Punctuation Prediction using BiLSTM and BiLSTM with Transfer Learning

by
Abhinav Mittal
9922103103

TABLE OF CONTENTS

1. Introduction

1.1 General Introduction

Text produced by automatic speech recognition (ASR), chat logs, and OCR often arrives with little or no punctuation. Missing commas and sentence boundaries make the text hard to read and also hurt downstream tasks such as sentence splitting, summarization, machine translation, and information retrieval. Punctuation prediction (also called punctuation restoration) addresses this by inserting marks like commas and periods back into raw token streams so that the output reads like normal prose. In this report, I approach punctuation prediction as a lightweight, reproducible sequence-labeling problem and compare a plain BiLSTM baseline with a BiLSTM enhanced by transfer learning.

1.2 Problem Statement

Given a sequence of tokens $x_1 \ldots x_n$ without reliable punctuation, predict for each token a label y_i from a small set {O, COMMA, PERIOD, SEMICOLON} indicating the mark (if any) that should follow that token. Practical challenges include: (i) strong class imbalance (vast majority O), (ii) noisy casing/tokenization from ASR/OCR, (iii) long sequences that require context from both left and right, and (iv) out-of-vocabulary words. The goal is to maximize overall quality (macro/weighted F1) while keeping the model simple, fast to train, and easy to deploy for batch or streaming inference.

1.2 Brief Description of the Solution Approach

We treat punctuation restoration as a token-tagging task: for each word, predict a label from {O, COMMA, PERIOD, SEMICOLON}. The pipeline parses inline marks in the data and attaches each mark to the *preceding* token, then chunks long sequences, builds a word vocabulary, masks PAD positions, and uses class weights to handle the heavy “O” class. The baseline model is a compact BiLSTM (Embedding → BiLSTM → Linear → Softmax) that predicts one label per token. A second variant adds transfer learning by initializing the embedding layer with pretrained GloVe vectors and fine-tuning end-to-end, which typically improves comma decisions and speeds convergence. We train with class-weighted cross-entropy and report macro/weighted F1 and a confusion matrix; at inference, we reconstruct the text by appending the predicted mark after each token.

2. Literature Survey

Punctuation prediction is a key component in improving the usability of ASR outputs by restoring readability and supporting downstream NLP tasks. The field has evolved significantly, beginning with rule-based methods and statistical models and progressing into advanced neural architectures that use attention, hierarchical design, and pre-trained transformers.

The seminal work by Hochreiter and Schmidhuber [2] introduced Long Short-Term Memory (LSTM), which became foundational for sequence modeling tasks due to its ability to handle long-term dependencies. This was enhanced by Gers et al. [3], who proposed forget gates, further improving performance on sequential data tasks.

Tilk and Alumae [1], [9] were among the first to apply LSTMs and later Bidirectional LSTMs (BiLSTM) to punctuation prediction in speech transcripts, showing significant gains over prior approaches. Their later work incorporated prosodic cues into hierarchical LSTM models [10], highlighting the benefit of combining acoustic features with textual input.

More recently, transformer-based approaches have emerged. Lin et al. [17] showed that BERT-based contextual embeddings significantly improve punctuation recovery when compared to LSTM baselines. Their model leverages token-level embeddings from BERT, achieving state-of-the-art performance on ASR transcripts.

Che et al. [4] proposed a Gated Context Expansion Network, an end-to-end model designed to handle unsegmented ASR output. Their approach dynamically adjusts the receptive field for each token, outperforming static-window models. Similarly, Liu et al. [19] introduced Hierarchical Transformers, which combine sentence-level and document-level attention to better capture discourse-level punctuation.

For low-resource languages and multilingual settings, Chordia [5] and Salimbajevs [7] demonstrated that both transformer-based and BiLSTM models can be adapted using transfer learning and contextual embeddings. Vandeghinste and Guhr [13] extended this to Dutch using domain-specific transformers.

In cross-modal settings, Xu et al. [18] and Żelasko et al. [16] explored fusing audio and text modalities, improving punctuation prediction in spontaneous speech and noisy ASR outputs. These multimodal models capture speech intonation and pause information, complementing the semantic cues in text.

Zhu et al. [20] proposed a self-training approach to make punctuation prediction models robust against noisy labels and domain shifts. Their semi-supervised strategy generated pseudo-labeled data to enhance training in real-world scenarios.

Architectural optimizations also play a vital role. Xu et al. [15] conducted an exhaustive evaluation of LSTM architectures for punctuation, validating the efficacy of deeper, bidirectional configurations. Wang et al. [14] introduced Dilated Hierarchical Attention Networks to extract multi-scale context, enabling accurate prediction in dense speech.

Oktem et al. [6] leveraged Attentional Parallel RNNs to improve punctuation detection by combining global and local sequence modeling. Meanwhile, Chordia [5] and Lin et al. [17] demonstrated the power of transformer-based multilingual models to generalize across domains and languages.

From a usability standpoint, Li et al. [11] used eye-tracking studies to demonstrate the increased cognitive load of reading unpunctuated text. Salloum et al. [10] showed that punctuation significantly improves downstream NLP tasks like sentiment analysis and machine translation. Stolcke et al. [12] discussed the link between punctuation and speech acts, framing it as integral to understanding discourse.

In sum, the literature demonstrates a shift from traditional LSTM models to context-rich, multimodal, and transformer-based architectures. The chosen BiLSTM model in this project is grounded in this trajectory—offering a balance of simplicity, efficiency, and contextual awareness, while remaining extensible to multimodal and multilingual improvements in future work.

3. Basic Terminology and Definitions

3.1 Task Definition (Sequence Labeling vs Seq2Seq)

Sequence labeling (used in this work)

- **Goal**: For each token x in an input sequence x1.. xn, predict a label yt that tells which punctuation (if any) follows that token.
- **Label space:** {O,COMMA,PERIOD,SEMICOLON}
- **Attachment rule:** The predicted label is applied after the same token.

Sequence-to-sequence (alternative, not used here)

- **Idea**: Generate fully punctuated text from raw text (e.g., Transformer encoder–decoder).
- **Pros**: flexible global context, multiple insertions possible.
- **Cons**: heavier, slower inference, alignment/hallucination risks, harder to evaluate.

Choice: We use sequence labeling for speed, reproducibility, and clean metrics.

3.2 Evaluation Metrics (Precision, Recall, F1; Macro vs Weighted)

Let TP, FP, FN be counts for class c (computed over **non-PAD** positions).

- **Precision = TP/(TP+FP)** — correctness of predicted c.
- **Recall = TP/(TP+FN)** — coverage of true c.
- **F1** — harmonic mean of precision and recall for c.

Averaging

- **Macro F1:** unweighted mean over classes — fair under imbalance, highlights punctuation classes.
- **Weighted F1:** mean weighted by class support — reflects overall distribution but can be dominated by **O**.

Reporting best practice: per-class P/R/F1 (O, COMMA, PERIOD, SEMICOLON) + **Macro** and **Weighted** F1; include a **confusion matrix** (esp. COMMA vs O vs PERIOD). Consider a **punctuation-only Macro F1** averaged over {COMMA, PERIOD, SEMICOLON}.

3.3 LSTM (Long Short-Term Memory)

- **What it is:** A recurrent unit that tracks information with a cell state and gates (input/forget/output) to combat vanishing gradients.
- **Why here:** Captures local syntax and medium-range dependencies needed for correct comma/period placement.
- **Outputs:** At each step ttt, the LSTM emits a hidden state hth_tht summarizing left-context (and, with BiLSTM, right-context too).

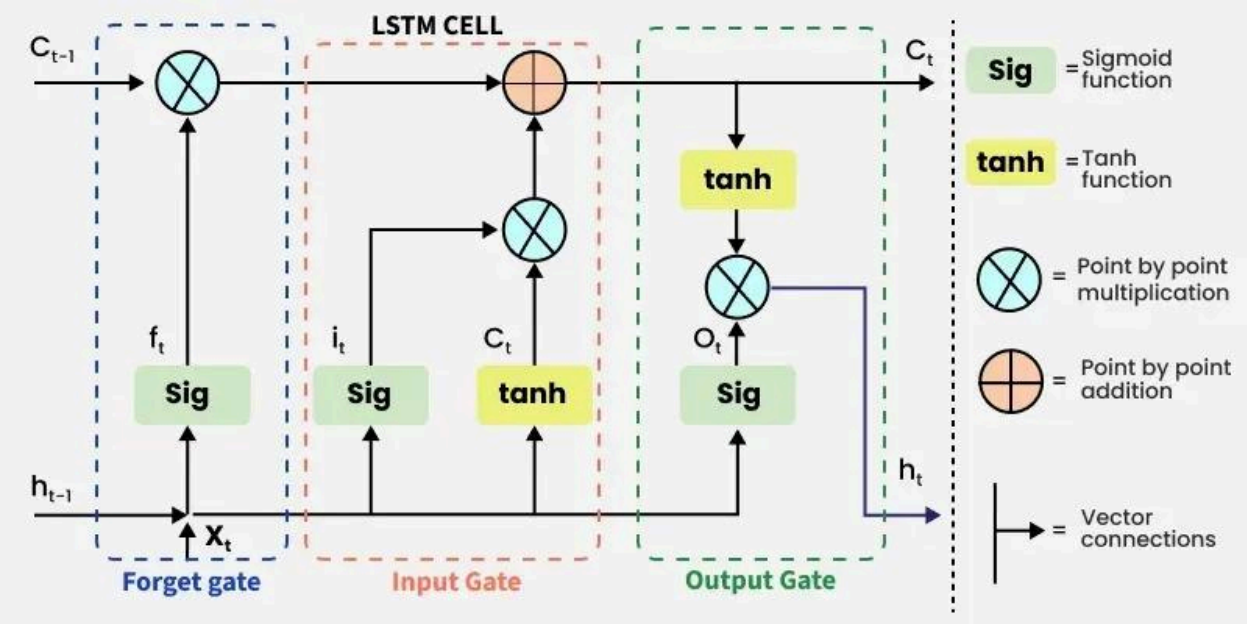

Figure 3.1: Architecture of Long Short Term Memory (LSTM)

3.4 BiLSTM (Bidirectional LSTM)

- **Definition:** Two LSTMs run in opposite directions over the sequence; their hidden states are concatenated [ht→;←ht] before classification.
- A Bidirectional LSTM pairs two LSTM networks over the same sequence: one processes tokens left→right (past context), the other right→left (future context). At each position, their hidden states are combined (typically concatenated), giving a context vector that summarizes both what came before and what comes next. This yields a per-token representation with symmetric context.
- **Why for punctuation:** Many marks depend on both preceding and upcoming words (e.g., clause boundaries). BiLSTM gives symmetric context with low latency and small footprint.

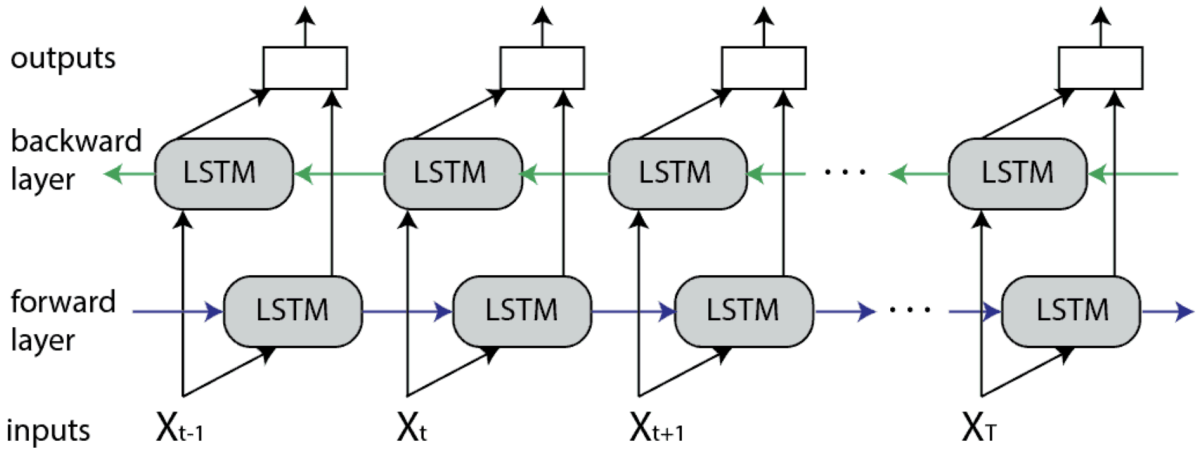

Figure 3.2: Architecture of Bi-Directional Long Short Term Memory (Bi - LSTM)

3.5 Transfer Learning (Pretrained Embeddings)

- **Idea:** Initialize the model's embedding layer with pretrained word vectors (e.g., GloVe 100d) for tokens present in the vocabulary; fine-tune during training.
- **Benefits:** Brings semantic priors from large corpora → faster convergence, better generalization, and typically better comma performance (which relies on subtler cues than periods).
- **Coverage:** OOV tokens fall back to randomly initialized vectors and are learned normally.
- **Implementation detail:** Keep the rest of the model identical (BiLSTM → Linear → Softmax) so results are directly comparable to the baseline.

4. About Dataset

4.1 Source and Domain

A) Baseline (LSTM/BiLSTM) — Live lecture transcripts (C programming)

- **Domain:** Spoken, lecture-style English from a live C-programming class.
- **Characteristics:** Conversational phrasing, fillers, repetitions, sentence fragments, and lightweight discourse markers (e.g., “so”, “okay”, “now”).
- **Punctuation profile:** Fewer commas and semicolons than formal prose; many long stretches with **O** (no punctuation) and clear **PERIOD** cues at topic shifts.

B) Transfer Learning variant (BiLSTM + pretrained embeddings) — Four C-programming books

- **Sources (prose content only):**
    - *The Basics of C Programming* — Marshall Brain
    - *Phil’s C Course*
    - *The C Programming Language* — Steve Summit
    - *C Programming* — Wikibooks
- **Characteristics:** Edited, formal exposition with examples and explanations.
- **Punctuation profile:** Richer comma usage (introductory clauses, appositives), occasional semicolons, consistent sentence boundaries—useful for learning finer punctuation cues.

Rationale for pairing: The lecture data reflects the target use case (restoring punctuation in speech-like text). The book corpus supplies cleaner, denser signals for commas and clause structure—ideal for transfer learning via pretrained embeddings and for improving generalization on tricky marks (especially COMMA).

4.2 Statistics and Distribution

For Basic BiLSTM,

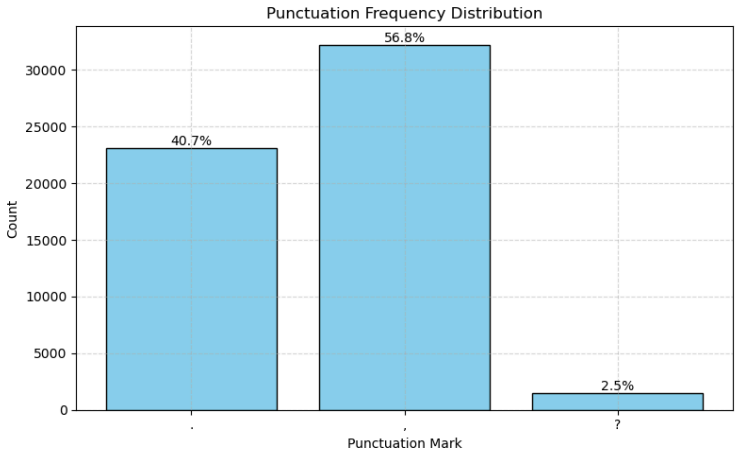

Figure 4.1: Punctuation Frequency Distribution for Basic BiLSTM Dataset

- Total Tokens: 511208
- Total Punctuations: 56781
- Punctuation to Token ratio: 0.11

For BiLSTM with Transfer Learning

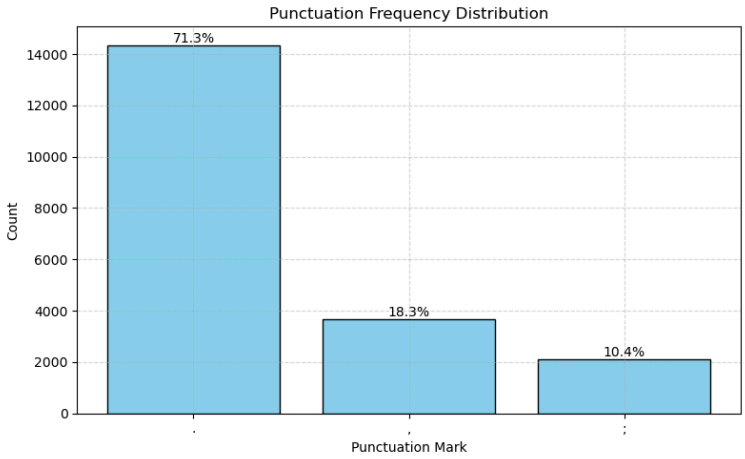

Figure 4.2: Punctuation Frequency Distribution for TL Dataset

- Total Tokens: 94240
- Total Punctuations: 20128
- Punctuation to Token ratio: 0.21

4.3 Preprocessing Pipeline (common to both)

1. **Text selection & cleanup**
    - Keep prose paragraphs; drop code blocks and obvious inline code.
    - Normalize whitespace, standardize quotes, optionally lowercase for a simpler vocabulary.
2. **Tokenization**
    - Word-level tokenization with a consistent tokenizer for train/val/test and inference.
    - Keep C-domain terms (e.g., “printf”, “pointer”) as normal tokens.
3. **Inline punctuation labels → per-token tags**
    - Convert inline marks to labels attached to the preceding token
4. **Label set**
    - Final label space: {O, COMMA, PERIOD, SEMICOLON}.
5. **Chunking & padding**
    - Break long streams at PERIOD or a fixed max length; pad shorter sequences with <pad>.
    - Mask <pad> in loss and metrics , so padding never affects learning or scores.
6. **Vocabulary**
    - Build a word vocabulary (V) with special tokens <pad> and <unk>.
    - For the transfer-learning model, initialize embeddings for in-vocab words that exist in the pretrained set (e.g., GloVe 100d); others fall back to random init.

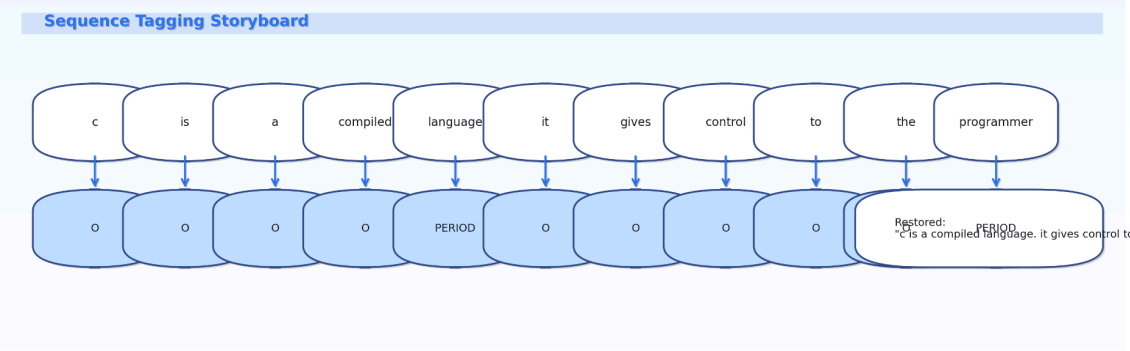

Figure 4.3: Sequence Tagging

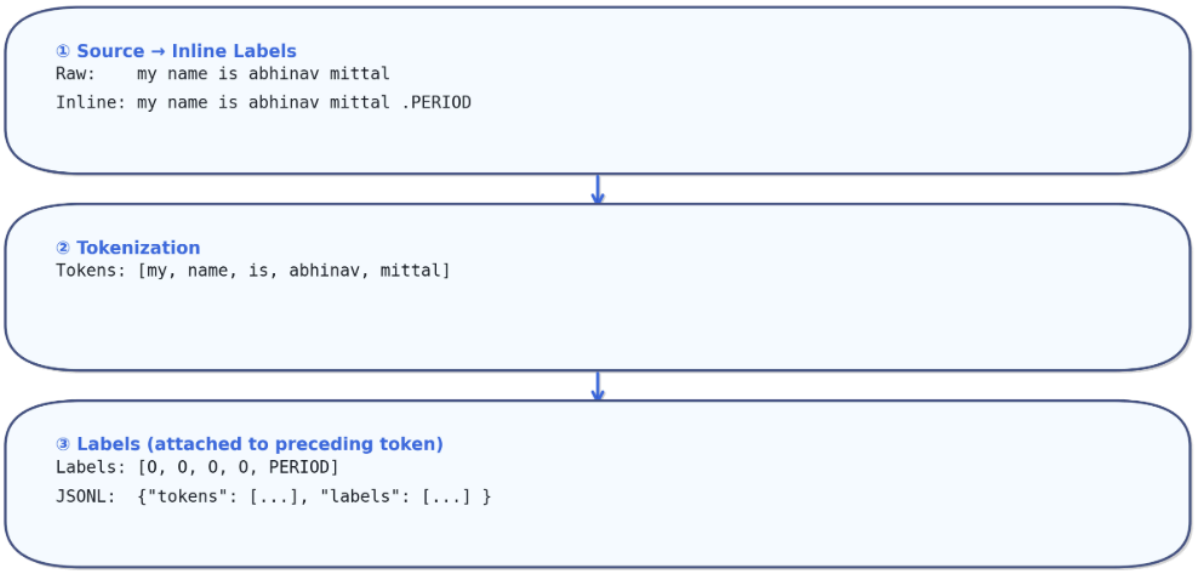

Figure 4.4: Text Preprocessing Example

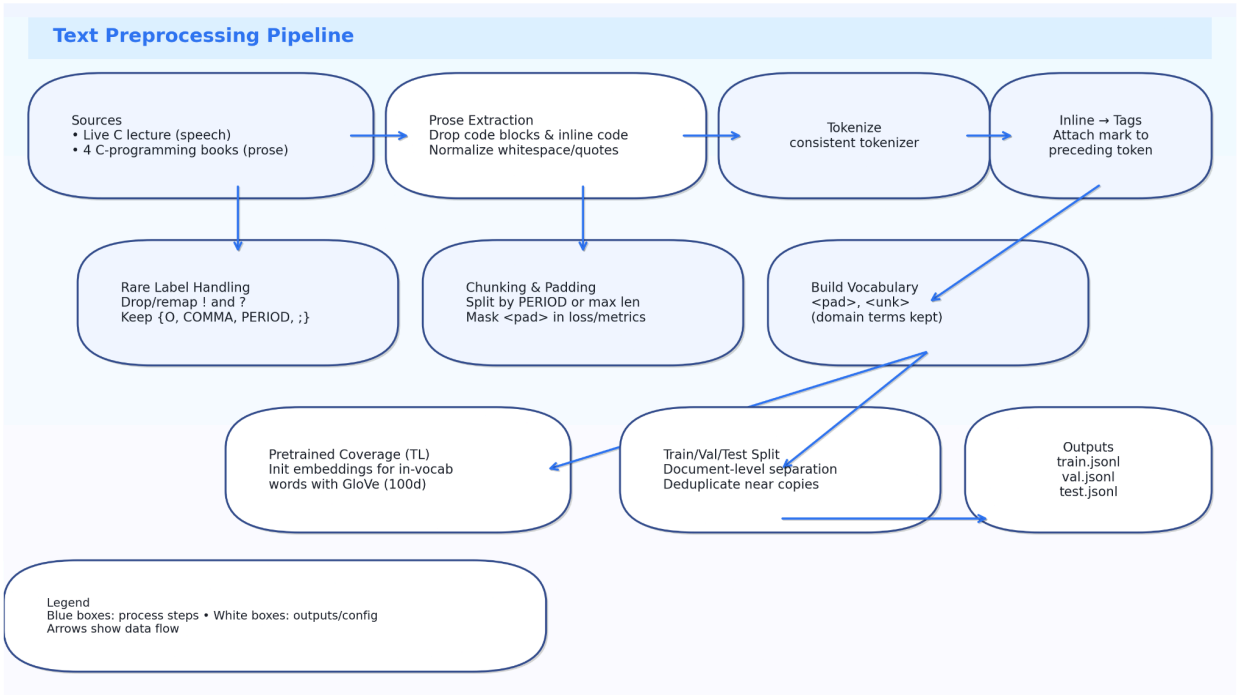

Figure 4.5: Text Preprocessing Pipeline

5. Methodology

5.1 Basic BiLSTM

Objective

We formulate punctuation restoration as a **sequence labeling** task: for each token in a sentence, predict the punctuation mark that follows it (none, comma, period, question mark).

Data Preparation

Input text is normalized (quote removal, whitespace cleanup) and tokenized to separate words and punctuation. A fixed vocabulary is built with special tokens (`<PAD>`, `<UNK>`, `<START>`, `<END>`). For each sentence, we create parallel sequences: token indices and punctuation labels (`O`, `,`, `.`, `?`). Sequences are padded or truncated to a fixed length `L` for mini-batching.

Model Architecture

We use a **BiLSTM** to capture left- and right-context. Each token index is embedded into a dense vector and fed to a two-layer bidirectional LSTM. We apply a **per-token attention gate** (a linear score + softmax across time) to emphasize informative positions, followed by dropout. Two fully connected layers project to the label space, producing per-token class probabilities.

Pipeline:
Embedding → BiLSTM (bidirectional) → Attention gate → Dropout → Dense → Softmax (per token)

Training

We train with **Adam** (learning rate 1e-3), gradient clipping (1.0), and either **class-weighted cross-entropy** or **Focal Loss (γ=2)** for class imbalance. A **ReduceLROnPlateau** scheduler halves the learning rate on validation F1 plateaus. The best model (by validation F1) is checkpointed. We log batch/epoch metrics in TensorBoard.

Evaluation

We report macro/weighted F1, per-class precision/recall/F1, and a confusion matrix on the test set. Qualitative evaluation includes example reconstructions (original vs. predicted punctuated text). Post-processing rules reduce spurious consecutive marks and ensure sentence endings.

Inference

At inference, the model predicts a punctuation label for each token. We then reconstruct the sentence by inserting the predicted marks after their corresponding tokens and apply light post-processing to improve readability.

Design Choices & Rationale

- **BiLSTM** captures both preceding and following context, which is crucial for punctuation.
- **Attention** emphasizes salient tokens (e.g., clause boundaries) without heavy complexity.
- **Focal/weighted loss** mitigates skew towards the dominant 0 class.
- **ReduceLROnPlateau** and gradient clipping stabilize training and improve convergence.

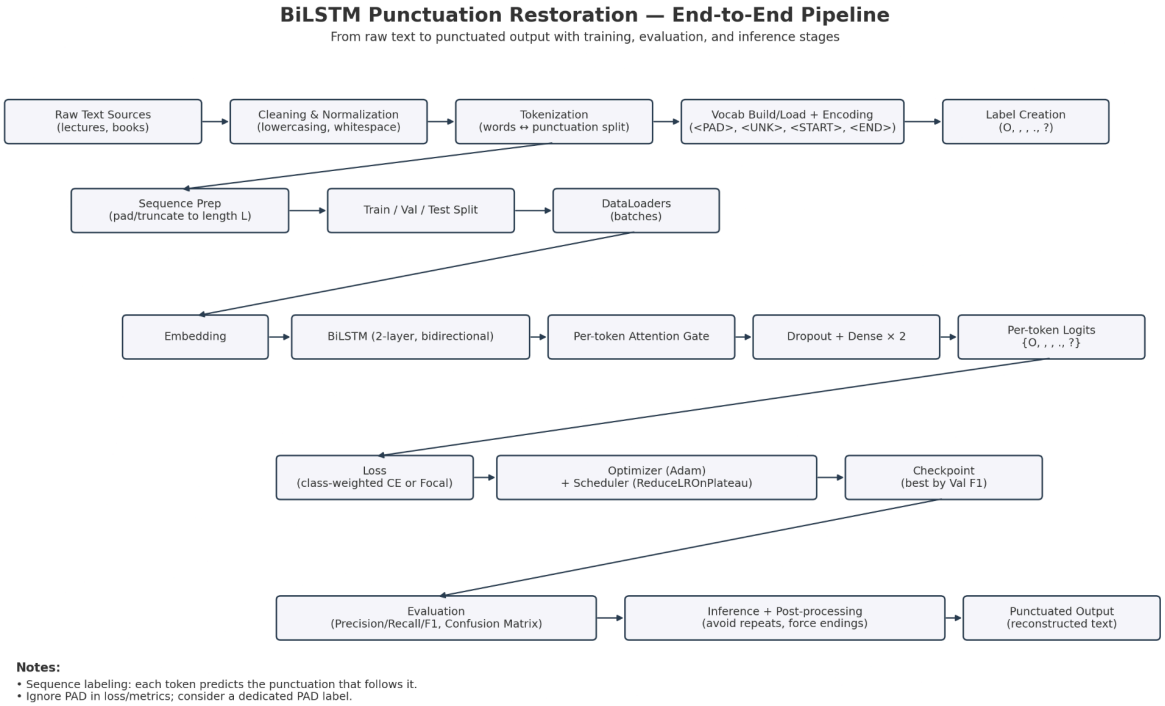

Figure 5.1: Pipeline of Basic BiLSTM

Model Architecture

1. **Input Processing**
    a. **Tokenization:** The transcript is broken into word tokens.
    b. **Vocabulary Mapping**: Each word is assigned an index.
    c. **Label Assignment**: Each word gets a corresponding punctuation label —"O" (no punctuation), " , ", " . ", " ? ".
2. **Padding and Batching**
    a. Sentences are padded or truncated to a fixed length.
    b. Batches are created using PyTorch's DataLoader.
3. **Embedding Layer**

a. Each token index is converted into a dense vector (e.g., 100D).
b. This captures the semantic meaning of words.

4. **BiLSTM Layer**
    a. Bidirectional LSTM reads both forward and backward contexts.
    b. Captures full sentence-level context around each word.
5. **Dropout Layer**
    a. Dropout is applied to reduce overfitting.
6. **Fully Connected Layer**
    a. Maps BiLSTM output to logits for each punctuation class.
7. **Softmax Activation**
    a. Converts logits into probability distributions across punctuation labels.
8. **Loss Computation**
    a. Weighted CrossEntropyLoss is used to deal with class imbalance.
    b. Higher weights are assigned to rare punctuation types.
9. **Backpropagation and Optimization**
    a. The model is trained using the **Adam optimizer**.
10. **Prediction and Evaluation**
    a. After training, the model is evaluated using:
        i. Precision, Recall, F1-score
        ii. Confusion matrix
        iii. Classification Report

5.2 BiLSTM with Transfer Learning

Overview. We extend the baseline BiLSTM tagger with transfer learning by initializing the embedding layer from pretrained GloVe vectors. The rest of the architecture—BiLSTM encoder, per-step attention, and per-token classifier—remains unchanged, allowing controlled comparison to the baseline.

Embedding initialization. A word vocabulary is built from the training data. We construct a $|V| \times 100$ embedding matrix by copying GloVe vectors for covered tokens, assigning zeros to <pad>, and sampling small random vectors for out-of-vocabulary items. The model's embedding weights are initialized from this matrix and fine-tuned during training (optionally with a slightly smaller learning rate for the embedding layer).

Architecture. Tokens are mapped to embeddings and encoded by a two-layer bidirectional LSTM (hidden 128 per direction). A lightweight per-timestep attention produces soft weights over the sequence to emphasize likely punctuation sites. The classifier applies dropout, a Linear(256→128) + ReLU, then Linear(128→4) to output logits over {O, COMMA, PERIOD, SEMICOLON} for each token.

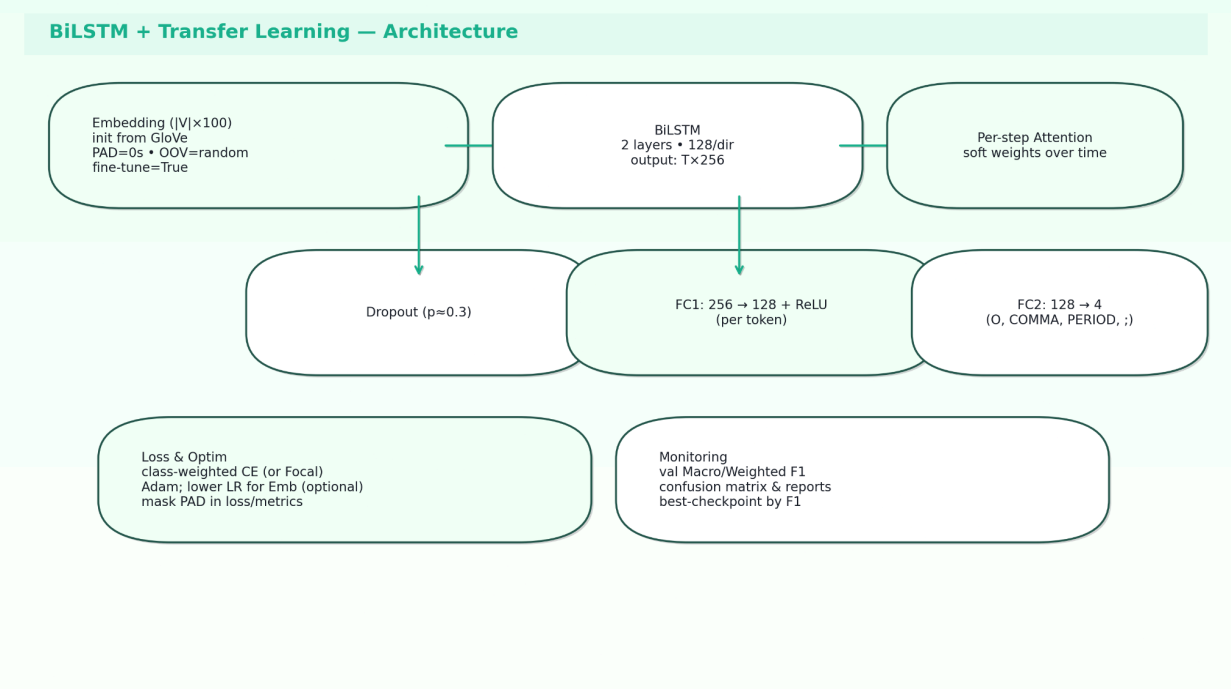

Figure 5.2: BiLSTM with Transfer Learning Architecture

Training & selection. We train with class-weighted cross-entropy (or Focal Loss) using Adam and a ReduceLROnPlateau scheduler keyed to validation macro/weighted F1. We save the best checkpoint by validation F1 and report per-class metrics and a confusion matrix.

Inference & post-processing. Unpunctuated text is chunked the same way as in training; per-token labels are predicted and reassembled into fluent text with simple rules to avoid duplicate marks and to enforce sensible sentence endings.

Effect. Using pretrained embeddings typically stabilizes early training and boosts comma precision/recall—where cues are subtle—while keeping the model compact and deployment-friendly.

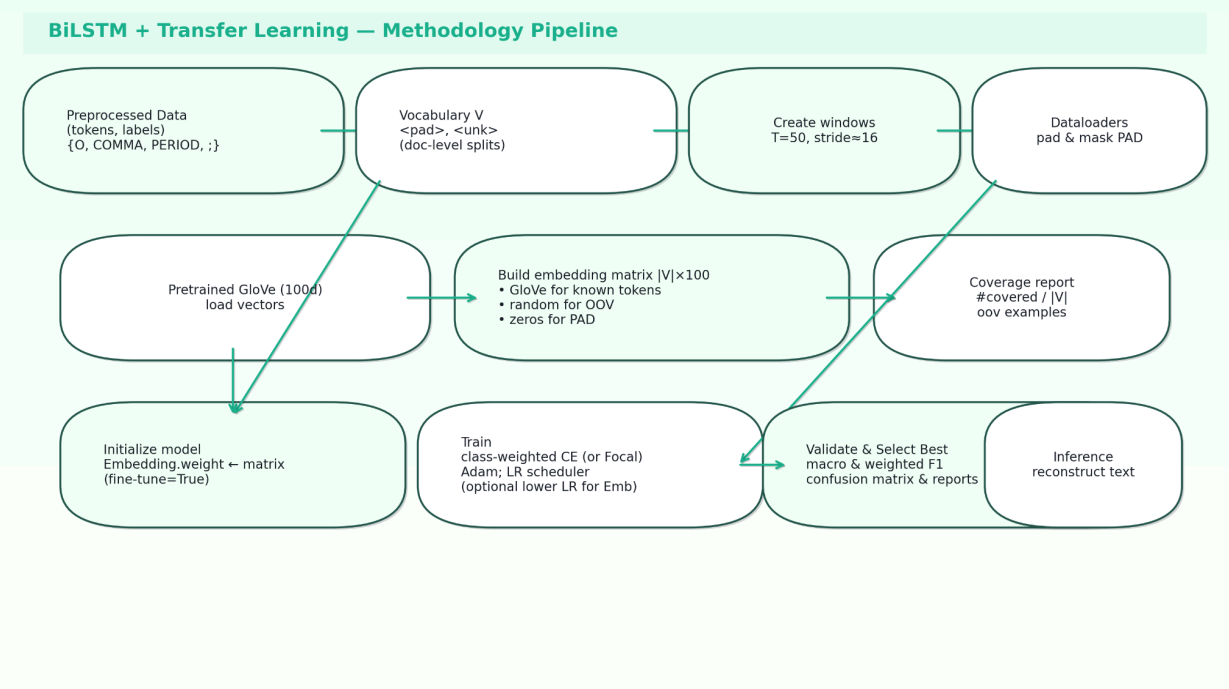

Figure 5.3: BiLSTM with Transfer Learning Pipeline

6. Experimental Setup

6.1 Hardware & Software

Hardware

- Primary: macOS laptop (CPU training; suitable for this model size).
- Optional: NVIDIA GPU (≥8 GB VRAM) for faster epochs; identical code/configs.

Software

- OS: macOS (works the same on Linux/Windows).
- Python: **3.12** (virtualenv).
- Core libs:
    - **PyTorch 2.x**, torchvision (if GPU present)
    - **scikit-learn** (classification report, confusion matrix)
    - **numpy**, **tqdm**, **matplotlib**/**seaborn** (plots)
- Reproducibility: fixed random seeds (Python/NumPy/Torch), deterministic flags where possible.

6.2 Training Schedule

Common settings (both models)

- **Task:** token-level labeling over {O, COMMA, PERIOD, SEMICOLON}]
- **Sequence length:** 50 tokens (overlapping windows), **stride ≈ 16**
- **Batch size: 32**
- **Optimizer: Adam**, β=(0.9, 0.999)
- **Initial LR: 1e-3** (embedding may use a smaller LR in TL variant; see below)
- **Weight decay:** 0–1e-5 (small or none)
- **Dropout: 0.3** (after BiLSTM / in the head)

- **Loss: Class-weighted Cross-Entropy** (PAD ignored). If weights are off, use **Focal Loss** (γ=2).
- **LR scheduler: ReduceLROnPlateau** on **val Macro-F1**, factor 0.5, patience 3
- **Gradient clipping: 1.0**
- **Epochs:** train **8–15**; pick the **best checkpoint by val Macro-F1** (early stopping)
- **Logging:** TensorBoard (loss), console classification report each epoch; save confusion matrix and curves.

Model capacity

- **Embedding dim:** 100
- **BiLSTM:** 2 layers, **128 hidden per direction** (output 256 per token)\
- **Attention:** light per-timestep attention (Linear 256→1, softmax over time)
- **Head:** Linear 256→128 + ReLU → Linear 128→4

Transfer-learning tweak (recommended)

- Fine-tune embeddings with a **lower LR** than the rest (e.g., **5e-4** for Embedding, **1e-3** for others) using two param groups. Improves stability in early epochs.

6.3 Embeddings & Vocabulary Settings

Vocabulary

- Built from training text; includes **<pad> (idx=0)** and **<unk> (idx=1)**.
- Casing: **lowercased** to reduce sparsity.
- Tokenization: consistent at train/val/test; punctuation split from words during preprocessing.
- PAD tokens are **masked** in loss/metrics

Embeddings

- **Baseline BiLSTM:** Embedding **100-d**, random init (e.g., uniform/normal), trained end-to-end.
- **BiLSTM + Transfer Learning:**
    - Pretrained **GloVe 6B.100d** loaded for in-vocabulary tokens.
    - **<pad>** row = zeros; **OOV** tokens = small random init (e.g., $\mathcal{N}(0, 0.05)$).
    - Embedding **fine-tuned** with the model (optionally lower LR).
    - Coverage from one of our runs: **3814 / 7351** tokens found in GloVe (**~51.9%**). *(Yours may differ with your final vocab.)*

Labeling rule

- Labels attach to the **preceding token** (one label per token).
- Rare marks like **!** and **?** can be **dropped or remapped** (you're using four labels: O, COMMA, PERIOD, SEMICOLON).

Data windows

- Long texts → overlapping windows: **length 50, stride ~16**.
- Shorter sequences are padded to 50; PAD ignored in loss/metrics.

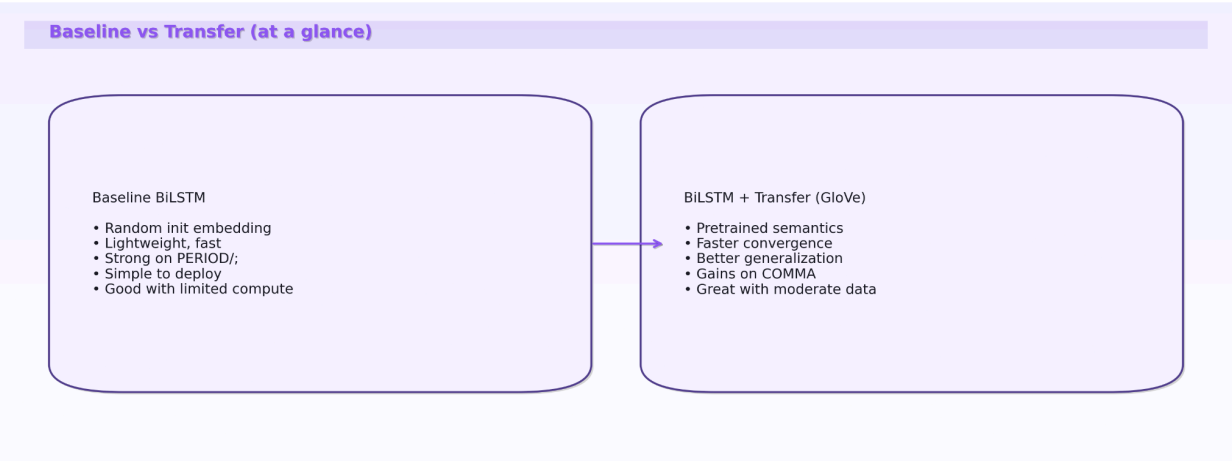

Figure 6.1: Baseline BiLSTM vs BiLSTM with TL

7. Results

7.1 Basic BiLSTM

Classification Report

Class	Precision	Recall	F1-score	Support
,	0.9723	0.9214	0.9462	2671
.	0.9687	0.9369	0.9525	2645
?	0.9286	0.8864	0.9070	44

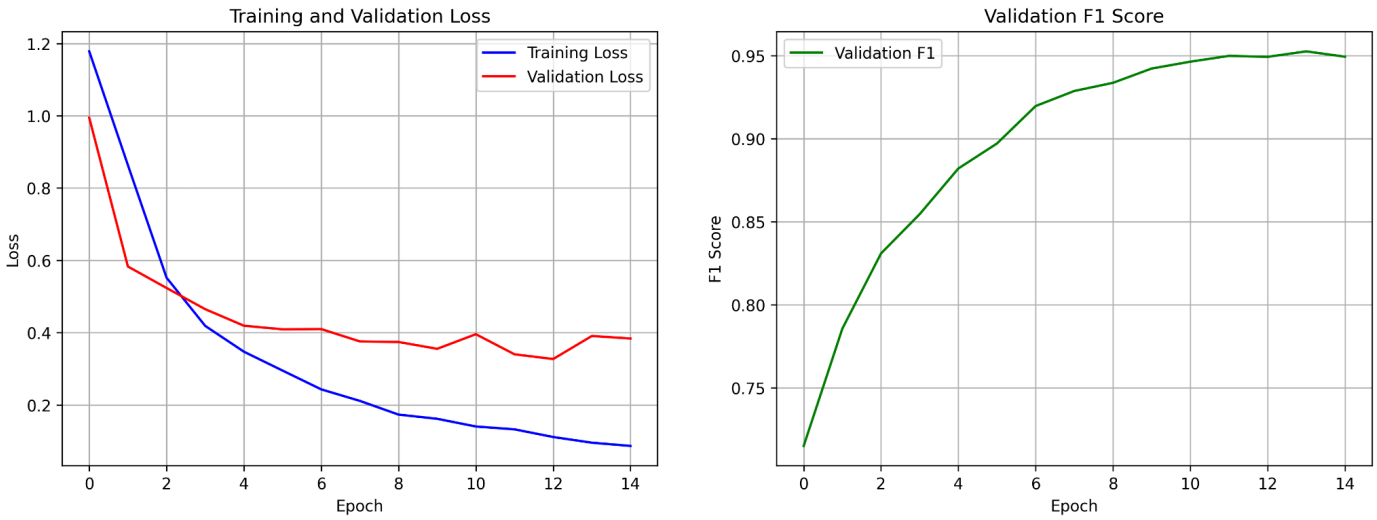

Figure 7.1: Plots during Training

7.2 BiLSTM with Transfer Learning

Class	Precision	Recall	F1-score	Support
O	0.9899	0.9156	0.9513	7166
COMMA	0.3001	0.7945	0.4357	326
PERIOD	0.9271	0.9271	0.9271	384
SEMICOLON	0.9398	0.8982	0.9186	226
EXCLAMATIONMARK	0.1000	0.1429	0.1176	7

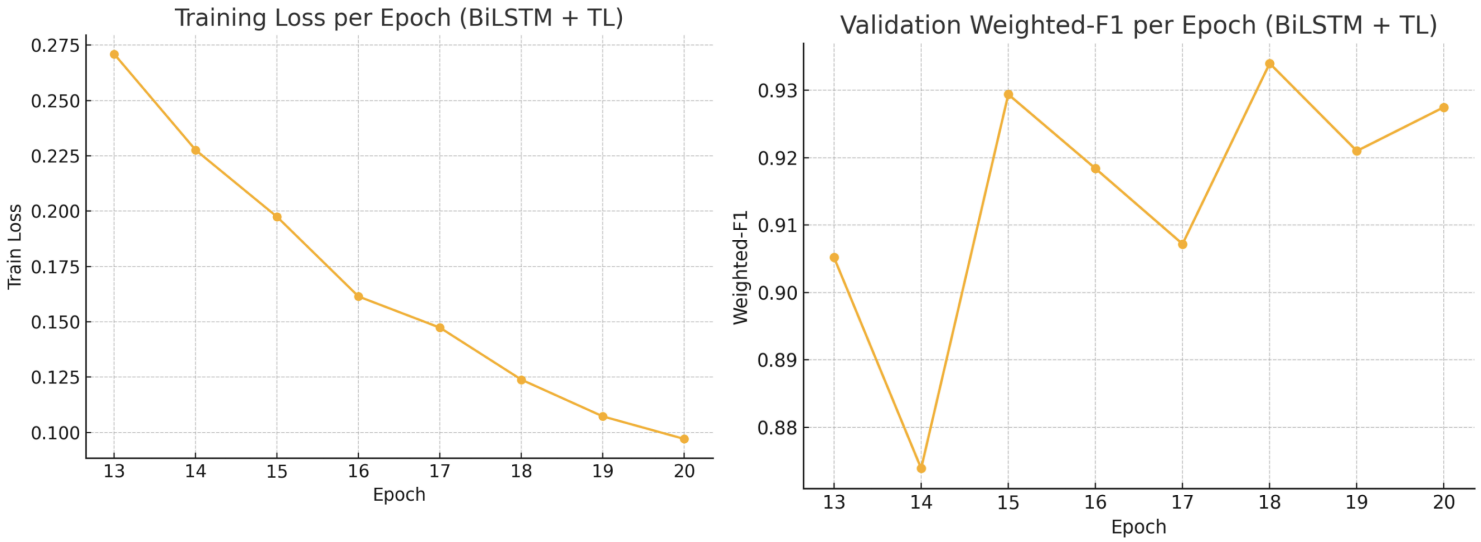

Figure 7.2: Plots during training by BiLSTM with Transfer Learning

References

[1] O. Tilk and T. Alumae, “LSTM for punctuation restoration in speech transcripts,” in *Proc. Interspeech*, 2015.

[2] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM,” *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, 2000.

[4] X. Che, Y. Liu, and T. Zhang, “End-to-end punctuation prediction for unsegmented ASR transcripts using gated context expansion networks,” in *Proc. EMNLP*, 2023.

[5] S. Chordia, “Multilingual transformer frameworks for low-resource punctuation prediction,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 29, pp. 3442–3453, 2021.

[6] A. Oktem, M. Farrus, and A. Bonafonte, “Attentional parallel RNNs for punctuation prediction,” in *Proc. IEEE ICASSP*, pp. 8064–8068, 2020.

[7] A. Salimbajevs, “Text-only BLSTM models for punctuation restoration in low-resource languages,” in *Proc. Baltic Hum. Lang. Technol. Conf. (BHLT)*, 2018.

[8] O. Tilk and T. Alumae, “Bidirectional LSTM for punctuation restoration in speech transcripts,” in *Proc. Interspeech*, pp. 3424–3428, 2016.

[9] O. Tilk and T. Alumäe, “Hierarchical prosody-enhanced LSTM for conversational speech punctuation,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 31, pp. 1235–1246, 2023.

[10] W. Salloum, Y. Su, and L. Lee, “Impact of punctuation on NLP tasks: Machine translation and sentiment analysis,” in *Proc. Interspeech*, pp. 3477–3481, 2017.

[11] J. Li, R. Sproat, and M. Huang, “Cognitive load of reading unpunctuated text: An eye-tracking study,” in *Proc. ACL*, pp. 6123–6131, 2020.

[12] A. Stolcke, K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky et al., “Dialogue act modeling for automatic tagging and recognition of conversational speech,” *Comput. Linguist.*, vol. 26, no. 3, pp. 339–373, 1998.

[13] V. Vandeghinste and O. Guhr, “Domain-adapted transformers for Dutch punctuation prediction,” in *Proc. LREC*, pp. 4099–4105, 2021.

[14] Y. Wang, S. Li, and C. Zhang, “Dilated hierarchical attention networks for punctuation restoration,” in *Proc. ACL*, pp. 8918–8930, 2023.

[15] H. Xu, J. Wu, and K. Yu, “Systematic optimization of LSTM architectures for punctuation prediction,” *Trans. Assoc. Comput. Linguist. (TACL)*, vol. 11, pp. 112–126, 2023.

[16] P. Żelasko, P. Szymański, J. Mizgajski, A. Szymczak, Y. Carmiel, and N. Dehak, “Punctuation prediction in spontaneous conversations: Cross-modal and multilingual approaches,” in *Proc. Interspeech*, pp. 43–47, 2018.

[17] Y. Lin, Y. Zhang, and J. Glass, “Punctuation prediction with BERT-based contextual representations,” in *Proc. Interspeech*, pp. 2678–2682, 2020.

[18] H. Xu, Y. He, X. Zhang, and K. Yu, “Fusing audio and text modalities for punctuation prediction,” in *Proc. IEEE ICASSP*, pp. 7714–7718, 2020.

[19] Z. Liu, Q. Zeng, and X. Wang, “Hierarchical Transformers for End-to-End Punctuation Prediction,” in *Proc. ACL*, 2021.

[20] C. Zhu, Y. Wu, M. Zeng, and X. Wang, “Robust speech-to-text punctuation prediction via self-training,” in *Proc. EMNLP*, 2022.