Programming Assignment 1: Report

Word Embeddings and Named Entity Recognition

Team Members: 22b1279, 22b1233

February 14, 2026

Task 1: GloVe Pre-training

1.1 Hyperparameters

To avoid an explosion of runs, we fixed the dimension at $d = 200$ to identify the optimal hyperparameters. Based on loss convergence and semantic quality of nearest neighbors, the final hyperparameters chosen for the full run ($d \in \{50, 100, 200, 300\}$) were:

- **Context Window (w):** 10 (Chosen to capture broader topical semantics).
- **Learning Rate:** 0.05 (Using Adagrad optimizer).
- **Iterations (Epochs):** 50
- **Weighting Hyperparameters:** $x_{max} = 100$, $\alpha = 1.0$.
- **Batch Size:** 32,768 (Optimized for GPU VRAM pre-loading).

1.2 Latency and Loss Analysis

The GloVe model was trained on a T4 GPU. As the embedding dimension increased, the model's capacity to learn complex representations improved, resulting in a strictly lower final weighted MSE loss for higher dimensions (with $d = 300$ achieving the lowest loss). Latency scaled linearly with the embedding dimension due to the increased matrix multiplication overhead during the forward and backward passes.

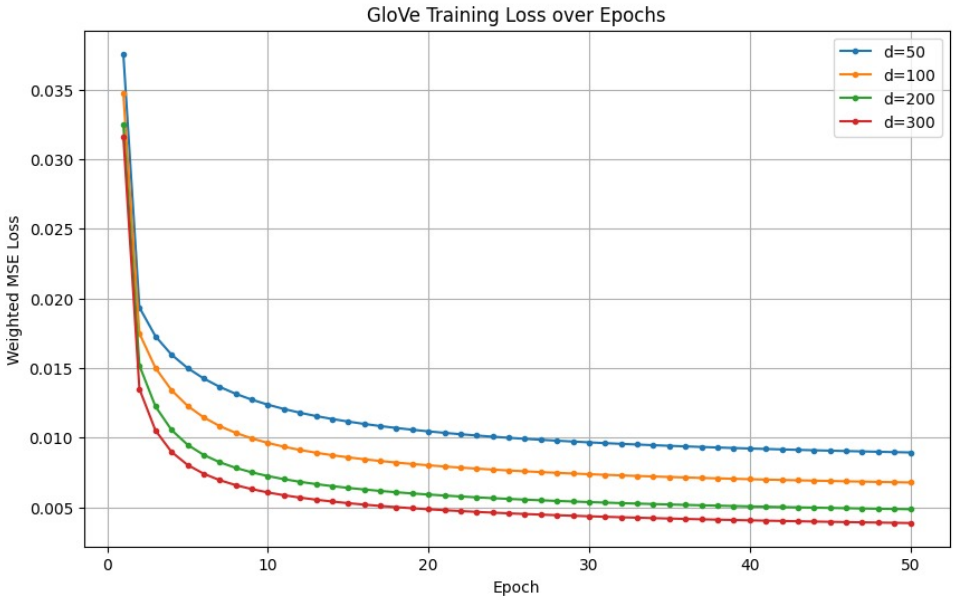

Figure 1: GloVe Training Loss over 50 Epochs for $d \in \{50, 100, 200, 300\}$

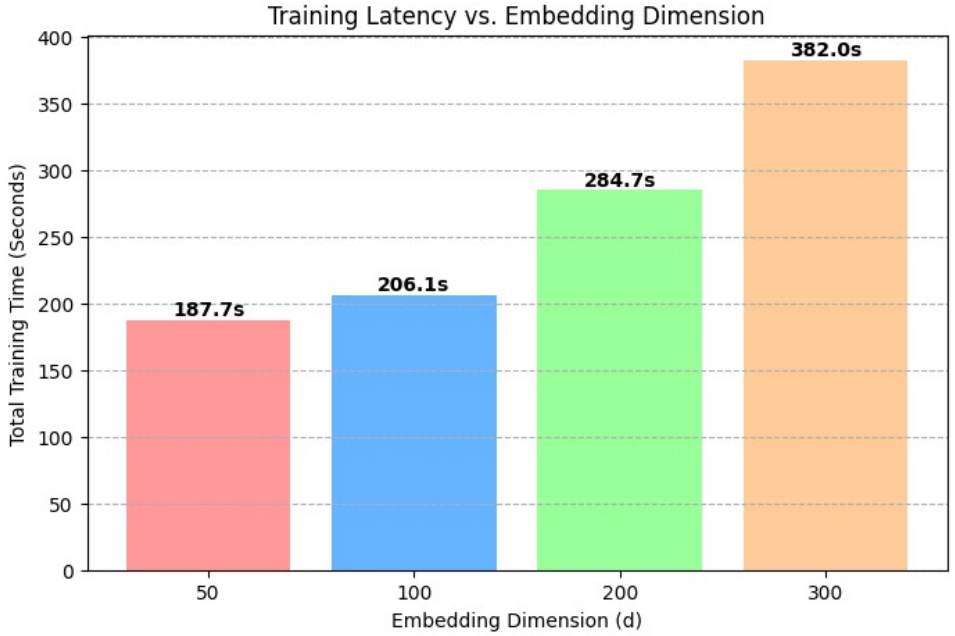

Figure 2: Training Latency vs. Embedding Dimension

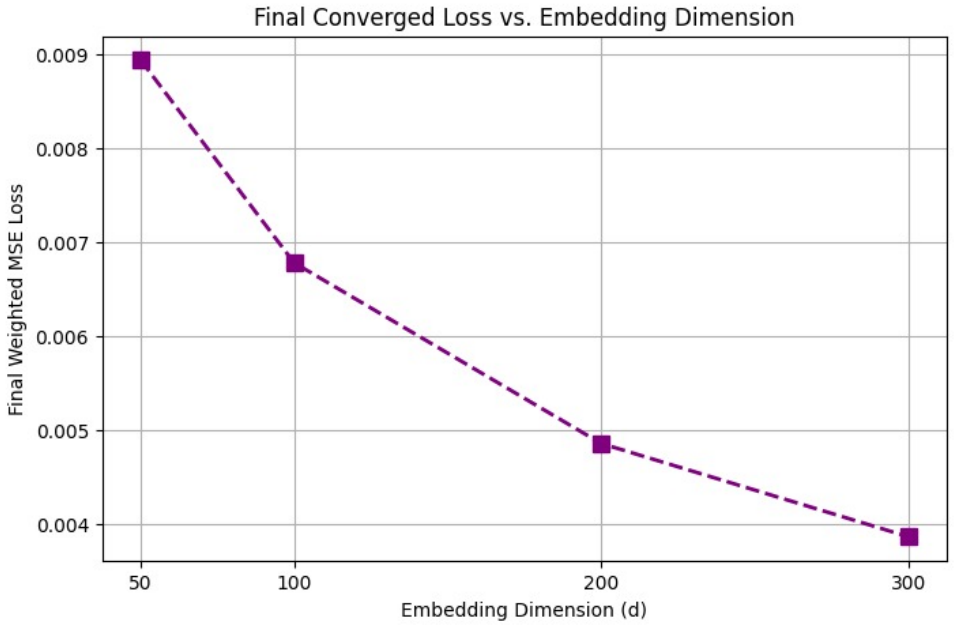

Figure 3: Training Loss vs. Embedding Dimension

1.3 Nearest Neighbors (GloVe, $d = 200$)

Target Word	**Top-5 Nearest Neighbors (Cosine Similarity)**
government	state, federal, officials, should, would
said	says, that, also, he, told
India	Indian, Pakistan, China, Australia, Japan

Task 2: SVD Pipeline

2.1 SVD Latency Analysis

The SVD approach decomposes a sparse Term-Document matrix (25013×67093). Because SVD is algebraic rather than iterative, execution was incredibly fast. Latencies recorded were: $d = 50$ (0.88s), $d = 100$ (2.14s), $d = 200$ (6.65s), and $d = 300$ (6.74s).

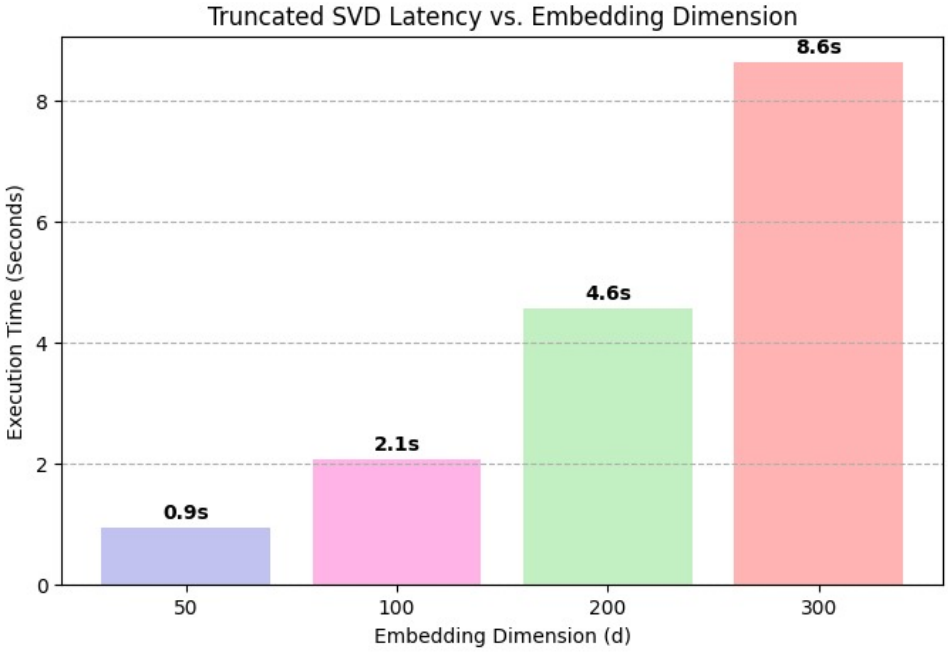

Figure 4: Truncated SVD Latency vs. Embedding Dimension

2.2 Nearest Neighbors (Raw SVD, $d = 300$)

Target Word	Top-5 Nearest Neighbors (Cosine Similarity)
government	raise, ugly, downgraded, consortium, Reuters
said	was, had, general, meetings, who
India	observation, model, engineer, Chief, technical

Table 1: Top-5 nearest neighbors using Raw SVD embeddings at $d = 300$.

Observation: The Raw SVD neighbors are noticeably noisier than GloVe. Words like "ugly" and "downgraded" as neighbors of "government" indicate that SVD on a raw term-document matrix captures document-level co-occurrence rather than fine-grained syntactic relationships.

Task 3: NER using CRF

3.1 Engineered Features

The Conditional Random Field (CRF) relied entirely on hand-crafted features. The features included for each token at index i were:

- **Lexical:** Current word (`word.lower()`), Previous word (`-1:word.lower()`), Next word (`+1:word.lower()`).
- **Shape:** Capitalization patterns (`word.isupper()`, `word.istitle()`), Presence of digits (`word.isdigit()`).
- **Sub-word:** Suffixes (`word[-3:]`, `word[-2:]`), Prefixes (`word[:3]`).
- **Boundary:** Beginning of Sentence (`BOS`), End of Sentence (`EOS`).

3.2 Feature Importance

The CRF heavily prioritized capitalization and local context to classify entities. The top learned features include:

Weight	Target Tag	Feature
-6.8545	O	`word.istitle()`
6.6912	O	`word[-2:]:0M`
6.4854	O	`word[-2:]:5M`
5.4754	B-ORG	`-1:word.lower():v`
5.3889	I-LOC	`-1:word.lower():colo`

Table 2: Top-5 highest weighted features learned by the CRF model.

3.3 Baseline Metrics

Evaluating the CRF on the CoNLL-2003 test set yielded:

- **Token-Level Accuracy:** 0.9571
- **Macro-F1 Score:** 0.7834

Task 4: NER using Feature Learning

4.1 MLP Architecture and OOV Strategy

Architecture: We implemented a Multilayer Perceptron (MLP) mapping the d-dimensional embedding to the 9 NER tags. The architecture is: `Linear(d, 128)` $\rightarrow$ `ReLU` $\rightarrow$ `Dropout(0.3)` $\rightarrow$ `Linear(128, 9)`. The model was trained for 10 epochs using **CrossEntropyLoss** and the **Adam** optimizer with a learning rate of 0.001.

OOV Strategy: For Out-Of-Vocabulary tokens in the CoNLL-2003 dataset, we implemented an `<UNK>` token generated via **Mean-Pooling**, with a lowercase fallback (i.e., if the exact token is OOV, we check its lowercased form before applying the `<UNK>` vector). *Justification:* By averaging all known embeddings in our $V \times d$ matrix, the `<UNK>` token represents the mathematical center of the semantic space. This provides a neutral, unbiased representation for unseen words, minimizing geometric distortion in the MLP compared to inserting a zero-vector or a random vector. The lowercase fallback further reduces the OOV rate by matching tokens like "Government" to the known embedding for "government".

4.2 Results Table

Model	Dim (d)	Token Accuracy	Macro-F1
GLOVE-MLP	50	0.8964	0.4724
GLOVE-MLP	100	0.9016	0.5304
GLOVE-MLP	200	0.9053	0.5497
GLOVE-MLP	300	**0.9072**	**0.5648**
SVD-MLP	50	0.8289	0.0431
SVD-MLP	100	0.8357	0.1206
SVD-MLP	200	0.8518	0.2800
SVD-MLP	300	0.8617	0.3333
CRF (Baseline)	**N/A**	**0.9571**	**0.7834**

4.3 Comparative Analysis

Best Overall Configuration: Among the feature-learning models, the **GloVe-MLP at** $d = 300$ was the best overall configuration, achieving the highest F1 score (0.5648). GloVe significantly outperformed Raw SVD across all dimensions because GloVe's sliding context window enforces fine-grained syntactic relationships (grouping nouns with nouns), whereas Raw SVD simply groups broad document-level topics.

MLP vs. CRF: Despite using dense, pre-trained embeddings, the **CRF Baseline utterly dominated the Neural MLPs** (F1 0.7834 vs 0.5648). This disparity highlights the critical importance of context in NER. Our MLP only evaluated an isolated token's embedding to predict its tag. The CRF, however, accessed surrounding words, suffixes, and crucially, capitalization rules. Because NER is heavily grammar- and context-dependent, human-engineered features easily outperformed context-blind feature learning.

Task 5: [Extra Credit] SVD Performance Boost

We applied a TF-IDF transformation to the raw Term-Document matrix to down-weight high-frequency words before running Truncated SVD at $d = 300$.

5.1 Quality Check 1: Nearest Neighbors Comparison

Table 3: Nearest Neighbors Comparison: Raw vs. TF-IDF SVD

Word	**Raw SVD Top-5**	**TF-IDF SVD Top-5**
government	raise, ugly, downgraded, consortium, Reuters	Papua, consortium, Guinea, raise, copper
said	was, had, general, meetings, who	had, was, which, on, who
India	observation, model, engineer, Chief, technical	Indian, bloodshed, partition, Partition, 32
year	million, platform, We, Our, start	million, Our, We, technology, business
economic	financing, invest, infrastructure, Wang, Xinhua	Latin, Xinhua, financing, Wang, China

Observation: TF-IDF forced the SVD to capture deeper semantic relevance. For example, "India" shifted from generic business nouns (`engineer`, `model`) to highly specific historical/regional concepts (`Indian`, `partition`).

5.2 Quality Check 2: MLP Evaluation

Training the MLP on the TF-IDF SVD vectors yielded:

- **Raw SVD (d=300) Baseline:** Acc: 0.8617 | F1: 0.3333
- **TF-IDF SVD (d=300) Score:** Acc: 0.8581 | F1: 0.3092

Conclusion: While TF-IDF vastly improved the topical quality of the vectors, it *decreased* NER performance. Because NER relies heavily on syntax and grammar, artificially crushing the weights of common structural words distorted the grammatical relationships the MLP relies on, proving TF-IDF is better suited for Document Classification than Named Entity Recognition.

0.1 Additional Interesting Visualizations

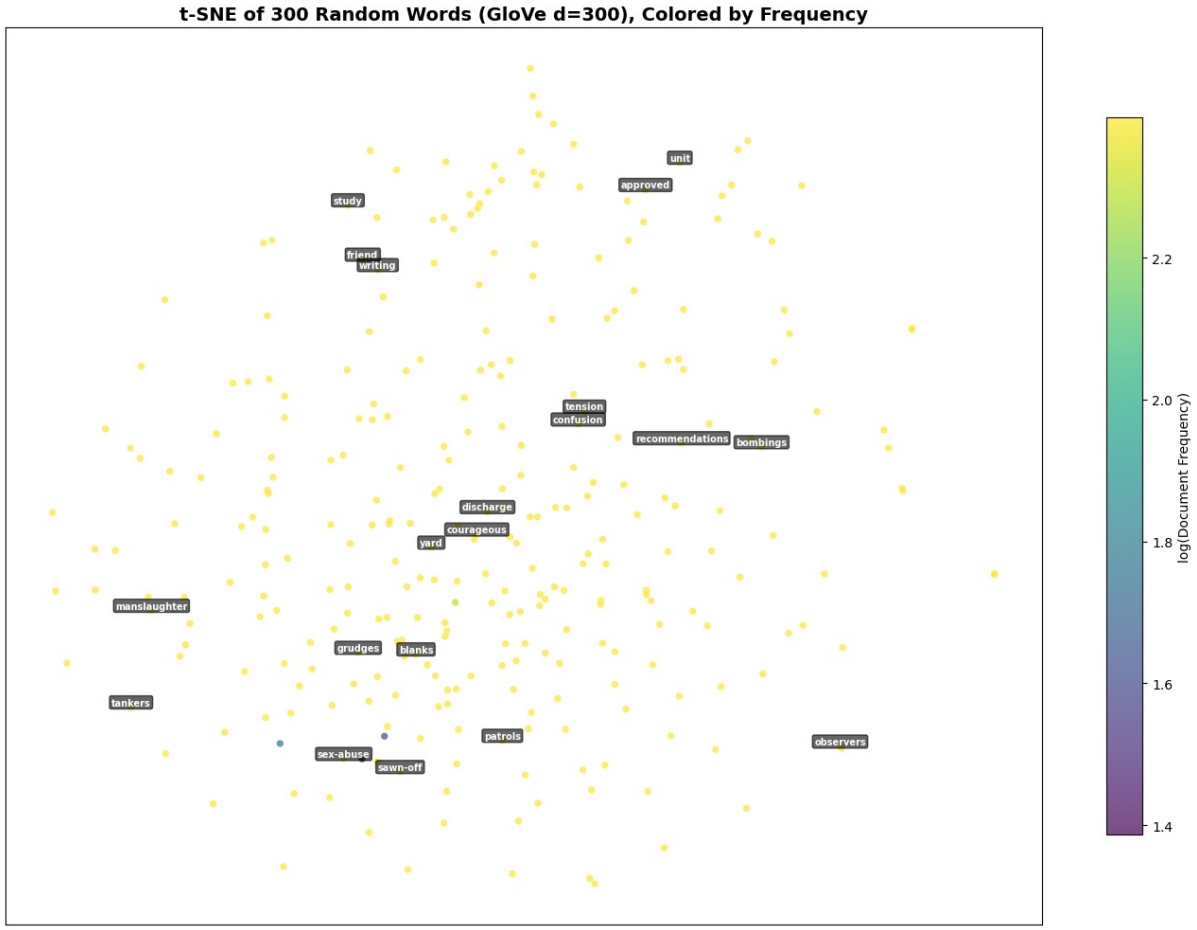

Figure 5: t-SNE of 300 Random Words colored by Frequency

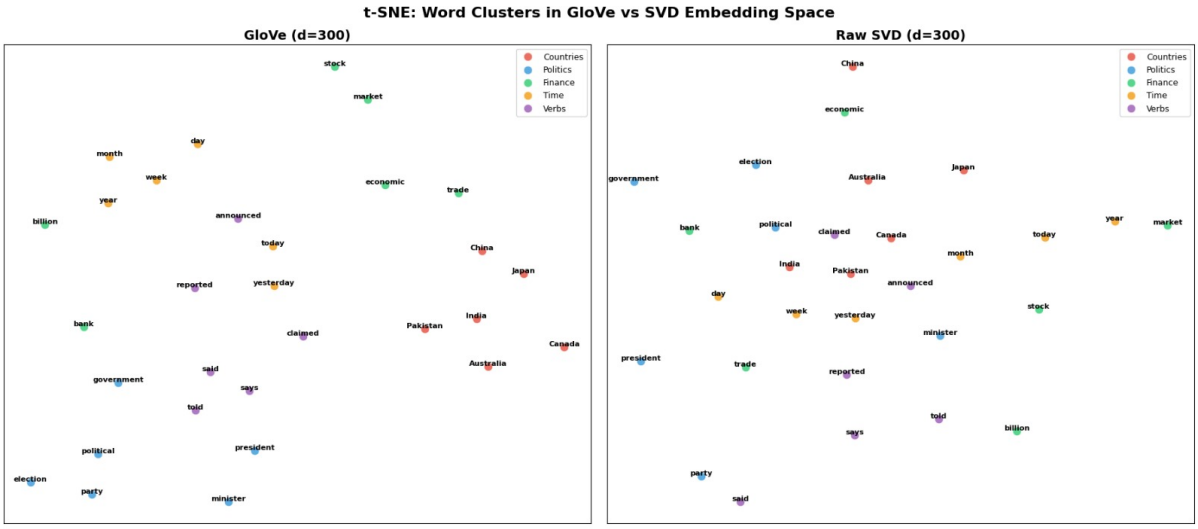

Figure 6: t-SNE Word Clusters for GloVe vs SVD

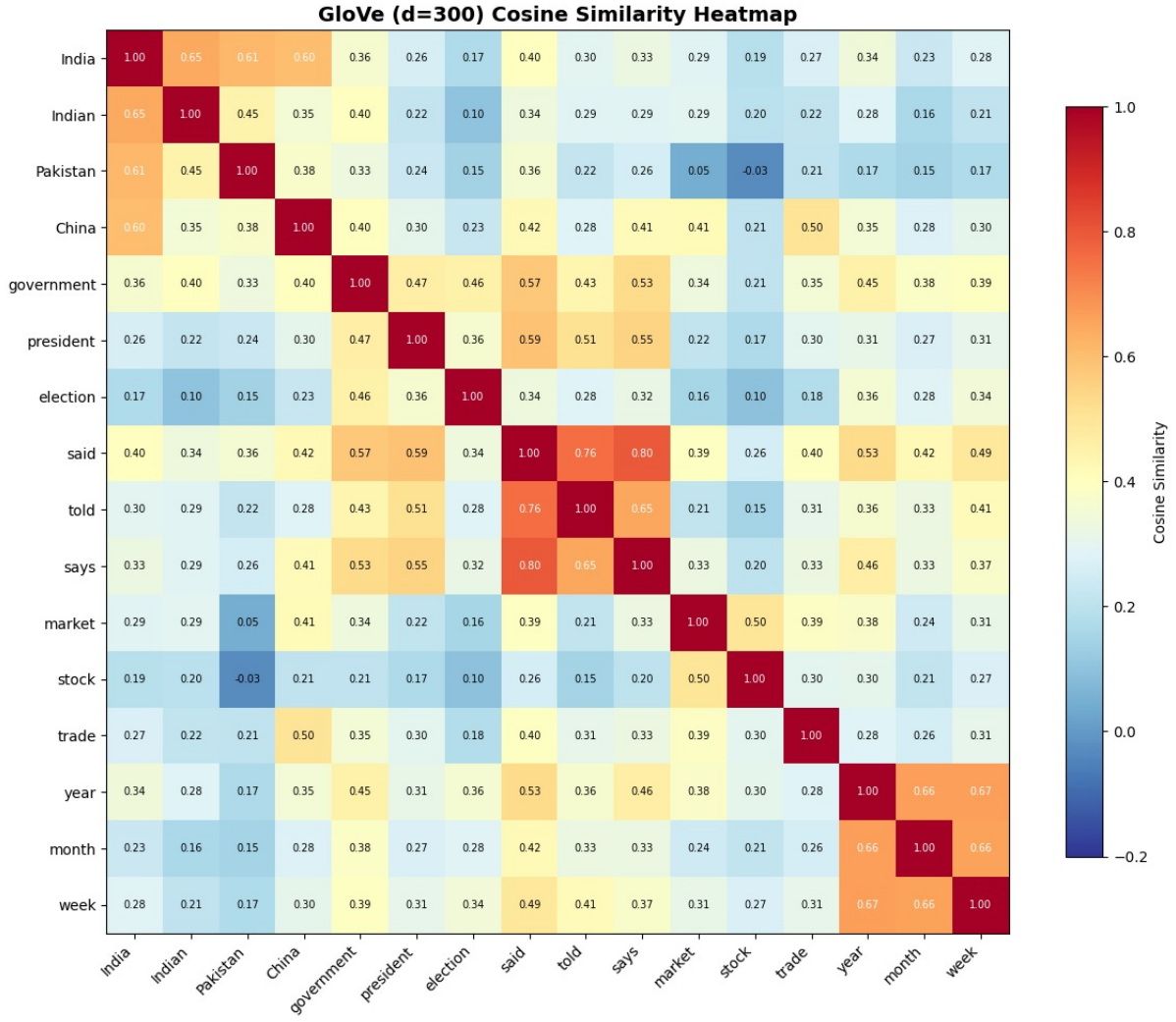

Figure 7: Cosine Similarity Heatmap

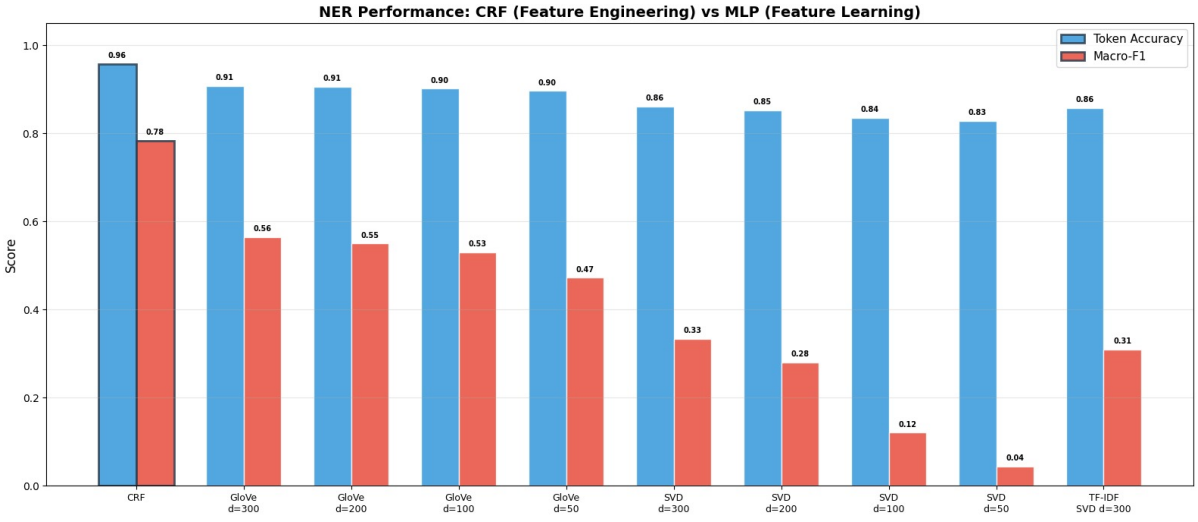

Figure 8: NER Performance Macro-F1

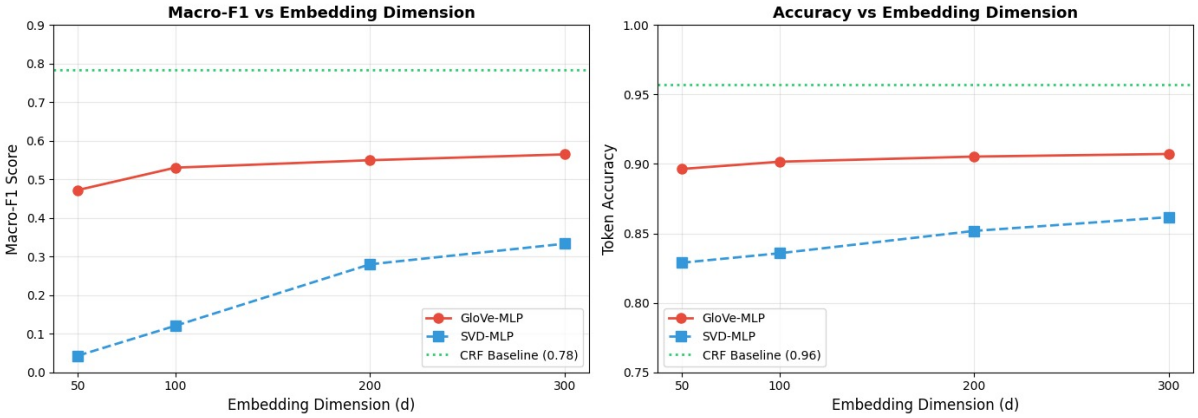

Figure 9: Macro-F1 and Accuracy vs Embedding Dimension

GenAI Usage Policy Declaration

In accordance with the course honor code, we declare the use of a Generative AI assistant (Google Gemini) to assist with planning, optimizing PyTorch memory management (sparse embeddings), handling Scipy sparse matrices, and generating boilerplate plotting code.

Shareable Chat Link: Click here to view the full prompt transcript.