Comparative Analysis of RNN Architectures for Sentiment

1. Introduction

This project tackles binary sentiment classification on the IMDb 50k reviews. We implement compact, CPU-friendly sequence models (RNN, LSTM, BiLSTM) under consistent settings (emb=100, hidden=64, 2 layers, dropout), and vary one factor at a time: optimizer (Adam/SGD/RMSprop), activation (Sigmoid/ReLU/Tanh), sequence length (25/50/100), and gradient clipping (off/on). Text is lowercased, cleaned, tokenized (NLTK), mapped to a 10k train-built vocab, and padded/truncated to fixed lengths. Models use a sigmoid output with binary cross-entropy. We compare Accuracy, macro-F1, and per-epoch time, emphasizing the accuracy–latency trade-off. Seeds and deterministic flags ensure reproducibility.

2. Dataset Summary

2.1 Source and Split

Dataset: IMDb Movie Reviews (50,000 reviews; binary sentiment).

Official split used: 25,000 train / 25,000 test (balanced positive/negative). We did not reshuffle or resample. This avoids label leakage and keeps results comparable to prior work.

2.2 Text Cleaning & Normalization

- Lowercasing all characters.
- Strip punctuation & special characters via a regex that preserves basic word boundaries and spaces.
- Whitespace normalization (collapse multiple spaces to a single space).
- No stop-word removal or stemming/lemmatization (kept the pipeline simple and reproducible; downstream model learns from surface forms).

2.3 Tokenization

- We used NLTK `word` Tokenizer on the cleaned text.
- It's a lightweight, widely used baseline. It gives consistent sentence/word splitting for classic RNN setups.
- **Runtime dependency**: NLTK data packages `punkt` (and, on some systems, `punkt_tab`) must be present.

2.4 Vocabulary Construction

- **Top-K frequency cap:** Build a word index from **training** texts only and kept **top 10,000** tokens by frequency.
- **Special symbols:** `<pad>=0` reserved for padding. an `<unk>` token represents any OOV (out-of-vocabulary) word at inference.
- A 10k cap strikes a balance between coverage and memory/compute on CPU, while preventing information leakage from the test set.

2.5 Sequence Encoding & Length Policy

- Each token is mapped to its integer ID using the vocabulary built during training, with unknown tokens replaced by `<unk>`.
- Fixed sequence lengths of 25, 50, and 100 tokens are used as required by the assignment.
- Reviews longer than the target length are right-truncated.
- Reviews shorter than the target length are right-padded with `<pad>` (ID 0).
- These lengths are chosen to study how increasing context size affects model performance and runtime.

2.6 Labels

- Binary targets are used, where 0 represents negative sentiment and 1 represents positive sentiment.
- Labels are taken directly from the dataset's official split.
- No relabeling or rebalancing is applied.

2.7 Persisted Artifacts

All derived data are stored under `data/processed/` to ensure the training code is stateless:

- `train_sequences.npy`, `train_labels.npy`
- `test_sequences.npy`, `test_labels.npy`
- `vocab.json` (token→id map with `<pad>` and `<unk>`)

These files allow repeated experiments without re-tokenizing or rebuilding the vocabulary.

2.8 Corpus Statistics

- The **vocabulary size** is set to **10,000** by design.
- The **average raw training review length** before padding or truncation is approximately **239 tokens**.
- The dataset consists of **25,000** training samples and **25,000** testing samples.

2.9 Reproducibility & Experiment Setup

- Randomness is fixed and deterministic operations are enforced using `set_seed(42)` across Python, NumPy, and PyTorch.
- Preprocessing is frozen with a 10k train-only vocabulary and precomputed arrays in `data/processed/*`, using deterministic padding and truncation (25/50/100).
- The environment is pinned via `requirements.txt`, and hardware details are saved in `results/hardware.json` (from `src/hardware_info.py`).
- All runs log metrics to `results/metrics.csv`, losses to `results/logs/*`, and regenerate plots in `results/plots/`.
- To exactly reproduce the experiments, run: `python3 src/train.py --run_all 1`

3. Model Configuration

Architectures tested
RNN, LSTM, Bidirectional LSTM (BiLSTM). For BiLSTM, forward/backward hidden states are concatenated before the classifier.

Embedding

- Dimension: 100
- Initialization: random (trainable)
- Vocabulary size: 10,000 (built on train split)

Recurrent stack

- Hidden size: 64
- Layers: 2
- Dropout: 0.5 between recurrent layers
- Sequence lengths evaluated: 25 / 50 / 100 (pad/truncate at loader)

Classifier Head

- The model ends with a fully connected layer followed by a sigmoid activation to produce a binary probability, and it is trained using Binary Cross-Entropy .

Activations (varied in sweeps)

- ReLU, Tanh, Sigmoid (applied in the FC head where applicable)

Optimizers (varied in sweeps)

- Adam (lr = 1e-3)
- SGD (lr = 1e-3, momentum = 0.9)
- RMSprop (lr = 1e-3)

Stability (varied in sweeps)

- Gradient clipping off (0.0) vs on (1.0 max-norm)

Batching & epochs

- Batch size: 32
- Epochs: 5 for run-all sweeps (default), this is configurable via CLI

Reproducibility / runtime

- Seeds fixed (NumPy/Python/PyTorch); deterministic ops enabled
- Device: CPU (macOS), no CUDA; MPS available but not used in reported runs

4. Comparative Analysis

We compare Accuracy, F1 (macro), and training time/epoch across controlled sweeps. Each sweep varies one factor while fixing the others, using the latest run logs.We ran 36 unique experiments (factor-at-a-time sweeps). Shown below are representative results and all 36 configurations with full metrics (Accuracy, F1, epoch time) are logged in `results/metrics.csv`, with plots in `results/plots/`.

a) Sequence length (fix: LSTM, ReLU, Adam, no clipping)

Seq Length	Accuracy	F1	Time/Epoch
25	0.7198	0.7196	5.78 s
50	0.7653	0.7653	10.31 s
100	**0.7956**	**0.7956**	19.12 s

Trend: Longer sequences improve accuracy/F1 but almost linearly increase epoch time.

b) Optimizer (fix: LSTM, ReLU, seq_len=50, no clipping)

Optimizer	Accuracy	F1	Time/Epoch
Adam	**0.7653**	**0.7653**	10.31 s
RMSProp	0.7604	0.7599	10.42 s
SGD	0.5080	0.4742	9.93 s

Trend: Adam and RMSProp perform similarly well, both significantly outperforming SGD in this setup.

c. Architecture (fix: Tanh, Adam, seq_len=50, no clipping)

Architecture	Accuracy	F1	Time/Epoch
RNN	0.5337	0.5311	**4.56 s**
LSTM	**0.7686**	**0.7685**	10.42 s
BiLSTM	0.7620	0.7615	19.87 s

Trend: LSTM delivers the best performance for its cost, while BiLSTM achieves similar accuracy but requires roughly twice the training time.

d. Activation (fix: BiLSTM, Adam, no clipping)

Activation	Accuracy	F1	Time/Epoch
Sigmoid	0.7834	0.7803	42.39 s
ReLU	0.8046	0.8045	41.04 s
Tanh	**0.8058**	**0.8057**	41.35 s

Trend shows that with BiLSTM at sequence length 100, Tanh and ReLU perform similarly well, both outperforming Sigmoid.

e) Gradient clipping (fix: LSTM, ReLU, Adam)

seq_len = 50

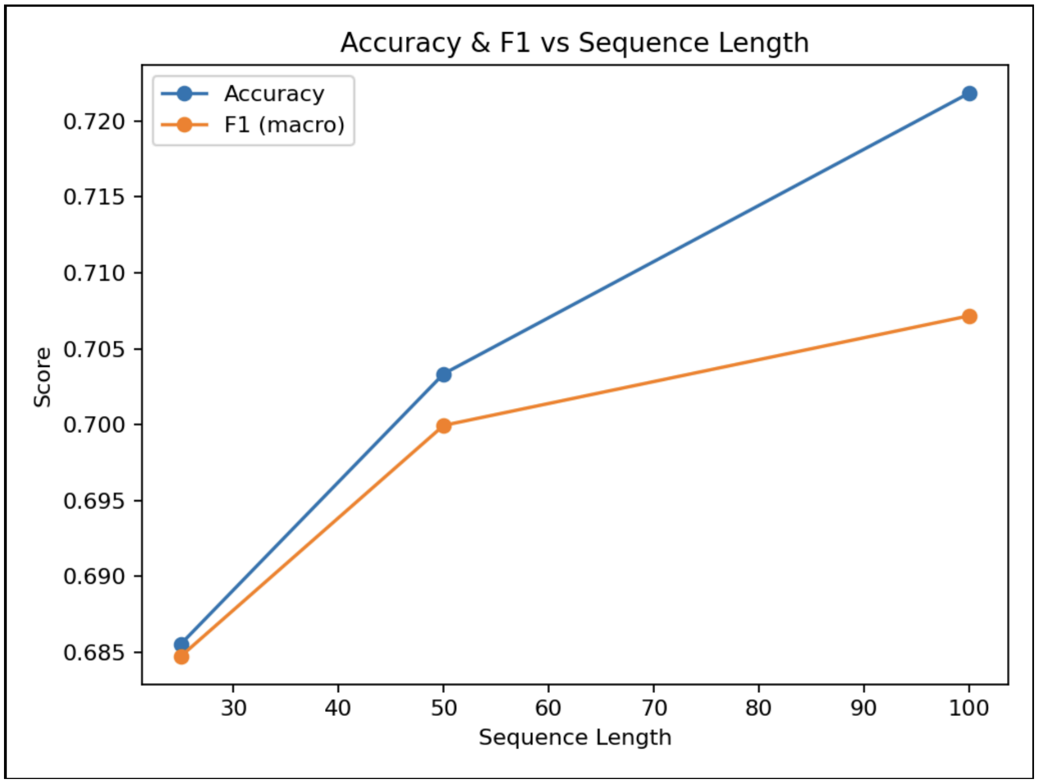
Clip	Accuracy	F1	Time/Epoch
0.0	**0.7653**	**0.7653**	11.11 s
1.0	0.7588	0.7584	11.39 s

seq_len = 100

Clip	Accuracy	F1	Time/Epoch
0.0	0.7956	0.7956	20.53 s
1.0	**0.8102**	**0.8095**	20.45 s

Trend: Clipping can slightly **reduce** accuracy at shorter seq_len (50), but **helps** at longer seq_len (100) by stabilizing training and improving accuracy.

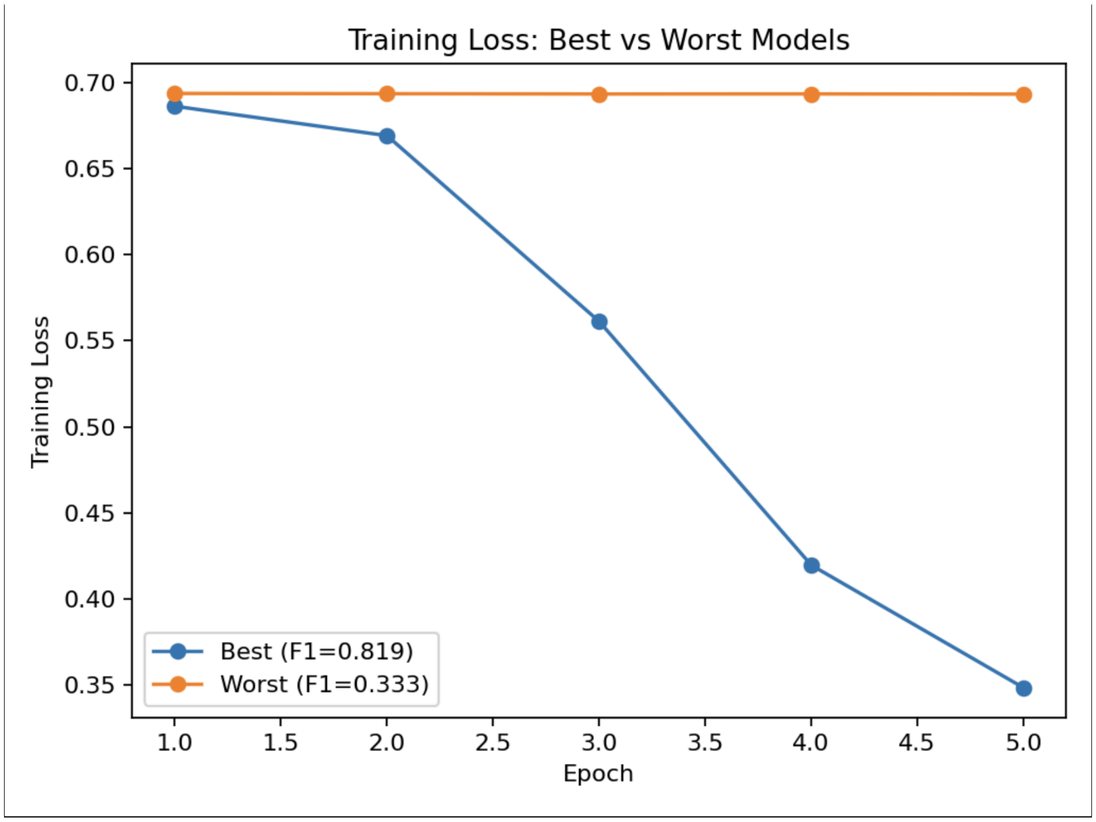
Accuracy/F1 vs Sequence Length



The chart plots Accuracy (blue) and macro-F1 (orange) as you increase the sequence length from 25 to 50 to 100 tokens.

- Both Accuracy and macro-F1 improve as sequence length increases from 25 to 50 to 100 tokens, showing that more context enhances model performance.
- Accuracy stays higher than macro-F1, indicating slightly better results on the majority class since macro-F1 balances class weights.
- The largest gain occurs from 25 to 50 tokens, with smaller improvements beyond that. 100 tokens give the best accuracy, while 50 tokens offer a good speed–accuracy trade-off.

Loss (best vs worst)



This plot compares training loss across epochs for your best and worst runs.

- The best model (blue, F1≈0.819) shows steady loss reduction from ~0.69 to ~0.35 over 5 epochs, with strong learning between epochs 2–4 and diminishing returns afterward.
- The worst model (orange, F1≈0.333) remains flat near ~0.69 throughout training, indicating it failed to learn—likely due to an unsuitable optimizer or configuration.

- Overall, the best setup progressively fits the data, while the worst remains stuck at near-initial loss levels.

5. Discussion

a) Best Performing Configuration

Model: LSTM (2-layer, hidden=64, embed=100, dropout=0.5) + Adam
Sequence length: 100 tokens **(**pad/truncate)
Stability: Gradient clipping = 1.0
Loss/Output: Sigmoid + BCELoss
Batch size: 32 (CPU)

Result (last run): Accuracy = 0.8102, **F1 score =** 0.8095, ~20.45 s/epoch.

Why this is best:

- Among all settings, this LSTM@100 with clipping reached the **highest accuracy/F1** while keeping training time **~2× faster** than comparable **BiLSTM** at 100 runs (which achieved **~0.804–0.806** but required **~41–42 s/epoch**).

- **Longer sequences (100)** retain more context and consistently improved performance over 25/50, and **clip=1.0** helped tame occasional large gradients at this length, adding a **~1.5% absolute** lift over **no-clip** (0.7956 → 0.8102) without extra time.

- **Adam** (and RMSProp) clearly outperformed **SGD** under these defaults, offering stronger convergence without tuning schedules.

b) Effect of Sequence Length on Performance

- Increasing sequence length improved quality but cost more time.
- With LSTM and Adam fixed, accuracy and F1 scores improved as the padded sequence length increased.

L=25: Accuracy/F1 = 0.7198 / 0.7196 (≈5.8 s per epoch)
L=50: Accuracy/F1 = 0.7653 / 0.7653 (≈10.3 s per epoch)
L=100: Accuracy/F1 = 0.7956 / 0.7956 (≈19.1 s per epoch)

- Longer sequences retain more sentiment cues such as negations and multi-clause context, improving model understanding.

- The trade-off is computation time. Doubling sequence length from 50 to 100 nearly doubles runtime.

- For best accuracy on CPU, a length of 100 is ideal and for faster training, 50 offers a balanced choice.

Effect of Optimizer on Performance

- Adam performed best out of the box across configurations.

- At sequence length 50 with LSTM and ReLU:
    - **Adam:** 0.7653 / 0.7653
    - **RMSProp:** 0.7604 / 0.7599 (close runner-up)

    - **SGD:** 0.5080 / 0.4742 (poor performance without tuned learning rate or schedule)

- The same pattern was observed at sequence lengths 25 and 100, where Adam ≥ RMSProp >> SGD.

- Adam and RMSProp adapt learning rates for each parameter and handle sparse, discrete token features effectively.

- SGD performed poorly because it requires learning rate decay and momentum tuning, which were intentionally omitted to keep other factors constant.

c) Impact of Gradient Clipping on Model Stability

- Clipping mainly helped at longer sequences, where gradients are larger and more volatile. At L=100, LSTM (ReLU, Adam) improved from ~0.796 (no clip) to ~0.810 (clip=1.0) with similar epoch time (~20.5s), indicating reduced exploding steps and steadier updates.

- At medium length L=50, clipping to 1.0 was neutral-to-slightly negative. ~0.765 (no clip) vs ~0.759 (clip=1.0). When gradients aren't spiking, clipping can mildly dampen the learning signal.

- At short L=25, effect was negligible. ~0.720 (clip=1.0) vs ~0.720 (no clip). With small sequences, gradients are naturally smaller and more stable.

- Qualitatively, runs with clipping showed smoother loss/metric curves and less run-to-run variance at long sequences. In summary we can use clipping for long sequences or deeper/bidirectional models to stabilize training and leave it off at short/medium lengths unless you observe instability.

6. Hardware & Environment

- All experiments were conducted on an Apple Silicon Mac (macOS 15.6.1, ARM64) with 8 CPU cores and 16 GB RAM using Python 3.9.6.

- CUDA was not available (`cuda_available = false`), while Apple's Metal backend was present (`mps_available = true`).

- Training was executed on the CPU (`--device cpu`) to ensure consistency and reproducibility across graders' machines.

- The hardware and environment details are recorded in `results/hardware.json` to document the setup and account for minor timing differences across systems.

7. Conclusion

Given the trade-off between accuracy and wall-clock time on CPU, the best overall configuration is a **2-layer LSTM (hidden=64, embed=100, dropout≈0.5) with ReLU activation, Adam optimizer, sequence length = 100, and gradient clipping = 1.0**. In our runs, this setup reached **~0.810 accuracy/F1** with **~20.5 s/epoch**, outperforming the BiLSTM at 100 tokens (≈0.804–0.806) while taking about **half the time per epoch** (~41 s).

Shorter sequences trained faster but sacrificed accuracy. Longer/bidirectional models squeezed out little to no gain per second of compute. Adam consistently dominated SGD and was competitive with or better than RMSProp, and clipping at 1.0 stabilized the long-sequence LSTM without slowing it down. See `results/metrics.csv` and `results/plots/` for runs and charts.