Ministry of Education and Science of the Russian Federation
Moscow Institute of Physics and Technology (State University)

PhysTech School of Applied Mathematics and Informatics
Department of Intelligent Document Processing

Bachelor's Degree Thesis

Using Internal Representations of Frozen Transformer Models to Solve Downstream Tasks

Author:
Student of Group B05-031
Dremov Aleksandr Olegovich

Scientific Supervisor:
Voropaev Pavel Mikhailovich

Moscow 2024

Abstract

Using Internal Representations of Frozen Transformer Models to Solve Downstream Tasks

Dremov Alexander Olegovich

Large transformer models are currently one of the main methods for solving various tasks. For example, they are used for text summarization, speech recognition, sentiment analysis of text, etc.

However, large models require significant resources, which creates problems when there is a need to solve multiple tasks simultaneously. This work proposes a solution according to which an existing transformer model, solving the primary task (base model), can be reused to solve related tasks on the same input data. The proposed solution efficiently reuses the computed results of the hidden layers of the base model to solve related tasks without making changes to the base model.

According to the experimental results, the proposed approach achieves quality comparable to SOTA (state-of-the-art) solutions in the field, competing with solutions that use full model retraining.

Contents

1 Introduction

1.1 Information on the Problem Domain

Transformers, first introduced in the paper "Attention Is All You Need" [1] by the Google team in 2017, demonstrated outstanding results compared to recurrent and convolutional neural networks. The algorithm have become the foundation for creating models such as BERT ("Pre-training of Deep Bidirectional Transformers for Language Understanding" [2]), GPT ("Language Models are Unsupervised Multitask Learners" [3]), and T5 ("Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer" [4]), which have radically changed the approaches to solving complex natural language processing (NLP) tasks. The main component of these models is the attention mechanism, which allows for the effective processing of data sequences.

The attention mechanism operates on the principle of "highlighting" key elements in the input sequence that the model should focus on to perform the task. This enables transformer models to consider context and relationships between elements regardless of their position in the sequence. This solves the problem characteristic of recurrent neural networks of poor performance with long dependencies.

Transformer models have found extensive application in automatic text processing, including tasks such as machine translation, text summarization, sentiment analysis, question answering, and text generation. For instance, the GPT-3 model not only enables meaningful dialogues with users but also performs deep text analysis for information search and extraction. In commercial applications, such models are used in recommendation systems, support service chatbots, and translators.

However, it should be noted that transformer models are resource-intensive. Thus, modern models that demonstrate the best results (state-of-the-art models) are usually very large. For example, the GPT-3 model has more than 175 billion parameters. This creates certain difficulties in their use, especially on devices with limited resources.

Therefore, the relevance of the research is due to the wide range of applications of transformer models (transformers) in various NLP tasks. Thus, it is necessary to develop more efficient methods for using transformer models.

1.2 Transformer Model Architecture

The classic transformer model consists of an encoder and a decoder, which contain layers made up of multihead-attention [1], normalization, and fully connected layers. A visualization of the architecture is presented in Fig. 1.

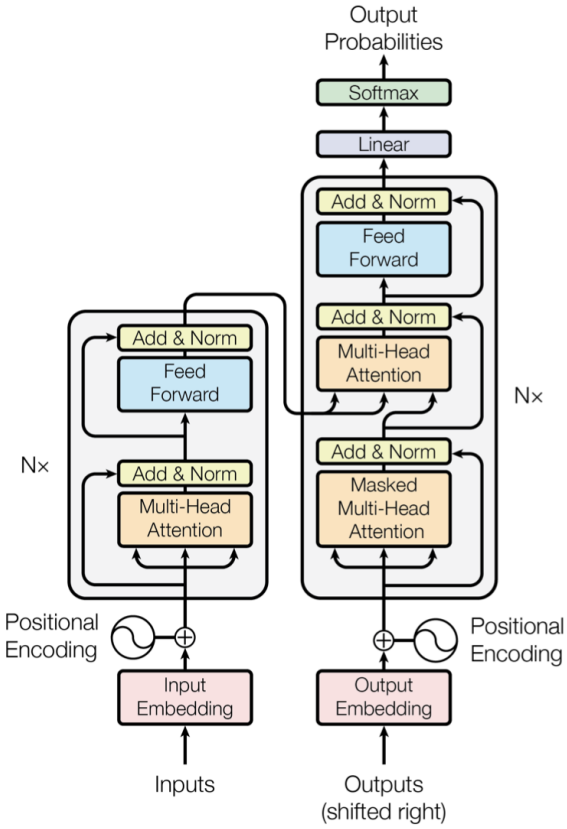


Figure 1: Visualization of the classic transformer model architecture from the article "Attention Is All You Need" [1]. On the left are the encoder blocks, on the right are the decoder blocks.

At the core of the architecture lies the attention mechanism (dot-product attention) [1]. For a triplet of tensors Q, K, V, the algorithm is described as follows:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where d_k is the dimensionality of features Q and K. The tensors Q, K, V are the inputs to the algorithm.

In the transformer layer, the multi-head attention algorithm is used:

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}\left(head_1, \ldots, head_h\right)W^O, \\ head_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right),\end{aligned}$$

where W_i^Q, W_i^K, W_i^V, W^O are trainable matrices.

In the encoder layers and in the masked attention layers of the decoder, $Q = K = V$— this setup is called "self-attention". In the decoder layers, K, V represent the outputs of the encoder, while Q is the output of the previous operation of the decoder layer.

In summary, the encoder consists of N identical layers, each of which includes:

1. A multi-head attention layer.

2. A fully connected feed-forward network (FFN) layer.
3. Layer normalization.
4. Residual connections.

Similarly, the decoder also consists of N identical layers, each of which includes:

1. A masked multi-head attention layer, preventing attention to future positions during sequence generation.
2. A multi-head attention layer, similar to the encoder's, which uses the outputs of the encoder as K, V.
3. A fully connected feed-forward network (FFN) layer.
4. Layer normalization.
5. Residual connections.

The internal layers of the transformer are called hidden layers, and the data at their output are referred to as hidden states. The final layer of the decoder converts the output representations into the predicted probability distribution.

2 Problem Statement

It is important to note that often, in addition to solving the main NLP task, there are several related tasks for the same input data. For example:

- summarizing text and identifying named entities within it,
- translating text and classifying it,
- recognizing text from a voice recording and classifying the user's emotion.

Moreover, one of the tasks is considered the main (primary) one, to which almost all computational resources are allocated to achieve high quality. The main task is the one that provides the key functionality of the product and should have, as far as possible, high quality.

Thus, the related task is forced to operate under conditions of limited computational power. In such a setup, it is not feasible to use separate large state-of-the-art models for solving related tasks due to the high demand for resources mentioned earlier.

On the other hand, a large number of independent big models to solve each separate task requires sequential processing due to resource demands. Such an approach slows down the entire service's operation and increases the potential response time.

In the case of computations on user devices, the use of a set of large transformer models for different tasks simply is not feasible due to severe limitations on available resources.

Therefore, the goal of this work can be formulated as follows: without negatively affecting the quality of the model solving the main task, it is necessary to efficiently solve related tasks under conditions of limited computational power.

3 Review of Existing Solutions

3.1 General Solutions

To save resources when using transformer models, various methods have been proposed, among which the main ones can be highlighted:

1. Quantization

   Quantization is a general approach to reducing the precision of calculations to speed them up, possibly at the expense of quality.

   For example, in "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference" [5], Google suggests a method of modifying the model to make predictions using faster integer arithmetic, rather than floating-point numbers.

2. Knowledge Distillation

   The main idea of distillation is that a compact model is trained on the output of a large model, which can contain richer and more useful information compared to standard data labeling. This allows simplifying the model while maintaining quality.

   For instance, DistilBERT [6] is a compact alternative to BERT with similar quality.

Such methods indeed save resources. However, when there is a primary task, the gained speed is often exchanged for an increase in the model size to improve quality. Therefore, when there are related tasks, the problem of limited resources remains, even when applying the above approaches.

3.2 Solutions Using a Single Model

To solve several tasks using a single transformer model, a series of approaches have been proposed:

1. Multi-task learning

   This approach involves training a single model to perform several tasks simultaneously, which allows for reducing computational costs by sharing common representations and parameters. An example is the T5 model [4], which uses a unified approach to processing various NLP tasks.

   However, this approach is not always applicable for several reasons: training a large model from scratch to solve an additional task can be too costly. Also, it is necessary to control the balance of data from different tasks to avoid overfitting on one of them, which can be difficult.

In addition to this, in such an approach, one model usually solves one task in one pass. Then, to solve several tasks, several passes are required, which negatively affects the performance of the system.

2. Domain-specific fine-tuning

   This method involves adapting the model based on data from different domains and tasks. Here, a transformer model can be trained on data from one domain and then fine-tuned on another, assuming that important knowledge is retained and transferred between tasks.

   The problem is that when fine-tuning on a new task, the quality on the original task may start to decline, which is unacceptable in the considered setting. This problem is described in the literature as the "Forgetting Problem" [7].

3. Cross-task adapters

   In this approach, adapter modules optimized for multitasking are used. They allow a single base model to perform multiple different tasks more efficiently. This method may include the dynamic embedding and training of new adapter layers, specific to each task. An adapter is a small neural network that is embedded into the architecture of transformer layers and modifies the internal computations.

   For example, the article "Cross-Lingual Transfer with Target Language-Ready Task Adapters" [8] explores the possibility of training adapter layers to adapt a task for different languages.

   The problem with this approach in the case of related tasks is the need for a complete recalculation of the base model. Since adapters are embedded into the model architecture, computations for different tasks will be completely different.

4. Methods of Efficient Fine-Tuning (PEFT).

   Methods of efficient fine-tuning (parameter efficient tuning, PEFT) have gained high popularity in the field of fine-tuning transformer models. The article "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning" [9] provides an overview of the most popular methods.

   The authors divide the approaches into several categories, with the division diagram presented in Fig. 2. However, upon examining each category, it can be concluded that many PEFT methods are not suitable for solving the given task. Thus, additive methods [10, 11], selective methods [12, 13, 14], and methods based on reparameterization [15, 16] alter the original model, which contradicts the task setting.

   The task could potentially be solved by soft-prompts [17]—methods of modifying the input of a transformer model and its internal states to adapt the model for a new task. However, this method is not capable of significantly altering the model's behavior, and

it changes the computations, which requires multiple applications of the model in the case of several tasks.

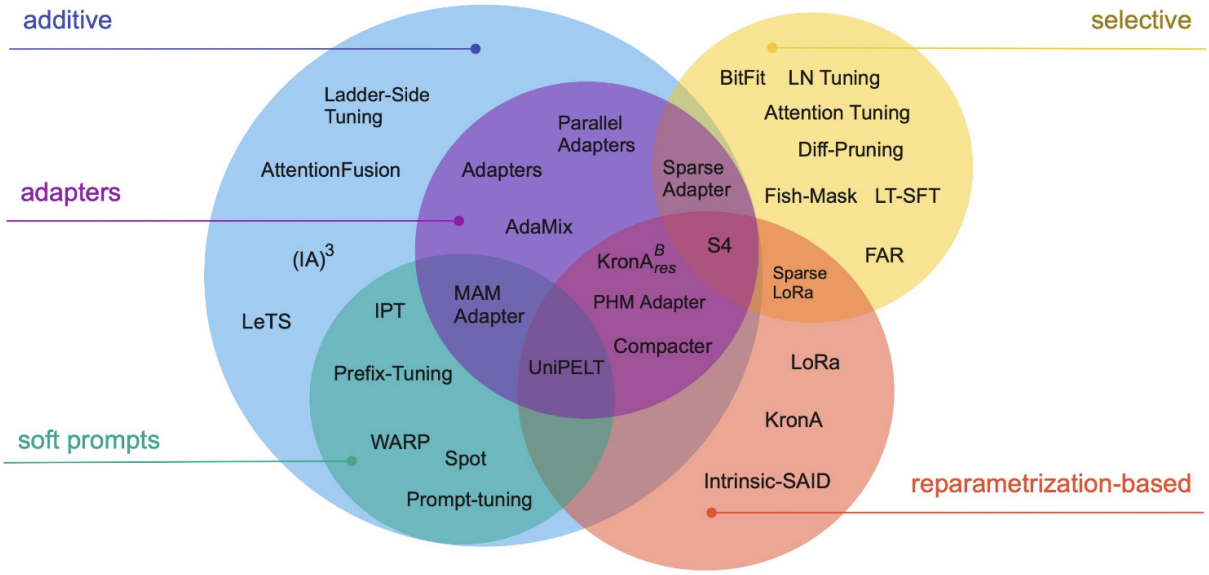


Figure 2: Categorization of PEFT algorithms. "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning" [9]

Thus, there are methods for saving resources when solving related tasks over the same input data. However, popular methods consider related tasks as equivalent and do not focus on the case of a primary task and a related one. This distinction is important because:

- the related task is forced to operate under conditions of limited resources,
- adding a related task should not negatively impact the quality of the primary task's execution.

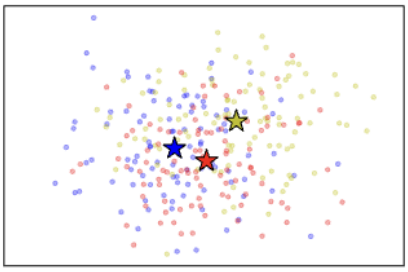
3.3 Solutions Using a Single Model Without Modifications

In addition to the literature reviewed above, there are approaches for solving related tasks based on supplementing the existing model without any modifications.

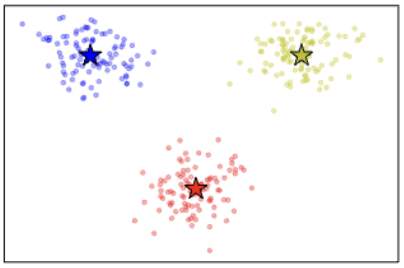
1. "Hidden State Variability of Pretrained Language Models Can Guide Computation Reduction for Transfer Learning" [18]

   In this article, an approach to solving the classification problem is considered. The essence of the method lies in the use of a pretrained transformer model, from which the "most informative" hidden layer is selected based on the metric of output variability. A layer will be considered informative if it has a large inter-class variance and a small intra-class variance. Examples are presented in Fig. 3.

   The method is interesting in that it does not require changes to the base model and efficiently reuses its calculations. However, the authors note that the layer selection



(subfigure) High intra-class variability, low inter-class variability



(subfigure) Low intra-class variability, high inter-class variability

Figure 3: Example from "Hidden State Variability of Pretrained Language Models Can Guide Computation Reduction for Transfer Learning" [18]. The figures show the output values of various hidden layers of the model. The color indicates the true class of the object that generated the depicted values. On the left (3subfigure): a layer that is not informative for the task. On the right (3subfigure): a highly informative layer for the task

method may not always work and is merely a heuristic. A trivial example of the algorithm's failure is when clusters are arranged in concentric circles. Such a structure is easily separable into classes but does not match the authors' heuristic.

Furthermore, the layer selection algorithm is tailored for the classification task, and cases involving other tasks are not discussed.

2. "An Algorithm for Routing Vectors in Sequences" [19]

   The solution proposed in the article [19] utilizes the idea from capsule neural networks [20] to combine all hidden states of the transformer model.

   The algorithm dynamically selects useful features from each layer of the model and uses them to compute the final prediction. However, the algorithm requires storing all hidden states of the base model, which for large transformer architectures can amount to several gigabytes of memory per input example.

   A practical drawback of the algorithm is the introduction of a new type of layer with non-trivial tensor operations. For the rapid application of this algorithm, it is necessary to efficiently implement the proposed operations using graphics processing unit (GPU) methods, which is not trivial.

   Figure 4 presents an example of the algorithm's operation on a single example. It is evident how the algorithm uses different hidden layers of the neural network for different tokens to make a prediction.

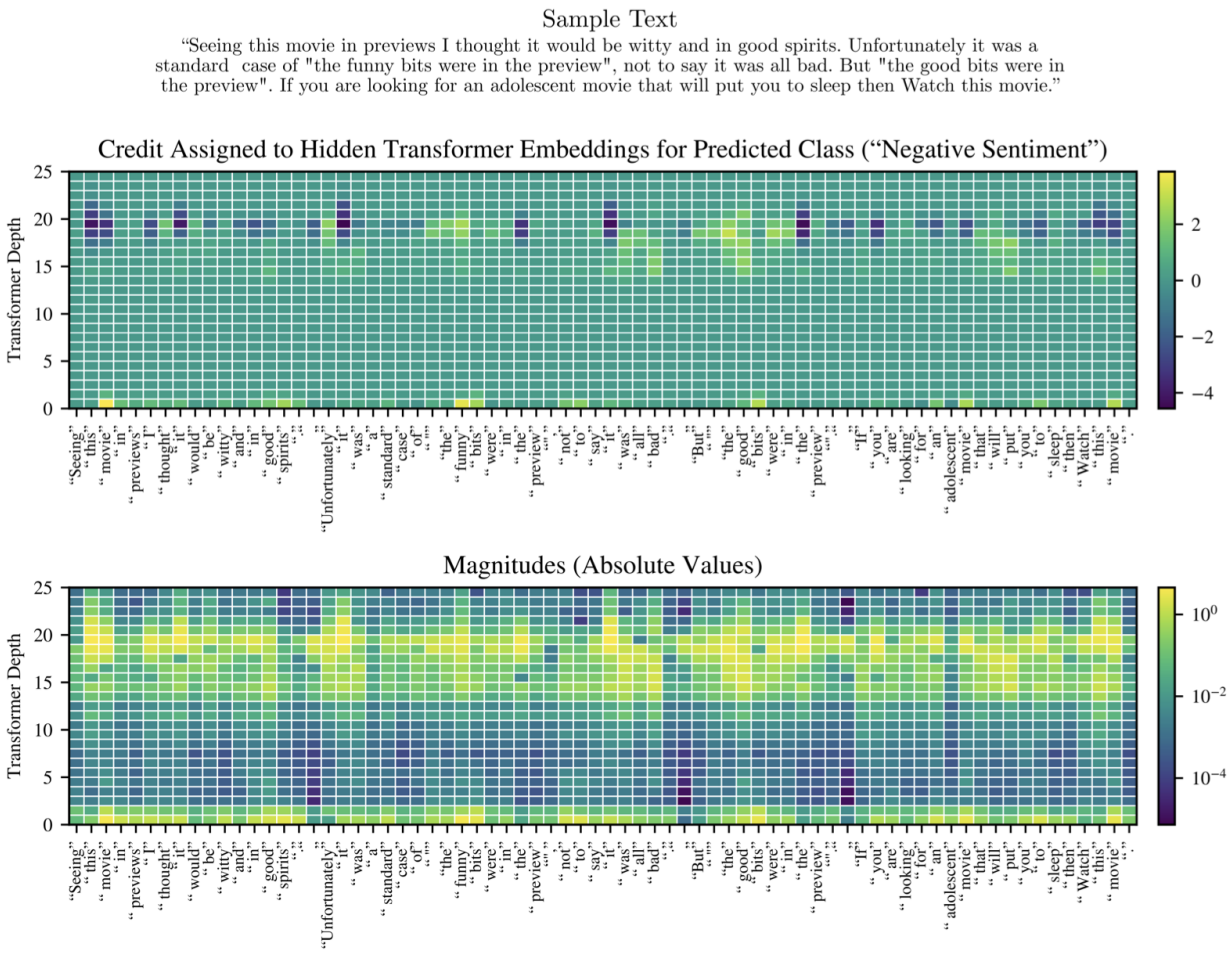


Figure 4: Visualization of the algorithm from the article "An Algorithm for Routing Vectors in Sequences" [19]. Shown is the contribution of each hidden layer of the model to the prediction. The influence of the layer is calculated by aggregating the influence across all features of the layer

3. "Head2Toe: Utilizing Intermediate Representations for Better Transfer Learning" [21]

   This method, developed by Google, suggests taking all hidden layers and training the required model on their concatenation. Let L be the number of hidden layers in the transformer model, and h_i, $i \in [1, L]$ be the hidden states, then the proposed algorithm can be represented as follows:

$$h_{all} = [a_1(h_1), a_2(h_2), \ldots, a_L(h_L)],$$
$$z'_L = h_{all} W_{all},$$

   where a_i, $i \in [1, L]$ are dimensionality reduction through average pooling and normalization, W_{all} is a trainable matrix, and z'_L is the feature tensor used for the final prediction.

   The obtained structure is trained in an end-to-end manner on the required task with group-lasso regularization of the matrix W_{all}. Thanks to regularization, the matrix W_{all} becomes sparse, and a subset of rows with the highest "importance" is further

selected from it. This allows reducing the number of features in h_{all}. The visualization of the algorithm is shown in Fig. 5.

Similar to the previous article, the base model used for the algorithm remains unchanged. The algorithm shows good results on test datasets and solves the set task.

However, the algorithm is not without drawbacks—the main problem is the need to save the outputs of all layers for calculating h_{all}, which requires large amounts of memory, as noted by the authors themselves.

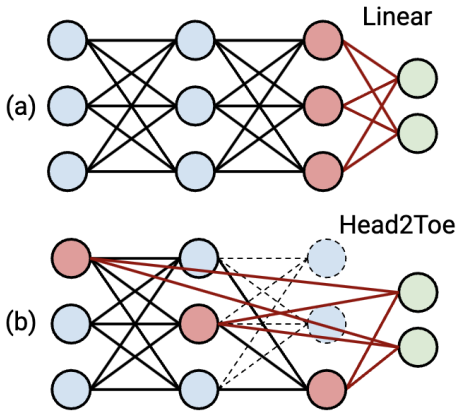


Figure 5: Visualization of the Head2Toe algorithm [21]. The key feature of the algorithm is shown — selecting various features from the hidden layers of the model for obtaining the final prediction

This problem is partially solved by regularization of W_{all} and the selection of a feature fraction. However, the optimal feature fraction obtained by the algorithm is often quite large, as follows from the appendix to the authors' work. In the case of large transformer models, storing all hidden states can reach gigabytes of additional memory for a single example.

4 Research and Solution Development for the Problem

4.1 Objective of the Study

To develop an algorithm that operates under resource constraints and in the presence of some large transformer model solving the main task. Without negatively affecting the quality of the main model, the algorithm should efficiently solve related tasks over the same input data.

4.2 Tasks of the Study

1. Select conditions for the algorithm development - choose a base model, and evaluation datasets (benchmarks).
2. Based on the literature and existing solutions, determine the quality of third-party solutions on the evaluation datasets.
3. Develop an algorithm that reuses the computations of the main model to solve related tasks.
4. Train the developed algorithm on the evaluation datasets and assess its quality on a held-out dataset (test dataset).
5. Evaluate the quality of the task/tasks solution using simple models (baseline).
6. Formulate conclusions about the algorithm's functionality.
7. Pose questions requiring further study.

4.3 Selection of Development Conditions and Quality Evaluation

As the base model, it was decided to choose two models: RoBERTa-Large [22] and BART [23]. RoBERTa-Large is a model based on the classic BERT architecture [1], which has been improved and refined.

The pretrained model solves the MLM (masked language modeling) task - predicting masked tokens in the text. This choice is motivated by the model's universality and its unbiasedness towards any specific task, which makes subsequent comparisons on various evaluation data more fair.In addition to RoBERTa-Large, the BART model, pre-trained on the task of text summarization, was selected for the experiments.

BART is an encoder-decoder model, in which the encoder corresponds to the BERT architecture, and the decoder has an autoregressive architecture, similar to GPT. This construction is trained end-to-end on the task of summarization. Unlike RoBERTa-Large, this model is narrowly specialized, which will help assess the quality of the algorithm in a setting close to real-world conditions.

As evaluation datasets, the following widely used datasets for assessing model quality were selected:

1. **IMDB Sentiment** [24].

   This dataset is used for text sentiment analysis, specifically in the context of movie reviews. The goal is to classify movie reviews as either positive or negative.

2. **GLUE CoLA** [25].

   The GLUE CoLA (Corpus of Linguistic Acceptability) dataset is aimed at evaluating the ability to distinguish between grammatically correct and incorrect sentences. CoLA is part of the GLUE benchmark (General Language Understanding Evaluation), which is used for comprehensive quality assessment of various NLP models.

3. **CoNLL** [26] NER.

   The CoNLL dataset is used for evaluating the quality of named entity recognition (NER). Thus, CoNLL-2003 contains English-language data with labels for various entities: names of people, organizations, locations, miscellaneous. Each input word from the text must be assigned exactly one of the listed classes.

This choice of data is motivated by the desire to test the algorithm on tasks of varying complexity and format: from simple sequence classification (IMDB) to token-level classification (NER). Thus, the obtained results will allow for a comprehensive assessment of the algorithm's quality. To understand how the achievable quality compares with the world's best solutions, a selection of the best results is listed in Appendix 6.4.

4.4 Motivation and Planning of the Algorithm

From the problem statement, it is known that there is a certain large (main) transformer model solving the primary task. The idea of the proposed approach is to reuse the computations of the main model without modifying it. Obviously, the quality of the solution to the main task will remain unchanged due to the invariability of the main model. Therefore, if the algorithm can obtain informative features from the already calculated outputs of the main model, it will be possible to effectively solve related tasks. Moreover, such a solution will be computationally simple if the processing of the features obtained from the main model is efficient and straightforward.

To obtain informative features, it is decided to use the output data of the hidden layers of the transformer model. The foundation of this idea is articles [27, 28, 29], which indicate that different hidden layers highlight features of varying levels of complexity and abstraction. In particular, the potential success of this approach is suggested by the previously reviewed article [21], in which the authors propose an algorithm for reusing the results of hidden layers to solve related tasks.

Thus, it is proposed to develop an algorithm that efficiently extracts useful information for an adjacent task from the features already calculated by the main model. With such a solution format, it will be possible to easily and quickly train on solving any set of adjacent tasks with minimal resource expenditure at the time of algorithm application.

5 Description of the Practical Part

5.1 Algorithm Description

Guided by the idea that the hidden states of a transformer model extract features of different levels, let us construct the following algorithm. Suppose a certain transformer model, consisting of l layers, receives input data X of dimension (B, T, K)—the dimension of the number of examples (batch), the dimension of the sequence, and the dimension of features, respectively. In the context of working with text, K corresponds to the dimension of the model's embedding layer. Then, the main model during its operation calculates the outputs of each of the l layers:

$$Z_i,\ \ i \in [1, l] \text{— the index of the hidden state.}$$

Apply the following transformation to each of the layers to obtain new features:

$$f_i = N(a(Z_i)W_i^T),\ \ i \in [1, l],$$

where a is some nonlinearity (activation function, for example, GELU [30]), $\underset{k\times K}{W_i}$ is a trainable parameter of the linear transformation of the base model's features into a smaller space of dimension k, N is normalization by the feature dimension (element-wise affine layer norm):

$$N(x) = \frac{x - \underset{dim=last}{\mathbb{E}} x}{\sqrt{\underset{dim=last}{\mathbb{D}} x + \varepsilon}}.$$

The idea of such a transformation is that the trainable linear mapping should extract useful information for the task from each specific hidden layer. Normalization brings features from different layers to a similar space. However, not every hidden layer potentially contains features useful for the task. To give the model the ability to select useful layers, we introduce the parameter $\underset{l\times 1}{p}$—a vector whose components determine the usefulness of each hidden layer for the task.

By using the introduced parameter, it is possible to combine features from each hidden layer, having first normalized p using softmax to obtain a distribution over the layers:

$$\begin{aligned}
&\text{Softmax}(p_i) = \frac{\exp(p_i)}{\sum_j \exp(p_j)},\\
&\tilde{p} = \text{Softmax}(p)\\
&F = \sum_{i=1}^{l} \tilde{p}_i \cdot f_i.
\end{aligned}$$

Thus, the final algorithm for obtaining features F to solve the necessary task can be written in matrix form as follows:

$$\begin{aligned}
&\tilde{p} = \text{Softmax}(p),\\
&F = \left(\underset{dim=last}{\text{Stack}} \left[N\left(a\left(Z_i\right)W_i^T\right) \mid i \in [1, l]\right] \times \tilde{p}\right) \in \mathbb{R}^{B\times T\times k}.
\end{aligned}$$

To summarize the algorithm briefly, the idea consists of mapping hidden states to a smaller, common space for all layers and in the weighted summation of the obtained features with weights reflecting the "importance" of each layer for the task. These weights are trainable, which gives the model the ability to select layers useful for the task.

On the obtained features F, one can further build an arbitrary neural network to solve the necessary task. The visualization of the algorithm is presented in Fig. 6.

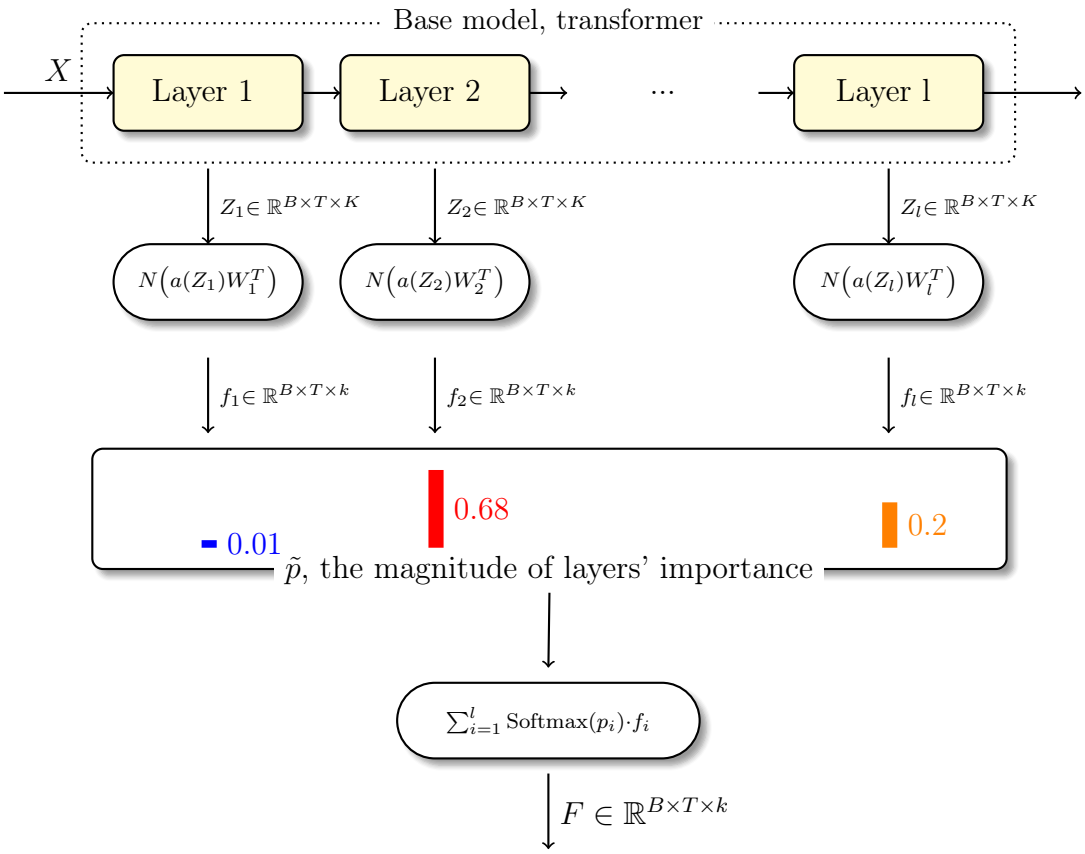


Figure 6: Visualization of the proposed algorithm

5.2 Adaptation to Specific Tasks

The obtained features can be further used for training on the required task in an end-to-end manner. It is evident that any algorithms for working with sequences of features can be applied to the features F. Depending on the task, the following transformations were performed on the tensor F in this work.

1. **Sequence Classification**

   In this task, the self-attention algorithm [1] is applied to the features F. Then, aggregation by averaging over the sequence dimension is performed $(B, T, k) \to (B, k)$, followed by further linear projection into the logits space (B, n_{classes}), where n_{classes} is the number of final classes.

2. **Token Classification (NER)**

Token classification required the addition of a more powerful neural network over the features F. However, the proposed transformation is still minor compared to the main model.

Thus, several classical encoder layers [1], described in the introduction of this work, were used. The only change was the placement of normalization—more recent articles [31] indicate that placing normalization before attention blocks leads to more stable training.

The obtained features (B, T, k) are projected by a linear transformation into the logits space for each of the input tokens: $(B, T, n_{\text{classes}})$, where n_{classes} is the number of classes.

5.3 Note on the Efficient Application of the Algorithm

It is easy to notice that due to the fact that the projected states f_i are weighted and summed, when applying the algorithm in practice, it is sufficient to keep only one additional tensor of size (B, T, k). Then, one can update the stored value by adding a new f_i with the corresponding coefficient. Thus, the algorithm requires very little additional memory when applied.

If we assume the presence of not one, but n additional tasks, then extending the algorithm for each of them will require a total tensor size of $n \times (B, T, k)$. Such scaling is quite efficient, allowing the algorithm to be applied simultaneously for a set of related tasks.

Furthermore, in the practical part, it will be shown that the importance component for the substantial part of the layers is nullified. This means that when applying the algorithm, it will not be necessary to aggregate all the hidden states, making the computations even more efficient.

It is worth noting separately that the base model remains unchanged, and the number of parameters and operations added by the proposed algorithm is small compared to the base model, making the application of the algorithm in practice efficient.

5.4 Training and Results

5.4.1 Training Setup

The models were trained using the PyTorch library [32] on NVIDIA graphics cards. The optimization algorithm for all trainings is gradient descent in the Adam modification [33].

Since each of the tasks being solved is a classification task, the optimized functional is cross-entropy:

$$H(p, q) = -\sum_x p(x) \log q(x).$$

As regularization techniques, weight decay and dropout [34] were applied. The optimization step size (learning rate) was halved in the absence of improvements over a certain period. Training was halted if no improvements were observed for an extended period.

To increase the model's ability to change the parameter p more rapidly during optimization, $\alpha \cdot p$ was used in the algorithm instead of p, where α is a hyper-parameter.

Regularization on p was also applied: entropy of the distribution on hidden layers was added to the optimized functional with a certain coefficient λ. Such regularization is intended to steer the optimization towards extracting only truly important layers, rather than remaining at a distribution close to uniform.

During training, it was also noted that changing the parameter p almost never resulted in the complete "zeroing" of less informative layers — some importance components were close to zero, but not equal to it. It turns out that nearly excluded layers do not add useful information but can be used by the model to overfit to the training data. Therefore, during training, a method of zeroing the importance component for less informative layers at a certain threshold (distribution cutoff) and subsequent renormalization of the distribution $\tilde{p}$ was tried.

In addition to this, an improvement in quality was observed on some tasks when positional embeddings were added to the tensor F—this can be explained by the fact that in such a modification, subsequent layers do not need to extract positional information from existing features, allowing the model to focus on solving the task and use parameters more efficiently.

In all experiments, the base model was either RoBERTa-large or BART—their weights were frozen and not trained.

5.4.2 IMDB

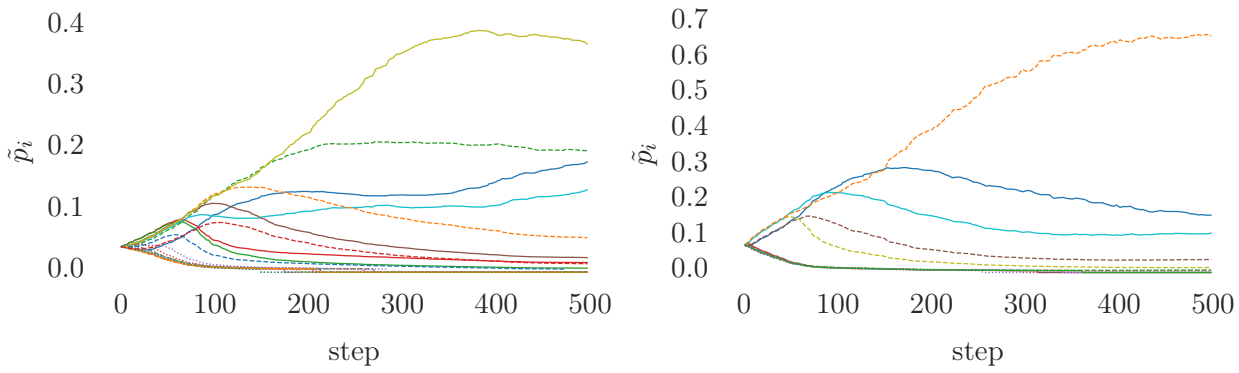
For the IMDB dataset, training was conducted with hyperparameters presented in the table in Fig. 7.

Parameter	**Value (RoBERTa-large)**	**Value (BART)**
init learning rate	10^{-4}	
weight decay	10^{-3}	
α	70	
λ	0	
distribution cutoff	$5 \cdot 10^{-3}$	
dropout	$3 \cdot 10^{-2}$	
attention heads	16	
k	64	
learning rate decay patience	5	
precision	amp fp16	
batch size	1200	588
add positional embeddings	no	
number of parameters	1.59M	0.87M
proportion of parameters from the base model	**0.4%**	**0.2%**

Figure 7: Training hyperparameters on the IMDB dataset and brief information about the number of added parameters

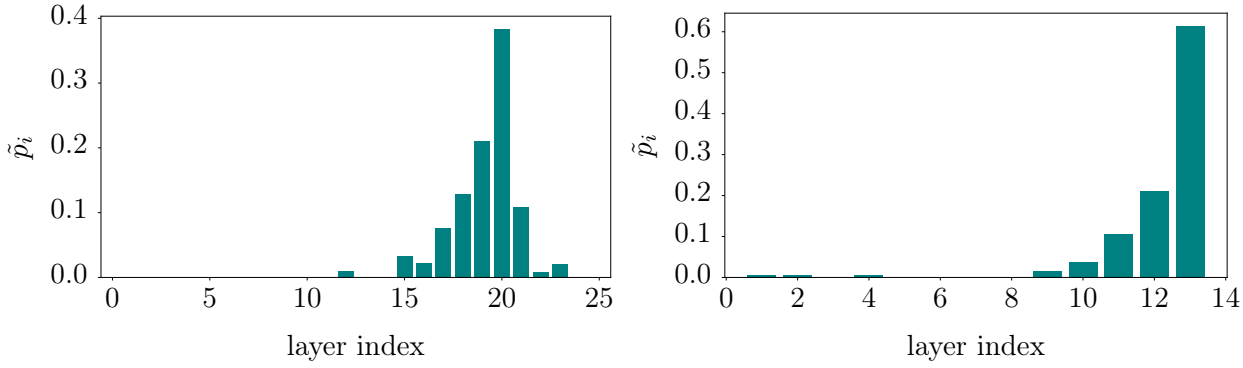
The model was selected based on the highest quality on validation. Figure 8 shows how the importance components for each layer changed during the training process for both base models. The trained distribution across layers $\tilde{p}$ is depicted in Figure 9.

As can be observed, the model selected a certain number of layers among the last ones. The low-level layers were zeroed out, indicating their minor significance for the task. Indeed, determining the sentiment of a text requires sufficiently high-level features.



(subfigure) RoBERTa-large (subfigure) BART

Figure 8: The history of changes in the components of the vector $\tilde{p}$ during the training process of the model trained on the IMDB sentiment task. On the left (8subfigure) — the base model RoBERTa-large, on the right (8subfigure) — the base model BART.



(subfigure) RoBERTa-large (subfigure) BART

Figure 9: The magnitudes of the components of the vector $\tilde{p}$ in the model trained on the IMDB sentiment task. The model with the best metrics on validation. On the left (9subfigure) — the base model RoBERTa-large, on the right (9subfigure) — the base model BART.

For RoBERTa-large, the quality on the test part of the dataset was **96.1% according to the accuracy metric**. As seen in Appendix 6.4, the best solutions achieve a quality of 96% - 96.68%. Thus, the quality achieved by the proposed algorithm is comparable to the best solutions in the field. It is worth noting that in this work, unlike many solutions in Appendix 6.4, the base model was not fine-tuned. Moreover, only **0.4%** of parameters were added compared to the size of the base model.Similarly, high quality is also achieved in the case of the base BART model. The distribution of layer influence is shown in Fig. 9subfigure. The quality achieved on the test portion of the dataset is **96.0% according to the accuracy metric**, which also places the solution among the best.

Some examples of erroneous predictions of the model based on RoBERTa-large are presented in Appendix 6.5. Upon visual analysis of many examples, it was noted that a significant portion of the errors are due to labeling errors or ambiguous examples—for

instance, a review with negative content but a positive rating, or vice versa.

For RoBERTa-large, baseline solutions were trained. For a fair comparison, the same architecture was chosen, but only one of the hidden layers is forcibly used:

$$p = \left(-\infty \quad \dots \quad \underset{i}{1} \quad \dots \quad -\infty\right)^T.$$

This allows us to understand how much better the proposed solution is compared to simple variants, where only one layer is chosen. The results are presented in Fig. 10. It is evident that there is a significant quality gap. Indeed, the proposed method uses features from various layers, effectively combining them. It is clear that each layer taken individually does not provide the same quality as the proposed approach.

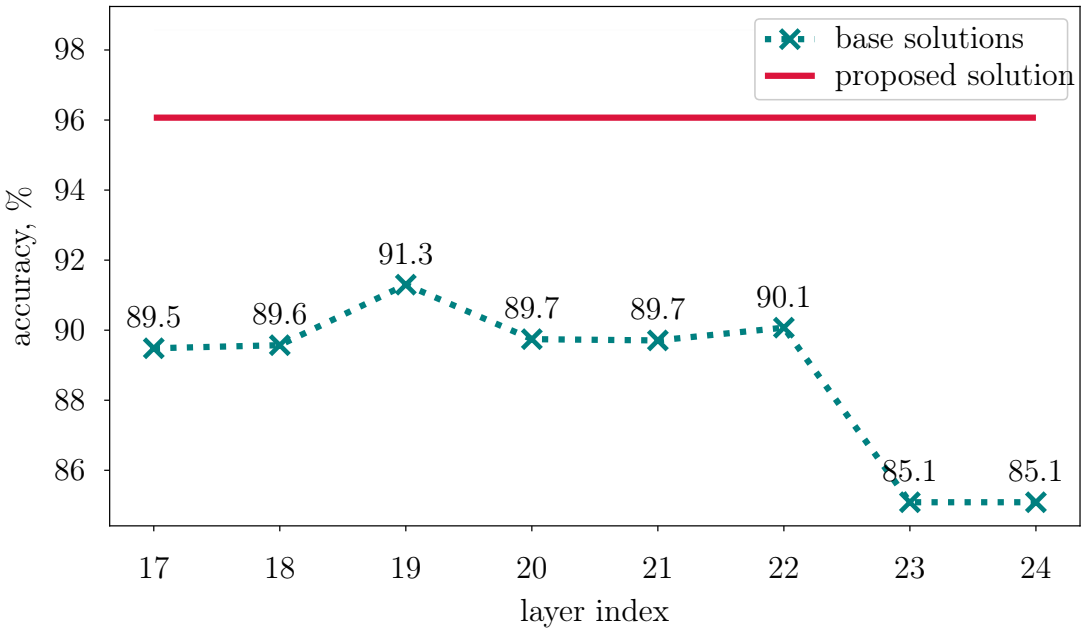


Figure 10: Comparison of the obtained solution with baseline solutions—solutions that use the states of only one hidden layer. Base model: RoBERTa-large.

Thus, on the IMDB dataset with both base models, results comparable to SOTA solutions were achieved. It is important to note that many of the solutions listed in Appendix 6.4 trained an entirely new model. In the proposed solution, the base models are frozen. Therefore, it can be concluded that the algorithm successfully copes with the task for both base models, while adding less than one percent of parameters to the size of the base model.

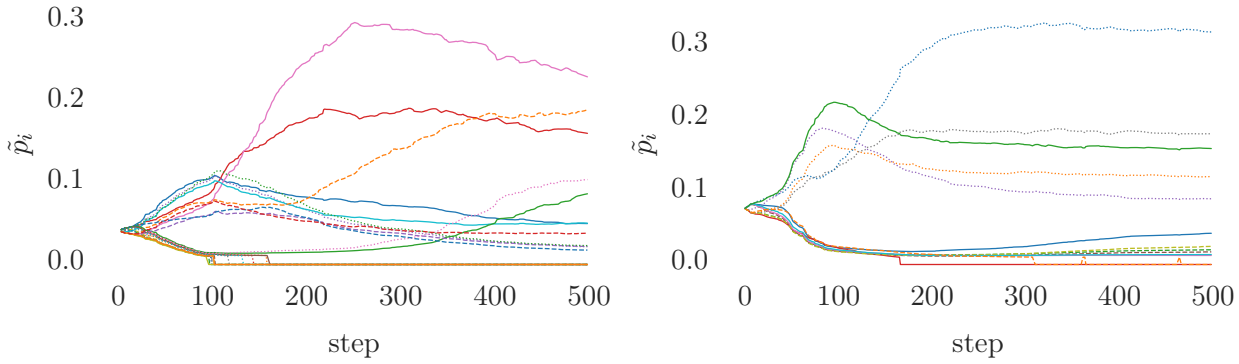
5.4.3 CoLA

Similarly, the model was trained on the CoLA dataset. On this dataset, one of the algorithm's drawbacks became apparent—under conditions of a small amount of training data, the model tends to overfit easily. Therefore, it is necessary to include additional regularizations and reduce the number of parameters. The training hyperparameters are presented in Fig. 11.

Parameter	Value (RoBERTa-large)	Value (BART)
init learning rate	10^{-4}	
weight decay	0	
α	64	
λ	0	
distribution cutoff	10^{-2}	
dropout	$3 \cdot 10^{-2}$	10^{-2}
attention heads	8	
k	64	
learning rate decay patience	25	16
precision	amp fp16	
batch size	600	900
add positional embeddings	yes	
number of parameters	1.59M	0.87M
proportion of parameters from the base model	**0.4%**	**0.2%**

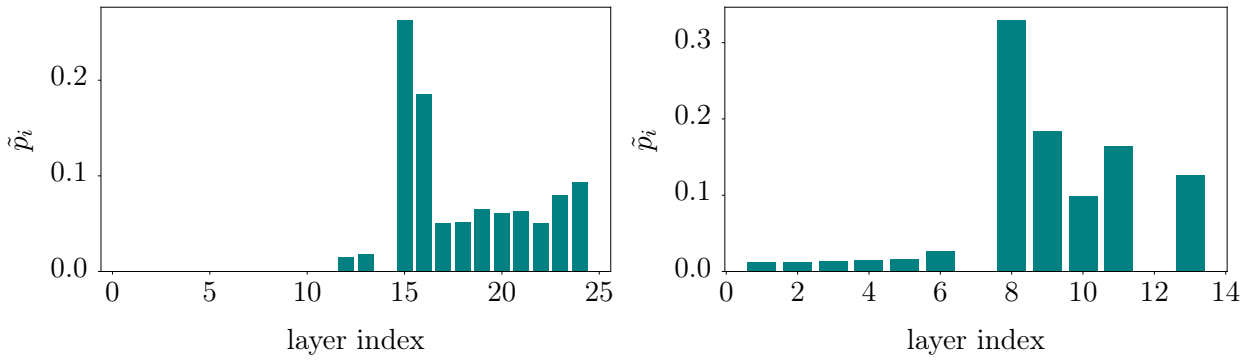
Figure 11: Training hyper-parameters on the CoLA dataset

The model was selected based on the highest quality on validation. The learned distribution of "importance" across layers is depicted in Fig. 13. Unlike the IMDB dataset, the obtained distribution extracted features from various layers.



(subfigure) RoBERTa-large (subfigure) BART

Figure 12: The history of changes in the vector components $\tilde{p}$ during the training process for the model, trained on the CoLA task. On the left (12subfigure) — the base model RoBERTa-large, on the right (12subfigure) — the base model BART.



(subfigure) RoBERTa-large (subfigure) BART

Figure 13: The magnitudes of the vector components $\tilde{p}$ in the model trained on the CoLA task. The model with the best metrics on validation. On the left (13subfigure) — the base model RoBERTa-large, on the right (13subfigure) — the base model BART.

The obtained quality is also comparable to the best solutions: **83.6% and 80.9% accuracy** for RoBERTa-large and BART respectively.

On this dataset, the proposed solution also shows excellent results. Unlike IMDB, on CoLA, a difference in quality for the two base models is observed. RoBERTa-large, trained on a more general task, achieves higher quality. Such a difference can be explained by the fact that when trained on a summarization task, the model may start to ignore factors responsible for linguistic correctness of the text, since they are not significantly important for capturing the meaning.

Another interesting result is the distribution shown in Fig. 13. It is evident that an entire set of layers has a high impact on the outcome. This can be explained by the fact that identifying errors in the text requires both high-level features and factors more specific to the word level.In Fig. 14, baseline solutions using only one layer of RoBERTa-large are presented. As can be seen, none of the selected layers individually provides quality as high as the proposed algorithm.

Comparing the solution with Appendix 6.4, it is evident that the best solutions have a quality of 82% - 88.6%. Thus, the quality obtained in this work falls within the range of the best solutions. It is important to note that among the presented solutions, there are no approaches that do not fine-tune the base model. It turns out that the proposed solution, without fine-tuning the base model, successfully competes with solutions that train the entire model.

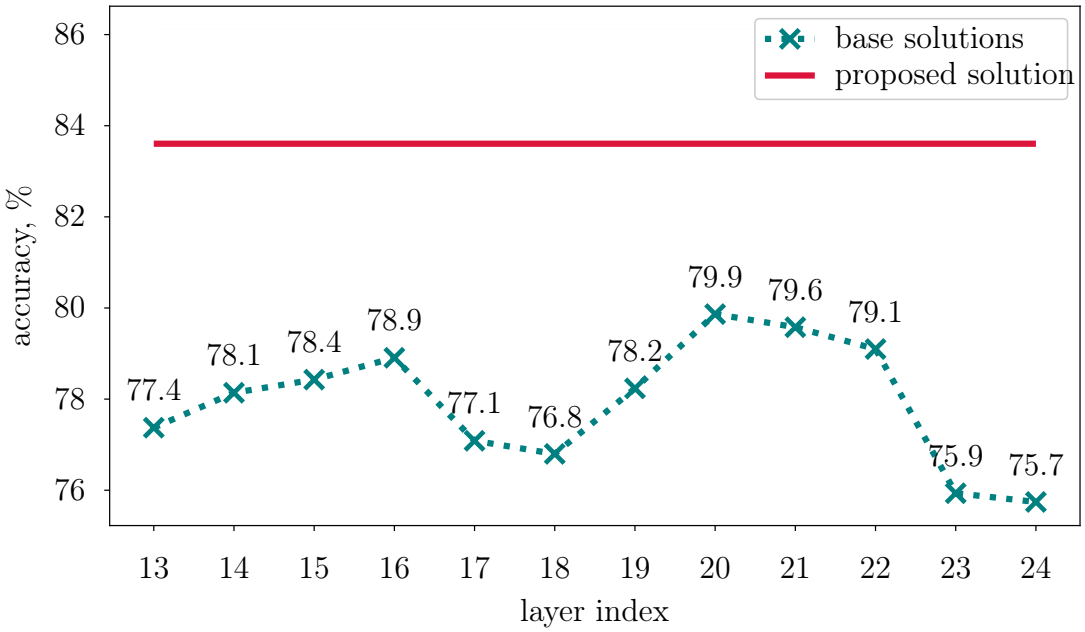


Figure 14: Comparison of the obtained solution with baseline solutions—solutions using states of only one hidden layer. Base model: RoBERTa-large.

5.4.4 CoNLL

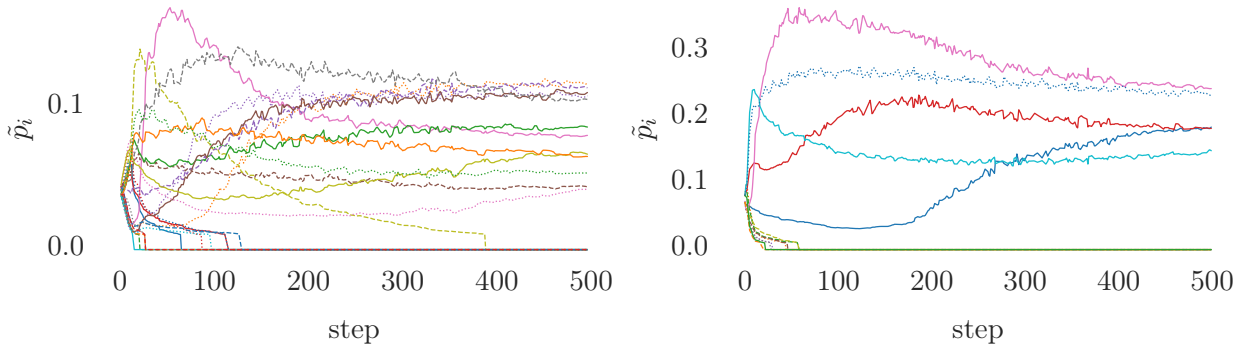
The last task discussed, CoNLL, is a named entity recognition task. In terms of neural networks, this task is equivalent to classifying each input token. Unlike the previous tasks, for this dataset, encoder layers were used as the final layers over the feature vector F.

This modification is due to the fact that, unlike classifying the entire sequence, classifying tokens requires learning additional relationships between tokens. Solutions with a single attention layer at the output, as in the previous sections, did not show the best results. The model's hyperparameters are listed in the table in Fig. 15.

The model was selected based on the highest quality on validation. The learned distribution of "importance" of layers is depicted in Fig. 17. The attached distributions show that the model uses both low-level features and higher-level ones. This makes sense in the context of named entity recognition. For prediction, both token-level features and higher-level features are important.

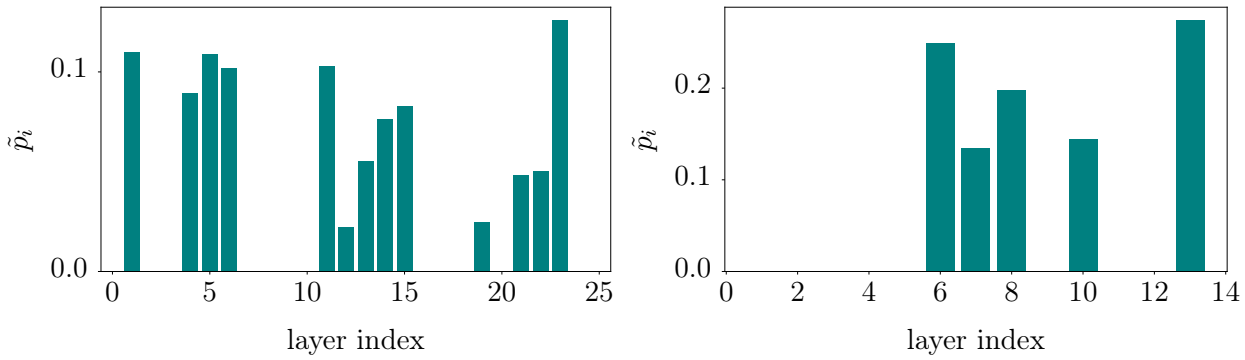
Parameter	Value (RoBERTa-large)	Value (BART)
init learning rate	10^{-4}	
weight decay	10^{-3}	
α	128	
λ	10^{-3}	
distribution cutoff	10^{-2}	
dropout	0.2	
attention heads	8	
k	128	
learning rate decay patience	16	
precision	amp fp16	
batch size	356	
encoder layers added	1	
dim feedforward	2048	
add positional embeddings	yes	
number of parameters	3.21M	1.77M
percentage of parameters from the base model	**0.9%**	**0.4%**

Figure 15: Training hyper-parameters on the CoLA dataset



(subfigure) RoBERTa-large (subfigure) BART

Figure 16: The history of changes in the vector components $\tilde{p}$ during the training process for the model, trained on the CoNLL task. On the left (16subfigure) — the base model RoBERTa-large, on the right (16subfigure) — the base model BART.



(subfigure) RoBERTa-large (subfigure) BART

Figure 17: The magnitudes of the vector components $\tilde{p}$ in the model trained on the CoNLL task. The model with the best metrics on validation. On the left (17subfigure) — the base model RoBERTa-large, on the right (17subfigure) — the base model BART.

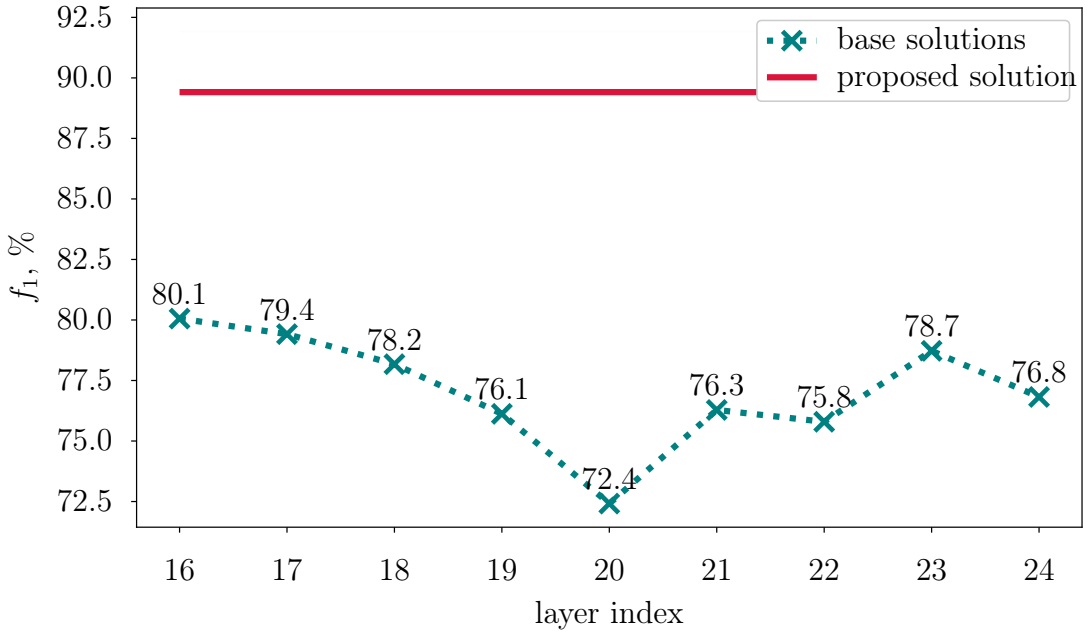


Figure 18: Comparison of the obtained solution with the baseline solutions—solutions that use the states of only one hidden layer. Base model: RoBERTa-large.

The quality obtained is quite high: **89.4% and 89.3% $\mathbf{f_1}$** for RoBERTa-large and BART, respectively. Comparing with Appendix 6.4, where the best solutions have an f_1 score of 93% - 94.6%, it can be concluded that the results obtained are still lower than the best solutions. However, as shown in Fig. 18, the baseline solutions that use only one layer of RoBERTa-large individually, achieve significantly worse quality than the proposed solution.

The table in Fig. 19 presents the NER quality for the algorithm with the base model RoBERTa-large. As can be seen, the quality significantly drops for one class—MISC. This can be explained by the fact that the model is not trained on specific named entities that fall precisely into this category.

Label	Precision	Recall	f_1	Count
LOC	90.93%	92.51%	91.71	1697
MISC	72.44%	81.62%	76.76	791
ORG	85.05%	90.43%	87.66	1766
PER	94.02%	95.36%	94.69	1640

Figure 19: Performance of the RoBERTa-large model with the proposed algorithm on CoNLL: breakdown by named entity classes.

5.4.5 General Observations

The trainable model proved to be quite robust to changes in hyperparameters. Thus, the most influential hyperparameters are: k — the dimensionality of the hidden layer space after projection, α — the "rate" of change in p.

It is important to note that with a large k, the algorithm starts to overfit quickly. Therefore, in this work, k is used relatively small. The parameter α significantly affects the rate of convergence: at low values, the contribution of less useful layers disappears slowly, which can negatively affect the quality. On the other hand, high values of α provoke the identification of important layers at the very beginning of training, which may not be optimal.

6 Conclusion

6.1 Results

This work proposes an algorithm for reusing computations of the main transformer model to solve related tasks on the same input data. The algorithm was tested on two different base models and three datasets.

In all tests, the approach shows high results, with outcomes on two datasets being comparable to the best solutions. It is important to note that in the proposed algorithm, the base model is not retrained, and yet, the proposed approach successfully competes with solutions that involve full model retraining.

Moreover, the proposed approach is efficient in application, as described in section 5.3. In particular, the algorithm can be efficiently applied in terms of additional memory, which distinguishes it from similar algorithms [21, 19].

A crucial detail of the approach is that the base transformer model remains unchanged. Thus, without negatively affecting the quality of the main task, an efficient algorithm has been developed, capable of solving related tasks with high quality under conditions of limited computational resources. All objectives of the work have been achieved.

6.2 Source Code and Reproducibility of Results

The source code is made publicly available in the repository at the link: `https://github.com/alexdremov/adalayers`. The repository contains all the training configuration files that can be used to reproduce the results presented in the work.

6.3 Questions for Further Research

In this work, two different models were used to test how much the quality of the algorithm depends on the choice of the base model. However, this question was not fully explored. For example, it is evident that the RoBERTa-large model, trained on MLM, performs better on NER than BART, trained on summarization. It is also unknown how significantly the domains of the base task and the related task can differ before significant drops in quality occur.

Besides the first question, the impact of the hyperparameter α and the contribution of entropy regularization to the parameter p have not been fully studied. During the experiments and in this work, it was noted that these parameters significantly affect the quality, but no separate experiments were conducted to determine the extent of this influence. Since the algorithm is prone to overfitting with a small amount of data, a deeper assessment of the impact of these parameters is important.

An interesting direction of work is the application of the algorithm over the features of the decoder layers, which is permissible within the proposed architecture.

References

[1] *Vaswani, Ashish.* Attention Is All You Need. — 2023.

[2] *Devlin, Jacob.* BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. — 2019.

[3] Language Models are Unsupervised Multitask Learners / Alec Radford, Jeff Wu, Rewon Child et al. — 2019.

[4] *Raffel, Colin.* Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. — 2023.

[5] *Jacob, Benoit.* Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. — 2017.

[6] *Sanh, Victor.* DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. — 2020.

[7] *He, Tianxing.* Analyzing the Forgetting Problem in the Pretrain-Finetuning of Dialogue Response Models. — 2021.

[8] *Parović, Marinela.* Cross-Lingual Transfer with Target Language-Ready Task Adapters. — 2023.

[9] *Lialin, Vladislav.* Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning. — 2023.

[10] *Houlsby, Neil.* Parameter-Efficient Transfer Learning for NLP. — 2019.

[11] *Wang, Yaqing.* AdaMix: Mixture-of-Adaptations for Parameter-efficient Model Tuning. — 2022.

[12] *Zaken, Elad Ben.* BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. — 2022.

[13] *Guo, Demi.* Parameter-Efficient Transfer Learning with Diff Pruning. — 2021.

[14] *Vucetic, Danilo.* Efficient Fine-Tuning of BERT Models on the Edge. — 2022. `http://dx.doi.org/10.1109/ISCAS48785.2022.9937567`.

[15] *Aghajanyan, Armen.* Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. — 2020.

[16] *Hu, Edward J.* LoRA: Low-Rank Adaptation of Large Language Models. — 2021.

[17] *Lester, Brian.* The Power of Scale for Parameter-Efficient Prompt Tuning. — 2021.

[18] *Xie, Shuo.* Hidden State Variability of Pretrained Language Models Can Guide Computation Reduction for Transfer Learning. — 2022.

[19] *Heinsen, Franz A.* An Algorithm for Routing Vectors in Sequences. — 2022.

[20] *Sabour, Sara.* Dynamic Routing Between Capsules. — 2017.

[21] *Evci, Utku.* Head2Toe: Utilizing Intermediate Representations for Better Transfer Learning. — 2022.

[22] RoBERTa: A Robustly Optimized BERT Pretraining Approach / Yinhan Liu, Myle Ott, Naman Goyal et al. // *CoRR*. — 2019. — Vol. abs/1907.11692. `http://arxiv.org/abs/1907.11692`.

[23] BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension / Mike Lewis, Yinhan Liu, Naman Goyal et al. // *CoRR*. — 2019. — Vol. abs/1910.13461. `http://arxiv.org/abs/1910.13461`.

[24] Learning Word Vectors for Sentiment Analysis / Andrew L. Maas, Raymond E. Daly, Peter T. Pham et al. // Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. — Portland, Oregon, USA: Association for Computational Linguistics, 2011. — June. — Pp. 142–150. `http://www.aclweb.org/anthology/P11-1015`.

[25] *Warstadt, Alex.* Neural Network Acceptability Judgments / Alex Warstadt, Amanpreet Singh, Samuel R Bowman // *arXiv preprint arXiv:1805.12471*. — 2018.

[26] *Rücker, Susanna.* CleanCoNLL: A Nearly Noise-Free Named Entity Recognition Dataset. — 2023.

[27] *Clark, Kevin.* What Does BERT Look At? An Analysis of BERT's Attention. — 2019.

[28] *Coenen, Andy.* Visualizing and Measuring the Geometry of BERT. — 2019.

[29] *Vig, Jesse.* Analyzing the Structure of Attention in a Transformer Language Model. — 2019.

[30] *Hendrycks, Dan.* Gaussian Error Linear Units (GELUs). — 2023.

[31] *Xiong, Ruibin.* On Layer Normalization in the Transformer Architecture. — 2020.

[32] *Paszke, Adam.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. — 2019.

[33] *Kingma, Diederik P.* Adam: A Method for Stochastic Optimization. — 2017.

[34] Dropout: A Simple Way to Prevent Neural Networks from Overfitting / Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky et al. // *Journal of Machine Learning Research*. — 2014. — Vol. 15, no. 56. — Pp. 1929–1958. `http://jmlr.org/papers/v15/srivastava14a.html`.

[35] SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems / Alex Wang, Yada Pruksachatkun, Nikita Nangia et al. // *arXiv preprint 1905.00537*. — 2019.

[36] Acceptability Judgements via Examining the Topology of Attention Maps / Daniil Cherniavskii, Eduard Tulchinskii, Vladislav Mikhailov et al. // Findings of the Association for Computational Linguistics: EMNLP 2022. — Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, 2022. — dec. — Pp. 88–107. `https://aclanthology.org/2022.findings-emnlp.7`.

[37] *Proskurina, Irina*. Can BERT eat RuCoLA? Topological Data Analysis to Explain / Irina Proskurina, Ekaterina Artemova, Irina Piontkovskaya // Proceedings of the 9th Workshop on Slavic Natural Language Processing 2023 (SlavicNLP 2023). — Association for Computational Linguistics, 2023. `http://dx.doi.org/10.18653/v1/2023.bsnlp-1.15`.

[38] *Sileo, Damien*. tasksource: A Dataset Harmonization Framework for Streamlined NLP Multi-Task Learning and Evaluation. — 2023.

[39] *Wang, Sinong*. Entailment as Few-Shot Learner. — 2021.

[40] *Charpentier, Lucas Georges Gabriel*. Not all layers are equally as important: Every Layer Counts BERT. — 2023.

[41] *Lee-Thorp, James*. FNet: Mixing Tokens with Fourier Transforms. — 2022.

[42] *Wang, Xinyu*. Automated Concatenation of Embeddings for Structured Prediction. — 2021.

[43] *Yamada, Ikuya*. LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention. — 2020.

[44] *Zhou, Wenxuan*. Learning from Noisy Labels for Entity-Centric Information Extraction / Wenxuan Zhou, Muhao Chen // Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. — Association for Computational Linguistics, 2021. `http://dx.doi.org/10.18653/v1/2021.emnlp-main.437`.

[45] *Liu, Tianyu*. Autoregressive Structured Prediction with Language Models. — 2022.

[46] *Schweter, Stefan*. FLERT: Document-Level Features for Named Entity Recognition. — 2021.

[47] *Ye, Deming.* Packed Levitated Marker for Entity and Relation Extraction. — 2022.

[48] *Wang, Xinyu.* Improving Named Entity Recognition by External Context Retrieving and Cooperative Learning. — 2022.

[49] Named Entity Recognition Architecture Combining Contextual and Global Features / Tran Thi Hong Hanh, Antoine Doucet, Nicolas Sidere et al. // Towards Open and Trustworthy Digital Societies. — Springer International Publishing, 2021. — P. 264–276. http://dx.doi.org/10.1007/978-3-030-91669-5_21.

[50] *Csanády, Bálint.* LlamBERT: Large-scale low-cost data annotation in NLP. — 2024.

[51] *Yang, Zhilin.* XLNet: Generalized Autoregressive Pretraining for Language Understanding. — 2020.

[52] *Haonan, Lu.* Graph Star Net for Generalized Multi-Task Learning. — 2019.

Appendix

6.4 Quality of the Best Models on Selected Datasets

- **IMDB Sentiment** [24], accuracy.

  1. **96.68%** — RoBERTa-large with LlamBERT [50]
  2. **96.54%** — RoBERTa-large [50]
  3. **96.21%** — XLNet [51]
  4. **96.2%** — Heinsen Routing + RoBERTa Large [19]
  5. **96.1%** — RoBERTa-large 355M + Entailment as Few-shot Learner [39]
  6. **96.0%** — GraphStar [52]

- **GLUE CoLA** [25], accuracy.

  1. **88.6%** — En-BERT + TDA + PCA [36]
  2. **88.2%** — BERT+TDA [37]
  3. **87.3%** — RoBERTa+TDA [37]
  4. **87.15%** — deberta-v3-base+tasksource [38]
  5. **86.4%** — RoBERTa-large 355M + Entailment as Few-shot Learner [39]
  6. **82.7%** — LTG-BERT-base 98M [40]
  7. **82.6%** — ELC-BERT-base 98M [40]
  8. **82.1%** — En-BERT + TDA [36]
  9. **78%** — FNet-Large [41]

- **CoNLL** [26] NER, f_1.

  1. **94.6%** — ACE + document-context [42]
  2. **94.3%** — LUKE 483M [43]
  3. **94.22%** — Co-regularized LUKE [44]
  4. **94.1%** — ASP+T5-3B [45]
  5. **94.09%** — FLERT XLM-R [46]
  6. **94.0%** — PL-Marker [47]
  7. **93.85%** — CL-KL [48]
  8. **93.82%** — XLNet-GCN [49]
  9. **93.8%** — ASP+flan-T5-large [45]

6.5 Examples of Model Errors on the IMDB Dataset

Below are some examples of items from the IMDB dataset, the true label, and the prediction.

> First off let me say, If you haven't enjoyed a Van Damme movie since Bloodsport, you probably will not like this movie. Most of these movies may not have the best plots or best actors **but I enjoy these kinds of movies** for what they are [...] **Good fun stuff!**

True label: negative
Prediction: positive

> Just watched on UbuWeb this early experimental short film directed by William Vance and Orson Welles. Yes, you read that right, Orson Welles! [...] I won't reveal any more except to say **how interesting the silent images were** as they jump-cut constantly. That's not to say this was **any good but it was fascinating** to watch even with the guitar score (by Larry Morotta) [...]

True label: negative
Prediction: positive

> The theme is controversial and the depiction of the hypocritical and sexually starved India is excellent.**Nothing more to this film**. There is a **lack of good dialogues** (why was the movie in English??). There was **lack of continuity** and **lack of passion/emotion** in the acting.

True label: positive
Prediction: negative

> If you want to see a brilliant performance of Mikado, played to perfection with expert timing and panache, **don't watch this version.** [...] It's a lot of fun and a good intro to Gilbert and Sullivan, but after this, **rush out and rent the Canadian Stratford version**. You'll see the difference between **good and great**. Nobody does G&S better than Brian McDonald and the Stratford group.

True Label: positive
Prediction: negative