Text-Based Multi-Label Classification

Project Final Report

Team Members: Varun Jain, Sudipta Halder

Abstract

Text-based multi-label classification (MLC) is a fundamental task in natural language processing (NLP) that involves predicting multiple relevant labels from textual input. This project focuses on performing multi-label classification on the GoEmotions dataset [2], a high-precision benchmark for emotion classification. The study explores deep learning architectures, integrating pre-trained embeddings and sequence modeling techniques to enhance classification performance. We primarily trained two models, a lightweight and a balanced model. The brief difference between the two modules is that Balanced uses Quasi-Recurrent Neural Networks (QRNN) [1] and Max Pooling layers, and Lightweight does not. Both variants achieved an F1-score of over 0.3. These models indicated promising performance and provided insights into key factors influencing classification accuracy. We also tested more model variants, with layers of Data and Zero-Fill Reverse Pooling, Projection [4], and more.

1 Introduction

Natural Language Processing (NLP) has significantly advanced over recent years, particularly in the domain of multi-label classification (MLC) tasks. Multi-label classification requires a model to predict multiple labels for a given input text, which presents unique challenges. The model must effectively capture the relationships between the labels while maintaining the overall semantic meaning of the text.

Our goal is to design novel neural network architectures that can classify emotion labels on text without prior context. The models will be designed explicitly for MLC. We will have two main types of models: Balanced and Lightweight neural network architectures. Balanced will be able to run on platforms with more computing power. Balanced will take advantage of GPUs by using QRNN [1] to run parallel operations. This way Balanced model can take significant and long sequences of text effectively and is less slow compared to an LSTM model that relies on a sequence of text in a series. On the other hand, Lightweight neural network architectures are used to less taxing on memory and computation; we want it to be able to run on low powered devices and still have high performance. Hence, we will remove QRNN so it can run on hardware that does not require too much parallel computing.

We utilize the GoEmotions [2] dataset, a large-scale dataset containing 27 labeled emotions alongside neutral, derived from 58,000 English Reddit comments. This dataset offers a balanced distribution of comment lengths and labels, making it ideal for training multi-label classification models. Additionally, the generalized nature of the dataset allows models trained on it to maintain consistent performance across different tasks, making it a suitable choice for our model evaluation. Our features will be text and characters in these comments. These comments can range from short comments with few words to long paragraphs worth of words. This. Make our MLC model versatile in input text length. This should allow our model to detect emotion labels with small input text to large input text. The Large input texts are better suited for the Balanced model since it uses QRNN; therefore, it would better classify long large text inputs.

Another one of our goals is to have a model that achieves or surpasses an F1 score of 0.46 on the GoEmotions dataset. This would demonstrate the model's ability to effectively classify input text into the appropriate 28 labels while ensuring the model size remains as small and efficient as possible. By achieving this, we aim to show that our architectures can be competitive in multi-label classification tasks, even when focusing on efficiency.

2 Related Work

The GoEmotions [2] Dataset is one of the largest manually annotated datasets comprising 58k English Reddit comments with 27 emotion labels. The dataset has a balanced distribution of comment length and label emotions. They have applied the Principal Preserved Component Analysis (PPCA) to find the most stable emotion patterns in the dataset by searching which emotions are consistently agreed by different raters. In their benchmark, they performed transfer learning experiments with the models trained on this dataset and then tested them on other Datasets(ISEAR, Emotion-Stimulus, EmoInt). All models performed well despite the limited target domain data given. Therefore, GoEmotions generalizes well to different domains, taxonomies, and especially with limited target domain data. For their experiment, they used a pre-trained BERT model that is fine-tuned. The model only achieved an average F1 score of 0.46, which can be improved by expanding taxonomy to other domains and exploring cross cultural robustness of emotion ratings.

The PRADO [3] is a novel projection attention neural network combining trainable projections with attention and convolution layers for multiple large document text classification tasks. PRADO was designed to run entirely on devices and IoTs. Having a small, accurate neural network model run on these IoTs and devices can enable user privacy. It would also be low latency, keeping a consistent user experience. PRADO also would focus on long text classification. PRADO effectively finds semantically similar phrases with a trainable projection model while reaching high performance and maintaining a compact size. Using the PRADO approach, it can train tiny neural networks of just 200 Kilobytes in size, which is an improvement over CNN and LSTM models. It performed near state-of-the-art on multiple long document classification tasks in 2019. In their paper, they were able to show PRADO and its learning transfer capabilities. PRADO can also be applied to transfer learning. Since it is a compact model, transferring the weight requires small storing and looking up.

The pQRNN [4] is a tiny and effective projection-based embedding-free neural encoder for natural language processing tasks. It was made to take up less memory and compute resources compared to mBERT and XLM-R to run on latency-critical applications on servers and edge devices. pQRNNS is a combination of QRNNs and projection layers. A non-pre-trained pQRNNs model can outperform a pre-trained LSTM model by .015 in accuracy with the same parameters, making it effective in the model in accuracy and parameters. While the pQRNNs model is 140x smaller, making it more compact than LSTM and lighter. The pQRNNs could also outperform mBERT by over 0.95 in MTOP, a multilingual semantic parsing dataset, and mATIS, a popular parsing task. At the same time, being 350x smaller than mBERT shows strong results of pQRNNs being low weight and suitable for latency-sensitive applications.

QRNN [1] quasi-recurrent neural networks. QRNN alternates convolutional layers and applies them parallel across timesteps. Then, a minimalist recurrent pooling function is applied in parallel across all the channels. QRNN was designed to make a recurrent neural network that can run for data in long sequences. Recurrent neural networks would depend on each timestep's computation on the previous ones, making it not ideal for parallelism and long sequences. Since QRNN performs operations in parallelism, it is 16 times faster at training and testing/validation than an optimized cuDNN LSTM model. QRNN outperfroms LSTM in achieves better accuracy on several language modeling tasks. Due to its parallelism, it runs well on a GPU, hence why the high performance. QRNN can take on character-level neural machine translation, sentiment classification, language modeling, and more, making it crucial for various sequence tasks, especially in characters.

3 Methodology

3.1 Data Preprocessing

Since the input data is a string, it has to be tokenized and converted into an integer representation. For this, we use TensorFlow's *TextVectorization* preprocessing layer, followed by an *Embedding.* Section 3.5 discusses the selection process for the hyperparameters of this layer.

3.2 Proposed Methods

Due to the existence of a variety of neural network architectures for text processing, we decided to use a mixture of these. We utilize Projection [4], Reverse Pooling, Quasi-Recurrent Neural Network (QRNN) layers [1], Convolution layers, a PRADO-based layer [3], Pooling and Flatten layers, and conclude with a Dense layer. More details about the model architectures can be found in the following sections.

3.3 Layer Descriptions

3.3.1 Text Vectorization

The Text Vectorization layer is responsible for converting raw text inputs into fixed-length sequences of integer token indices. It performs preprocessing such as lowercasing, punctuation handling, and splitting based on whitespace. This layer standardizes the input format and maps tokens to a vocabulary of fixed size, allowing the model to process variable-length textual input in a consistent way.

3.3.2 Embedding

The Embedding layer transforms each token index into a dense, continuous-valued vector. These embeddings serve as learned representations of words, capturing syntactic and semantic relationships. This layer outputs a three-dimensional tensor corresponding to batch size, sequence length, and embedding dimension, which is passed to the subsequent projection layer.

3.3.3 Projection

The Projection layer introduces a lightweight, trainable transformation to improve model efficiency and non-linearity. It consists of a fully connected layer that first performs a dimensionality-reducing transformation (down-projection), followed by a non-linear activation function (such as ReLU), and finally a second fully connected layer that projects the features back to the original embedding dimension (up-projection). This bottleneck design encourages feature compression while maintaining expressive capacity.

3.3.4 QRNN

The QRNN layer models sequential dependencies by combining the efficiency of convolution with a recurrent-like gating mechanism. It consists of two parallel 1D convolution operations:

- z_t: the main convolution output
- f_t: the forget gate, produced by a sigmoid-activated convolution

The QRNN computes its output using the element-wise formula:

$$\text{output}_t = f_t \odot z_t + (1 - f_t) \odot \text{stop\_gradient}(z_t)$$

where $\odot$ denotes element-wise multiplication. This formulation allows the model to selectively retain or discard information over time without explicit recurrence, and supports parallel computation across timesteps.

3.3.5 PRADO Sparse Layer

The PRADO Sparse Layer is a custom fully connected layer that incorporates sparsity into its activations. By enforcing a sparsity mask during training, it allows only a subset of the neurons to be active, reducing overfitting and encouraging the network to learn compact, interpretable representations. This is especially useful for high-dimensional input spaces common in NLP.

3.4 Model Architectures

Based on the layer types currently available, we have created and tested two architecture types: Lightweight and Balanced. A variation of the Lightweight and Balanced architectures is provided in Figures 1 and 2 respectively.

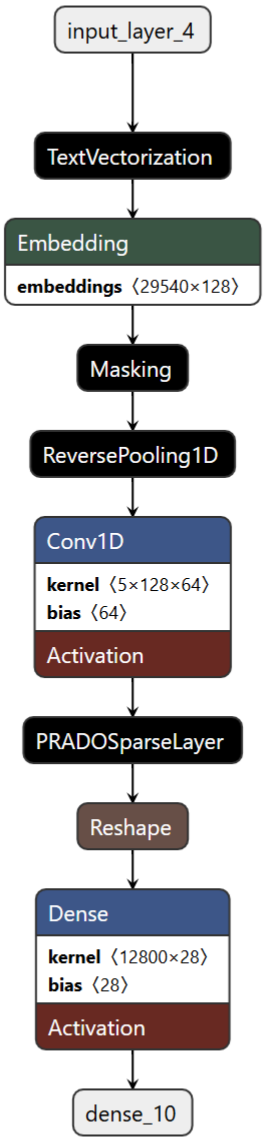


Figure 1: Lightweight Model Architecture

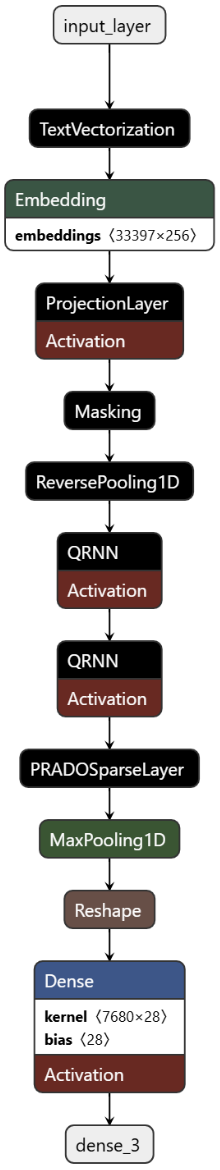


Figure 2: Balanced Model Architecture

Both of the architectures have different types of variations, with some of them being common across both. Table 1 provides the variations within the architectures.

Variation Type	Values
TextVectorization Sequence Length	30, 50
Embedding Maximum Tokens	29540, 33397
Reverse Pooling Type	Data-Fill, Zero-Fill
Balanced Architecture CNN Present	True, False
Reshape Output Shape	7680, 12800
Projection Layer Present	True, False

Table 1: Architectural Variations

Going forward, the notation provided in Table 2 will be used to denote the different variations of the architectures. These notations will be placed in the order of their appearance on the table. Hence, a lightweight architecture with data-fill reverse pooling and the 30 sequence length TextVectorization will be denoted as LD3S, while a balanced architecture with zero-fill reverse pooling, 33397 maximum token embedding, no CNN layer, and with a Projection layer will be denoted as BZCVNCP. Section 3.5 provides an explanation for the absence of the Reshape variation shorthand notations in the table.

Variation Type	Short Hand Notation
Lightweight Architecture	L
Balanced Architecture	B
Data-Fill Reverse Pooling	D
Zero-Fill Reverse Pooling	Z
Embedding with 33397 Maximum Tokens	CV
Balanced Architecture CNN Absent	NC
TextVectorization with 30 Sequence Length	3S
Projection Layer Present	P

Table 2: Architectural Variation Shorthand Notations

3.5 Hyperparameters and Training

3.5.1 Activation Functions

Each layer in the model utilizes activation functions suited to its role in the network. The activation functions used are as follows:

- **Conv1D:** Uses the ReLU activation function for non-linear feature extraction.
- **Dense:** Uses the sigmoid activation function to output probabilities for multi-label classification.
- **PRADO Sparse Layer:** Contains an internal Dense layer with a ReLU activation function.
- **QRNN:** Comprises two parallel Conv1D layers as described in [1]:
  - The main convolution branch (z_t) uses a tanh activation.
  - The forget gate branch (f_t) uses a sigmoid activation.
- **Projection:** Applies a ReLU activation on the down-projected input.

3.5.2 Layer-wise Hyperparameters

The following outlines the key hyperparameters for each layer type used in the Lightweight and Balanced models:

- **Text Vectorization:** The output sequence length is a tunable parameter that determines how many tokens are passed to downstream layers. Although the initial value was set to 50, further analysis revealed that a length of 30 sufficiently captured 95% of the dataset, and was thus used for half of the experiments.
- **Embedding:** The embedding layer outputs a dense vector representation of shape $(\text{SequenceLength}, \text{EmbeddingDim})$. The embedding dimension is 128 for the Lightweight model and 256 for the Balanced model.
- **Projection:** This layer applies a down-projection followed by a non-linear activation and an up-projection. The bottleneck dimension is set to 64 in the Lightweight model and 128 in the Balanced model. The input and output dimensions of the layer match the embedding size.
- **Reverse Pooling:** This layer upsamples the sequence by a fixed factor to approximate previously reduced temporal resolution. The upsampling factor is 2 in the Lightweight model and 4 in the Balanced model.
- **QRNN:** Present only in the Balanced model, the QRNN layer has 256 hidden units and consists of two parallel convolutional layers: one for the main transformation and another for the forget gate.
- **Conv1D:** The Lightweight model uses 64 filters with a kernel size of 5, while the Balanced model uses 128 filters with a kernel size of 3.
- **PRADO Sparse Layer:** This layer applies a sparsity mask to its 128 output units during training. It is used in both Lightweight and Balanced configurations.
- **Max Pooling:** Applied only in the Balanced model, the 1D Max Pooling layer uses a pool size and stride of 2 to reduce the sequence length.
- **Reshape:** The output of the PRADO layer, with shape $(2 \times \text{SequenceLength}, 128)$, is flattened into a 1D vector of size $256 \times \text{SequenceLength}$ before being passed to the Dense layer.
- **Dense:** The final Dense layer contains 28 units, corresponding to the number of emotion classes in the GoEmotions dataset, and uses a sigmoid activation for multi-label classification.

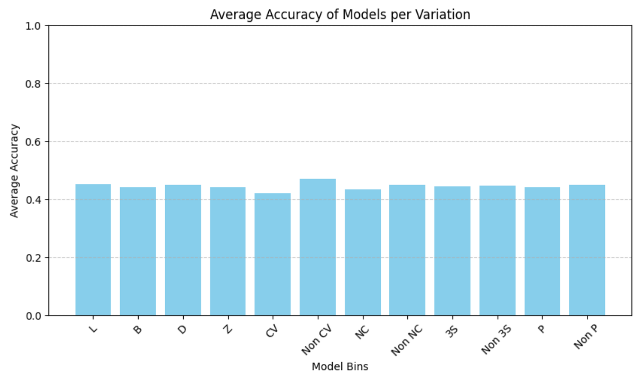
3.5.3 Training Procedure

The dataset is divided into training, validation, and test splits using the TensorFlow Datasets library. After preprocessing, the training dataset is used to adapt the `TextVectorization` layer. Each model variant is trained one epoch at a time using a batch size of 32 and evaluated on the test dataset after each epoch. The model with the best macro F1-Score is saved. Training is performed using the Adam optimizer and Binary Cross Entropy as the loss function.

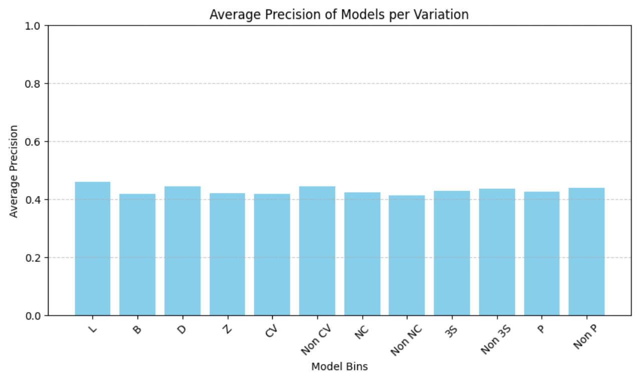
3.6 Evaluation Metrics

To evaluate model performance, we use the macro F1-score at the end of each training epoch. Training is terminated when the F1-score begins to decline, and the model from the previous epoch is selected as the final version. Because of differences in sparsity and architecture depth across model variants, the number of training epochs varied, typically ranging between 3 and 9.

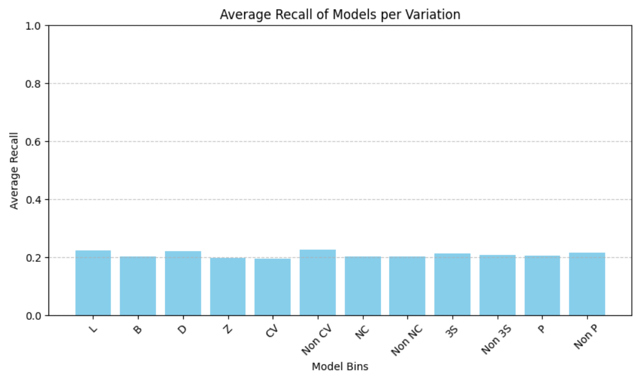
4 Experimental Results



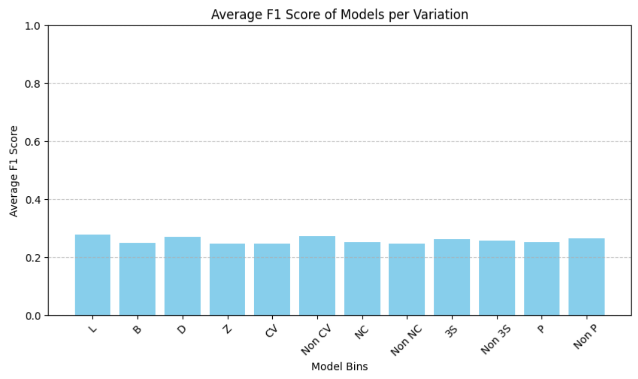
(a) Average Accuracy per Bin



(b) Average Precision per Bin



(c) Average Recall per Bin



(d) Average F1-Score per Bin

Figure 3: Per-bin average performance metrics across all model categories.

As can be seen from the results in Figure 3, the lightweight models have the best precision and F1-Score on average, while default standardization models have the best accuracy and recall. This suggests that using a lightweight model with default standardization would provide one of the best models across all metrics. However, taking a look at Table 3, we can see that this is not completely true. While most of the best models do use the default standardization, they are also using a balanced architecture. Not only that, they all skip the convolution layer before the sparse layer. This suggests that by building on the BDNC or BDNC3S variants could possibly provide a model that would be more capable.

Metric	Short Hand Notation	Accuracy	Precision	Recall	F1-Score
Accuracy	BDNC	49.88%	0.47	0.24	0.29
Precision	LDCVP	43.41%	0.51	0.20	0.26
Recall	BDNC3S	43.82%	0.42	0.28	0.32
F1-Score	BDNC3S	43.82%	0.42	0.28	0.32

Table 3: Best Models by Metric

Table 4 shows the per-class Precision, Recall and F1-Score of the model with the best F1-Score, i.e., BDNC3S. As can be seen, some classes have an F1-Score of over 0.70, which indicates these classes are detected best by the model. On the other hand, there are classes like 16 and 19 which the model is not able to learn at all. This can be due to the model either overfitting on the other classes, or the model not having enough layers to be able to learn these classes sufficiently.

The appendix shows the Per-model Accuracy, Precision, Recall, and F1-Scores in Figure 4 and Figure 5.

Table 4: Per-Class Metric of Best Model

Class	Precision	Recall	F1-Score
0	0.6410	0.4464	0.5263
1	0.7248	0.5985	0.6556
2	0.4076	0.3232	0.3606
3	0.2683	0.1375	0.1818
4	0.3105	0.1937	0.2386
5	0.2785	0.1630	0.2056
6	0.2464	0.1111	0.1532
7	0.3460	0.2570	0.2949
8	0.3333	0.2651	0.2953
9	0.2292	0.0728	0.1106
10	0.2734	0.1311	0.1772
11	0.5111	0.1870	0.2738
12	0.3333	0.0541	0.0930
13	0.2556	0.2233	0.2383
14	0.7778	0.3590	0.4912
15	0.8905	0.8778	0.8841
16	0.0000	0.0000	0.0000
17	0.4706	0.4472	0.4586
18	0.7604	0.6134	0.6791
19	0.0000	0.0000	0.0000
20	0.5373	0.4992	0.5176
21	0.5372	0.3495	0.4235
22	0.6667	0.1250	0.2105
23	0.3143	0.0759	0.1222
24	0.2000	0.1818	0.1905
25	0.5200	0.4643	0.4906
26	0.5054	0.3013	0.3775
27	0.3824	0.2766	0.3210
Average	0.4266	0.2787	0.3281
Std Dev	0.2265	0.2223	0.2273

5 Conclusion

This project explored novel neural network architectures for text-based multi-label emotion classification using the GoEmotions dataset. Through rigorous experimentation with both Lightweight and Balanced model variants, we demonstrated the efficacy of integrating techniques such as QRNNs, projection layers, and sparse PRADO layers in enhancing model performance under different computational constraints. Our best-performing model, `BDNC3S`, achieved a macro F1-score of 0.32—falling short of the 0.46 benchmark reported by the original GoEmotions paper, yet showcasing a competitive tradeoff between accuracy, model complexity, and inference efficiency.

Notably, the Balanced architecture variants—particularly those omitting convolution layers before the sparse layer and employing default text standardization—consistently outperformed others across key metrics like accuracy and recall. However, analysis of per-class F1-scores revealed performance inconsistencies, with some emotions being well-identified (e.g., class 15 with a 0.88 F1-score), while others (e.g., classes 16 and 19) were not learned at all. This disparity is likely due to class imbalance or insufficient representational capacity.

For future work, we plan to address these shortcomings by incorporating advanced regularization strategies, class balancing techniques, and further architectural tuning. We also intend to explore knowledge distillation and transfer learning methods to bridge the performance gap with state-of-the-art models while maintaining the lightweight design necessary for edge deployment.

6 References

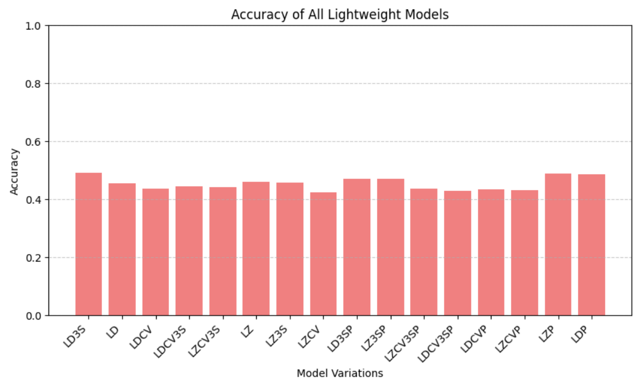
[1] James Bradbury et al. *Quasi-Recurrent Neural Networks.* 2016. arXiv: `1611.01576 [cs.NE]`. URL: `https://arxiv.org/abs/1611.01576`.

[2] Dorottya Demszky et al. "GoEmotions: A Dataset of Fine-Grained Emotions". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* 2020, pp. 4040–4054.

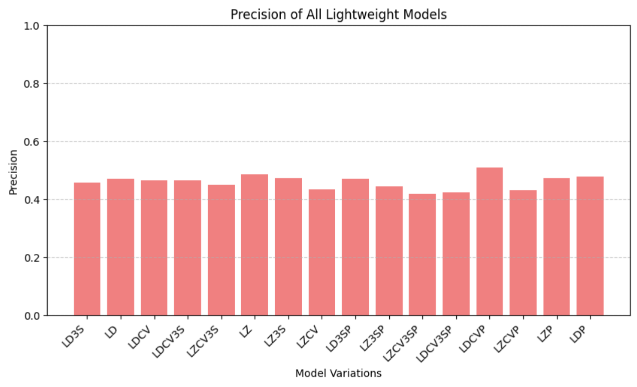
[3] Prabhu Kaliamoorthi, Sujith Ravi, and Zornitsa Kozareva. "PRADO: Projection Attention Networks for Document Classification On-Device". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).* Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5012–5021. DOI: `10.18653/v1/D19-1506`. URL: `https://aclanthology.org/D19-1506/`.

[4] Prabhu Kaliamoorthi et al. *Distilling Large Language Models into Tiny and Effective Students using pQRNN.* 2021. arXiv: `2101.08890 [cs.CL]`. URL: `https://arxiv.org/abs/2101.08890`.

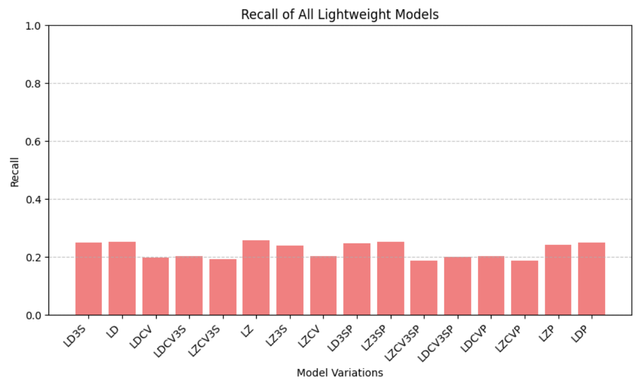
7 Appendix: Per-Model Results



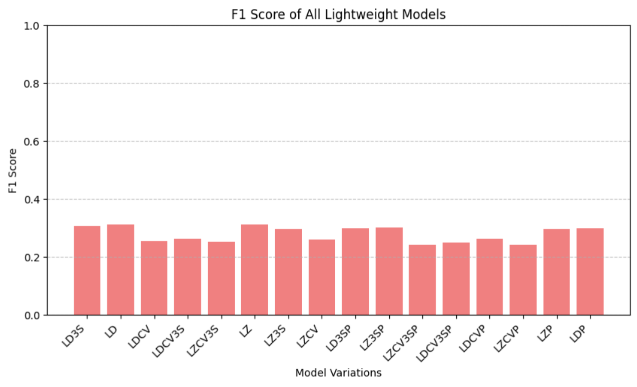
(a) Lightweight Model Accuracies



(b) Lightweight Model Precisions

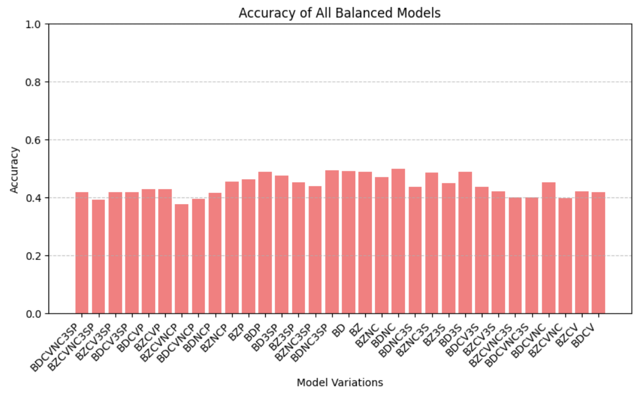


(c) Lightweight Model Recalls

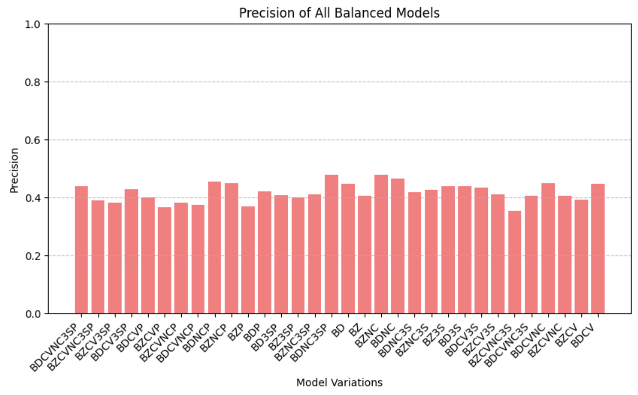


(d) Lightweight Model F1-Scores

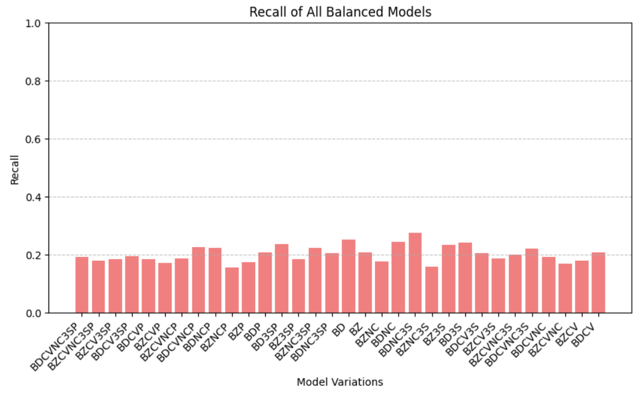
Figure 4: Performance metrics across all Lightweight models



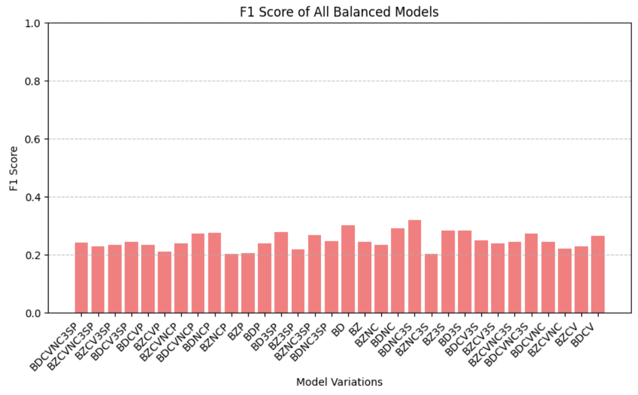
(a) Balanced Model Accuracies



(b) Balanced Model Precisions



(c) Balanced Model Recalls



(d) Balanced Model F1-Scores

Figure 5: Performance metrics across all Balanced models