Modeling Semantic Relationships for Natural Language Inference

Ridwanul Hasan Tanvir
Department of Computer Science and Engineering
Pennsylvania State University
rpt5409@psu.edu

Abstract

We address the problem of natural language inferenceby developing a Bidirectional Long Short-Term Memory(BiLSTM) model. Our approach leverages pretrained GloVe embeddings for word representation and optimizes model performance through dropout regularization and hyperparameter tuning. Further analysis explores the impact of attention mechanisms and model generalization, paving the way for improved entailment detection without relying on Transformer-based architectures.

1 Problem Definition

Natural Language Inference (NLI) is a fundamental task in Natural Language Processing (NLP), where a model determines the logical relationship between a given *premise* and a *hypothesis*. Given a premise sentence P and a hypothesis sentence H, the objective is to classify their relationship into one of the following categories:

- **Entailment:** H logically follows from P.
- **Contradiction:** H contradicts P.
- **Neutral:** H and P are unrelated or have no definite inference relation.

Formally, given a dataset $D = \{(P_i, H_i, y_i)\}_{i=1}^{N}$, where $y_i \in \{0, 1, 2\}$ denotes the label for entailment, contradiction, or neutrality, the goal is to learn a function $f_\theta : (P, H) \rightarrow y$ that maps a sentence pair to a correct inference label.

The challenge in NLI arises from lexical ambiguity, syntactic variations, and the need for deep semantic reasoning.

2 Dataset

We use the Stanford Natural Language Inference (SNLI) dataset [1], a large-scale corpus designed for evaluating natural language inference (NLI) models. SNLI consists of human-annotated sentence pairs, each labeled with one of three inference relations: Entailment, Contradiction, or Neutral. Each sample comprises:

- A **premise** P, a sentence taken from an image caption dataset.
- A **hypothesis** H, a new sentence written by human annotators based on the premise.
- A **label** $y \in \{0, 1, 2\}$, where 0 represents Entailment, 1 represents Contradiction, and 2 represents Neutral.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

The dataset is split into predefined training, validation, and test sets:

- **Training Set:** 549,367 sentence pairs.
- **Validation Set:** 10,000 sentence pairs.
- **Test Set:** 10,000 sentence pairs.

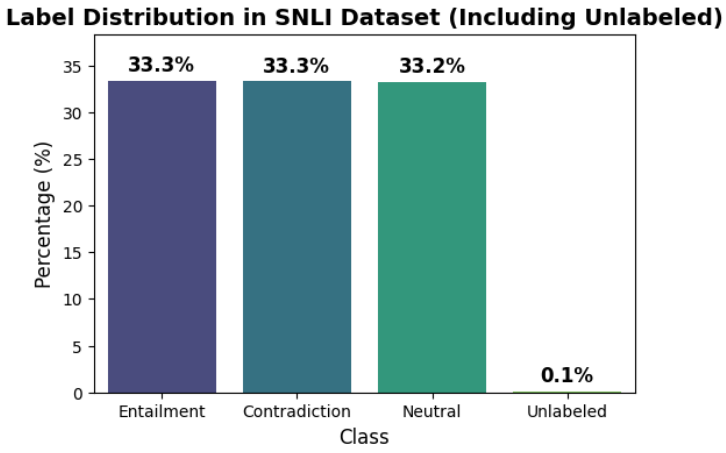

Figure 1: Label distribution in the SNLI dataset, including unlabeled data. The dataset is approximately balanced across the three main classes: Entailment, Contradiction, and Neutral.

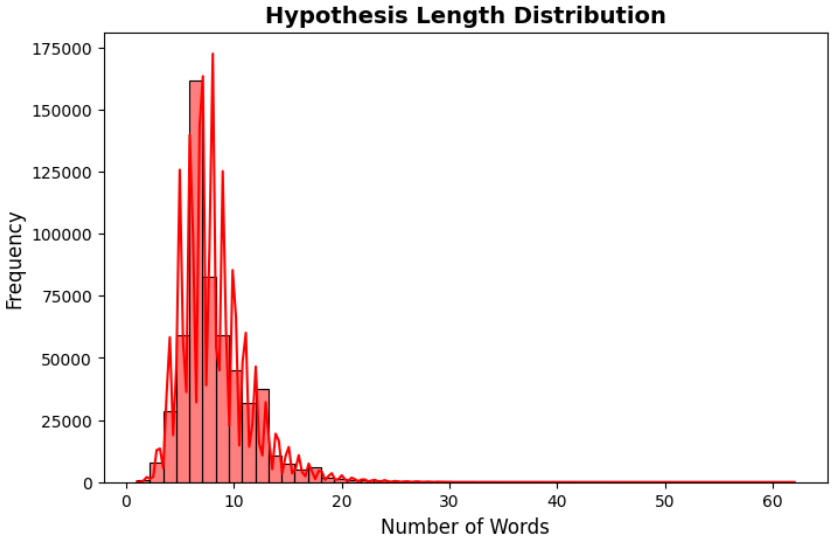

Figure 2: Histogram showing the distribution of hypothesis sentence lengths in the SNLI dataset. Most hypotheses are relatively short.

3 Preprocessing

Effective preprocessing is essential for ensuring that the model can generalize well on the Natural Language Inference (NLI) task. We apply a series of preprocessing steps to clean and normalize the Stanford Natural Language Inference (SNLI) dataset before passing it to the model.

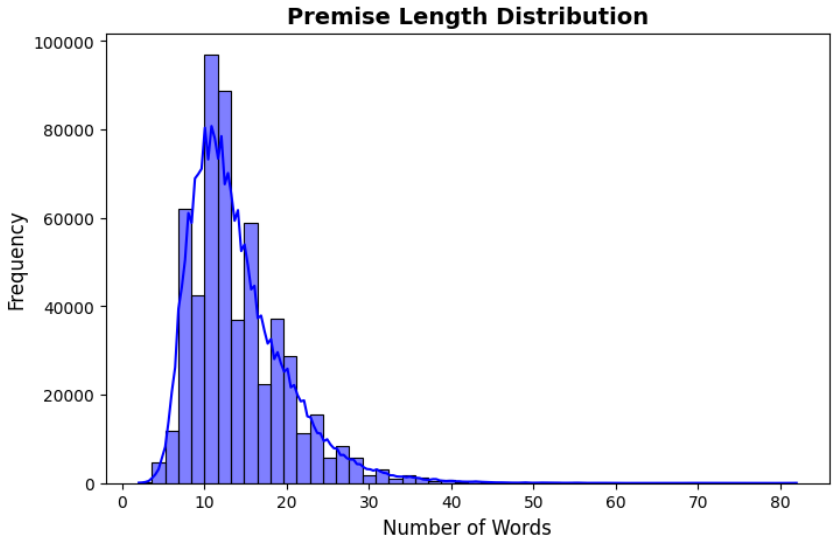

Figure 3: Histogram showing the distribution of premise sentence lengths in the SNLI dataset. Premises tend to be longer than hypotheses.

3.1 Tokenization

Each input sentence, whether a premise P or hypothesis H, is tokenized into individual words. We utilize the standard NLTK tokenizer, which ensures consistency in handling contractions, punctuation, and whitespace.

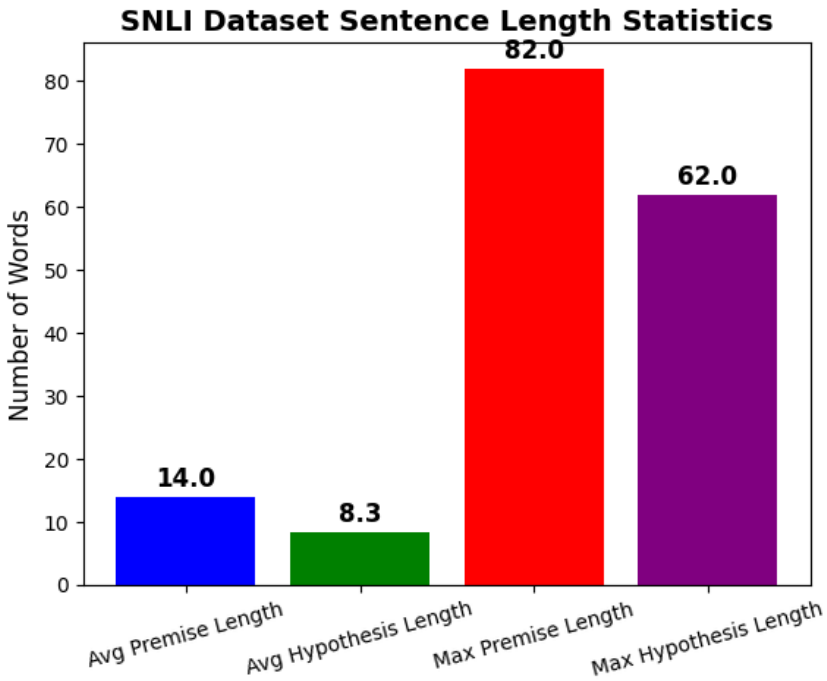

Figure 4: Sentence length statistics for SNLI dataset, showing the average and maximum lengths of premises and hypotheses.

3.2 Lowercasing

All text is converted to lowercase to reduce the sparsity in the vocabulary. This prevents the model from treating words such as *"The"* and *"the"* as distinct entities.

3.3 Punctuation Removal

Punctuation marks (*.,!?;*) are removed since they do not contribute to the semantic understanding required for inference classification. This step prevents unnecessary noise in the embeddings.

3.4 Label Encoding

We remove all unlabelled premises and hypotheses. So if the label is "-1" it is removed. Each label in the SNLI dataset is mapped to a numerical representation:

$$\begin{aligned} \text{Entailment} &\rightarrow 0, \\ \text{Contradiction} &\rightarrow 1, \\ \text{Neutral} &\rightarrow 2. \end{aligned}$$

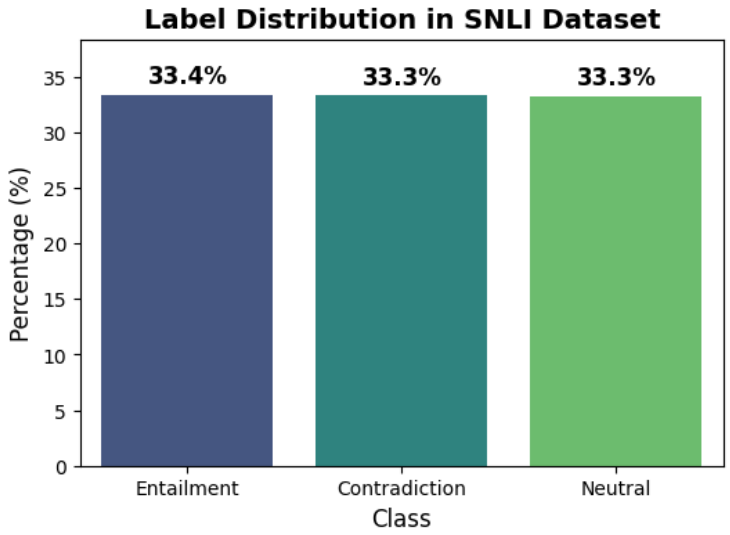

Figure 5: Class distribution in the SNLI dataset. The dataset is balanced across the three categories: Entailment, Contradiction, and Neutral.

3.5 Handling Out-of-Vocabulary (OOV) Words

Words that are not present in the pretrained GloVe embeddings are assigned a special token `<UNK>`. This ensures that the model can still process unseen words without failing.

3.6 Sentence Padding and Truncation

To handle varying sentence lengths, we pad or truncate each sentence to a fixed length L, where:

$$L = \max(\text{sentence length distribution})$$

Shorter sentences are padded with a special `<PAD>` token, while longer sentences are truncated. This step standardizes input size, ensuring efficient batch processing.

4 Algorithm and Training

Algorithm 1 BiLSTMWithAttention: Frozen, Fine-Tuned, and Scratch

Initialize:
  Define model parameters: *embedding_dim*, *hidden_dim*, *output_dim*
  Define optimizer and loss function: *Adam optimizer*, *CrossEntropyLoss*
  Define number of epochs: *num_epochs*
  Define batch size: *batch_size*
Initialize BiLSTMWithAttention models:
  model_frozen:
    Initialized with pretrained word embeddings, which remain frozen during training.
  model_finetune:
    Initialized with pretrained word embeddings, which are updated during training.
  model_scratch:
    Initialized with randomly generated word embeddings, which are trained from scratch.
for each model in {*model_frozen*, *model_finetune*, *model_scratch*} **do**
  for epoch = 1 to num_epochs **do**
    for each mini-batch $(prem, hyp, labels)$ in train_loader **do**
      Reset gradients: *optimizer.zero_grad*()
      Forward pass: $outputs \leftarrow model(prem, hyp)$
      Compute loss: $loss \leftarrow$ CrossEntropyLoss$(outputs, labels)$
      Backpropagate: $loss$.backward()
      Update model parameters: *optimizer.step*()
    end for
    Evaluate $model$ on val_loader
  end for
  Evaluate $model$ on $test\_loader$
end for

4.1 Word Embeddings and Model Architecture

Word Embeddings. We employ pre-trained GloVe embeddings as the basis for our model. These embeddings are either:

- **Frozen:** The embeddings remain fixed, preserving their original pre-trained representations.
- **Fine-tuned:** The embeddings are allowed to update during backpropagation, enabling task-specific adaptation.

Additionally, in one variant, we initialize the embeddings from scratch (random initialization) to compare against pre-trained embeddings.

BiLSTM: A bidirectional LSTM (BiLSTM) is chosen due to its ability to capture long-range dependencies from both the forward and backward directions in a sequence. Natural Language Inference (NLI) tasks often hinge on subtle contextual cues found throughout entire sentences. By processing each sentence in both directions, the BiLSTM more effectively captures these context-sensitive features compared to a unidirectional model.

Attention Mechanism. On top of the BiLSTM outputs, we integrate a simple attention mechanism. Attention allows the model to learn which tokens in the premise and hypothesis are most important for predicting the relationship (entailment, contradiction, or neutral). By computing attention scores over the BiLSTM outputs, the model focuses on critical parts of the sequence before making a final classification decision. In essence, the attention layer assigns higher weights to more relevant words and phrases.

4.2 Training Configuration and Optimization

We train all models with mini-batch stochastic gradient descent (SGD) via the Adam optimizer. Specifically, we use cross-entropy loss for multi-class classification, given three labels: entailment,

neutral, and contradiction. Hyperparameters, such as the learning rate and number of epochs, are determined empirically by monitoring validation accuracy and loss. We use standard techniques, including gradient clipping (where appropriate) and dropout, to prevent overfitting.

The Adam optimizer is used due to its adaptive learning rate mechanism, which helps in stabilizing updates and accelerating convergence. Adam combines the benefits of both momentum and adaptive learning rate methods, making it effective for handling sparse gradients, which are common in natural language processing tasks. In our implementation, we set the learning rate to 0.0001 to ensure gradual and stable learning, avoiding sudden fluctuations in model performance.

To prevent overfitting, we incorporate dropout in the fully connected layers, setting a dropout probability of 0.3. Dropout randomly deactivates a portion of neurons during training, forcing the model to learn more robust representations by reducing reliance on specific neurons.

4.3 Motivation for Three Variants

Model_frozen (Pre-trained + Frozen). We keep the pre-trained GloVe embeddings fixed. This provides a baseline performance to compare whether fine-tuning actually offers gains in representation quality.

Model_finetune (Pre-trained + Fine-tuned). Here, the pre-trained embeddings are allowed to update during training. If the target domain or task data significantly differ from the embeddings' original training corpus, fine-tuning can yield meaningful performance improvements, as it adapts the embedding space to the current task.

Model_scratch (Random Initialization). Finally, we train embeddings entirely from scratch without leveraging any external linguistic knowledge. This variant helps us see how much benefit is gained from using pre-trained embeddings.

5 Results Analysis

We compare the three variants of our BiLSTMWithAttention model—frozen embeddings, fine-tuned embeddings, and randomly initialized embeddings.

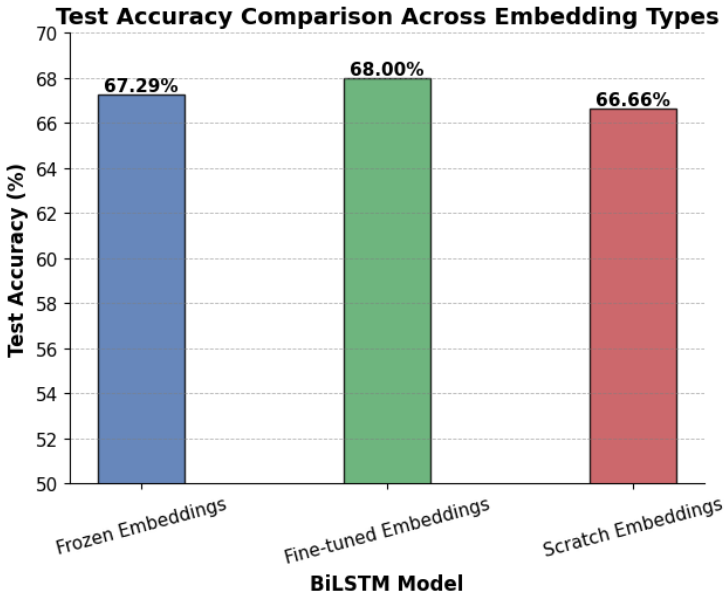

Figure 6: Test accuracy comparison of Frozen, Fine-tuned, and Scratch embeddings.

Fine-tuned vs. Frozen Embeddings. Among all the models, the version with fine-tuned GloVe embeddings achieved the highest test accuracy (68.00%). This indicates that allowing the embedding layer to update its parameters during training provides more task-specific representations, helping the model learn subtle relationships necessary for Natural Language Inference (NLI). By contrast, the frozen model reached a slightly lower test accuracy (67.29%), despite benefitting from high-quality pre-trained embeddings. This gap highlights that even strong pre-trained word vectors can be further improved by adaptation to the target domain and task.

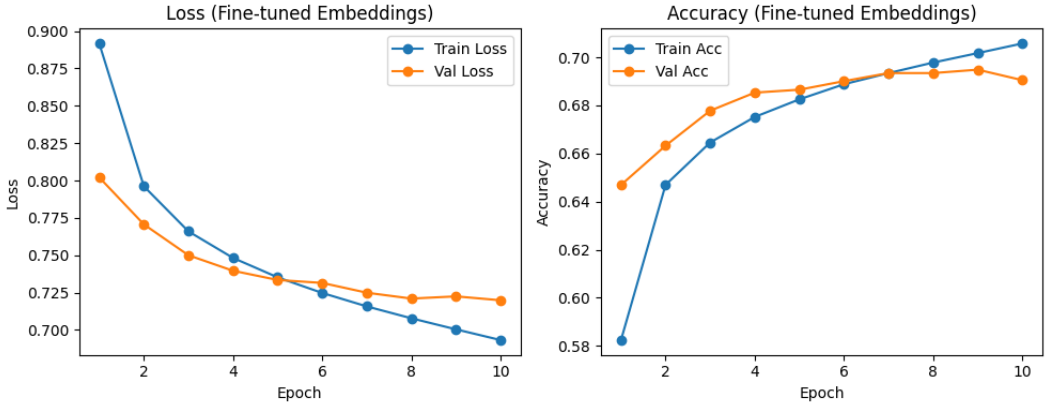

Figure 7: Training and validation loss/accuracy curves for the **Fine-tuned Embeddings** model.

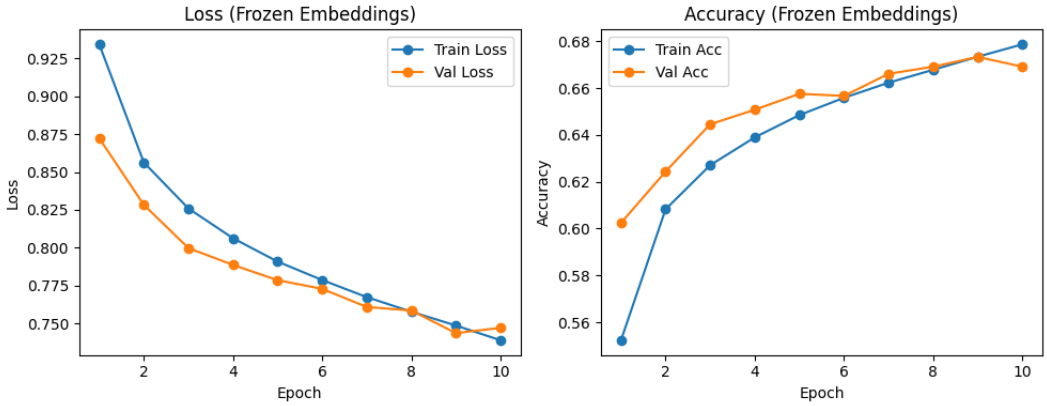

Figure 8: Training and validation loss/accuracy curves for the **Frozen Embeddings** model.

Frozen vs. Scratch Embeddings. Although the frozen embeddings do not adapt to our specific task, they still surpass the random (scratch) initialization. This is expected because the GloVe vectors are trained on a large corpus, yielding a robust semantic space from the outset. By leveraging this prior knowledge, the frozen model already encodes valuable semantic signals. In contrast, the scratch model starts with no linguistic biases, forcing it to learn both low-level word representations and higher-level semantic relationships in parallel, which is inherently more challenging.

Domain Adaptation. Even if the SNLI text domain partially overlaps with the corpus used to train GloVe, subtle domain differences (e.g., the style of premise-hypothesis pairs) may require specific tuning. Fine-tuning captures these nuances better than freezing weights.

Model Performance Overall, the best-performing approach involved **fine-tuning pre-trained GloVe embeddings**, underscoring the value of allowing domain adaptation on top of large-scale pre-trained word vectors. The experiments confirm the common finding in NLP research that **pre-trained word embeddings can significantly boost performance**, particularly for tasks requiring nuanced semantic understanding like NLI.

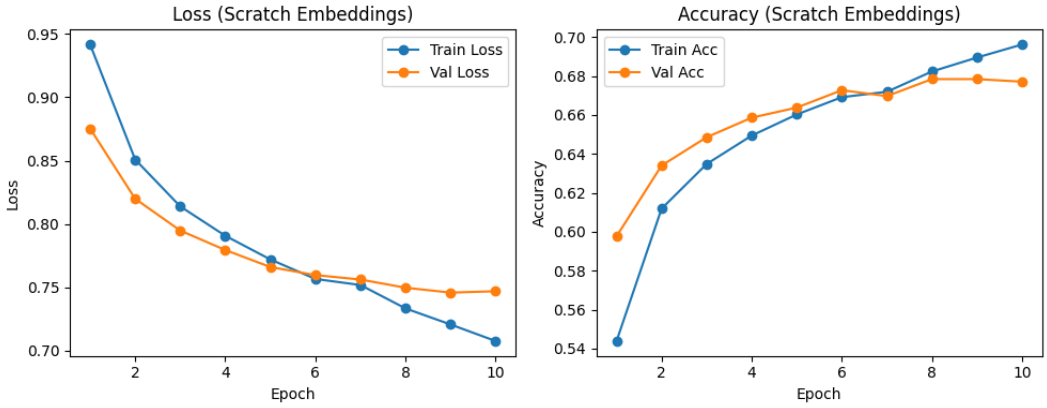

Figure 9: Training and validation loss/accuracy curves for the **Scratch Embeddings** model.

6 Future Work

As part of our efforts to further improve model performance, we explored a Stacked BiLSTM architecture. This approach builds on the standard BiLSTM by adding multiple stacked layers, allowing the model to capture deeper hierarchical representations of textual data.

The primary motivation behind stacking multiple BiLSTM layers was to enhance the model's ability to capture long-range dependencies and complex sentence structures, which are crucial for Natural Language Inference (NLI). While a single-layer BiLSTM is capable of modeling sequential dependencies, deeper architectures often enable better feature extraction, leading to improved generalization.

In our Stacked BiLSTM model, we increased the number of BiLSTM layers to **2**, allowing the network to learn more abstract hierarchical representations of sentence pairs. The hidden dimension was kept the same as in the single-layer BiLSTM to ensure a fair comparison, and dropout was applied between layers to mitigate overfitting and maintain stable training. This architecture enables the first BiLSTM layer to capture lower-level lexical and syntactic features, while the second BiLSTM layer refines these representations to better model sentence relationships for the NLI task.

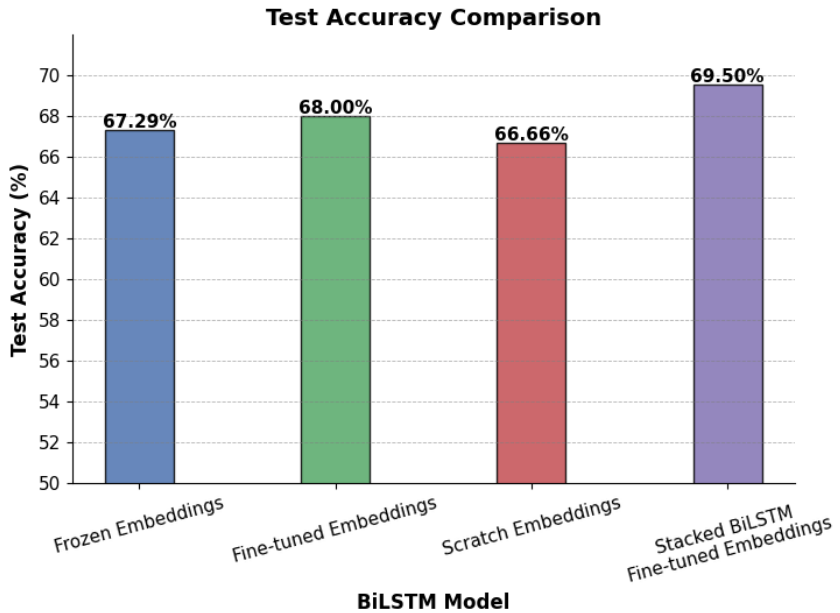

Figure 10: Test Accuracy Comparisons. The Stacked BiLSTM with fine-tuned embeddings achieves the highest accuracy.

The Stacked BiLSTM model achieved a test accuracy of **69.50%**, outperforming all previous models, including the single-layer fine-tuned BiLSTM (**68.00%**). With multiple stacked layers, the model can learn low-level lexical patterns in the first layer and higher-level semantic relationships in deeper layers.

Limitations and Future Exploration Despite the improvements, there were several constraints during our experimentation, primarily due to hardware limitations:

- The model was trained and tested on **Google Colab**, which imposed restrictions on available GPU resources and session timeouts.
- Due to these limitations, we could only test the Stacked BiLSTM with fine-tuned embeddings.
- Higher depth models typically require more training time to fully converge. Due to resource constraints, we could not train with a significantly higher number of epochs, though we presume that performance could improve with extended training.
- While we used a modestly deeper BiLSTM architecture, we were unable to experiment with even deeper networks due to memory and computational restrictions.

7 Disclaimer

This report acknowledges the use of AI-based tool ChatGPT[2] for rephrasing certain sections to enhance clarity and readability. Although this project was conducted individually, the pronoun "we" has been used for generality and consistency in academic writing. Additionally, all code implementations and experiments were executed in the free version of Google Colab[3], which imposed certain computational limitations.

References

[1] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642. Association for Computational Linguistics, 2015.

[2] OpenAI. Chatgpt: A large language model for conversational ai, 2024. Accessed: February 2025.

[3] Google Research. Google colaboratory: Cloud-based python notebook environment, 2024. Accessed: February 2025.