Lab 3.2 - FMRI, Stat 214, Spring 2025

April 28, 2025

1 Introduction

Masked language models are introduced in Chapter 11 of [1]. In this lab, we implement a simplified BERT-style Transformer encoder in PyTorch and pre-train it on 70 narrative texts using a masked language modeling objective. We then extract fixed embeddings from the encoder and fit linear ridge regression models to predict voxel-wise fMRI responses to these texts. Finally, we compare the predictive performance of our contextual embeddings against bag-of-words, Word2Vec, and GloVe embeddings, evaluating embedding complexity and data requirements under the PCS framework.

2 Pre-training

2.1 Encoder

We implemented a simplified BERT-style Transformer encoder model using PyTorch. The core components include a `MultiHeadSelfAttention` module that projects input embeddings into multiple attention heads to compute self-attention scores, a `FeedForward` network that further processes each token independently, and a `TransformerBlock` that stacks self-attention and feedforward networks with residual connections and layer normalization to maintain stability. These blocks are then stacked together to form the full `Encoder` class, which processes input token IDs, position IDs, and type IDs by embedding them, adding them together, and passing them through the Transformer layers. After going through all the layers and a final layer normalization, the model produces output logits for masked language modeling (MLM) tasks.

The design follows the structure of the original BERT model but simplifies some components, like the mask handling and the feedforward network. We included dropout (dropout rate = 0.1 by default) for regularization in both `FeedForward` and `TransformerBlock` and used GELU as the activation function in the feedforward layers. This encoder can be trained for tasks such as masked token prediction by comparing the output logits with the true token labels, and it can also serve as a feature extractor for downstream tasks once pre-trained, as we would use in the following part.

2.2 Masked-Language Model Training

We set up a full training pipeline for a BERT model with the MLM objective. The `mask_tokens` function is responsible for corrupting the input sequences by masking tokens according to a defined probability: 80% of the time, a token is replaced with a [MASK], 10% of the time it is swapped with a random token, and for the remaining 10%, the original token is kept unchanged. Importantly, padding tokens are excluded from masking. This process generates both masked inputs for the model and corresponding labels, where only masked positions contribute to the loss calculation.

The `train_bert` function handles model training by moving the network to the target device, initializing an optimizer and loss criterion, and setting up TensorBoard for visualization. For each epoch, it processes batches by applying the masking function, performing a forward pass to obtain predictions, computing the loss, and updating model weights through backpropagation. The model is evaluated after each epoch without updating its parameters if validation data is available. After completing the training, the code plots the loss curves and saves both the model and the tokenizer.

2.3 Training & Validation

We augmented our masked-language model pipeline to include a held-out validation split for monitoring generalization performance. Using `data.py`, we loaded all 101 stories, then applied a deterministic 70/30

train/validation split (`random_seed` = 42) to ensure reproducibility. This gave 70 stories for training and 31 for validation. Each split was converted into a `TextDataset`, which tokenizes and pads sequences to a fixed length (`max_len` = 32), and wrapped in a PyTorch `DataLoader` (`batch_size` = 2) for efficient batch processing.

We then modified the `train_bert` function in `train_encoder.py` to accept an optional `val_dataloader`. During each epoch, after performing the usual mask-and-predict steps, that is, masking tokens at random, computing the cross-entropy loss only over masked positions, and updating parameters with AdamW, the model is switched to evaluation mode to compute the average validation loss without gradient updates. Both the training and validation losses are logged per epoch and plotted together. These curves allow us to assess convergence behavior, spot overfitting early, and guide our subsequent hyperparameter tuning.

2.4 Hyper-parameter Tuning

In this project, we chose to tune `learning_rate`, `hidden_size`, `num_layers` and `dropout`. `learning_rate` controls the step size in the AdamW optimization process. `hidden_size` determines the dimensionality of the embeddings. A larger hidden size allows the model to capture richer patterns but also increases the risk of overfitting and computational cost. `num_layers` controls the depth of the transformer model. Deeper models can model more complex relationships, but are more likely to overfit, especially since we have limited text data. `dropout` can randomly zero out parts of the hidden layers during training, which is a regularization technique that helps control overfitting. A small dropout can lead to overfitting, while a large dropout may undercut the model's performance.

We keep the following hyperparameters fixed. `batch_size` affects training stability and convergence speed but does not fundamentally change the model architecture. `max_len` controls how many tokens the model can process at once, but since most of our text sequences are short, further tuning will not substantially improve the performance. `intermediate_size` controls the Feedforward Neural Network, and keeping it fixed at a reasonable value like 512 should be good enough for our model. `epoch` determines how long we train our model. We found that in most cases, the train loss functions become steady between 5-10 epochs, and validation losses also stop decreasing and begin to increase after 10 epochs, which is a sign of overfitting. So we keep `epochs` = 10 fixed and believe that this is a suitable value that prevents both under-fitting and overfitting.

We tested `learning_rate`: $\{1 \times 10^4, 5 \times 10^4, 1 \times 10^3\}$; `hidden_size` : $\{128, 256, 512\}$; `num_layers` : $\{2, 4, 8\}$ and `dropout` : $\{0.1, 0.2\}$. Validation loss was used as the evaluation metric to assess the performance of the pretrained BERT model. However, we observed noticeable fluctuations in the validation loss curves across many hyperparameter settings. Therefore, instead of relying solely on the validation loss at the final epoch, we opted to use the average validation loss over the last five epochs to obtain a more stable and reliable performance measure. The result is in Table 1 and stored as a `csv` file in the embeddings folder.

Hidden Size	Num Layers	Dropout	Learning Rate	Final Val Loss	Min Val Loss	Avg Val Loss
128	2	0.1	0.0001	7.4471	7.0797	7.6340
128	2	0.1	0.0005	7.2155	6.7370	7.2966
128	2	0.1	0.0010	7.2285	6.6653	7.1188
256	2	0.1	0.0001	6.8319	6.8055	7.0488
256	2	0.1	0.0010	7.7876	6.8360	7.4162
512	2	0.1	0.0001	6.4030	6.4030	6.8883
512	2	0.1	0.0005	7.4276	7.1769	7.3800
512	2	0.1	0.0010	7.6030	7.2626	7.7470
128	4	0.1	0.0001	6.8338	6.8338	7.4905
128	4	0.1	0.0010	7.2206	6.8646	7.4592
128	4	0.1	0.0005	7.1374	6.4830	6.8766
256	4	0.1	0.0001	7.1774	6.6787	7.0792
256	4	0.1	0.0010	7.6359	7.1822	7.4915
256	4	0.1	0.0050	7.5075	7.4146	7.5561
512	4	0.1	0.0001	7.1619	6.9696	7.0922
512	4	0.1	0.0005	7.9773	7.3526	7.7560
512	4	0.1	0.0010	7.9315	7.1387	7.6797
128	8	0.1	0.0001	7.3398	7.2569	7.4341
128	8	0.1	0.0005	6.9226	6.7639	7.2242
128	8	0.1	0.0010	7.3233	7.2229	7.4850
256	8	0.1	0.0001	7.7423	6.7313	7.2402
256	8	0.1	0.0005	7.7202	7.0535	7.5601
256	8	0.1	0.0010	7.6945	7.5697	7.7342
512	8	0.1	0.0001	7.8248	6.9344	7.2909
512	8	0.1	0.0005	7.3636	6.7504	7.5646
512	8	0.1	0.0010	7.8710	7.6106	7.9144
128	2	0.2	0.0001	7.1915	7.1915	7.6964
128	2	0.2	0.0005	7.2613	6.7628	7.2384
128	2	0.2	0.0010	6.9798	6.7340	7.0758
256	2	0.2	0.0005	6.6444	6.4202	6.9555
256	2	0.2	0.0001	6.8246	6.8246	7.1200
256	2	0.2	0.0010	7.6319	7.1895	7.6058
512	2	0.2	0.0010	7.0597	6.8077	7.4190
512	2	0.2	0.0001	6.8909	6.8909	7.1299
512	2	0.2	0.0005	7.0139	6.5147	7.0611
128	4	0.2	0.0001	7.4041	7.1851	7.6394
128	4	0.2	0.0005	7.0296	6.8357	7.1121
128	4	0.2	0.0010	6.8160	6.8160	7.0956
256	4	0.2	0.0005	7.2772	6.6507	7.2505
256	4	0.2	0.0001	6.8862	6.8862	7.1074
256	4	0.2	0.0010	7.1654	6.8206	7.4366
512	4	0.2	0.0001	7.0879	6.7902	7.2029
512	4	0.2	0.0010	8.2185	7.0820	7.6767
512	4	0.2	0.0005	6.7834	6.7834	7.4985
128	8	0.2	0.0001	7.5773	7.1536	7.6376
128	8	0.2	0.0010	7.6005	6.7634	7.5827
128	8	0.2	0.0005	7.7396	6.8384	7.5149
256	8	0.2	0.0010	7.6060	7.2990	7.4773
256	8	0.2	0.0001	6.7994	6.7994	7.1694
256	8	0.2	0.0005	7.5713	7.0718	7.5167
512	8	0.2	0.0001	7.2629	6.4473	6.9825
512	8	0.2	0.0005	6.6985	6.6985	7.4076
512	8	0.2	0.0010	7.9833	7.1109	7.8745

Table 1: Hyperparameter tuning results

We have three possible candidates for the best hyperparameter setting since they have smaller validation losses compared to other models:

1. `hidden_size`=128, `num_layers`=4, `learning_rate`=5e-4, `dropout`=0.1;
2. `hidden_size`=256, `num_layers`=2, `learning_rate`=5e-4, `dropout`=0.2;
3. `hidden_size`=512, `num_layers`=8, `learning_rate`=1e-4, `dropout`=0.2.

We plotted the corresponding training and validation loss curves for these models. Model 2 exhibited the noisiest validation loss curve, indicating unstable generalization, and was therefore excluded from further consideration. Both Model 1 and Model 3 demonstrated an initial decline in validation loss followed by moderate fluctuations. Ultimately, we selected Model 1 for subsequent tasks as it exhibited slightly smaller fluctuations and a smaller discrepancy between training and validation losses during the final five epochs.

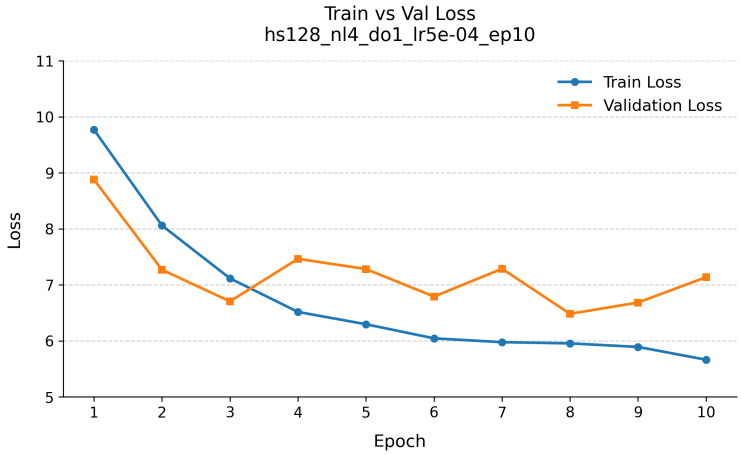


Figure 1: Training and validation loss curves for Model 1

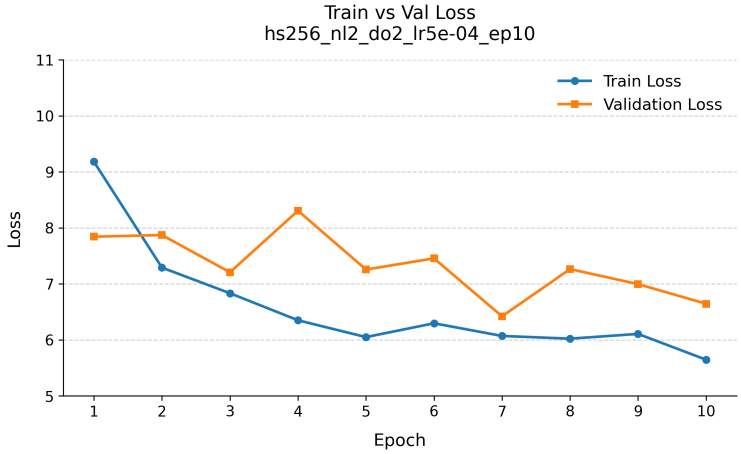


Figure 2: Training and validation loss curves for Model 2

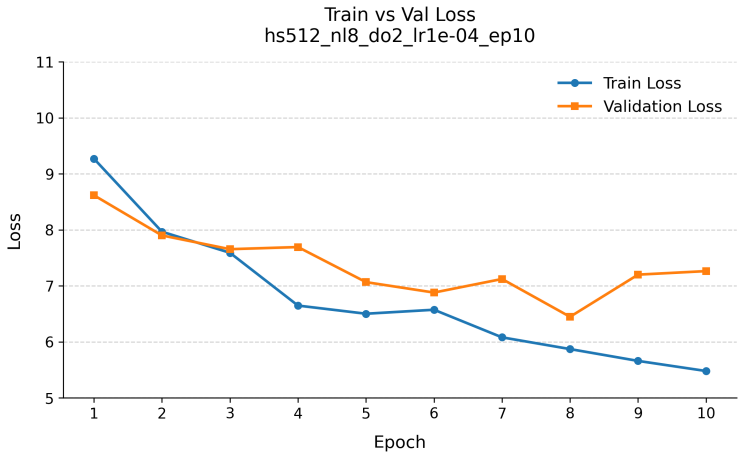


Figure 3: Training and validation loss curves for Model 3

3 Modeling & Evaluation

3.1 Short EDA Comparing W2V, GloVe, and BERT

Before fitting models, we briefly explored the embeddings of the BERT model with the best hyperparameters (`bert_embeddings_hs128_nl4_do1_lr5e-04.bin`). We confirmed that the model contains approximately 30,522 tokens, including special placeholders like `[unused0]`, `[unused1]`, etc. These tokens, typical in BERT vocabularies, do not correspond to real words but are reserved for future use.

Table 2 summarizes the vocabulary sizes across the embedding models. Word2Vec (trained on Google News) and GloVe (trained on Common Crawl) have substantially larger vocabularies than our BERT-based model, which reflects the limited size of our training corpus (around 70 stories).

Embedding Model	**Total Number of Words**
Word2Vec (GoogleNews)	3,000,000
GloVe (6B.300d)	400,000
BERT-based (trained by us)	30,522

Table 2: Total number of words (vocabulary size) available in each embedding model.

Each token in the BERT model is associated with a 128-dimensional vector, whereas Word2Vec and GloVe embeddings have 300 dimensions. However, when examining the cosine similarities between semantically related words (e.g., *dog* and *cat*, *king* and *queen*), BERT embeddings yield near-zero or even negative values, which suggests that the BERT model may not effectively capture their semantic relatedness in this context.

BERT is more technically advanced than Word2Vec and GloVe because it produces contextual embeddings using deep Transformer networks, allowing the same word to have different representations depending on the sentence. In contrast, Word2Vec and GloVe generate static embeddings. However, our small training dataset was insufficient for BERT to learn broad semantic relationships effectively.

Table 3 compares word similarities across the three models. As expected, GloVe and Word2Vec can capture semantic similarities better than our BERT embeddings, suggesting they may perform better in predictive tasks like ridge regression.

3.2 Ridge model fitting

We now fit a linear ridge model to the best BERT-based embeddings generated in the previous step, following the same procedures used in Lab 3.1. Among the 101 stories, 70 were used for training and 31 for testing. To tune the ridge regularization strengths (α) for each voxel, we applied the same 5-fold cross-validation

Word Pair	GloVe Similarity	Word2Vec Similarity	BERT Similarity
man - woman	0.700	0.766	-0.183
king - queen	0.634	0.651	-0.090
dog - cat	0.682	0.761	-0.056
university - school	0.577	0.508	-0.026

Table 3: Cosine similarities between semantically related word pairs, computed using GloVe embeddings, Word2Vec embeddings, and the BERT-based embeddings we trained. (The closer to 1, the better.)

procedure as before: within each fold, we again split the 70 training stories into 80% training and 20% validation sets, tested a range of α values (from 10 to 10000), and selected the best α per voxel based on mean validation performance. The tuned α values were then used to fit the final model on all training stories and evaluate performance on the held-out test set.

3.2.1 All Alphas Set to 1

As before, we first manually set the regularization parameter α to 1 for all voxels and compared the mean correlation coefficient (CC) across four different embeddings (Table 4).

Embedding	Mean CC
Bag-of-Words	0.00216
GloVe	0.00592
Word2Vec	0.00499
BERT	0.00319

Table 4: Mean correlation coefficient for different embeddings when all α values are set to 1.

As expected, when α is poorly tuned, the mean CC across voxels remains very low for all embeddings. The BERT-based embeddings outperform Bag-of-Words but underperform compared to GloVe and Word2Vec. This result highlights that setting a single α across all voxels disregards voxel-specific differences, leading to uniformly low CC values regardless of the embedding type.

3.2.2 Alphas selected via 5-fold CV

Embedding	Mean CC	Median CC	Top 1 Percentile CC	Top 5 Percentile CC
Bag-of-Words	0.00421	0.00372	0.03648	0.02444
GloVe	<u>0.01369</u>	<u>0.01094</u>	0.07089	<u>0.04824</u>
Word2Vec	0.01298	0.01027	<u>0.07103</u>	0.04782
BERT	0.00497	0.00455	0.03552	0.02517

Table 5: Cross-validation results for Subject 2 (Test CC).

Embedding	Mean CC	Median CC	Top 1 Percentile CC	Top 5 Percentile CC
Bag-of-Words	0.00490	0.00398	0.04508	0.02705
GloVe	0.02029	0.01636	<u>0.09529</u>	0.06236
Word2Vec	<u>0.02081</u>	<u>0.01686</u>	0.09457	<u>0.06346</u>
BERT	0.00768	0.00659	0.04733	0.03246

Table 6: Cross-validation results for Subject 3 (Test CC).

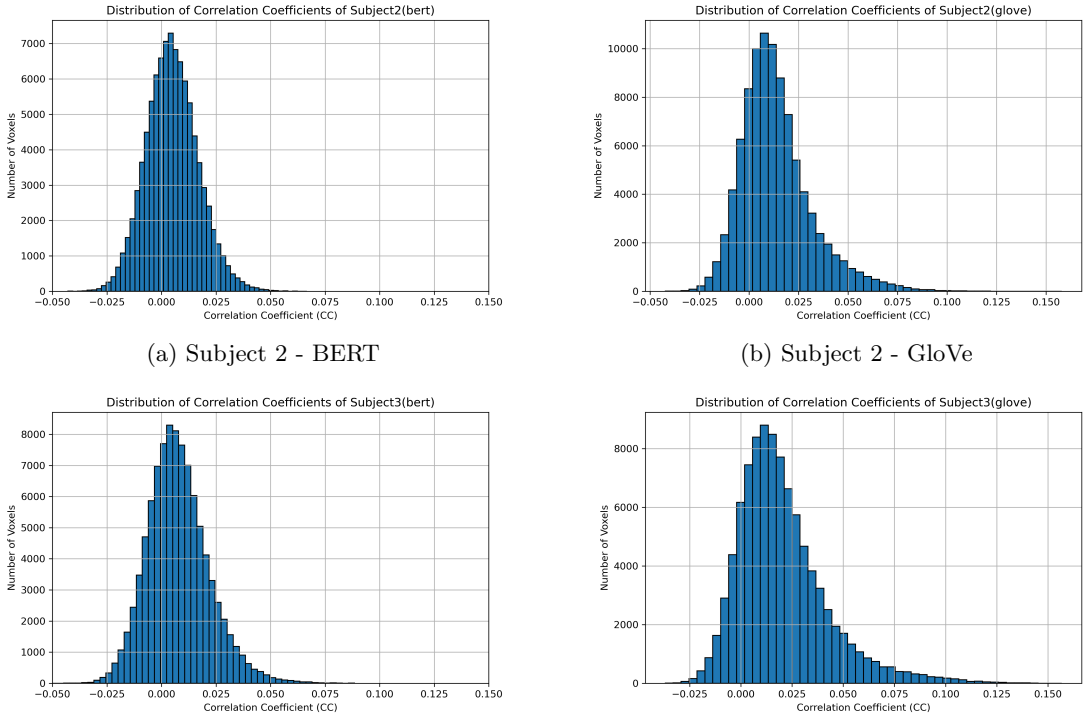
Table 5 and 6 show the results of test CCs for each subject. Across both subjects, the BERT-based embeddings perform much worse than GloVe and Word2Vec, and instead show test CC values close to those of the Bag-of-Words model. Despite BERT's technically advanced architecture, which generates contextual embeddings based on deep Transformer networks, its predictive performance is poor in our setting.

While GloVe and Word2Vec capture semantic relationships between words and achieve significantly higher CCs, the BERT-based embeddings fail to leverage their potential, behaving similarly to Bag-of-Words, which

almost completely ignores word semantics and treats words independently. This result again highlights that advanced models like BERT cannot realize their full advantage without sufficiently large-scale training data, and may underperform compared to simpler, pre-trained embeddings.

3.3 Model Evaluation/Comparison (BERT and GloVe)

We examined the distribution of CCs across voxels for subjects 2 and 3 for BERT and compared them with the CC distributions of our previous best model, GloVe.



(a) Subject 2 - BERT

(b) Subject 2 - GloVe

(c) Subject 3 - BERT

(d) Subject 3 - GloVe

Figure 4: Distribution of CCs for Subject 2 and Subject 3 using BERT and GloVe embeddings.

From the CC distribution plots in Figure 4, we observe that for both GloVe and BERT embeddings, most correlation coefficients are tightly centered around zero, suggesting that neither model achieves consistently high predictive performance for fMRI responses based on story text inputs. However, there is a clear difference between the two distributions: GloVe shows a right-skewed distribution, with a longer tail extending toward higher positive CC values, whereas the distribution of BERT appears nearly symmetric and resembles a normal distribution centered at zero.

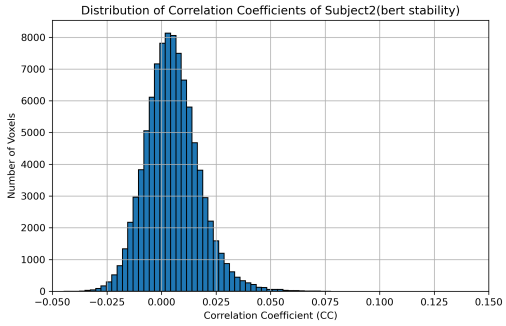
This observation aligns with the earlier results, which show that GloVe yields generally higher CCs than BERT. While both models struggle overall, GloVe produces more voxels with relatively strong positive correlations, indicating better predictive capacity than the BERT-based embeddings.

3.4 Result Interpretation & Comparison

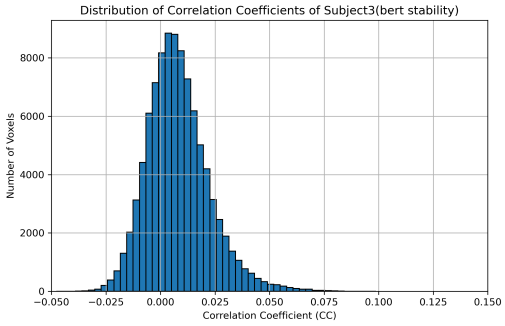
The BERT model performs poorly across all voxels, with most correlation coefficients centered near zero. This supports our intuition that the training dataset was too small to capture semantic relationships effectively and fully leverage BERT's advanced Transformer-based features.

According to the PCS framework (Predictability, Computability, Stability), the model shows poor prediction performance for most voxels, as indicated by the low CCs. Computability is satisfied, as the model can be trained and evaluated efficiently across the full voxel set.

We assessed the stability of the BERT model by reshuffling all stories and refitting the model on a new



(a) Subject 2 - BERT Stability



(b) Subject 3 - BERT Stability

Figure 5: Distribution of correlation coefficients (CCs) after reshuffling the stories for the BERT model.

train/test split, following the same procedure as in Lab 3.1. As shown in Figure 4 (original split) and Figure 5 (reshuffled split), the CC distribution shapes for both subjects remain highly similar. This suggests that train/test variation has a minor impact on model performance and that the BERT model demonstrates stability and robustness to data perturbations. This result is consistent with the GloVe findings from Lab 3.1, where GloVe also showed low CCs across voxels but robustness against data reshuffling.

3.5 Conclusion

All four models (Bag-of-Words, Word2Vec, GloVe, and BERT) demonstrated limited predictive ability. These results suggest that to fully realize the potential of BERT's architecture, training the model on a substantially larger dataset would be necessary. Only then could BERT leverage both its advanced modeling capabilities and the advantages of large-scale input data.

4 Bibliography

References

[1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. 2025. URL: `https://web.stanford.edu/~jurafsky/slp3/`.

A Academic honesty

A.1 Statement

We hereby make the following statements: We personally designed and conducted all data analysis processes presented in this report. We have written and produced all text and graphs included in this report. Additionally, we have incorporated and documented all procedures into our workflow, ensuring that the results are fully reproducible. We have properly attributed all references to others' work.

We strongly believe that integrity in academic research is of utmost importance. First and foremost, we must take responsibility for the results of our research. Since science thrives on collaboration, our findings will serve as the foundation for future research. Dishonest or non-reproducible research can lead to a cascade of erroneous results. Furthermore, plagiarism contributes nothing new to the scientific community and is disrespectful to the original authors, as it misappropriates credit for their work. Therefore, we are committed to maintaining honesty, transparency, and reproducibility in all aspects of our research.

A.2 LLM Usage

Coding

The code portion of this report only uses chatGPT to help refine formatting and annotation writing. We ensure that data exploration, preprocessing, model selection, comparison, interpretation, and tuning are entirely manual by the team members.

Writing

The writing portion of this report only uses chatGPT to help with the layout of the images and the rephrasing of very few sentences. We make sure that the vast majority of the content is brainstormed, discussed, organized, and typed by us in person.