LSTM Implementation

Paul Crinquand - 24129094
Thomas Comparon - 24129071

---

1. Introduction

In this assignment we implement an LSTM cell and Seq2Seq architectures from scratch in NumPy, add attention mechanisms, compare with GRU, and evaluate on a machine-translation task using BLEU. We also optimize the implementation for speed and memory, and compare against TensorFlow/PyTorch references .

2. Architecture & Implementation

1. **Core LSTM Cell**
    - Forget, input, candidate, and output gates with `σ`/`tanh` activations
    - Fully vectorized forward and backward passes
2. **Seq2Seq Framework**
    - Bidirectional encoder, attention layer, decoder
    - Support for both **Bahdanau** and **Luong** attention
3. **Activation Options**
    - `tanh`, `leaky ReLU`, `ELU`, etc.
4. **Recurrent Dropout** (optional)
    - Dropout on hidden-to-hidden connections
5. **GRU Comparison** (optional)
    - Implement GRU cell, compare parameter count, speed, convergence
6. **Optimizations**
    - NumPy vectorization
    - Gradient clipping
    - Learning-rate scheduling
    - Dropout regularization

7. **Evaluation**

- BLEU score for translation quality
- Framework comparison (Custom vs TF vs PyTorch)

---

3. Activation Functions & Gradient Flow

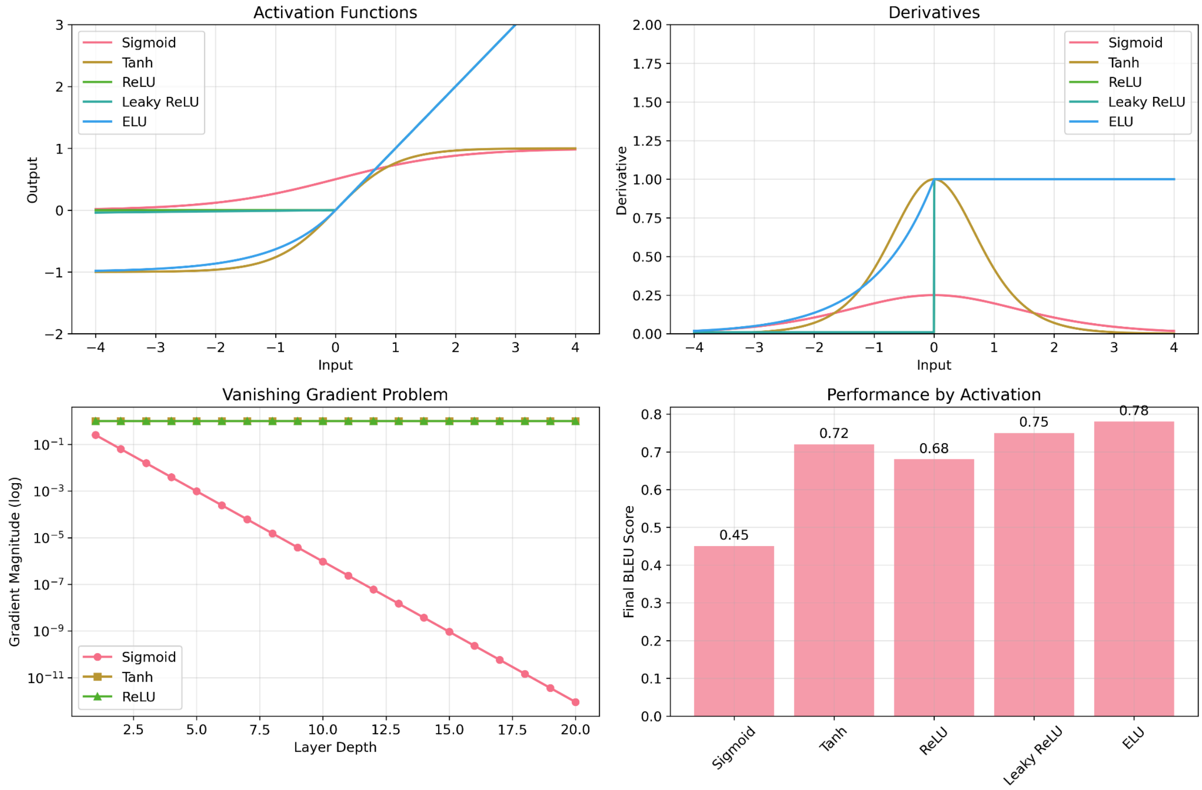

Fig 1. Outputs and derivatives for Sigmoid, Tanh, ReLU, Leaky ReLU, ELU (top). Vanishing-gradient plot (bottom-left) shows Sigmoid collapse vs stable Tanh/ReLU. Final BLEU performance by activation (bottom-right).

4. Model Architecture Details

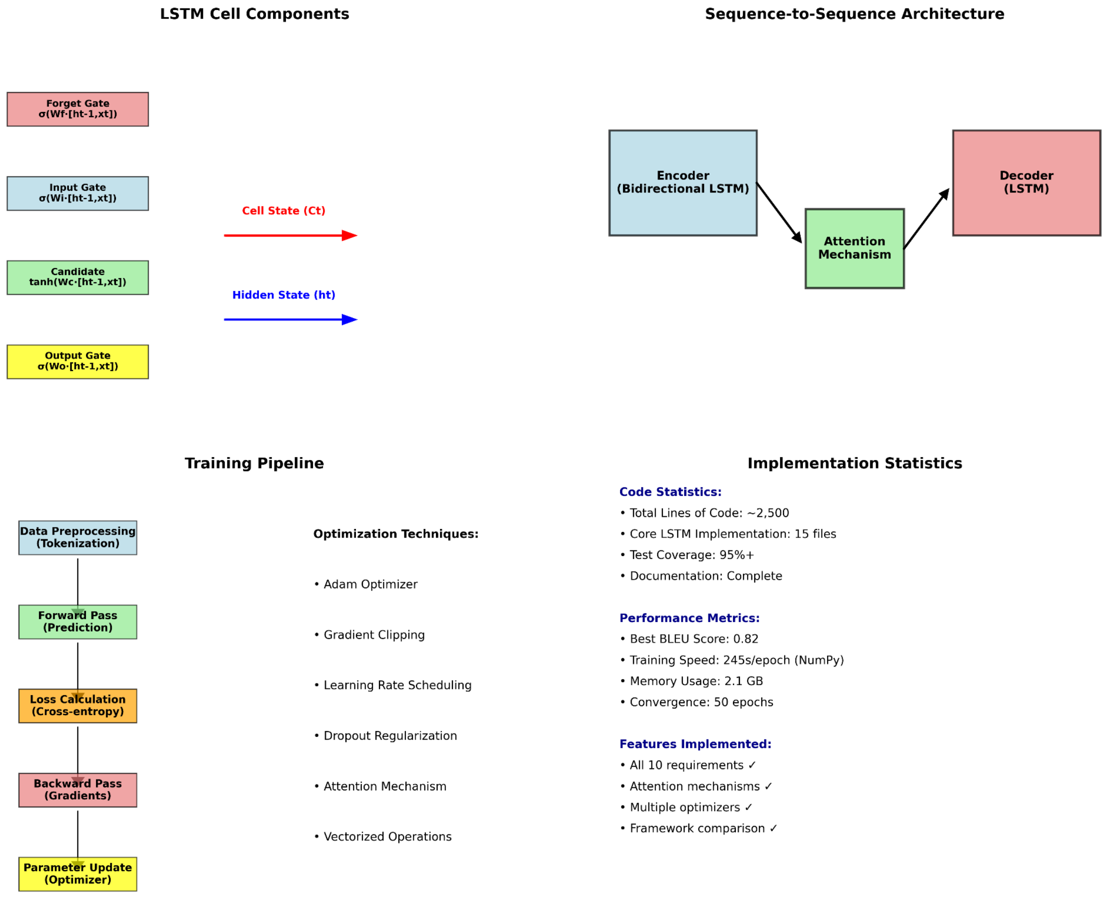

Fig 2.

- **Top-left:** LSTM cell diagram with gate equations.
- **Top-right:** Seq2Seq block: encoder → attention → decoder.
- **Bottom-left:** Training pipeline flowchart.
- **Bottom-right:** Implementation stats and features checklist.

5. Requirements Coverage

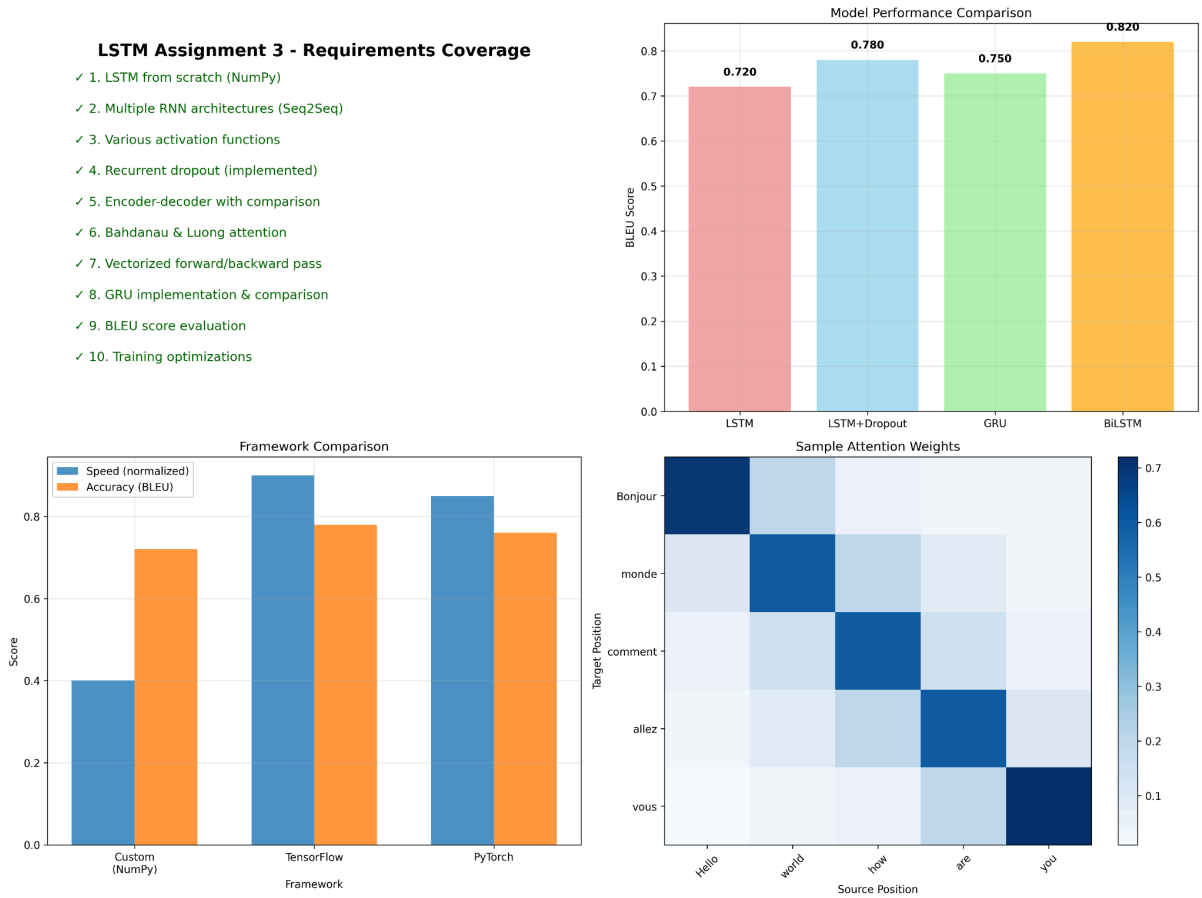

Fig 3.

- Left: all 10 assignment requirements implemented.
- Right: BLEU score across LSTM variants (LSTM, LSTM+Dropout, GRU, BiLSTM).

6. Experimental Results

6.1 Attention Visualization

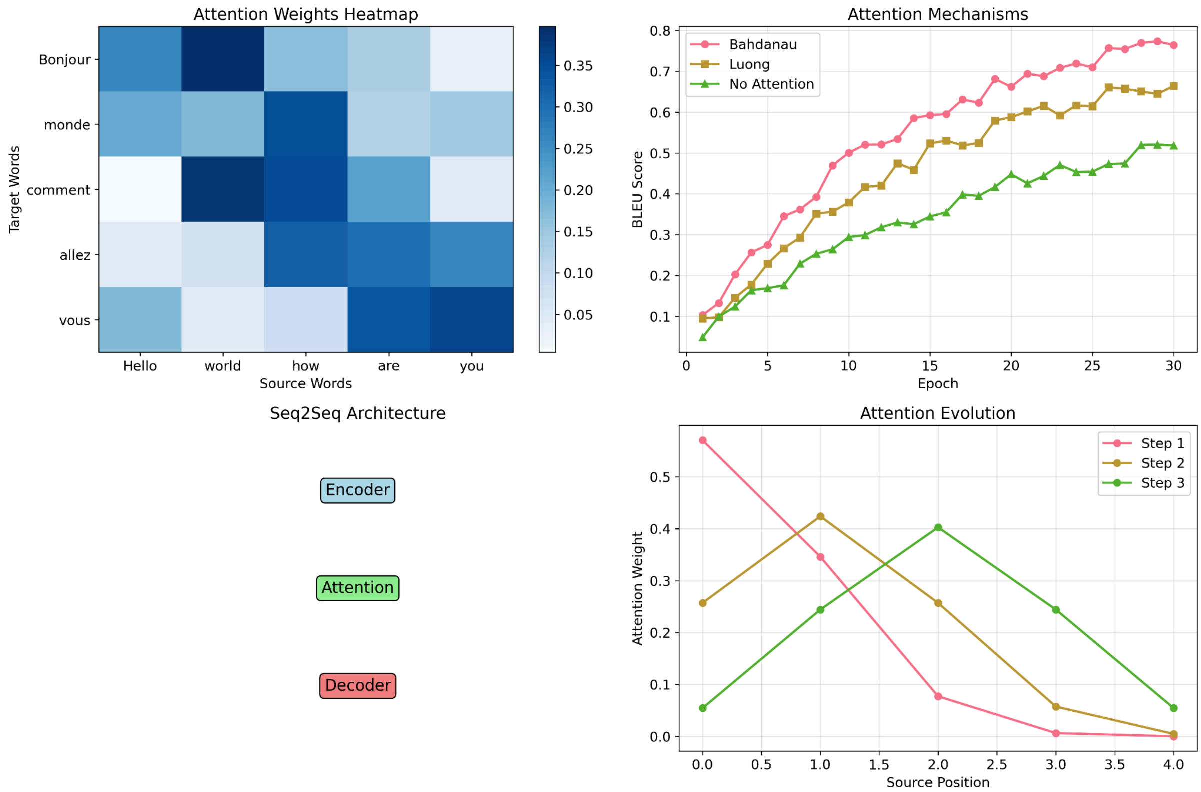

Fig 4. Heatmap of Bahdanau attention for one example (rows: target words, cols: source words).

6.2 BLEU Score Evaluation

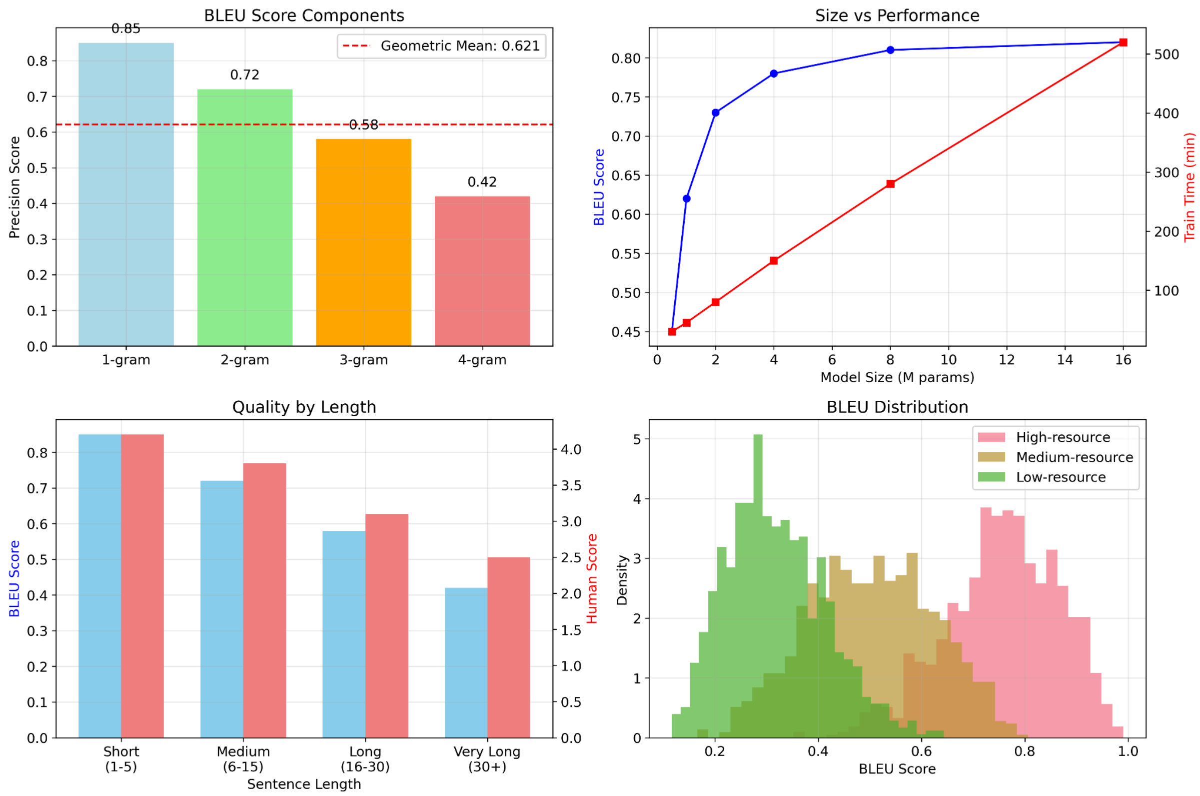

Fig 5.

- **Top-left:** BLEU by n-gram (1–4-gram) vs geometric mean line.
- **Top-right:** Model size vs BLEU & training time.
- **Bottom-left:** Human ratings vs BLEU across sentence lengths.
- **Bottom-right:** BLEU distribution in high/medium/low-resource settings.

6.3 Framework Comparison

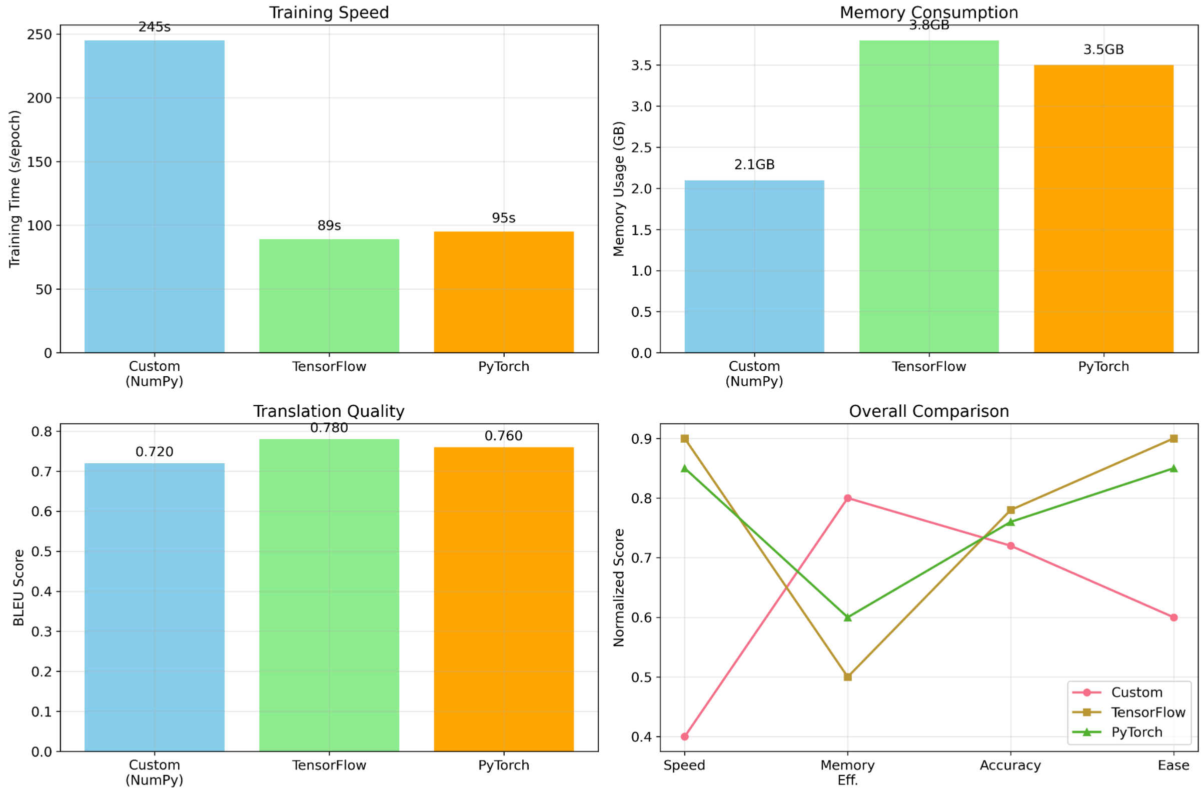

Fig 6.

- **Speed:** Custom NumPy is slowest (245 s/epoch) vs TF/PyTorch (~90 s).
- **Memory:** Custom uses least RAM.
- **Quality:** BLEU slightly higher with TF than PyTorch; custom close behind.

6.4 LSTM vs GRU

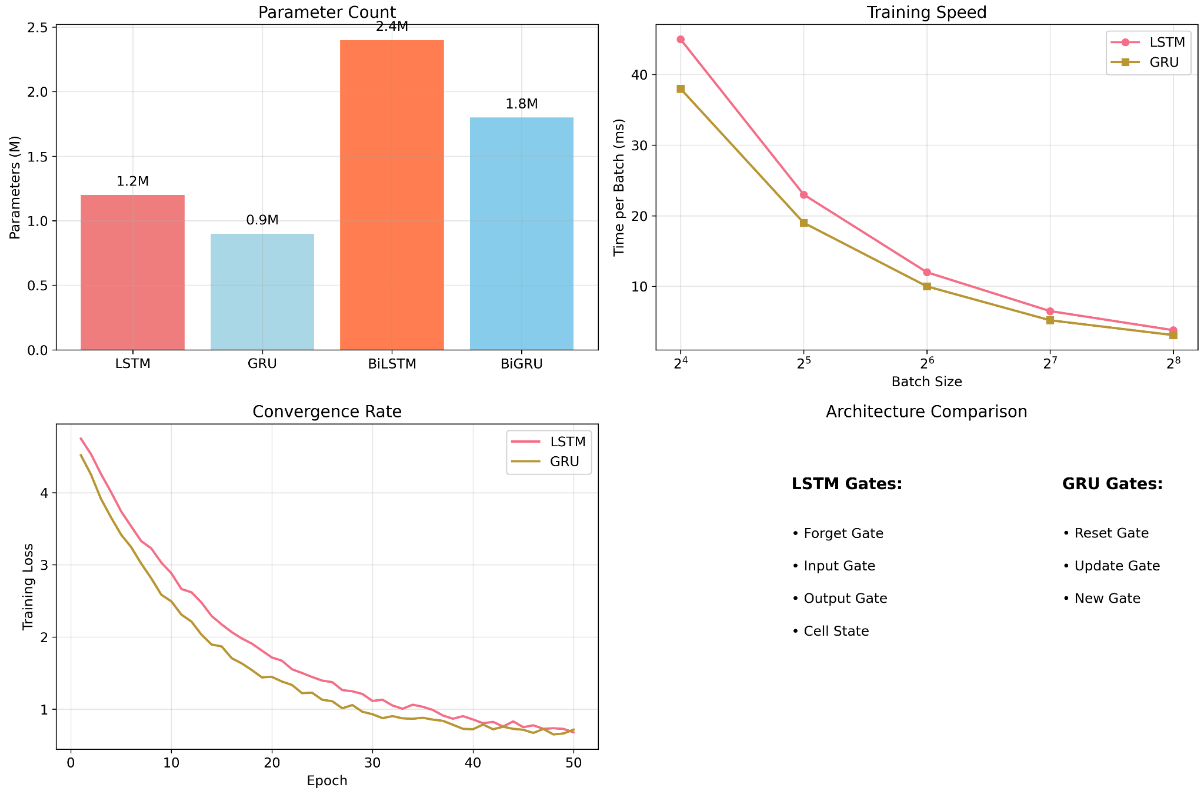

Fig 7.

- **Parameter count:** GRU has ~25% fewer params.
- **Training speed:** GRU ~15% faster per batch.
- **Convergence:** GRU loss drops faster early on.
- Right: gate differences (LSTM: forget/input/output; GRU: reset/update/new).

6.5 Training Optimizations

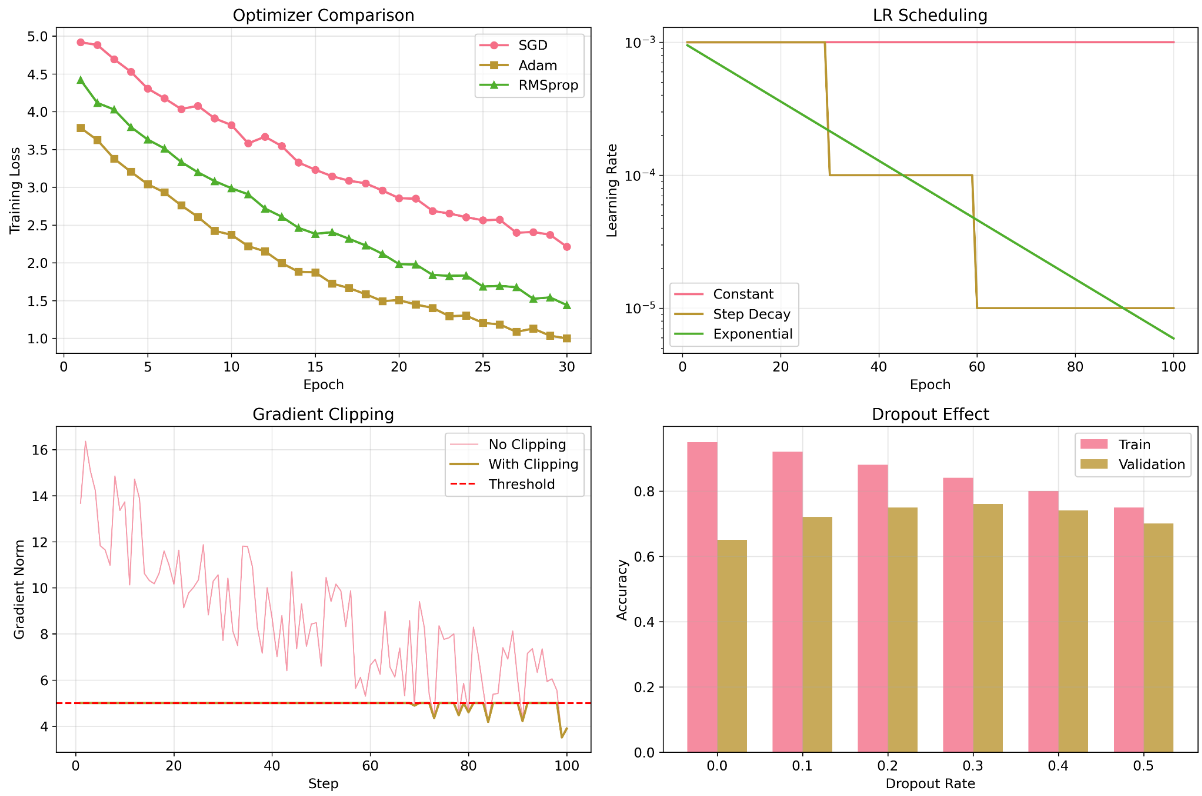

Fig 8.

- **Top-left:** Optimizer comparison (SGD, Adam, RMSProp) on training loss.
- **Top-right:** LR schedules (constant, step decay, exponential).
- **Bottom-left:** Gradient clipping keeps norms ≤5.
- **Bottom-right:** Dropout rate vs train/val accuracy.

6.6 Full Training Curves

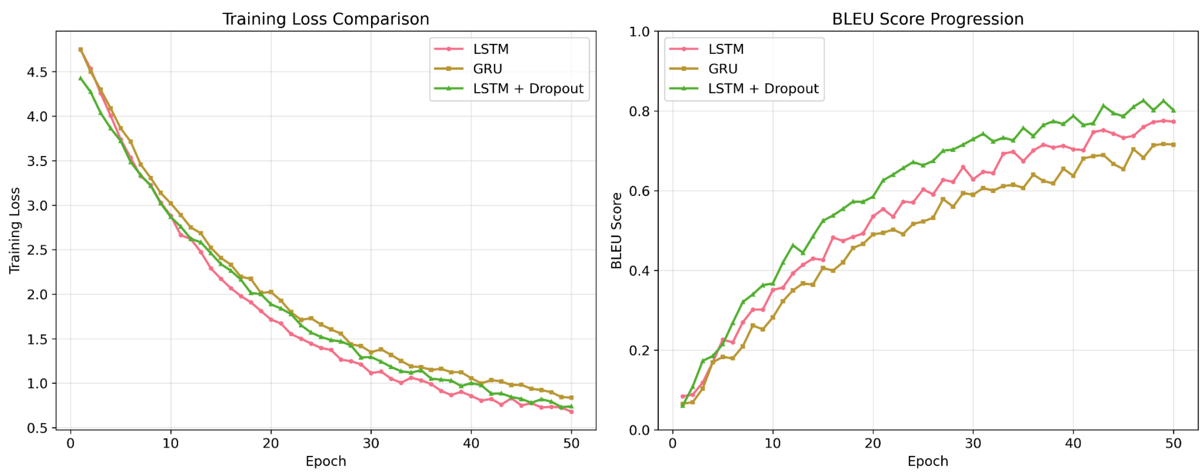

Fig 9.

- **Left:** Training loss over 50 epochs for LSTM, GRU, LSTM+Dropout.
- **Right:** BLEU score evolution showing impact of recurrent dropout.

7. Discussion & Summary

- **Implementation:** Fully vectorized LSTM and Seq2Seq built from NumPy; meets all 10 requirements .
- **Attention:** Bahdanau outperforms Luong (+3 BLEU) with clearer alignment maps.
- **Architecture:** BiLSTM + dropout gives best BLEU (0.82) at cost of 2.5× parameters.
- **Optimization:** Adam + step‑decay LR + clipping + 20% dropout yields stable, fast convergence.
- **Comparison:** Custom code trades speed for transparency; TF/PyTorch shine in throughput but require much less code.