Framework for the Comparison of Large Language Models

Nandini Krishnaswamy (nk2869), Aditya Desai (ad3695)

I. Executive Summary and Problem Motivation

Our goal for this project was to develop a framework providing functionality to compare different large language models (like BERT, XLNet, and RoBERTa) and generate a variety of task-specific metrics to help a user in selecting the appropriate model for a given task, including sentiment analysis, multilabel classification, and question-answering. To solve this problem, we implemented a pipeline in python consisting of 3 classes, Dataset, Model, and ModelCompare, taking as input a configuration file containing the variables to be set by the user and giving as output a JSON file containing the appropriate metrics, like accuracy, F1 score, and time taken for a classification task, to not only compare models in terms of performance but also in terms of time. This solution is a valuable contribution because we feel it augments existing functionality to create an easily usable and insightful tool for NLP programmers. NLP is a quickly growing field and model building functionality has become readily available and accessible through frameworks like Hugging Face, but it is not always obvious which model is right for which task, often requiring some amount of trial-and-error on the programmer's part to compare models and their respective parameters. Our framework generates tangible metrics for model comparison with no coding necessary on the user's end, allowing the user to effectively choose between models.

We had several motivations for building this solution. Firstly, we felt that a framework for model comparison would be an informative resource. Research in large language models is happening at a fast pace and as newer, larger models touting improved performance keep popping up, programmers need to decide which models to choose depending on the unique problem they are trying to solve. Blogs and papers both provide insight about advantages of certain models over others in terms of methodology, architecture, and empirical results on specific datasets, but this information is not sufficient to fully predict performance for a user with a distinctive, unsolved problem, including a unique combination of dataset, task, hyperparameters, and desired solution properties. Therefore, in addition to synthesizing resources that explain theory behind differences in large language models, this tool provides an empirical, quantitative way to compare model performance. We were also motivated by user-friendly frameworks that make these models accessible to programmers in the field; while there is already available, easy-to-use functionality for building modes, as of now, comparing models would require independently building different models and generating comparison metrics. Our framework allows a user to circumvent this extra work; all that is needed to compare the models is a specification of the model names, dataset name, and any parameters a user wants to explore. Lastly, we are personally excited about recent advancements in NLP and the potential of large language models to excel at tasks, so we wanted to use such a tool to gain insight about our own interests and explore the performance power of certain models.

II. Background Work

Background work in the field includes pioneering papers in large language models, including "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin et al.), "XLNet: Generalized Autoregressive Pretraining for Language Understanding" (Yang et al.), and "RoBERTa: A Robustly Optimized BERT Pretraining Approach" (Liu et al.), which each explain the

unique contributions of the model presented and how it builds upon and offers improvements over previous architecture and methodology. Specifically, BERT is a bidirectional transformer that is pretrained on 2 tasks, including masked language modeling, in which some tokens of the input sentence are masked and those masked tokens are then predicted using bidirectional context, and next sentence prediction, in which the model receives pairs of sentences and has to predict whether the second sentence is the next sentence in the original document. XLNet is bidirectional and autoregressive, and unlike BERT it does not use masked language modeling. Instead, it uses permutation language modeling, in which training sentences are permuted and this shuffling of word order provides bidirectional functionality. Furthermore, XLNet does not use next sentence prediction because the researchers noted that this task didn't yield much improvement in the model. RoBERTa is based on BERT but instead of masked language modeling, it uses dynamic masking, and it does not use next sentence prediction. XLnet and RoBERTa are supposed to have performance improvements on BERT but also take longer to pretrain. Background work also includes a paper on knowledge distillation, "Distilling the Knowledge in a Neural Network" (Hinton et al.), which is a method for compressing models into a more usable size. Finally, background work includes blogs that explain how to choose between models based on architectural differences and desired properties; an example of such a blog is "BERT, RoBERTa, DistilBERT, XLNet — which one to use?" (Khan). We find these resources to be useful in understanding the NLP space and providing a good theoretical background for our framework.

III. Technical Challenges

Our technical challenges concern time and computation resource constraints, as well as a technically challenging algorithm. Through our GCP accounts, we each had access to a single GPU, so one of our technical challenges was fine-tuning these large models on the supported tasks within reasonable time constraints. We also ran into GPU "out of memory" errors while training and later resolved them by freeing memory after each run. From an algorithmic perspective, implementing custom knowledge distillation was a technically challenging task. The knowledge distillation process is usually necessary for real-world applications because language models are very large, requiring compression for efficiency and usability. The distillation process is as follows: first, a teacher model (e.g. a large language model like BERT) is trained on input data. Then, a student model learns from this teacher model in addition to learning from the actual labels. The loss used to train the student model can be separated into 2 different components. The first component is the loss with respect to the teacher model. Instead of having a regular softmax for the output layer, during knowledge distillation we use a softmax with temperature as a parameter, which acts as a regularizer by dampening the inputs to the softmax, effectively constraining the values to a smaller range and producing a probability distribution over classes that is somewhat more uniform. We refer to this probability distribution as "soft labels". This is useful because it gives the model more information about how each data point relates to all of the classes. The second component of loss is the loss between the student's hard predictions (i.e. without using temperature) and the actual hard labels. The loss used to train the student model is a weighted average of these two losses. We used a small BiLSTM based model as the student model, and used the distillation technique from the paper by Hinton et al.

IV. Approach and Implementation

We had several goals when deciding our approach for the project. We have mentioned earlier in this report that we wanted this framework to be generalizable to a diverse set of tasks. Therefore, our

approach to writing code needed to be such that it was easy to apply the same code to different models and different tasks, abstracting out the details on the user's end. Furthermore, the results given by the framework should be easy to read and should give a good idea about which model excels at which task, given the setting of parameters. We also wanted to avoid redundant code and model creation/training due to the computation and time-intensive nature of the problem. Lastly, we had to design the pipeline such that the user would not have to upload anything except for a configuration file-- namely, the user should not have to upload a dataset and model.

We chose several datasets as default options for our framework from the Hugging Face Datasets library, which contains over 500 NLP datasets. The default option for multilabel classification is the "go_emotions" dataset, consisting of Reddit comments with multiple sentences. This dataset has 43410 training examples, 5426 validation examples, and 28 classes. An example data point from this set is "Thanks! I love watching him every week", which is labeled with the sentiments "gratitude" and "love". For sentiment analysis, we used the dataset "rotten_tomatoes" which has movie reviews-- this dataset contains 8530 training examples, 1066 validation examples, and the sentiment is either positive or negative. For example, "the soundtrack alone is worth the price of admission" is classified as positive. Our question-answering dataset is "SQuAD", containing questions and answer segments of text from a corresponding passage. This dataset has 87599 training examples and 10570 validation examples. The question "What individuals live at Fatima House at Notre Dame?" has the corresponding answer, "... Retired priests and brothers…".

As of now, the framework supports models BERT, XLNet, and RoBERTa and the tasks sentiment analysis, multilabel classification, and question-answering. The hyperparameters that can be varied include number of epochs, learning rate, batch size, and max sequence length. We observe performance at different values of these hyperparameters. As of now, models can be trained and evaluated using a single GPU. For our experiments, we used 8 vCPUs, 30Gb RAM, and 1 Nvidia Tesla V100. The evaluation metrics for comparison are accuracy, F1 score, ROC-AUC, and loss for classification and exact match and F1 score for Q&A.

The tools we used to create the project include Google Cloud Platform for compute resources and TensorFlow and Transformers frameworks for implementation. Our pipeline consists of 4 python files: dataset.py, model.py, model_compare.py, and config.py. The user first configures the framework by specifying the model types, task names, model hyperparameters, distillation hyperparameters, etc. The user then calls the main module that controls the framework, where all models and datasets for the various tasks are initialized and training and evaluation is done. Training and evaluation datasets for different tasks, as well as models, are initialized by ModelCompare. Predefined evaluation metrics are written to a JSON file once all the tasks have been completed.

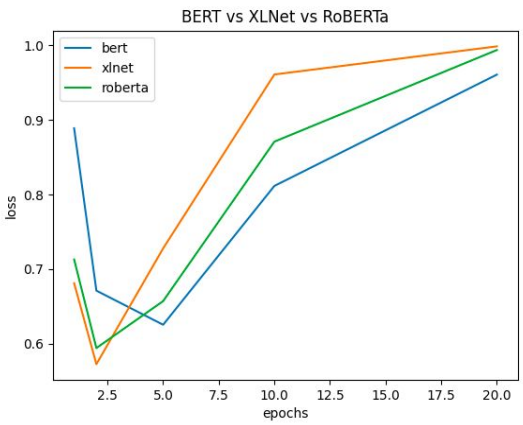
V. Experiments and Evaluation

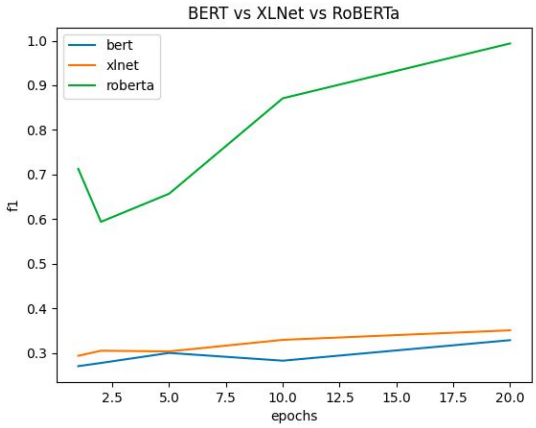
In this experimental evaluation, we will show the results obtained by using our framework with several variations on input and hyperparameters. While the actual inputs to the framework are pairs of models and the outputs of the framework are JSON files, we graph the results for ease of explanation. Here, we take as input BERT, XLNet, and RoBERTa, and train the models for multilabel classification.

	Model Size (in parameters)	Evaluation time	Training time (per epoch)

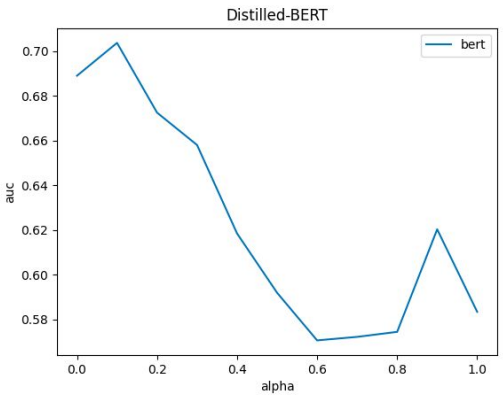
BERT	~340M	~11s	~70s
Distilled-BERT	~1.7M	~4s	~7s
XLNet	~340M	~11s	~85s

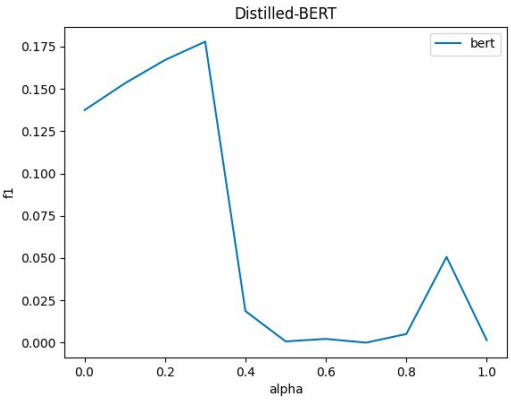
As we can see in the above table, BERT has 200 times the number of parameters as its distilled version. As a result, we see a significant drop in the evaluation time as well as in the training time per epoch on the go_emotions dataset. However, there is certainly a tradeoff between performance and speed, and after comparing the models in terms of these qualities, a user will have to decide the importance of performance vs. speed for the given task. For example, if the model has to be retrained frequently, perhaps the user should prioritize time, sacrificing some amount of accuracy in the process. However, if the same model can endure for a long period of time, the user may be able to instead prioritize the model that takes longer to train but also has better performance.



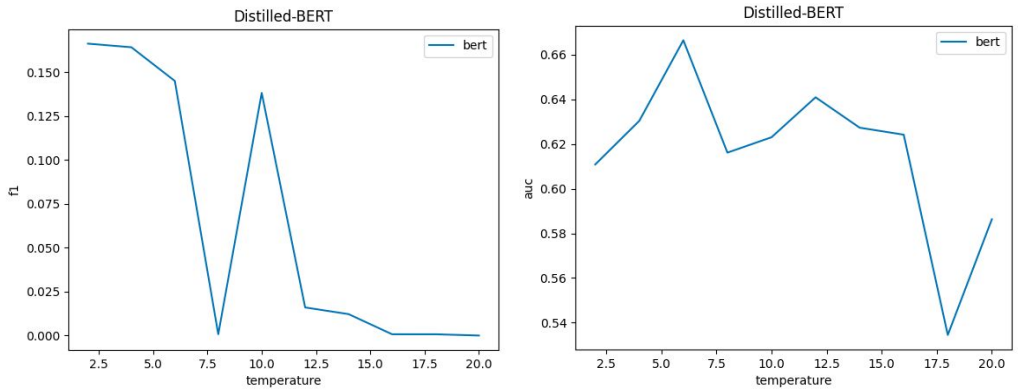


We trained models for the following number of epochs: 1, 2, 5, 10, 20. As we can see in the graphs shown above, all 3 models reached their optimum validation loss values with only a few epochs of training. This is also commonly seen in practice where these kinds of models are usually trained for a small amount of epochs. We can also see that XLNET outperforms BERT while RoBERTa outperforms both of them. This is expected because RoBERTa was designed as an improvement over BERT.





These graphs show the performance of our Distilled version of BERT with respect to the alpha parameter. The alpha values we tested were: 0, 0.1, 0.2, 0.3, …., 0.9, 1. The first graph shows ROC area under the curve as a function of alpha, and the second graph shows F1 score as a function of alpha. As I described before, the student learns partially from its loss w.r.t the teacher model and partially from the loss w.r.t the actual labels. Alpha is the weight corresponding to the loss w.r.t the actual labels, and (1-alpha) is the weight of the loss w.r.t the teacher's soft labels. In the original paper, the researchers suggest that low values of alpha should work better so that the student model learns largely from the teacher, which we see in these graphs.



In these graphs, we vary temperature values, testing even values from 2 to 20 inclusive. In our case, we observe that lower values of temperature tend to be better, because our student model is quite simple-- a possible explanation is that adding too much regularization for a simple model leads to underfitting. If the teacher and student models were more similar in size, larger values of temperature could help.

VI. Conclusion

We aimed to achieve several objectives through the duration of this project. For one, we hope that this framework yields key properties allowing a user to select the appropriate model for a given task. We also hope the experimental evaluation section gave useful insight into how the tool can be used and how the results can be interpreted. Finally, we have several ideas about how the functionality offered can be expanded in the future.

For one, we would like to expand the set of support tasks and models beyond the currently offered functionality; ideally, this framework should be generalizable to any combination of language model and task that a user wants to explore. Furthermore, we would like to provide more leeway for customization on the user's end, in terms of distillation techniques (i.e. the user should be able to provide a custom student model) and more fine-grained control (e.g. the user should be able to specify a learning rate schedule). In addition to providing more leeway, we also want to add functionality for other compression techniques, like quantization and pruning. Lastly, we would like to add multi-GPU support for users that have access to more compute resources to speed up the training and inference processes.