation magnitude (using L2-norm comparisons and an additional scaling hyperparameter), and then fuse the two representations before classification (details in subsection C.4). In practice, this did **not yield a significant improvement** over our simpler fusion and optimizer/SWA gains. We observed that the adaptive scaling can dominate the classifier input (effectively shifting most of the signal into the rescaled embedding branch), while also increasing training time due to extra computation and hyperparameter tuning.

6.4 Limitations

Our evaluation is constrained by the weak labeling mechanism (smileys removed from tweets), which introduces label noise and domain-specific shortcuts. Online leaderboard feedback can also bias iterative development; we mitigated this by selecting hyperparameters using a local validation split and treating the leaderboard as a final check rather than a tuning target.

From a systems perspective, full transformer fine-tuning is computationally expensive and sensitive to random seeds, learning-rate schedules, and batch sizing. While we report strong single-run results, a more exhaustive analysis would include confidence intervals across multiple random seeds and a broader ablation of fusion/LoRA/optimizer settings.

7 Ethical Risks

Although this project is an academic benchmark on public tweets, sentiment classification systems can be used downstream for moderation, analytics, or decision-making, which creates ethical risks. A primary risk is **disparate error** across language varieties: tweets containing slang, dialectal forms, code-switching, reclaimed slurs, or community-specific expressions may be misclassified more often, potentially harming affected groups (e.g., by triggering disproportionate moderation actions or skewing analyses). A second risk is **context collapse**: short texts are ambiguous and depend on pragmatics, sarcasm, and conversational context; false positives/negatives can therefore be common even when aggregate accuracy is high.

We evaluated these risks qualitatively by considering common failure modes in short-text sentiment (negation scope, sarcasm markers, and polysemy),

and by recognizing that our supervision is **weak** (labels inferred from removed emoticons), which can encode dataset-specific shortcuts and amplify noise. In the scope of this course project, we mitigated the risk primarily through **documentation and scope control**: we frame results as competition/benchmark performance rather than a deployable product; we report a local validation protocol to reduce overfitting to the leaderboard; and we recommend that any real-world use would require targeted audits (error slices by language variety), robustness checks (spelling and paraphrase perturbations), and stakeholder review before deployment.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. In *UAI*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. 2020. Bertweet: A pre-trained language model for english tweets. In *EMNLP (System Demonstrations)*.

Matteo Pagliardini, Pierre Ablin, and David Grangier. 2024. The ademamix optimizer: Better, faster, older. *Preprint*, arXiv:2409.03137.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *ICML*.

A Dataset Details

A.1 Training files

Small vs. full training sets. The dataset provides two versions of the labeled training data: `train_pos.txt` / `train_neg.txt` and `train_pos_full.txt` / `train_neg_full.txt`. Each file contains one whitespace-tokenized tweet per line. The `-use-full` flag selects the full set.

A.2 Test file format

`test_data.txt` contains one example per line as:

```
<Id>,<tweet text>
```

where `Id` is an integer.

A.3 Submission format

All provided training scripts write a submission CSV with header:

```
Id,Prediction
```

Predictions are mapped to $\{-1, +1\}$: negative $\rightarrow -1$, positive $\rightarrow +1$.

B Reproducibility

B.1 Exact commands

All experiments and the final submission can be reproduced from the repository root (`project_text_classification_EPFL/`). Below are the command sequences corresponding to the main pipelines.

(Optional) Build GloVe resources.

```
./build_vocab.sh
./cut_vocab.sh
python3 pickle_vocab.py
python3 cooc.py
python3 glove_solution.py
```

Baseline classifier (hashed / embedding-bag).

```
python3 baseline_classifier.py --use-full --device auto \
  --representation hash --ngram-max 2 --num-features 262144 \
  --epochs 5 --batch-size 2048 --output baseline_submission.csv
```

BERTweet fine-tuning (+ optional fusion). The recommended configuration used in our runs is documented in `run_distilbert.sh`. For example (head-only training with optional averaged-embedding fusion):

```
python3 distilbert_classifier.py --use-full --device cuda \
  --model-name vinai/bertweet-base \
  --val-size 0.05 --epochs 3 --output baseline_submission_distilbert_head.csv \
  --embedding-path embeddings.npy --embedding-vocab vocab.pkl \
  --estimate-embedding-scale
```

B.2 Outputs

All scripts write a submission CSV with header `Id,Prediction`, where predictions are mapped to $\{-1, +1\}$ (negative $\rightarrow -1$, positive $\rightarrow +1$).

C Implementation extra details

This appendix provides additional implementation details for our transformer setup.

Parameter	Value
General Model Details	
Base Model	`distilbert-base-uncased`
Architecture	Sequence Classification
Vocab Size	30,522
Transformers Version	4.44.2
Architecture Dimensions	
Hidden Dimension (d_{model})	768
FFN Intermediate Dimension	3072
Num. Hidden Layers (L)	6
Num. Attention Heads (A)	12
Max Position Embeddings	512
Regularization & Training	
Activation Function	GELU
Dropout (General)	0.1
Dropout (Attention)	0.1
Dropout (Seq. Classif.)	0.2
Initializer Range	0.02

Table 3: Hyperparameters and configuration for DistilBERT.

C.1 Model configuration

We report the specific hyperparameters and architectural details of the model in Table 3. In our best-performing runs we fine-tune **BERTweet** (`vinai/bertweet-base`); the training script `distilbert_classifier.py` supports switching backbones via `-model-name`.

C.2 Threshold tuning details

For models that produce a real-valued score s_i per example (e.g., linear regression predictions or a linear SVM decision function), we convert scores to binary labels using a threshold τ. Rather than fixing τ a priori, we optionally choose τ to maximize accuracy on the training split:

$$\tau^* = \arg\max_{\tau} \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{\mathbf{1}[s_i \geq \tau] = y_i\}.$$

This can be computed efficiently by sorting the scores once and evaluating candidate thresholds at score change points. We then apply the selected τ^* to validation/test scores to obtain the final predictions.

C.3 GPU-friendly hashed baseline implementation

Our *hashed* baseline in `baseline_classifier.py` is implemented using a **sum-pooling `EmbeddingBag`** over hashed token indices. Concretely, each tweet is mapped to a list of feature indices (hashed unigrams and optionally bigrams) and the model learns a scalar weight per feature index; the tweet logit is the sum of the selected weights plus a bias. This design is **GPU-friendly** because it reduces sparse linear classification to batched embedding-bag lookups and reductions, which are efficiently supported by PyTorch.

For the TF-IDF baselines (BoW/Char), feature extraction uses scikit-learn's `HashingVectorizer` followed by a `TfidfTransformer` (CPU), which remains memory-bounded via hashing. We then train linear models on these sparse matrices (e.g., `LinearSVC` / SGD-based linear regression), and optionally apply the threshold tuning procedure from subsection C.2.

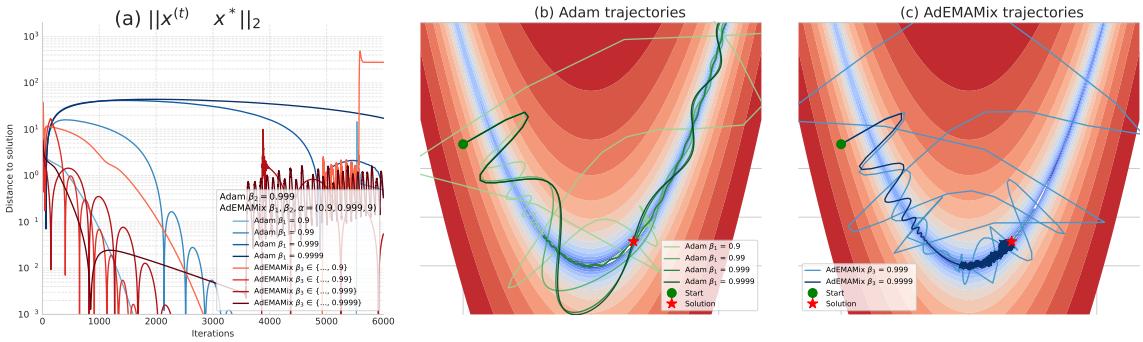

Figure 1: **Comparing Adam and AdEMAMix on the Rosenbrock function.** Starting from $x^{(0)} = [-3, 5]$, we minimize $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$. The global minimum ($\star$) is $x^\star = [1, 1]$. We use $\beta_2 = 0.999$ for Adam and $(\beta_1, \beta_2, \alpha) = (0.9, 0.999, 9)$ for AdEMAMix. Sweeping the momentum parameters shows Adam's trade-off between speed and oscillations, while AdEMAMix can move faster without large oscillations.

C.4 Negative result: norm-adapted GloVe + [CLS] fusion

Motivated by the fact that transformer representations and averaged word vectors can differ substantially in scale, we tested a fusion module that **adapts the GloVe embedding norm** before combining it with the transformer sequence representation. Let $c \in \mathbb{R}^d$ be the transformer [CLS] embedding and $g \in \mathbb{R}^d$ be the averaged (in-domain) GloVe vector for the same tweet. We experimented with L2-based rescaling:

$$g' = \alpha \cdot \frac{\|c\|_2}{\|g\|_2} g,$$

where α is a tunable scalar controlling the strength of the norm adaptation. We then fused c and g' (e.g., concatenation followed by an MLP head). Empirically, this modification did not provide a consistent accuracy gain over the simpler fusion variant and, in some settings, made optimization less stable: for larger α, the classifier relied heavily on the rescaled embedding branch. It also increased training time due to the additional hyperparameter sweep and the extra per-batch computation.

D Optimizer and SWA (additional details)

This appendix provides additional intuition for our optimizer choice and the use of stochastic weight averaging (SWA).

D.1 Why AdEMAMix?

AdEMAMix (Pagliardini et al., 2024) is an Adam-family optimizer that combines a mixture of two exponential moving averages (EMAs). In our setting (large-scale, weakly supervised tweets), gradients are noisy and the optimum can be sharp; we found that AdEMAMix offered a stable training behavior for full transformer fine-tuning compared to AdamW (Loshchilov and Hutter, 2019).

D.2 Why SWA?

Stochastic weight averaging (SWA) (Izmailov et al., 2018) maintains an average of model weights over the later part of training. Intuitively, averaging iterates from nearby solutions tends to move parameters toward wider optima, which often improves generalization.

D.3 Illustration on Rosenbrock

To build intuition, Figure 1 and Figure 2 visualize trajectories on the Rosenbrock function. The goal is not to claim novelty in optimization, but to motivate why optimizer dynamics and weight averaging can matter when fine-tuning large models.

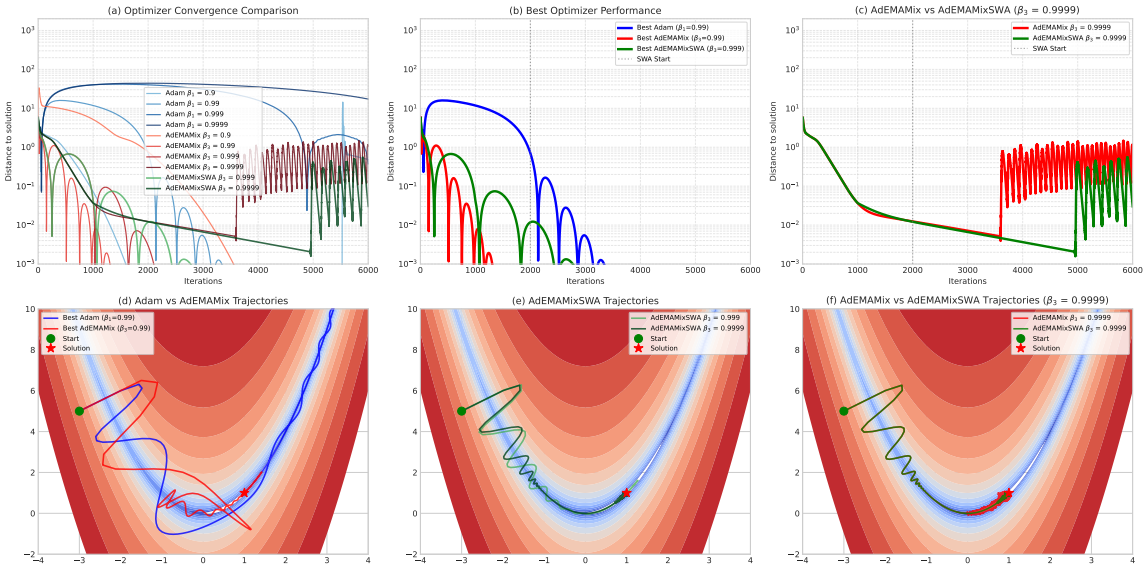

Figure 2: **Comparing Adam, AdEMAMix, and AdEMAMix+SWA on Rosenbrock.** SWA performs coordinate averaging over the later optimization trajectory. In this illustration, SWA begins at iteration 1000 and averages every 50 iterations. The averaged solution can be closer to $x^\star$ than the final iterate, motivating its use during transformer fine-tuning.