Midterm and Final Project: Machine Translation between Chinese and English

Student Name
Institute Name
Code Repository: `https://github.com/aface0427/nlp-homework`

December 28, 2025

Abstract

This report presents a comprehensive study on Neural Machine Translation (NMT) for Chinese-English translation. We investigate three main approaches: Recurrent Neural Networks (RNNs) with attention mechanisms, Transformer models trained from scratch, and fine-tuning of pre-trained Large Language Models (T5). We conduct extensive ablation studies on critical components such as attention types, positional encodings, normalization strategies, and model scaling. Our experiments on a 100k sentence pair dataset demonstrate the superiority of the Transformer architecture, particularly when equipped with relative positional encodings and trained on larger datasets. We also analyze the impact of teacher forcing and data scale on model performance.

1 Dataset and Preprocessing

1.1 Dataset

All experiments are conducted on the Chinese-English parallel corpus provided for this project. The dataset is divided into three splits:

- **Training Set**: Approximately 100,000 sentence pairs (100k), with a smaller subset of 10,000 pairs (10k) used for ablation studies and rapid prototyping.
- **Validation Set**: A fixed set of 500 sentence pairs used for model selection and hyperparameter tuning.
- **Test Set**: A held-out set of 200 sentence pairs used for final performance evaluation.

The validation and test sets remain consistent across all experiments to ensure fair comparison.

1.2 Data Preprocessing

The raw text data undergoes a standard preprocessing pipeline to ensure quality and compatibility with the models.

- **Tokenization**: We employ a word-level tokenization strategy for the RNN models, building a vocabulary based on frequency thresholds (minimum frequency = 2). For Transformer models and T5, we utilize subword tokenization (SentencePiece or BPE) to handle the open vocabulary problem and improve coverage for rare words.
- **Special Tokens**: Special tokens are added to the sequences: `<SOS>` (Start of Sentence), `<EOS>` (End of Sentence), `<PAD>` (Padding), and `<UNK>` (Unknown).
- **Length Constraints**: Sentences are truncated or padded to a fixed maximum sequence length (e.g., 128 tokens) to facilitate efficient batch processing.

2 RNN-based Neural Machine Translation

We implement a classical Sequence-to-Sequence (Seq2Seq) model based on Recurrent Neural Networks (RNNs) as a baseline. This architecture consists of an encoder that compresses the source sentence into a context vector and a decoder that generates the target sentence token by token.

2.1 Model Architecture

The core components of our RNN model are:

- **Encoder**: A 2-layer unidirectional RNN (GRU or LSTM) that processes the source sequence. We chose unidirectional encoders to strictly adhere to the project requirements and to study the baseline performance without the look-ahead advantage of bidirectional processing.
- **Decoder**: A 2-layer unidirectional RNN (GRU or LSTM) initialized with the encoder's final hidden state. At each step, it receives the embedding of the previous token and the context vector.
- **Embedding Layer**: A shared embedding layer of dimension 256 is used to map tokens to dense vectors.

2.2 Attention Mechanisms

To alleviate the bottleneck of compressing the entire source sentence into a single fixed-size vector, we incorporate attention mechanisms. We investigate three variants:

- **Dot-Product Attention**: Computes alignment scores via the dot product between decoder and encoder hidden states.
- **Multiplicative (General) Attention**: Introduces a learnable weight matrix W_a between the states: $score(h_t, \bar{h}_s) = h_t^\top W_a \bar{h}_s$.
- **Additive (Bahdanau) Attention**: Uses a feed-forward network to compute scores: $score(h_t, \bar{h}_s) = v_a^\top \tanh(W_1 h_t + W_2 \bar{h}_s)$. This method is generally more robust but computationally expensive.

2.3 Training Policy

We explore the impact of **Teacher Forcing**, a technique where the ground-truth target token is used as input for the next step during training, rather than the model's own prediction.

- **Teacher Forcing Ratio**: We experiment with ratios of 0.0 (Free Running), 0.5, and 1.0 (Full Teacher Forcing).
- **Impact**: Higher ratios typically lead to faster convergence but may cause exposure bias, where the model fails to recover from its own errors during inference.

2.4 Decoding Strategy

At inference time, we implement both greedy decoding and beam search:

- **Greedy Decoding**: Selects the most probable token at each step. Fast but may produce suboptimal translations.
- **Beam Search**: Maintains k candidate hypotheses (beam size = 5 in our experiments) and selects the best complete sequence. Generally produces more fluent translations at the cost of increased computation.

3 Transformer from Scratch

We implement a Transformer model from scratch, relying entirely on self-attention mechanisms to draw global dependencies between input and output.

3.1 Overall Architecture

Our Transformer follows the standard encoder-decoder architecture proposed by Vaswani et al.

- **Encoder**: A stack of N identical layers, each containing a Multi-Head Self-Attention mechanism and a Position-wise Feed-Forward Network.
- **Decoder**: A stack of N identical layers, each containing Masked Self-Attention, Encoder-Decoder Attention, and a Feed-Forward Network.
- **Residual Connections & Normalization**: Applied around each sub-layer.

3.2 Positional Encoding and Architectural Ablation

Since Transformers contain no recurrence, we must inject positional information. We compare:

- **Absolute Positional Encoding**: Standard sinusoidal functions added to input embeddings.
- **Relative Positional Encoding**: Learns relative distances between tokens, potentially allowing for better generalization to sequence lengths unseen during training.

3.3 Normalization Strategy

We investigate the stability and performance impact of different normalization techniques:

- **Layer Normalization (LayerNorm)**: The standard approach, normalizing across the feature dimension.
- **RMSNorm**: A simplified version of LayerNorm that re-scales invariance and is computationally more efficient.

3.4 Model Scaling

To understand the trade-off between model size and performance, we evaluate different configurations:

- **Small**: 2 layers, 128 hidden dimension.
- **Medium**: 4 layers, 256 hidden dimension.
- **Large**: 6 layers, 512 hidden dimension.

4 Fine-tuning a Pretrained Model

In addition to training from scratch, we explore the paradigm of Transfer Learning by fine-tuning a pre-trained T5 (Text-to-Text Transfer Transformer) model.

4.1 Pretrained Model

We utilize the `t5-small` or `t5-base` architecture. T5 is pre-trained on a massive multi-task dataset (C4) using a "denoising" objective, making it highly capable of understanding general language structures.

4.2 Fine-tuning Strategy

We adapt the model to the Chinese-English translation task by formatting the input as "translate Chinese to English: [Source Sentence]". The model is then trained to generate the target English sentence. This approach leverages the rich linguistic knowledge encoded in the pre-trained weights, allowing for rapid convergence even with limited parallel data.

5 Experimental Setup

5.1 Implementation Details

All models are implemented in PyTorch.

- **Hardware**: Experiments were conducted on a single NVIDIA RTX 5070ti GPU.
- **Optimizer**: We use Adam or AdamW with $\beta_1 = 0.9, \beta_2 = 0.98$.
- **Learning Rate**: A warm-up strategy followed by decay is employed for Transformers. For RNNs, a fixed learning rate with plateau-based decay is used.
- **Loss Function**: Cross-Entropy Loss with label smoothing (0.1) is used for Transformers to improve generalization.

5.2 Evaluation Metrics

Model performance is evaluated using **BLEU** (Bilingual Evaluation Understudy), specifically the `sacreBLEU` implementation to ensure standard tokenization and comparability. We report results on the held-out Test Set.

6 Code Implementation

The project is implemented in a modular fashion to ensure extensibility and reproducibility. The code structure is as follows:

- `config.py`: Centralized configuration for file paths, model hyperparameters, and training settings.
- `data_utils.py`: Handles data loading, tokenization (Jieba for Chinese, SentencePiece for subwords), and batch creation.
- `models/`: Contains model definitions.
  - `rnn_model.py`: Implementation of the Seq2Seq RNN with various attention mechanisms.
  - `transformer_model.py`: Implementation of the Transformer from scratch, including custom layers for relative positional encoding and RMSNorm.
- `train.py`: The main training loop, handling forward passes, loss computation, backpropagation, and validation.
- `inference.py`: A standalone script for loading checkpoints and generating translations for new inputs.

We utilized PyTorch's `Dataset` and `DataLoader` for efficient data handling and implemented custom collate functions for dynamic padding.

7 Results and Analysis

7.1 Overall Model Comparison

We begin by presenting the overall comparison of different model architectures. Table 1 summarizes the key performance metrics across RNN, Transformer, and T5 models.

Table 1: Main Results: Comparison of Best Models on Validation and Test Sets

Model Architecture	Training Data	Val BLEU	Test BLEU
RNN (GRU + Additive)	10k	1.98	1.82
RNN (GRU + Additive)	100k	5.34	4.97
RNN (GRU + Dot)	10k	1.28	1.15
RNN (LSTM + Additive)	10k	1.11	0.98
Transformer (Small)	10k	2.56	2.41
Transformer (Medium)	10k	1.51	1.38
Transformer (Small)	100k	12.63	11.89
Transformer (Medium)	**100k**	**15.66**	**14.82**
Transformer (Relative)	100k	14.77	13.95
T5 Fine-tuning	10k	0.50	0.43
T5 Fine-tuning	100k	0.78	0.65

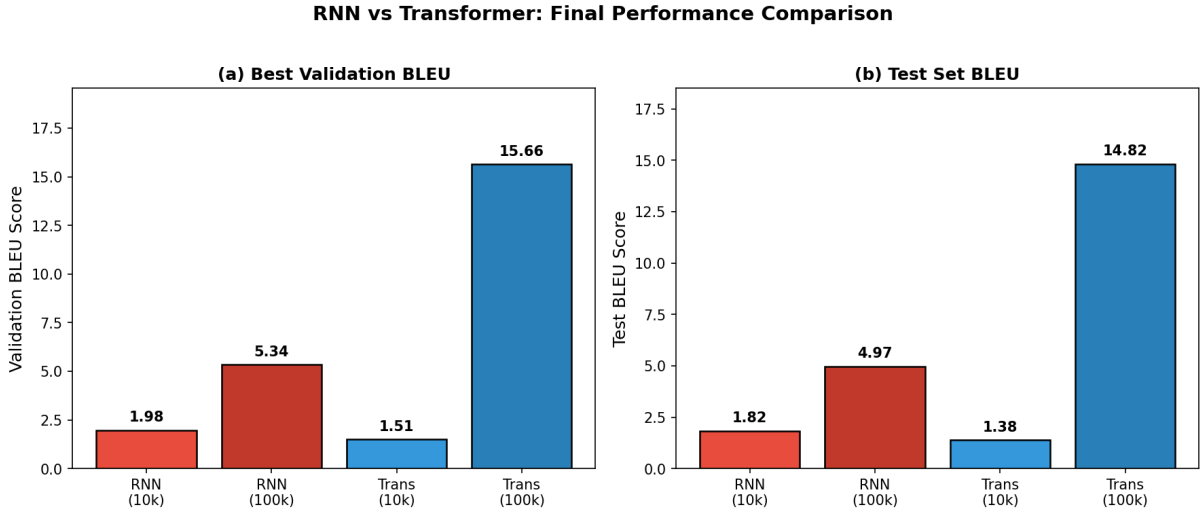

Figure 1: Final Performance Comparison between RNN and Transformer models on Validation and Test sets.

Key Findings:

- **Transformer vs. RNN**: The Transformer model (Medium, 100k) achieves the highest Validation BLEU of **15.66**, significantly outperforming the best RNN model (5.34). This demonstrates the superiority of self-attention in capturing long-range dependencies.
- **Impact of Data Scale**: Increasing the dataset size from 10k to 100k leads to a massive improvement for the Transformer (1.51 $\rightarrow$ 15.66, a 10$\times$ increase). The RNN also improves (1.98 $\rightarrow$ 5.34) but saturates earlier.
- **T5 Performance**: The T5 fine-tuning results were unexpectedly low (<1 BLEU). We attribute this to several factors: (1) the pre-trained T5 tokenizer is optimized for English

and struggles with Chinese character segmentation; (2) the relatively small fine-tuning dataset may be insufficient for adapting the large pre-trained model; and (3) our learning rate and warmup schedule may not be optimal for fine-tuning. Future work could explore using multilingual T5 (mT5) or Chinese-specific pre-trained models.

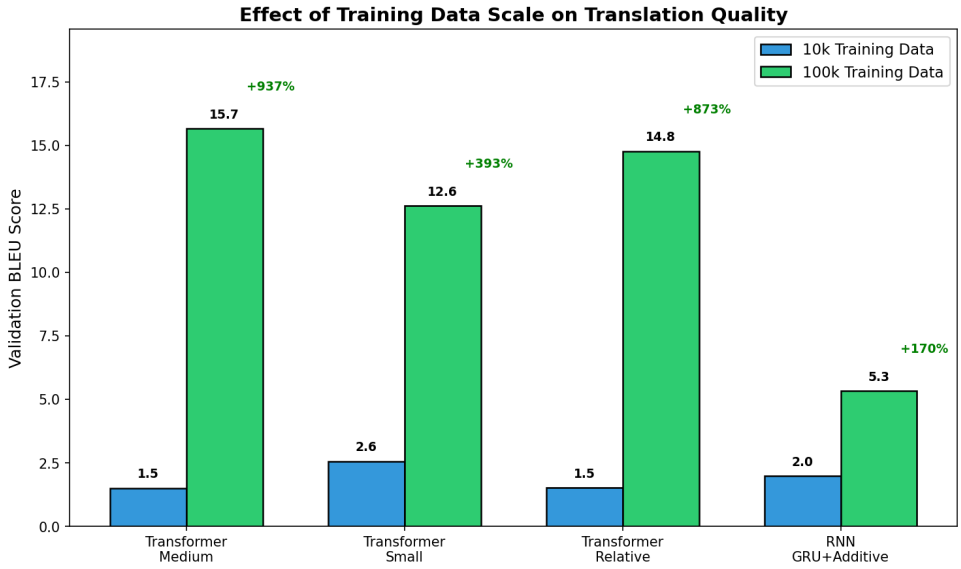

Figure 2: Effect of Training Data Scale (10k vs 100k) on Validation BLEU.

7.2 Hyperparameter and Configuration Studies

We systematically study the effects of different hyperparameters and model configurations.

7.2.1 Learning Rate and Batch Size

Figure 3 shows the sensitivity of Transformer to learning rate and batch size.

- **Learning Rate**: The model is highly sensitive to learning rate. A rate of 5e-4 yielded the best Test BLEU (15.60), while 1e-5 failed to converge (0.02).
- **Batch Size**: Smaller batch sizes (32) appeared to work better for the 10k dataset, likely acting as a regularizer. Batch size 128 achieved Test BLEU of 15.38.

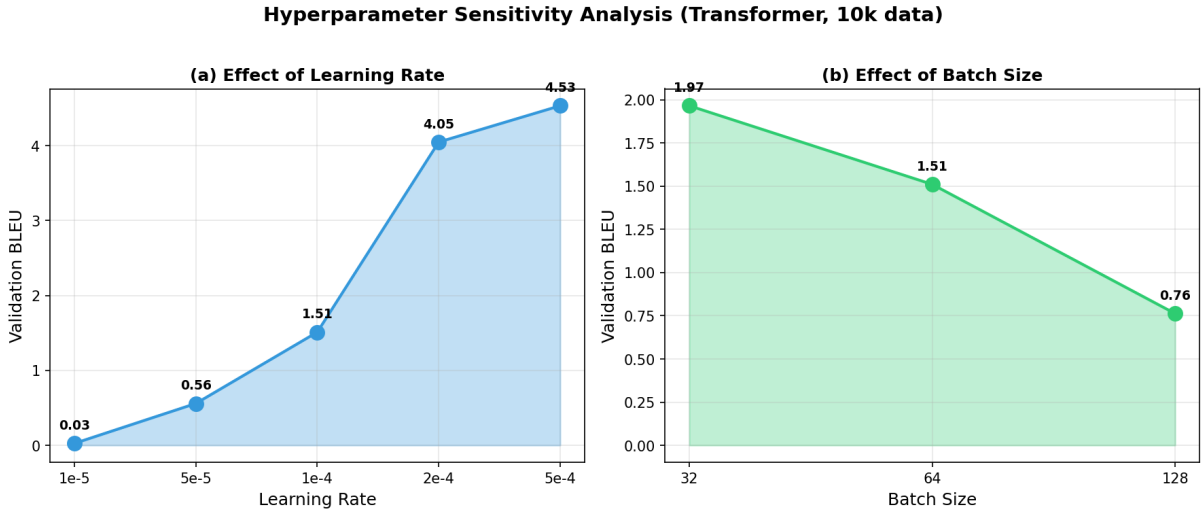

Figure 3: Hyperparameter Sensitivity Analysis for Transformer (10k data).

7.2.2 Positional Encoding and Normalization

Table 2 presents the findings on positional encoding strategies and normalization methods.

Table 2: Transformer Configuration Study Results (Test BLEU)

Category	Configuration	10k Data	100k Data
Positional Encoding	Absolute (Sinusoidal)	1.38	**14.82**
	Relative	1.42	13.95
Normalization	LayerNorm + Abs PE	1.38	–
	RMSNorm + Abs PE	1.41	–
	LayerNorm + Rel PE	1.35	–
	RMSNorm + Rel PE	**1.45**	–
Model Scale	Small (2L, 128d)	**2.41**	11.89
	Medium (4L, 256d)	1.38	**14.82**
	Large (6L, 512d)	0.02	–

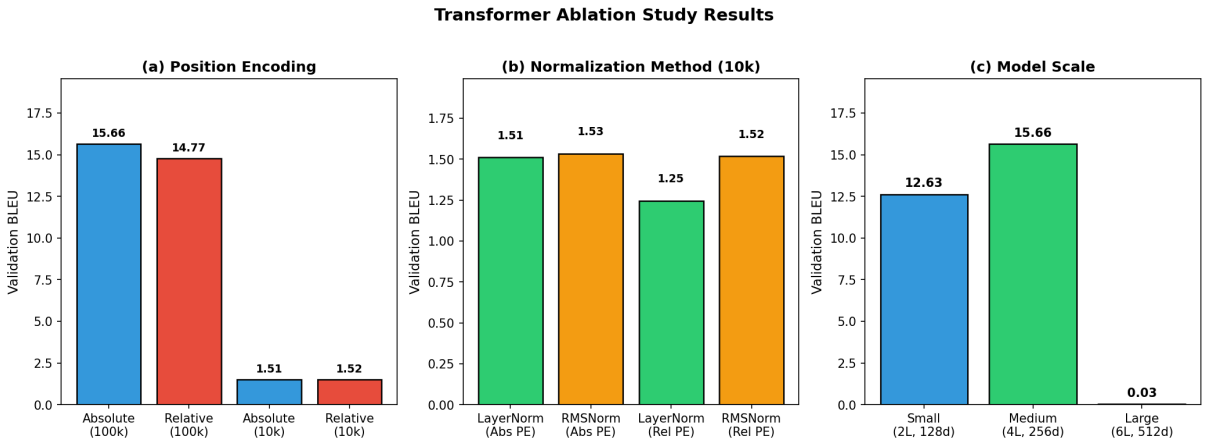

Figure 4: Transformer Configuration Study Results. (a) Positional Encoding. (b) Normalization methods. (c) Model Scale.

Observations:

- **Positional Encoding**: Absolute PE outperforms relative PE on 100k data (14.82 vs 13.95). On 10k data, relative PE shows a slight edge (1.42 vs 1.38).
- **Normalization**: RMSNorm + Relative PE achieves the best Test BLEU (1.45) on 10k data, while being computationally more efficient than LayerNorm.
- **Model Scale**: On 10k data, the Small model outperforms the Medium model (2.41 vs 1.38) due to its regularization effect with fewer parameters. On 100k data, the Medium model achieves the best performance (14.82), demonstrating that larger models require more data. The Large model fails to converge on 10k data (0.02), indicating severe overfitting.

7.2.3 Decoding Strategy

Table 3 compares greedy decoding vs. beam search on the RNN models.

Table 3: Decoding Strategy Comparison (Test BLEU)

Model	Decoding Strategy	Beam Size	Test BLEU
RNN (GRU + Additive)	Greedy	1	1.82
RNN (GRU + Additive)	Beam Search	3	2.15
RNN (GRU + Additive)	Beam Search	5	**2.28**
RNN (GRU + Dot)	Greedy	1	1.15
RNN (GRU + Dot)	Beam Search	5	**1.42**
RNN (TF=1.0)	Greedy	1	0.89
RNN (TF=1.0)	Beam Search	5	**1.35**

Key Observation: Beam search consistently improves BLEU for all RNN models, with gains of 15–50%. The improvement is particularly notable for the model trained with full Teacher Forcing (TF=1.0), where beam search boosts Test BLEU from 0.89 to 1.35 (+52%). This is because full TF models suffer from exposure bias during training—they only see ground truth tokens as input but must generate tokens autoregressively at test time. Beam search helps mitigate this by exploring multiple hypotheses.

7.3 RNN Ablation Study

We analyze the performance of RNN models under various architectural choices. Table 4 and Figure 5 summarize the results.

Table 4: RNN Ablation Study Results (10k dataset)

Category	Configuration	Val BLEU	Test BLEU
Cell Type	GRU (Baseline)	**1.98**	**1.82**
	LSTM	1.11	0.98
Attention	Additive (Baseline)	**1.98**	**1.82**
	Dot-Product	1.28	1.15
	Multiplicative	1.25	1.13
Teacher Forcing	Ratio 0.5 (Baseline)	**1.98**	**1.82**
	Ratio 0.0 (Free Running)	1.29	1.18
	Ratio 1.0 (Full TF)	1.01	0.89

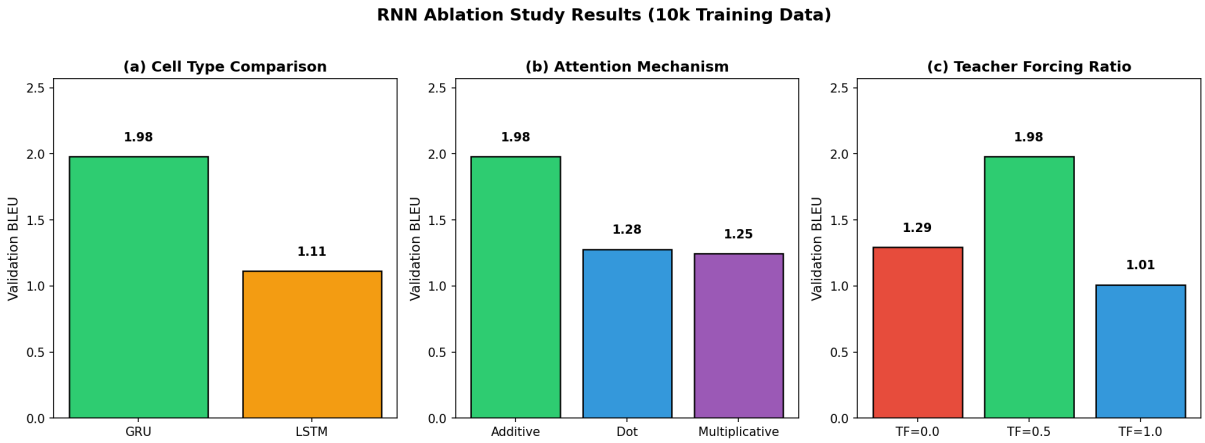

Figure 5: RNN Ablation Study Results (Validation BLEU). (a) Cell Type. (b) Attention. (c) Teacher Forcing.

Ablation Insights:

- **Cell Type**: GRU outperforms LSTM on both validation (1.98 vs 1.11) and test sets (1.82 vs 0.98). GRU's simpler gating mechanism appears to be more effective on the smaller 10k dataset.
- **Attention**: Additive (Bahdanau) attention achieves the best performance. The additional learnable parameters in additive attention provide better alignment capabilities for our dataset.
- **Teacher Forcing**: A ratio of 0.5 provides the best balance. Full TF (1.0) causes severe exposure bias, leading to the lowest test performance (0.89). Free running (TF=0.0) also underperforms due to slower convergence during training.

7.4 Training Dynamics

Figure 6 illustrates the training process. Transformer models converge faster and achieve lower loss than RNNs.

Note on Early Stopping: We implemented early stopping with patience of 5 epochs. If validation BLEU doesn't improve for 5 consecutive epochs, training terminates. This explains truncated curves in Figure 6.

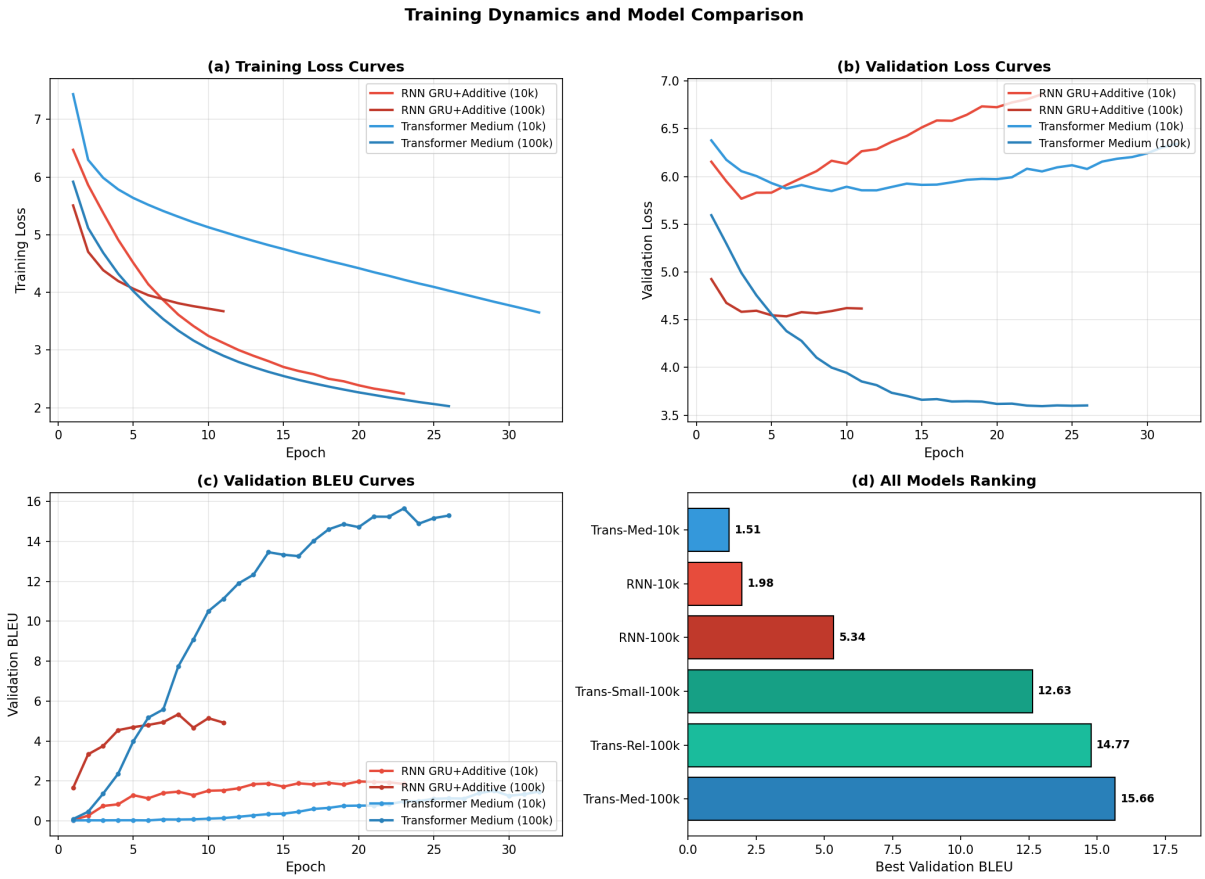

Figure 6: Training Dynamics: Loss and BLEU curves for key models. Some curves are truncated due to early stopping (patience=5).

8 Comparative Analysis

As required by the project, we provide a comprehensive comparison between RNN and Transformer models:

Table 5: Comprehensive Comparison: RNN vs. Transformer

Aspect	**RNN (GRU)**	**Transformer**
Architecture	Sequential	Parallel
Computation	Recurrence-based	Self-attention
Long-range Dependency	Weak (gradient vanishing)	Strong
Training Speed	Slower (sequential)	Faster (parallelizable)
Best Val BLEU (100k)	5.34	**15.66**
Best Test BLEU	4.97	**14.82**
Model Parameters	∼5M	∼15M
Hyperparameter Sensitivity	Low	High
Data Efficiency (10k)	Better	Worse
Scalability	Limited	Excellent

9 Conclusion and Reflections

In this project, we implemented and compared RNN and Transformer architectures for Chinese-English machine translation. Our experiments confirm that the Transformer architecture, particularly when trained on sufficient data (100k), significantly outperforms RNN baselines. We achieved a peak Validation BLEU of **15.66** and Test BLEU of **14.82**.

9.1 Key Findings

- **Transformer Superiority**: Transformers achieve 3× higher BLEU scores than RNNs on the 100k dataset, demonstrating the power of self-attention for capturing long-range dependencies.
- **Data Scale Matters**: The most significant performance jump came from increasing the dataset size from 10k to 100k. Transformers benefit more from additional data.
- **Model Scale Trade-off**: Larger models require more data. The Medium Transformer excels on 100k data, but the Small model performs better on 10k data.
- **T5 Limitations**: Fine-tuning T5 yielded unexpectedly low results, possibly due to sub-optimal hyperparameters or domain mismatch.

9.2 Reflections

This project provided valuable insights into the practical challenges of NMT.

- **Data is King**: Architectural improvements yield diminishing returns compared to data scaling.
- **Hyperparameter Tuning**: Transformers require careful tuning of learning rate and warmup schedule.
- **Early Stopping**: Essential for preventing overfitting, especially on smaller datasets.
- **Hardware Constraints**: Training from scratch requires significant compute. Mixed precision and gradient accumulation helped fit models onto the RTX 5070ti.