Answer Forecasting: ML-Driven Predictions in Lernnavi

Luca Zunino, Elena Grazia Gado, Tommaso Martorella
Machine Learning for Behavioral Data (CS421) - Milestone 7 - EPFL, Switzerland

***Abstract*—Answer forecasting is a task that aims to predict what a student would answer to a given question based on their learning profile and context. In this work, we present a system that leverages BERT, a state-of-the-art NLP model, to perform answer forecasting for multiple-choice questions. We describe the architecture and components of our system and evaluate its performance on a dataset of real-world questions and answers from an online learning platform: Lernnavi.**

I. Introduction

In recent years, educational applications have become increasingly prevalent as a means of supplementing traditional classroom learning. Machine learning algorithms have the potential to revolutionize this education sector by providing insights into student learning behavior and enabling the creation of tailored learning experiences. In this regard, developing a machine learning model capable of predicting a user's response could enable the provision of adaptive learning experiences that cater to the needs and abilities of individual students. This has indeed been the interest of various studies, such as [1]. This project aims to explore the development of such a machine learning model for Lernnavi, an educational platform created to support middle and high school students studying Math and German. This report provides a comprehensive overview of the process involved in building our machine learning model. It begins with an in-depth exploration of the data (Section II), followed by a thorough explanation of the methodology employed, outlined in section III. Finally, the report concludes by discussing the future possibilities and potential advancements in our approach.

Our project aims to understand if it is possible to harness the power of a language model, specifically GermanBERT, to accurately predict the responses of Lernnavi users.

This research questions implied others. Specifically:

1) Understanding if the available data allows extracting enough relevant features to predict the user's knowledge and learning;
2) Comprehending if we can represent the user's knowledge in a compact format (the embeddings);
3) Understanding if these are helpful to improve the predictions of the model.

II. Exploratory Data Analysis

Before starting to work on our model, a comprehensive exploratory data analysis (EDA) was conducted, in order to provide crucial insights into the initial understanding and characterization of the data used in our machine learning project. Firstly, an individual EDA was performed by each group member, where we delved into the dataset, examined its key properties, visualized the distributions and looked for any patterns or anomalies. We especially noticed that the number of users that remain active across time is a distribution highly skewed towards the left. That means that most users stop using the platform after just one or a few weeks.

Subsequently, we conducted a detailed exploration of the data with the specific aim of determining its relevance and availability for our project. This exploration involved assessing the usefulness of different data variables and understanding the quantity and extent of the data accessible to us. The subset of the Learnavi dataset we are interested in includes the following tables:

1) *events.csv*: user-generated events on the platform;
2) *transactions.csv*: detailed information regarding the questions, referred to as "documents" in Lernnavi, that were answered. Each transaction is associated with a transaction token, which can be linked to the events table.
3) *documents.csv*: the documents represent the actual questions, problems, and textual pages provided to students. The information on the topics can be of great use in modelling the student's knowledge.
4) *topics-translated.csv*: the taxonomy of categories shown in the Deutsch and Math dashboards.

The subsequent EDA consisted of several steps:

1) Data cleaning and Inspection: We initiated the EDA process by carefully examining the selected dataframes and filtering out any information that was deemed irrelevant to our project. Additionally, we computed examples of the data for each question type and stored them in separate files. This allowed us to gain a comprehensive understanding of the available data, allowing us to identify which question types could be suitable for an NLP-based approach. Once having picked the question types, we analyzed the amount of available data. The results can be seen in Figure II-1. For the remainder of this project we chose to focus on Multiple Choice Questions (MCQs).

2) Data merging and preprocessing: By cleaning and merging the different dataframes we built the question-and-answer pairs dataset that we later used both in the creation

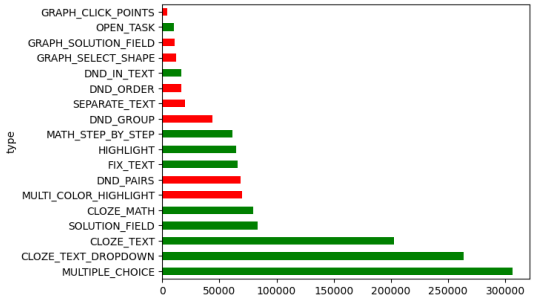

Figure 1. Number of answers per question type. Green: question types deemed useful, Red: question types deemed not suitable.

of the embeddings and in answer forecasting.

Since the Lernnavi dataset contained multiple versions of the same question, we first matched the student's events with the documents based on the time they did the exercise. We then extracted the options available, the correct answers, and the answers given by the student. Since MCQs in Lernnavi might have multiple selections, we decided to treat each question option as a binary choice (i.e. select or not select) independently of the other choices available.

III. Methods

A. Feature Extraction & Engineering

To accomplish our ambitious goal of building a machine learning model that predicts user responses in the Lernnavi application, we undertook several preprocessing and feature extraction steps. Our initial focus was to gain a comprehensive understanding of the available data by merging the provided datasets and applying necessary data cleaning procedures. Subsequently, we performed feature engineering, a vital phase in our project. This involved manipulating the datasets related to Lernnavi users to extract meaningful features that reflect the students' comprehension level and contribute to predicting their responses.

1) Initial features: The features that were initially computed are the following:

- the mastery level of the user for each topic;
- the mean mastery level of the user;
- the the sum of the scores of the student on the questions answered, normalized by the number of questions answered;
- the sum of the scores of the student on the questions answered weighted on the difficulty, normalized by the number of questions answered;
- the mastery level of the user for each topic.

It must be noted that the difficulty of each question and the associated scores were also extracted from the Lernnavi datasets. A limitation of this initial set of features is that the mastery scores were computed based on all the available answers given by the student, which means they may not be up-to-date with the user's current level of knowledge and therefore will not correspond to the mastery level of the user at the point in time at which they answered the question. This "data leaking issue" was fixed in the second version of computed features.

In this case, the train/val/test split is performed simply by dividing the students in the respective groups.

2) Intermediate version: As illustrated in the "feature engineering 2" notebook, we replaced the absolute mastery score we had in III-A1 with a temporally-synced version to reflect the student's knowledge at the time they answered the question. For this new feature, we first calculated the topic rolling median for each user and then matched the results with the questions-and-answers dataset from II-2. This allows the scores to better reflect the present level of knowledge of each student. The other features are calculated in the same way as in III-A1.

In this case, the train/val/test split is performed taking into account the temporal information. The validation and test set only contain datapoints that are temporally after the ones in the training set.

3) Final version: The third iteration of our feature engineering pipeline directly uses the student's questions and answers to estimate his knowledge instead of the previous proxy (i.e. the mastery scores). We directly use the question-answers pairs along with a more sophisticated LSTM-based autoencoder to create the student embedding in the hope of extracting more meaningful and task-relevant information from the data available.

In this case, taking temporality into account when performing the train/val/test split is even more important. The validation and test set only contain datapoints that are temporally after the ones in the training set.

B. The student embedding

The second phase of the pipeline involves generating concise student representations known as embeddings. Through this step, our goal is to encapsulate the student's characteristics in a condensed and computationally efficient form that ideally reflects their present level of knowledge.

1) Initial approach: Our initial approach consisted in using an autoencoder to generate the embeddings, employing the features described in section III-A1. The autoencoder architecture we employ consists of a symmetric multilayer perceptron (MLP) with two hidden layers in both the encoder and decoder. The hidden layers have sizes of 5000 and 1000 for the encoder, and 1000 and 5000 for the decoder. The input layer and output layer both have a size of 11907, as the autoencoder aims to reconstruct the input data. The bottleneck layer, known as the latent layer or embedding layer, has a size of 512.

While the mean mastery level and the scores were features easily usable to train the network, the topics masteries required an intermediate step. To incorporate the topic information into the input features, we

first represented the topic with a high-dimensional embedding using a sentence transformer based on ``paraphrase-multilingual-MiniLM-L12-v2'', such that similar topics have similar vectors. Each topic embedding has a size of 384. To combine the 31 topic embeddings with the mastery level of the student for each topic, we used a simple method: we first normalized the topic embedding vector to have a unit norm ($\|embedding\|_2 = 1$), and then we scaled it by the student's mastery level for that topic. This way, we obtained a scaled topic embedding vector that reflects both the semantic meaning of the topic and the student's proficiency in it. This way, the final input features for the autoencoder have a size of 11907, which is composed of 3 base features (mean mastery level, score, weighted score) and 31 scaled topic embeddings (384 x 31).

The autoencoder was trained using mean squared error (MSE) as the loss function, and Adam as the optimizer. The details of the autoencoder training can be found in the notebook "autoencoder.ipynb". We reached a validation loss of $4.7e^{-6}$. However, a closer analysis reveals that our embeddings are not ideal, as there is a notable difference in the norm2 of the input and output vectors across the training, validation, and test data (average $\|input\|_2$: 1.67, average $\|input - output\|_2$: 0.15).

2) Intermediate approach: In order to improve our embeddings, we then operated a few changes to the approach presented in section III-B1. Indeed, we used the same autoencoder presented in section III-B1, but instead of using the mastery scores computed in the initial set of features, we employed the rolling median of the mastery scores for each topic presented in section III-A2. This means that the data available in this case is an order of magnitude bigger (because we now have an embedding for each question instead of an embedding for each student). The autoencoder performance significantly dropped to a validation error of $6.139e^{-1}$. We noted no significant difference with the results presented in IV-A3 so we won't include them in this report for the sake of conciseness.

3) Final approach: In the final version, we made another adjustment to our approach. Specifically, we opted to utilize a stacked LSTM-based autoencoder. LSTM, short for Long Short-Term Memory, is a type of recurrent neural network (RNN) architecture. Instead of relying on the features computed in section III-A1 and the rolling median mastery scores discussed in section III-A2, we chose to generate student embeddings based on their previous answers. This is because nothing can aptly convey the depth of students' learning as effectively as their prior responses. Indeed, as Learnavi is a learning application, throughout it the students should be able to improve. Moreover, the answers they give are probably the best tool to identify their profile. As LSTM networks are indeed well-suited for modelling and predicting sequential data, switching to this type of autoencoders seemed to be the best choice to generate embeddings that could capture the temporal aspect of the data. LSTM networks consist indeed of an encoder, that takes a sequence in input and has a vector (the student embedding in our specific case) as an output, and of a decoder, that from the given vector out returns the starting sequence. The library sequitur[1] was used to create and train the autoencoder and is ideal for working with sequential data ranging from single and multivariate time series. The input layer and output layer both have a size of 768, the three hidden layers have a size of 512, and the bottleneck layer has a size of 512 as well. The input sequence given to the LSTM autoencoder is the concatenation of the 10 question-answer pairs previous to the point in time we are interested in. Each of those question-answer pairs is obtained by the concatenation of the question embedding and the answer embedding, each of size 384.

C. Answer forecasting

1) From GermanBERT to MCQBert - the pipeline: As previously stated, the objective of our project is to develop a model which can predict the students' answers to Multiple Choice Questions (MCQs) given the text of the question and the embeddings of the students. In order to obtain such a model, we decided to leverage a pre-trained model, which we then finetuned for answer prediction. After reviewing the literature, we selected the GermanBERT model, part of the BERT family, as the starting point. This model (``dbmdz/bert-base-german-uncased'') has been pre-trained on an extensive text corpus, learning effective latent representations of sentences in context. This extensive pretraining allows for finetuning the model on specific tasks with less computational effort.

In order to transform the pre-trained GermanBERT model into our final model able to predict the students' answers to MCQs, we decided to follow the approach below:

1) Finetuning GermanBERT on a language modelling task considering the Lernnavi dataset (and, in particular, the questions and possible answers of MCQs). For clarity, we will call the finetuned model "LernnaviBERT";
2) Finetuning LernnaviBERT to predict the correct answers to MCQs. In this case, the downstream task is being able to identify the correct answers among the possible ones for each MCQ in the dataset;
3) Further finetuning the model on the final downstream task, that is, predicting the answers of a given student to MCQs, taking into account the embedding of the student. We will call the model finetuned on the final downstream task "MCQBert".

In the following sub-subsections, we will present the procedures used in each of these steps.

[1]https://github.com/shobrook/sequitur

2) Step 1 - From GermanBERT to LernnaviBERT: The language models of the BERT family are usually extensively pretrained on a vast and varied corpus and can therefore be effectively finetuned on a variety of downstream tasks without the need for further finetuning them on language modelling tasks. This approach, which is possible thanks to transfer learning, is likely to produce good results as long as the corpus used for pretraining the BERT model is not too different from the corpus used for finetuning on the downstream task (in our case, MCQ prediction). This is due to the fact that if the dataset used for finetuning on the downstream task contains domain-specific words which were not observed while training, the model would treat these words as rare tokens, causing a rapid decrease in the resulting performance. As stated previously, we decided to consider GermanBERT as our BERT model (``dbmdz/bert-base-german-uncased'' on HuggingFace Hub), which according to the model card [2] has been trained on a dataset of 16GB (2,350,234,427 tokens) that includes various sources: a Wikipedia dump, EU Bookshop corpus, Open Subtitles, CommonCrawl, ParaCrawl and News Crawl. Although the dataset used for training seems extensive, and our dataset based on Lernnavi MCQs does not seem too domain-specific, we decided to finetune GermanBERT on our dataset (domain adaptation) to verify whether this leads to a performance boost. An advantage of this approach is that the obtained model, LernnaviBERT, could be used on various downstream tasks on the Lernnavi dataset, and the finetuned model by itself could be valuable to the scientific community in the future.

In order to finetune GermanBERT on a language modelling task, we followed HuggingFace's tutorial [3]; the procedure followed is somewhat similar to the one used to train the original GermanBERT model. The main steps that we perform to go from GermanBERT to LernnaviBERT are the following:

1) We preprocess the Lernnavi dataset containing MCQs (text of the questions and possible answers) to be suitable for our task. In particular, since we are interested in letting the model adapt to the language used in the MCQs, we are not directly interested in preserving the entity of each question. For this reason, we start by concatenating all the samples and splitting the corpus into chunks of equal size (which should be smaller than the maximum context size of the model);
2) In masked language modelling tasks such as the one we are considering, the objective is to predict the randomly masked tokens and compare the prediction with the ground truth. For this reason, we mask certain tokens by inserting `[MASK]` tokens at random positions in the input samples using a data collator (`DataCollatorForLanguageModeling`). A parameter which needs to be chosen is the fraction of tokens to be masked: we have decided to mask 15% of the tokens, a common choice in the literature and the value used to train BERT;
3) The rest of the training loop follows the standard procedure. In particular, we used an AdamW optimizer with linear learning rate decay and the model's built-in loss.

The procedure followed is schematized in Figure 8 in the Appendix.

Finally, we saved the finetuned model (LernnaviBERT) on the HuggingFace Hub to have it ready for the next step. It should be noted that this model (although, for the moment, it is kept private since it has been trained on confidential data) could be used by the ML community as a starting point for various downstream tasks involving the Lernnavi dataset.

3) Step 2 - Finetuning LernnaviBERT for correct answers prediction: In the second step of the pipeline, we aim to finetune LernnaviBERT to predict the correct answer to MCQs. In this case, the downstream task that we consider is being able to identify the correct answers among the possible ones for each MCQ in the dataset.

As previously stated, the project's final goal is to create a model able to predict a student's answer to an MCQ given the question, the possible answers and some relevant information concerning the student (in the form of an embedding). To reach this goal, we wanted the model to know the correct answers to all the MCQs present in the dataset before attempting to predict the students' answers. Indeed, we hypothesized that the model could predict the student's answers more accurately by exploiting the knowledge of the correct answer to an MCQ. For example, if the model detects (by analyzing the embedding) that a student has very high mastery in a particular topic, it may "decide" to predict that the student will answer correctly to a question on that topic and would therefore need to know which is the correct answer to make this prediction.

In this step, we consider two models based on LernnaviBERT (the ones which obtained the best results in the study conducted for the previous milestone), on which a binary classification head has been added. The classification head we considered is binary since we decided to frame the MCQs prediction problem as a binary classification problem. More in detail, we split each MCQ datapoint into multiple datapoints, each containing the question and one of the possible answers. The objective of the model is then to predict 1 if the possible answer is correct (or, in the next step, if the student selected that possible answer) and to predict 0 if the possible answer is not correct (or, in the next step, if the student did not select that possible answer). A scheme representing how the task has been framed as a binary classification task can be found in Figure 5 in the Appendix.

Two experiments have been conducted as part of the finetuning of the models on the correct answer prediction

task:

1) The two models have been finetuned on a training set, and their generalization capabilities have been assessed on a test set. This experiment has been performed to verify if the ability of the model to answer MCQs correctly generalizes to unseen questions. Regarding the dataset split into a training dataset and a validation/test dataset, it should be taken into account that each MCQ is present more than once in the dataset since there is a datapoint for each student's answer. Since we would like to test the models on new MCQs to verify whether they have learnt how to answer MCQs correctly, we performed the split by ensuring that each individual MCQ was present only in the training set or in the test set. In other words, all the instances of a given MCQ are located in one of the two datasets created.
2) The two models have been trained on the whole MCQ dataset, and it has been verified whether they remember the correct answers by evaluating them on the same dataset. This experiment has been performed since we would like our models to know the correct answers to all the MCQs. In this way, we could be sure that a failure to predict a student's answer to an MCQ does not stem from the model not knowing which answer is correct.

It should be noted that, in the third step of our pipeline (finetuning the models to predict the students' answers to MCQs), we will consider the models trained on the complete dataset (second experiment). This approach does not risk introducing data leakage, as our final goal will be to predict the answer of students to MCQs, not the correct answer. Moreover, we would like the model to learn the correct answers to all MCQs and exploit this information while predicting the answers given by the students.

For what concerns the models, we decided to consider "MCQBert1" (see Figure 6 in the Appendix) and "MCQBert3" (see Figure 7 in the Appendix) since these were the models which obtained the best results in the previous milestone. These models are based on LernnaviBERT (obtained in the first step of our pipeline), to which the students embedding are passed and to which a classifier head (two linear layers with a ReLU activation in between) has been added in order to predict whether each of the possible answers is correct or not. The main difference between the two models is that in "MCQBert1", the embeddings of the students are concatenated just before the classification layer, while in "MCQBert3", the student embeddings are summed at the input of the model. Furthermore, we also finetune a baseline model (that will be useful in the next step), which does not contain the embeddings. The scheme contained in Figure 9 in the Appendix shows the complete architecture of the models we considered (the embeddings are not depicted in this case).

Another essential aspect to consider is that, in this step, we are not interested in the embeddings of the students, as we would like to predict the correct answer to MCQs regardless of the student with which the datapoint is associated. Therefore, to avoid introducing noise which could confuse the models, we decided to pass all zero embeddings to them. Nevertheless, we still define models in which the embeddings can be inserted, as in the next step of the pipeline, we will be interested in considering the same models and adding the student embeddings.

4) Step 3 - Finetuning the model for student answers prediction: In the third and final step of the pipeline, we finetune our models to predict the answers given by students to MCQs, given the text of the MCQs and some information about the students in the form of embeddings. Therefore, the downstream task is different with respect to the previous step: we are not interested in identifying the correct answers among the possible ones but in the answers given by the students. Although the task is different, the procedure used to finetune the model is very similar to the one described in the second step. We will consider the two models finetuned in the previous step, "MCQBert1" and "MCQBert3", but this time we will also pass the embeddings of the various students both during training and during evaluation (previously, all the embeddings have been zeroed). Furthermore, the problem is again framed as a binary classification problem.

An important aspect to consider is how we split the dataset into a training dataset and an evaluation/test dataset. As explained in detail in the previous milestone, we considered (and coded) two main methods to perform the dataset split:

- The first method splits the dataset into a training and a validation/test datasets using the standard scikit-learn function ``train_test_split''. Since each individual MCQ is present multiple times in the dataset (once for each time a user attempted to answer that given MCQ), if we use this approach, the same MCQ can appear both in the training dataset and in the test dataset. In other words, the model could have already seen that particular MCQ but not for the same student. This method is graphically presented in Figure 3 in the Appendix (Figure 2 represents the original dataset).
- The second method splits the dataset by ensuring that each individual MCQ is present only in the training dataset or in the test/validation dataset. In other words, all the instances of a given MCQ are inserted in only one of the two datasets. This approach poses an additional challenge for the model since it should generalize to an MCQ which has never seen before. This method is graphically presented in Figure 4 in the Appendix (Figure 2 represents the original dataset).

We decided to consider the first method to prepare the datasets used to finetune the models. This choice is motivated by the fact that we want the model to leverage the

prior history related to the specific questions it is attempting to evaluate and to rely on the answers provided by students with similar profiles to the one for whom it is predicting the answer.

IV. Experimental Evaluation

A. Answer forecasting

1) Step 1 - From GermanBERT to LernnaviBERT: In order to verify the performance of the model before and after the finetuning on a language modelling task, we can perform both a quantitative and a qualitative evaluation:

- The quantitative evaluation is based on the computation of perplexity, which (roughly) indicates how much the model is "surprised" or "perplexed" by text examples, which it did not see during training. In particular, we will compute the perplexity as the exponential of the cross-entropy loss on the test set. It should be noted that this approach is accurate as long as most of the sentences are grammatically correct; otherwise, a very good model would be highly surprised by encountering errors. Since some of the MCQs' possible choices contain grammar errors, this approach is probably not entirely accurate, but it is a helpful indication. Before finetuning, the perplexity obtained is 1.20; after finetuning on the Lernnavi corpus, the perplexity is 1.01. This decrease in perplexity indicates that the finetuning process is useful in adjusting the model to the language corpus we consider for the downstream tasks.
- The qualitative evaluation is based on verifying the most probable words suggested by the models on a language modelling task. More in detail, we use the Hosted inference API provided by the HuggingFace Hub, and we verify how the models are completing sentences in which a `[MASK]` token is inserted. These experiments can be easily reproduced by using the same Hosted inference API available on HuggingFace's model cards ([2] for GermanBERT and [4] for LernnaviBERT).

Example 1: "Dies ist eine gute [MASK]" ("This is a good [MASK]")

GermanBERT → . - ! - sache - nachricht - idee (. - ! - thing - news - idea)
LernnaviBERT → nachricht - . - sache - wahl - ! (news - . - thing - choice)

Example 2: "Wählen Sie den richtigen [MASK]" ("Choose the right [MASK]")

GermanBERT → ort - weg - preis - browser - job (place - path - price - browser - job)
LernnaviBERT → ausdruck - weg - namen - ton - . (expression - path - name - tone - .)

Example 3: "Paris ist die [MASK] Frankreichs" ("Paris is the [MASK] of France")

GermanBERT → hauptstadt - zukunft - flagge - stadt - mitte (capital - future - flag - city - centre)
LernnaviBERT → hauptstadt - stadt - heimat - mutter - heimatstadt (capital - city - home - mother - hometown)

Example 4: "Das Ziel des Lebens ist [MASK]" ("The goal of life is [MASK]")

GermanBERT → : - " - , - " - . (: - " - , - " - .)
LernnaviBERT → : - , - das - immer - ? (: - , - the - always - ?)

From the four examples considered, we can notice that most of the predictions are similar for the original model (GermanBERT) and the finetuned one (LernnaviBERT), indicating that the finetuning process did not cause a decrease in the language modelling capabilities of the model. However, some of the predictions are more related to learning and education after the finetuning; this can be observed in particular in the second example, where the words "expression", "name", and "tone" seems more relevant to education and question answering than "place", "price" or "browser". These results are encouraging because they suggest that the model finetuning on in-domain data was successful: the language modelling capabilities of the model are maintained, but the predictions tend to align better to the language and the expressions characterizing the dataset we will consider for the next steps.

2) Step 2 - Finetuning LernnaviBERT for correct answers prediction: As stated when presenting our pipeline, we performed two different experiments (associated with two different training procedures) to verify which are the capabilities of the models on the downstream task of predicting the correct answers of MCQs. We will therefore present the results of the two experiments separately.

- **Experiment 1**: finetuning the models on a training set and testing them on a test set to verify the model's ability to answer unseen MCQs correctly. As part of this experiment, we considered three different models: "Baseline" (in which the student embeddings are not present), "MCQBert1" (in which the student embeddings are concatenated just before the classification layer), and "MCQBert3" (in which the student embeddings are summed at the input of the model). However, it should be noted that all the embeddings have been set to zero for the models requiring the embeddings since, in this case, we do not want the answer predictions to be affected by the students. After finetuning for one epoch, we computed various metrics on a test/validation set to determine the different models' performances. The metrics which we considered are the following (the same metrics will also be used for the second experiment and the third step):
  - Accuracy score, which measures the proportion of correct predictions out of the total number of predictions.
  - F1 score, which is the harmonic mean of precision and recall, and allows balancing both false positives and false negatives.
  - Matthews Correlation Coefficient (MCC), which is a measure of the quality of binary classification. This metric is effective even if the classes are strongly unbalanced, and it is a correlation coefficient value between -1 and +1 (+1: perfect prediction, 0: average random prediction, -1: inverse prediction).

  The results obtained are the following:
  - Baseline → Accuracy score: 0.630, F1 score for class 0: 0.677, F1 score for class 1: 0.565, Average F1 score: 0.621, MCC: 0.243.
  - MCQBert1 → Accuracy score: 0.605, F1 score for class 0: 0.660, F1 score for class 1: 0.529, Average F1 score: 0.5945, MCC: 0.190.
  - MCQBert3 → Accuracy score: 0.740, F1 score for class 0: 0.796, F1 score for class 1: 0.642, Average F1 score: 0.719, MCC: 0.472.

  All three models demonstrate good performance, showing that they can identify, up to a certain point, the correct answers to MCQs. It should be noted that this task is quite challenging for the models, as determining the correct answer to a multiple-choice question is not a trivial task, especially for models based on the BERT architecture. A result which is perhaps surprising is that MCQBert3 performed significantly better than the other two models: the average F1 score is 0.719 against the 0.621 of the baseline and the 0.5945 of MCQBert1. While MCQBert1 and MCQBert3 are quite different from an architectural perspective even if zero embeddings are considered (mainly since MCQBert1 has a classifier in which the input dimension is larger than the one used with MCQBert3, since it also receives the concatenated embeddings), it is harder to find differences between the Baseline model and MCQBert3 which could justify such difference in performance.
- **Experiment 2**: finetuning the models on the whole MCQ dataset and testing them on the same dataset to verify whether they remember the correct answers. For this experiment, we considered the same three models ("Baseline", "MCQBert1", "MCQBert3"), and we again set the embeddings to zero. The results obtained are the following:
  - Baseline → Accuracy score: 0.992, F1 score for class 0: 0.993, F1 score for class 1: 0.989, Average F1 score: 0.991, MCC: 0.983.
  - MCQBert1 → Accuracy score: 0.983, F1 score for class 0: 0.986, F1 score for class 1: 0.978, Average F1 score: 0.982, MCC: 0.964.
  - MCQBert3 → Accuracy score: 0.835, F1 score for class 0: 0.881, F1 score for class 1: 0.731, Average F1 score: 0.806, MCC: 0.673.

  From the results, it is possible to notice that the models tend to recall the correct answers very well, as both the Baseline model and MCQBert1 achieve almost perfect performance on this task. This means that after the finetuning process, the models have learnt the correct answers to the vast majority of the MCQs present in the dataset, and we can therefore exploit this knowledge in the next step. Once again, the performance obtained by MCQBert3 is puzzling, this time negatively. Indeed, although the performance remains more than reasonable in this case, this model seems to recall the correct answer much worse than the other two. This difference, which was not observed during the previous runs of the notebooks (with the same datapoints and the same model), may be caused by stochasticity in the training process.

3) Step 3 - Finetuning the model for student answers prediction: In the third step, we finetune the same models introduced in the second step ("Baseline", "MCQBert1", and "MCQBert3") on the final downstream task, which is the prediction of the answers given by the students to MCQs. The main difference with respect to the procedure followed in the previous step is that, in this case, we pass the embeddings of the students to the models so that they can leverage the additional information to arrive at the final prediction. For the sake of this report, we will describe the results obtained using the second variation of our student embeddings (mastery level-based and time-synced with the MLP autoencoder) even though we unexpectedly obtained analogous results with the other embedding types. We also finetune for longer (three epochs instead of one), while

Table I
PERFORMANCE OF THE BASELINE MODEL ON THE DOWNSTREAM TASK OF STUDENTS' ANSWERS PREDICTION.

Baseline	Epoch 1	Epoch 2	Epoch 3
Accuracy score	0.774	0.782	0.780
F1 score - class 0	0.823	0.821	0.825
F1 score - class 1	0.688	0.719	0.704
Average F1 score	0.7555	0.770	0.7645
MCC	0.527	0.544	0.540

Table II
PERFORMANCE OF THE MCQBERT1 MODEL ON THE DOWNSTREAM TASK OF STUDENTS' ANSWERS PREDICTION (EMBEDDINGS VERSION 1 AND 2).

MCQBert1	Epoch 1	Epoch 2	Epoch 3
Accuracy score	0.787	0.792	0.789
F1 score - class 0	0.823	0.829	0.827
F1 score - class 1	0.732	0.733	0.730
Average F1 score	0.7775	0.781	0.7785
MCC	0.556	0.565	0.560

all the other details of the training procedure are precisely the same as in the second step. In this case, we evaluate the models on an evaluation/test dataset (containing MCQs already seen by the model but not for the particular student under analysis) by considering the same metrics introduced above (accuracy score, F1 score, MCC). The results of the different models are presented in Table I, Table V and Table IV.

For all the three models considered, the epoch in which they reach the best performance is the second one (the performance slightly decreases in the third training epoch for all the models). We will therefore discuss the results reached at the end of the second epoch. From the results listed above, we can notice that both MCQBert1 and MCQBert3 outperform the baseline model both in terms of the average F1 score and of the MCC. In particular, the percentual improvements over the baseline are the following:

- MCQBert1: +1.43% over the baseline on the average F1 score, +3.86% over the baseline on the MCC;
- MCQBert3: +1.88% over the baseline on the average F1 score, +4.78% over the baseline on the MCC.

Therefore, the embeddings seem beneficial for the task we aim to perform, as the performance improvement over the baseline is quite significant. However, it should be noted that these results tend to oscillate slightly from run to run, and a more extensive evaluation would be necessary to draw more precise conclusions. Since training the models is computationally very expensive, we were limited in the number of times a complete training could be performed. In any case, the models seem to be able to exploit the additional information received in the form of student embeddings to refine the prediction and to reach better results than the baseline model (in which the embeddings are not considered). Therefore, our approach seems very promising in obtaining a model able to predict which answers of MCQs will be selected by a student. This can also be noticed by considering the absolute results instead of the relative ones: at the second epoch, MCQBert1 got an average F1 score of 0.781, while MCQBert3 got an average F1 score of 0.7845, which is relatively high for a complex task such as the one we are considering. An interesting, and perhaps surprising, aspect is that the model which obtained the best results was MCQBert3, the one which obtained the worst results on the second experiment of the second step (i.e. remembering the correct answers to MCQs). This may signify that knowing the correct answers to MCQs is not necessarily improving the prediction capabilities of the models.

Table III
PERFORMANCE OF THE MCQBERT3 MODEL ON THE DOWNSTREAM TASK OF STUDENTS' ANSWERS PREDICTION (EMBEDDINGS VERSION 1 AND 2).

MCQBert3	Epoch 1	Epoch 2	Epoch 3
Accuracy score	0.786	0.793	0.783
F1 score - class 0	0.824	0.825	0.820
F1 score - class 1	0.726	0.744	0.726
Average F1 score	0.775	0.7845	0.773
MCC	0.553	0.570	0.547

Table IV
PERFORMANCE OF THE MCQBERT3 MODEL ON THE DOWNSTREAM TASK OF STUDENTS' ANSWERS PREDICTION (EMBEDDINGS VERSION 3 WITH LSTM).

MCQBert3	Epoch 1	Epoch 2	Epoch 3
Accuracy score	0.784	0.789	0.785
F1 score - class 0	0.825	0.827	0.824
F1 score - class 1	0.720	0.732	0.723
Average F1 score	0.773	0.780	0,774
MCC	0.549	0.561	0.550

V. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

In our project, we worked to build a model that could predict how students would answer multiple-choice questions (MCQs). We used information about the students in the form of embeddings to guide our model, and our tests showed that it performed well.

The first step of our project was to fine-tune a pre-trained GermanBERT model for a language modelling task. This allowed us to lower the model's perplexity on our MCQ dataset, making its predictions more relevant to the realm of education and learning. From this, we developed a fine-tuned model we called LernnaviBERT. This became our starting point for the project.

Moving on to the second part of our project, we fine-tuned LernnaviBERT specifically to predict the correct answers of MCQ answers. We were pleased to see that this part of the project was also successful. The model showed a good ability to generalize, and it was able to remember almost all the correct answers after we trained it on the full dataset.

This was particularly true for the Baseline and MCQBert1 models.

Based on the success of the second step, we moved on to the third part of our project. Here, we fine-tuned the model for its final task: predicting student answers to MCQs. Under this configuration, the student embeddings showed their worth, boosting the performance of the model. For example, MCQBert1 showed a 1.43% improvement over the baseline on the average F1 score, and a 3.86% improvement on the MCC. Similarly, MCQBert3 showed a 1.88% improvement over the baseline on the average F1 score, and a 4.78% improvement on the MCC. Even more encouraging was the absolute performance of the models. Considering the complex nature of our task, the results were quite good. MCQBert1 had an average F1 score of 0.781 (MCC: 0.565), and MCQBert3 had an average F1 score of 0.7845 (MCC: 0.570). With these results, we could answer our third research question: the student embeddings proved helpful to improve the predictions of the model, and allowed us to obtain a more accurate model. Consequently, our results demonstrate that the available data is substantial enough to enable precise answer prediction, effectively addressing the first sub-question of our research.

B. Future prospects

In assessing whether the embeddings can be regarded as a concise depiction of the student's knowledge (second sub-question), the response is nuanced. While the embeddings did result in some enhancement, it fell short of the anticipated level, particularly in the case of the LSTM-based autoencoder approach, which was initially deemed highly promising. Therefore, new approaches should be tried to better convey the student's knowledge in the embeddings, starting from the available data.

We strongly believe in the potential of our model to revolutionize the learning experience for Lernnavi users. By accurately forecasting the responses students are likely to give to MCQs, our model can facilitate personalized tutoring, assisting learners during their answer formulation process. This, in turn, could transform the way students interact with educational material, making learning more engaging and personalized. It could potentially spotlight areas where students struggle the most, paving the way for more targeted teaching and a richer, more interactive learning experience. Moreover, future works could focus on extending the prediction to open questions, to enable for a more comprehensive assistance. The advancements we have made serve as an exciting indication of how machine learning can contribute to the future of education, enhancing teaching methods, and better catering to individual student needs. This project has given us a glimpse into the future of education, and we are excited about the possibilities it presents.

REFERENCES

[1] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein, "Deep knowledge tracing," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/bac9162b47c56fc8a4d2a519803d51b3-Paper.pdf

[2] HuggingFace - dbmdz/bert-base-german-uncased. [Online]. Available: https://huggingface.co/dbmdz/bert-base-german-uncased

[3] HuggingFace - NLP Course. [Online]. Available: https://huggingface.co/learn/nlp-course/chapter7/3?fw=tf

[4] HuggingFace - lucazed/LernnaviBERT. [Online]. Available: https://huggingface.co/lucazed/LernnaviBERT

VI. Appendix

This appendix includes additional figures and material (to increase the clarity of the report) that were not included in the main report due to a lack of space.

(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) B
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) C
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) A
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) C
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) C
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) B
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) B
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) B
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) B
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) B
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) B

Figure 2. Example of dataset to illustrate the dataset split into a training and a validation/test dataset. For clarity, each individual MCQ is characterized by the same colour.

Table V
Performance of the MCQBert1 model on the downstream task of students' answers prediction (embeddings version 3).

MCQBert1	Epoch 1	Epoch 2	Epoch 3
Accuracy score	0.781	0.787	0.785
F1 score - class 0	0.826	0.823	0.825
F1 score - class 1	0.706	0.734	0.722
Average F1 score	0.766	0.779	0.774
MCC	0.543	0.558	0.551

TRAINING DATASET
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) B
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) B
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) C
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) A
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) D
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) B
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) D
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) B
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) D
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) B
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) C

VALIDATION DATASET
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) A
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) B
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) C
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) B

Figure 3. Training and validation/test datasets obtained by splitting the dataset using the first method presented in the report. Different instances of the same MCQ can be in both datasets.

TRAINING DATASET
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) B
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) C
(Q) Compute 5 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) A
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) D
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) C
(Q) Compute 7 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) C
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) B
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) A
(Q) Compute 6 + 6 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) B

VALIDATION DATASET
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 159684 (ANS) B
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 695847 (ANS) D
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 079685 (ANS) B
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 333695 (ANS) A
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 229583 (ANS) B
(Q) Compute 6 + 7 (A) 12 (B) 13 (C) 11 (D) 14 (ID) 669584 (ANS) B

Figure 4. Training and validation/test datasets obtained by splitting the dataset using the second method presented in the report. Different instances of the same MCQ can be in only one of the two datasets.

Which of these are Swiss cities?
A. Lausanne
B. Rome
C. Paris
D. Bern
The student selected: [A, C]

From sentences to IDs:
Which of these are Swiss cities? → [11453 145 787 12 6954 111 9]
Lausanne → [67896]
Rome → [55543]
Paris → [18756]
Bern → [6554]

Question - Answer A:
“ids” → [11453 145 787 12 6954 111 9] + [SEP_ID] + [67896]
“label” → 1
“student_embedding” → [STUDENT EMBEDDING]

Question - Answer B:
“ids” → [11453 145 787 12 6954 111 9] + [SEP_ID] + [55543]
“label” → 0
“student_embedding” → [STUDENT EMBEDDING]

Question - Answer C:
“ids” → [11453 145 787 12 6954 111 9] + [SEP_ID] + [18756]
“label” → 1
“student_embedding” → [STUDENT EMBEDDING]

Question - Answer D:
“ids” → [11453 145 787 12 6954 111 9] + [SEP_ID] + [6554]
“label” → 0
“student_embedding” → [STUDENT EMBEDDING]

Figure 5. Scheme of the procedure used to transform the MCQ answer prediction task in a binary classification task (and of the data preprocessing steps).

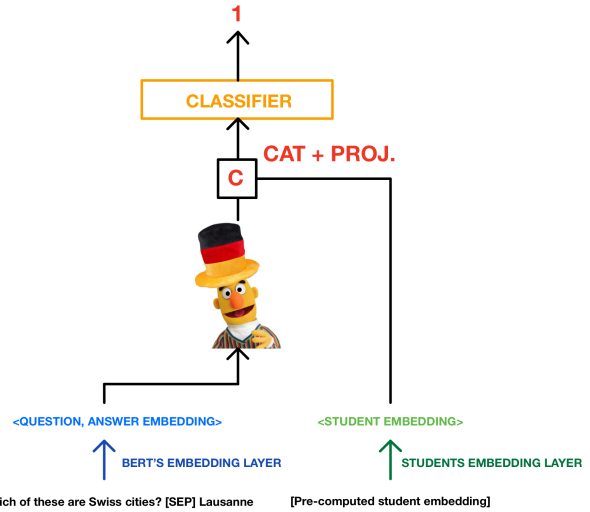

Figure 6. Scheme of the model “MCQBert1”, in which the student embeddings are concatenated just before the classification layer.

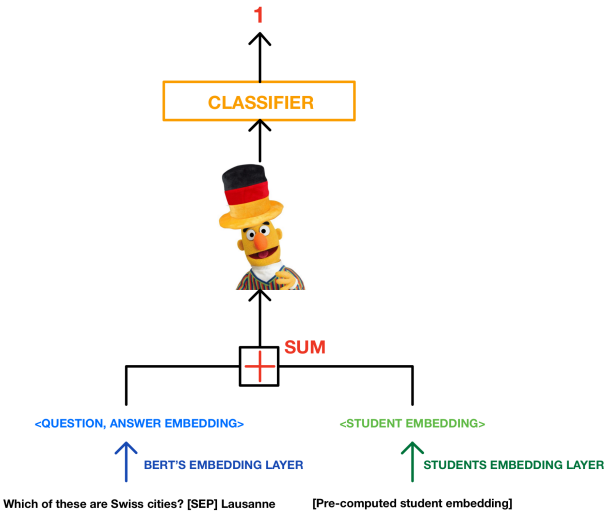

Figure 7. Scheme of the model “MCQBert3”, in which the student embeddings are summed at the input of the model.

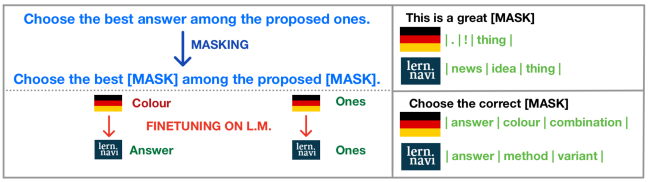

Figure 8. Scheme of the finetuning process on a language modelling task done as the first step of our pipeline.

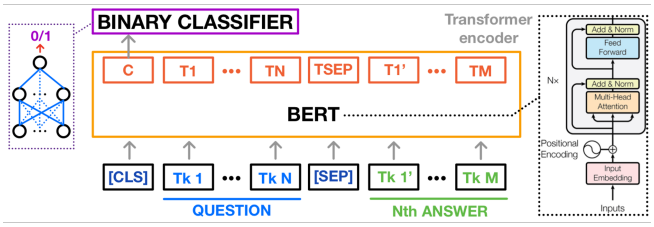

Figure 9. Complete architecture of the models based on BERT that we used in the project. Note: the student embeddings are not considered for this scheme.