Multilabel discriminative modeling on TinyStories dataset

Ilaria Antonellini, Simone Pasquini

Introduction to Machine Learning course – 2024/2025

1 Introduction

Among the major tasks in Natural Language Processing, Text Classification (TC) has been of utmost importance for numerous real applications, ranging from information retrieval [1] to sentiment analysis [2]. The aim of TC consists in taking input texts, extracting their relevant features and categorizing each text into various predefined classes. This work focuses on two discriminative approaches for multilabel (or multinomial) TC, which are well suited to handling each label as an independent binary classification problem.

Prior to introducing the specific models architectures, it is useful to clarify more formally the task addressed by supervised models for multilabel TC. Given a collection of N labeled documents $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, where $\mathbf{x}_i = \{x_{i1}, \ldots, x_{iT}\}$ denotes a document of length T words and $\mathbf{y}_i \in \{0,1\}^L$ is a multi-hot encoded label vector over $l = 1, \ldots, L$ possible labels, multilabel models are trained to learn the presence or the absence of each label l independently. Discriminative models address this task by learning, for each label, an optimal decision boundary through the maximization of the conditional probability $p(y_{il} = 1|\mathbf{x}_i)$, which represents whether each label l is relevant for document $\mathbf{x}_i$.

This report aims to present and compare two different discriminative supervised models for multilabel TC, trained on the TinyStories dataset [3] and implemented using `Python` with `scikit-learn` [4] and `PyTorch` [5] libraries. The TinyStories dataset consists of short children stories written with simple words, where each story is annotated with zero or more semantic tags extracted from a fixed set of six classes: *BadEnding, Conflict, Dialogue, Foreshadowing, MoralValue* and *Twist*. For each tag, the two models solve six binary classification tasks, in accordance with the aforementioned principles of multilabel classification.

The selection of the two models was driven by the intent of comparing a traditional method, namely the Logistic Regression [6], with a modern Transformer-based model, DistilBERT [7]. Logistic Regression is a well-established linear classifier which, trained on the TinyStories dataset, serves as a reliable baseline model. In contrast, DistilBERT represents one of the state of the art architectures, by excelling in learning complex features and capturing deep semantics [8]. The DistilBERT model, originally pre-trained on corpora including Wikipedia [9] and BookCorpus [10], is fine-tuned on TinyStories dataset.

Both Logistic Regression and DistilBERT models share the preparation of the TinyStories dataset. Firstly, the dataset is downloaded from [3]. Secondly, the dataset is randomly split into training set (50 000 stories), used to learn model parameters, and test set (10 000 stories), used to provide the final evaluation of the models. The choice of the training set size was driven by the compromise of employing sufficient data for learning the classification task obtaining generalization, while still keeping reasonable computational time. Furthermore, to maintain the test set untouched during development, the full training set is split into a training subset (40 000 stories) and a validation set (10 000 stories), which allows to evaluate model performances during the training phase. In addition, for both models, the class labels are defined in a list called `TAGS`, encoded to multi-hot vector through `scikit-learn`'s `MultiLabelBinarizer`.

Finally, several metrics are used to evaluate the performance of both models. To address the imbalance of labels in the dataset, optimized per-class thresholds are applied as decision boundaries. The evaluation criteria include accuracy and F1 score per-label, as well as Receiver Operating Characteristic (ROC) curves and the Area Under the Curve (AUC), fundamental metrics for assessing multilabel binary classification.

2 Multinomial Logistic Regression with a one-versus-the-rest classifier

The first implemented model is a multinomial Logistic Regression [11]. In particular, it is a one-versus-the-rest architecture which consists of L classifiers [12], each of which solves the binary problem of separating points in a particular class C_l from points not in that class. Each binary classification problem is solved by a Logistic Regression model. In this model, the aim is to train the weights $w \in \mathbb{R}^{d\times 2}$ of the function $f_w(\mathbf{x}) = w^T\mathbf{x}$, where $f_w : \mathbb{R}^d \to \mathbb{R}^2$, d is the dimension of the input and 2 are the possible outputs for each label. To have a probabilistic interpretation of the binary classification, $f_w(\mathbf{x})$ is passed to a softmax function $\tilde{f}_w(x) = \text{softmax}(f_w(x))$. The best weights are selected by minimizing the cost function:

$$\min_w \frac{1}{S} \sum_{i=1}^{n} s_i(-y_i \log(\tilde{f}_w(x)) - (1-y_i)\log(1-\tilde{f}_w(x))) + \frac{r(w)}{SC}, \quad (1)$$

where $S = \sum_{i=1}^n s_i$ with s_i the weights assigned to a specific training sample, C is the regularization strength and $r(w)$ is the regularization term. An input is assigned to a certain class if the predicted probability is above a certain threshold. A limitation of this approach is class imbalance, which arises from the uneven frequency of classes in the dataset. The multinomial Logistic Regression model is developed using standard classes from `scikit-learn` library [4].

2.1 Training and validation phase

Several steps must be followed to train the model. Firstly, the stories are converted in a matrix of TF-IDF (Term-Frequency Inverse-Document-Frequency) features, which considers terms that are more informative relative to the most frequent ones. This is implemented by combining two objects of the class `TfidfVectorizer` through the use of `FeatureUnion`. These objects allow to recognize the most frequent words and characters inside the stories. To perform a binary classification of the input as in Eq. 1, an object of the class `LogisticRegression` is defined. Here, the arguments passed are `class_weight="balanced"`, to properly treat class imbalance, and `solver="liblinear"`, which improves the performance of binary classification. The `LogisticRegression` is then passed as an argument to `OneVsRestClassifier`, which allows to perform a one-versus-the-rest multiclass strategy. This consists in fitting one classifier per label, where each classifier is trained to distinguish a given class from all the others.

In the context of multilabel classification, `OneVsRestClassifier` estimator uses the binary relevance method, which involves training one binary classifier independently for each label. To train the model a 3 fold K-fold validation is implemented using the class `GridSearchCV`. In particular, a set of parameters to test is defined together with two scores: macro F1 score[1] and the logarithmic loss. The K-fold validation is performed by applying the `fit` method on the training dataset. The best parameters are saved and then used

[1]Macro F1= $\frac{1}{L}\sum_{l=1}^{L} \text{F1}_l$, where F1_l is the F1 score computed independently for each label l.

to fit again the training dataset to produce the final model. The performances of the models in the K-fold validation are shown in Fig. 1.

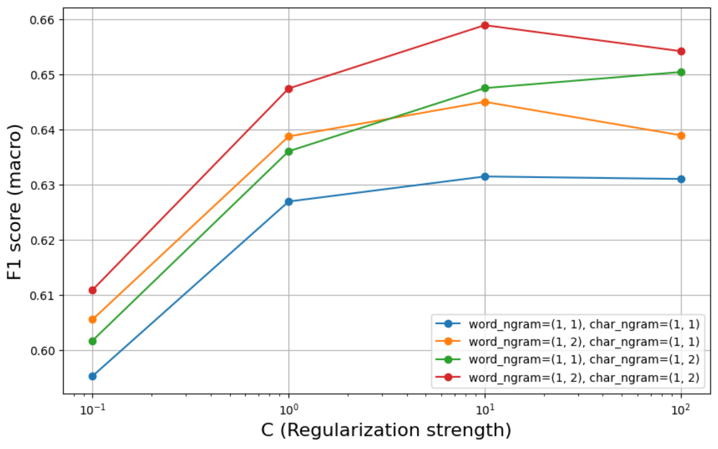


Figure 1: The plot illustrates the results of the K-fold validation. It shows the average F1 score with respect to the C coefficient for different combinations of unigrams and bigrams.

During training, a threshold tuning on the validation set is performed. The function `best_threshold` selects the best threshold for each label, i.e. the value that the predicted probability from the Logistic Regression must exceed for the label to be assigned. The model is evaluated with the validation set with the function `evaluate` which computes the accuracy and the F1 score for each label. These results are then printed out by the function `print_results` to have an order of magnitude of the F1 score as a quality parameter for the best model. The last step is to save the final parameters of the model and the tuned thresholds in a file.

2.2 Testing phase

The test dataset is used to assess the true performance of the model. The final model parameters are applied to predict the labels for each story in the test set. The model's performance is evaluated and printed with the same `evaluate` and `print_results` functions aforementioned. The ROC curve for each label is plotted as well as the confusion matrices for each label vs all the other labels. The results are reported in Sec. 4.

3 Pre-trained DistilBERT architecture

The second model considered is a pre-trained DistilBERT base model (uncased) [7], fine-tuned on the TinyStories dataset. The choice of this architecture is firstly motivated by the suitability of Bidirectional Encoder Representations from Transformer (BERT) models for text classification tasks. Indeed, bidirectional encoders use self-attention layers to map sequences of input embeddings $\{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iT}\}$ to sequences of output embeddings $\{\mathbf{h}_{i1}, \dots, \mathbf{h}_{iT}\}$,[2] where each output embedding has been contextualized using information from the input sequence [13].[3] These resulting output embeddings constitute contextualized representation of each input token, making masked language models particularly effective for interpretative tasks such as TC. Secondly, employing DistilBERT as a pre-trained architecture leverages transfer learning,[4] where general linguistic features extracted from large corpora adapt to downstream, task-specific applications. In this regard, finetuning consists of post-training the DistilBERT model on TinyStories supervised data to specialize the input representations to the specific narrative structure of the stories.

The selection of DistilBERT was driven by its favorable trade-off between performance and computational efficiency, being 40% smaller (with 66 millions parameters) and 60% faster than BERT [14], while retaining 97% of its language understanding capabilities [7]. DistilBERT achieves these properties through an optimized BERT architecture which reduces the Transformer layers from 12 to 6, removes token-type embeddings[5] and the pooler, while still retaining multi-head self-attention mechanism.

The DistilBERT model is instantiated through a task-specific variant called `DistilBertForSequenceClassification`, where an application-specific circuitry (the classifier head) is already appended on top of the Transformer encoder. This classifier head, consisting of a feed-forward neural network composed of linear layers and dropout, maps the final hidden state representation produced by the Transformer layers to a six-component logit vector, being the same size of `TAGS` list. Only the parameters of the last two Transformer layers and of the classifier head are updated during training, maintaining computational efficiency and avoiding catastrophic forgetting [15]. The final architecture of the used DistilBERT model is summarized in Fig. 2.

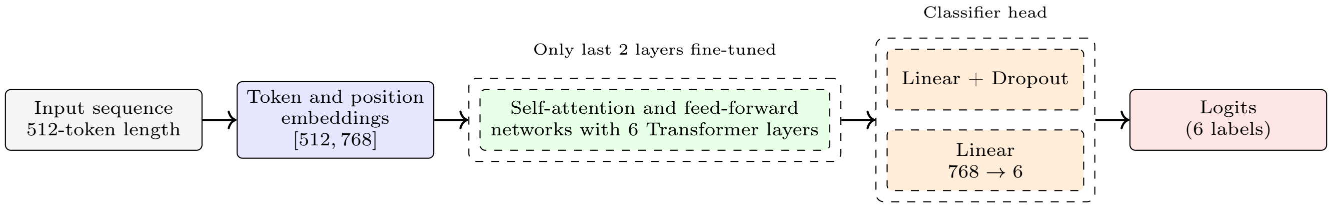

Classifier head
Only last 2 layers fine-tuned
Input sequence 512-token length
Token and position embeddings $[512, 768]$
Self-attention and feed-forward networks with 6 Transformer layers
Linear + Dropout
Linear $768 \to 6$
Logits (6 labels)

Figure 2: Horizontal overview of the `DistilBertForSequenceClassification` architecture used for multilabel TC. Dashed lines indicate the layers affected by fine-tuning. Total trainable parameters are 14 770 950.

[2] For simplicity, T is directly treated as the number of tokens for each i-th story (already accounting for subword tokens and for any padding or truncation applied) rather than number of words, as previously mentioned in Sec. 1.

[3] In bidirectional models the contextualization for each token is based on attending to both preceding and subsequent tokens.

[4] Method of acquiring knowledge from one task or domain, and then applying it (transferring it) to solve a new task [13].

[5] Although this lowers the performances on Next Sentence Prediction (NSP), this task does not constitute the objective of this work.

3.1 Training and validation phase

In accordance with the dataset preparation and the notation outlined in Sec. 1, the second model is trained to independently estimate the probability that each label applies through a sigmoid-activated multilabel classifier $\sigma(\cdot)$:

$$\hat{y}_{il} = \sigma\left(f_\theta\left(\mathbf{x}_i\right)_l\right), \quad l = 1, \ldots, L \quad \text{with} \quad L = 6, \tag{2}$$

where $\mathbf{x}_i = \{x_{i1}, \ldots, x_{iT}\}$ denotes the i-th tokenized story, with $i = 1 \ldots, N$, and $f_\theta(\mathbf{x}_i)_l$ is the l-th component of the logit vector returned by the model. For notational simplicity, in Eq. 2, T denotes the fixed number of tokens in each story after tokenization, truncation and padding, instead of the number of words per story. In particular, tokenized stories are truncated or padded to a fixed sequence length $T = 512$. This choice preserves to a large extent the full narrative structure of input samples (the average story length is approximately 160 words), and is compatible with the context window of BERT-based architectures.

The tokenization is performed through `DistilBertTokenizerFast`, which not only provides end-to-end tokenization including punctuation and WordPiece subword decomposition [16], but also an attention mask. The attention mask is employed by the DistilBERT self-attention mechanism to ignore tokens that do not contribute to contextual representations, such as padded tokens. Both the tokenization and the attention mask are stored into a custom `PyTorchDataset` object. Subsequently, each tokenized input is mapped to an embedding vector $\{\mathbf{z}_{i1}, \ldots, \mathbf{z}_{iT}\}$, with $\mathbf{z}_{ij} \in \mathbb{R}^{768}$, which is then processed by the DistilBERT encoder to produce contextualized representations $\{\mathbf{h}_{i1}, \ldots, \mathbf{h}_{iT}\}$, with $\mathbf{h}_{ij} \in \mathbb{R}^{768}$. The final hidden representation is passed to the task-specific classifier head, producing the aforementioned logit vector $f_\theta(\mathbf{x}_i)_l$.

The training procedure, implemented in the `train` function, is based on the minimization of `BCEWithLogitsLoss`, a Binary Cross-Entropy loss with logits $\mathcal{L}$, weighted to address per-class imbalance:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{L}\left[w_l \cdot y_{il} \cdot \log \hat{y}_{il} + (1 - y_{il}) \cdot \log\left(1 - \hat{y}_{il}\right)\right], \tag{3}$$

where w_l represents the negative-to-positive ratio for label l, computed through `class_imbalances` function, and $N = 64$ is the mini-batch size. The model parameters are updated via `AdamW`[6] optimizer with a weight decay parameter $\lambda = 0.01$ and a learning rate $\eta = 0.001$, the latter diminished by a factor 10 (through `StepLR`) during training.

As discussed in Sec. 3, to achieve efficient learning while still limiting computational costs, training affects only the parameters of the classifier head and of the last two layers of the Transformer. To train the model using the training subset, each 64-stories mini-batch is processed sequentially until the end of one epoch. At this point, the model is evaluated on the validation set using `evaluate2`, which reports the average validation loss, the accuracy per-class, and the F1 score per-class. This operation is repeated for 5 epochs. The batch size as well as the number of epochs are chosen to balance performance stability on the validation set with computational time. To compare the predicted probabilities found in Eq. 2 with the true multi-hot encoded labels (the target), per-class thresholds are applied to define a decision boundary for each class. To find the best decision boundary per class, a threshold sweep is performed on the validation set using the `threshold_tuning` function, which computes and prints the threshold vector that maximizes the F1 scores. Successively, the optimized thresholds are incorporated into the model evaluation on the testing set. Subsequent to the training-validation procedure and the threshold tuning, the model is retrained on the full training set using the same procedure described above.

3.2 Testing phase

The testing phase is implemented to evaluate the model's performance on the unseen testing set. To assess the model, metrics include the accuracy and F1 score for each class, computed by the `evaluate2` function. In addition, ROC-AUC curves and confusion matrices are generated for each tag. In Sec. 4, the results of the Transformer model evaluation are discussed and compared with those of the Logistic Regression approach.

4 Results

The results of the models' evaluation on the test set are shown in Tab. 1. One observes that for both models the accuracies are high, above 0.88, for each label. Despite this result, the accuracy is not a good estimator of the performance for multiclass classification problems because it does not take into account class imbalances. Hence, in Tab. 1, the F1 score is illustrated. This metric provides a reliable estimate of model performance, as it is robust to class imbalances by combining precision and recall. From the obtained F1 score values, it can be stated that the Transformer model performs better in five out of six classes. The only class in which the multinomial Logistic Regression works better is the classification of the stories labeled with *Dialogue*.

Label	Accuracy	F1 score
BadEnding	0.9607	0.7911
Conflict	0.8822	0.3990
Dialogue	0.9021	0.9130
Foreshadowing	0.8822	0.4062
MoralValue	0.9599	0.8090
Twist	0.9329	0.8290

(a) Accuracy and F1 score for each label evaluated on the test dataset of the multinomial linear regression model.

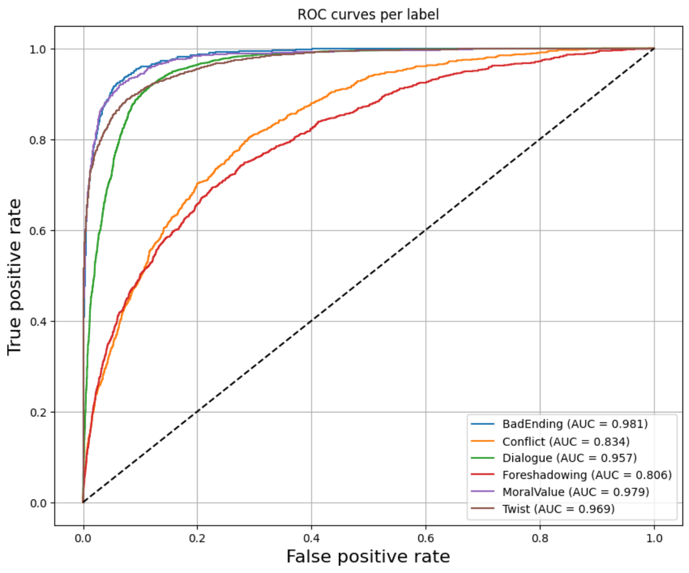
Label	Accuracy	F1 score
BadEnding	0.9866	0.9263
Conflict	0.9021	0.4576
Dialogue	0.9015	0.9110
Foreshadowing	0.8885	0.4220
MoralValue	0.9643	0.8366
Twist	0.9389	0.8485

(b) Accuracy and F1 score for each label evaluated on the test dataset of the Transformer architecture using pre-trained DistilBERT model.

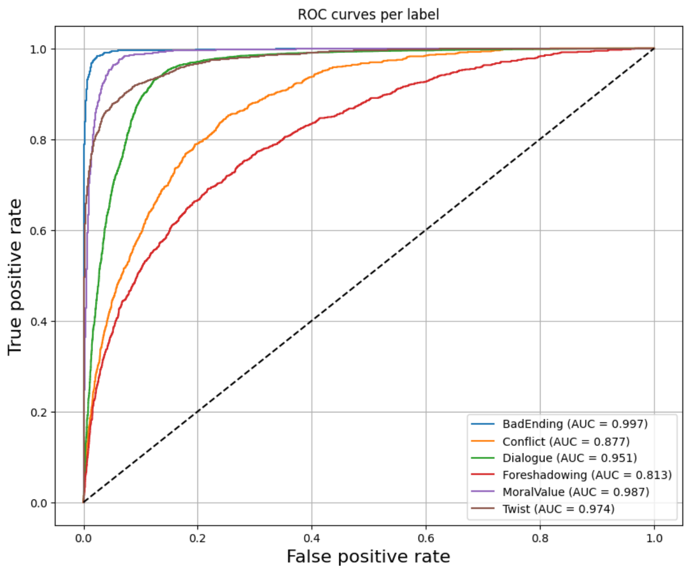
Table 1: Accuracy and F1 score for the two models.

The ROC curves, one per label, are shown in Fig. 3 for both models. The value of the Area Under the Curve (AUC) is reported for each label. These plots are a graphical representation of the results aforementioned. The closer is the curve to the top left corner, the better is the model. Instead, the diagonal dashed lines in Fig. 3 represent the performance of the random classifier.

[6] `AdamW` improves `Adam` generalization by decoupling weight decay from the gradient update [5].



(a) ROC curves for the multinomial linear regression model.



(b) ROC curves for the Transformer architecture using pre-trained DistilBERT model.

Figure 3: ROC curves for the two models.

To compare the two models as a whole and not only label by label, the macro F1 is evaluated. For the multinomial Logistic Regression it is 0.6912 while for the pre-trained DistilBERT architecture achieves 0.7337. These results support the selection of the pre-trained DistilBERT model over the multinomial Logistic Regression model.

The DistilBERT model demonstrates higher quality in comparison with the Logistic Regression model. This can be attributed to DistilBERT's ability to learn more successfully the contextual relationships among the words in the text. Moreover, the Logistic Regression relies on TF–IDF features, which evaluates word and character frequencies independently, whereas the DistilBERT architecture leverages self-attention mechanisms to learn contextualized representations of words. Contextual learning is particularly effective for narrative features such as *BadEnding*, *Conflict*, *Twist*, *Foreshadowing*, and *MoralValue*, which often depend on the narrative structure rather than isolated words. As a result, DistilBERT model is more appropriate for handling a narrative text, leading to higher F1 scores for most labels. The comparable performance observed for the *Dialogue* label suggests that this feature is more directly associated with well defined patterns such as punctuation (e.g. quotation marks), which can be effectively captured even by the Linear Regression model. Overall, these results highlight the advantage of DistilBERT for multiclass story feature classification tasks.

5 Code execution modes

The developed code supports two execution modes that determine which code cells are executed. For each model described in Sec. 2 and Sec. 3, the training-validation phase and the test phase can be executed independently through the `set_mode` function, called three times throughout the code. The default execution mode is `'TEST'`, which loads the trained model parameters and directly evaluates the model on the test set, avoiding retraining and validation procedures. This design allows the test phase to be performed as a standalone execution. To retrain the models, the execution mode can be switched to `'TRAIN'`, enabling training, validation, and testing while saving the updated model parameters.

6 Personal contributions

Ilaria Antonellini developed the first model of the Colab notebook: "1. Implementation of a multinomial Logistic Regression by using a one-versus-the-rest classifier". Simone Pasquini developed the second model of the Colab notebook: "2. Implementation of Transformer architecture". The Colab notebook section "Notebook execution modes" is realized by both contributors, as well as this report.

References

[1] C. Manning, P. Raghavan, and H. Schütze. "Introduction to Information Retrieval". Cambridge University Press, 2008.

[2] B. Pang and L. Lee. "Opinion Mining and Sentiment Analysis". *Foundations and Trends in Information Retrieval* 2 (Jan. 2008), pp. 1–135.

[3] skeskinen. "TinyStories-GPT4 Dataset". Accessed: 2026-01-14. Hugging Face Datasets, 2025.

[4] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[5] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". 2019.

[6] D. Cox. "Analysis of binary data, second edition". Routledge, Feb. 2018.

[7] V. Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". 2020.

[8] Y Song, X Liu, and Z. Zhou. "A comprehensive review of text classification algorithms". *J. Electron. Inf. Sci.* 9.2 (2024).

[9] W. Foundation. "Wikimedia Downloads".

[10] Y. Zhu et al. "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". *The IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[11] C. M. Bishop. "Pattern Recognition and Machine Learning". Information Science and Statistics. New York, NY, USA: Springer, 2006.

[12] K. P. Murphy. "Machine Learning: A Probabilistic Perspective". Adaptive Computation and Machine Learning. First edition. Cambridge, MA, USA: MIT Press, 2012.

[13] D. Jurafsky and J. 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 6, 2026. Prentice Hall, Pearson Education International, 2026.

[14] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". 2019.

[15] D. Lopez-Paz and M. Ranzato. "Gradient Episodic Memory for Continual Learning". 2022.

[16] M. Schuster and K. Nakajima. "Japanese and Korean voice search". *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152.