Programming questions classifications

Deivis Zolba
Faculty of Mathematics and Informatics
Vilnius, Lithuania
deivis.zolba@mif.stud.vu.lt

***Abstract*—This project investigates the classification of programming-related questions into predefined categories using semantic embeddings and machine learning models. The dataset consists of synthetic examples from HuggingFace, combining text and code. Multiple approaches were evaluated, including SVM classifiers with MiniLM and RoBERTa embeddings, and fine-tuning of the BERT transformer model. Results show that transformer-based models, particularly BERT, significantly outperform simpler alternatives, achieving up to 91% accuracy.**

***Index Terms*—text classification, programming questions, sentence embeddings, BERT, SVM, NLP**

I. Introduction

This project focuses on classifying programming-related questions into predefined categories such as *General Question*, *Help Fix Code*, *Explain Code*, *Help Write Code*, and *Question from Code*. The dataset from HuggingFace consists of synthetic examples containing both natural language queries and associated code snippets.

To tackle this problem, a semantic classification approach based on sentence embeddings was used. Pre-trained transformer models were employed from the sentence-transformers library to convert question-code into numerical representations. Transformed values are then classified by Support Vector Machine (SVM), and one based on the BERT architecture. Models' performances are evaluated using accuracy, precision, recall, and F1-score.

II. Literature review

Text embeddings convert text into numerical vectors that reflect the semantic relationships among words or phrases. Early methods like Word2Vec and GloVe generate static embeddings, assigning a single vector representation to each word regardless of context. These techniques have been instrumental in various NLP tasks because they capture syntactic and semantic information [4].

SVM, a supervised learning model, is used for classification and regression tasks. It works by finding the optimal hyperplane that separates data points of different classes with the maximum margin [3]. Embeddings provide dense, informative text representations, while SVMs offer robust classification capabilities. This combination performs various text classification tasks, including spam detection and sentiment analysis.

BERT (Bidirectional Encoder Representations from Transformers) is a language model introduced by Google AI in 2018. It advances natural language processing (NLP) by enabling a deep, bidirectional understanding of language context. Unlike previous models that process text either left-to-right or right-to-left, BERT reads entire sentences simultaneously, allowing it to grasp the full context of a word based on all adjacent words. This bidirectional approach enhances the model's language understanding, resulting in better performance on various NLP tasks [2].

Incorporating a basic classification layer on the pre-trained model enables BERT to be tailored for tasks like sentiment analysis, named entity recognition, and question answering. This adaptability, combined with its deep contextual understanding, has led BERT to achieve state-of-the-art results across numerous benchmarks, setting a new standard in the field of NLP [1].

III. Methods

A. Data Set

Public dataset from HuggingFace with about 7000 synthetically created programming questions. Each record has a short text question and, sometimes, a code snippet. Every question is tagged with one of five labels: *General Question*, *Help Fix Code*, *Explain Code*, *Help Write Code*, or *Question from Code*. If a question field was empty, it was filled with the task description or intention that came with the data. If the code snippet was available, it was also combined with the question part. To keep classes balanced, the first experiments were kept simple, 250 examples for each label; other iterations used more of the dataset to test data imbalance. Data distribution is shown in Figure 1.

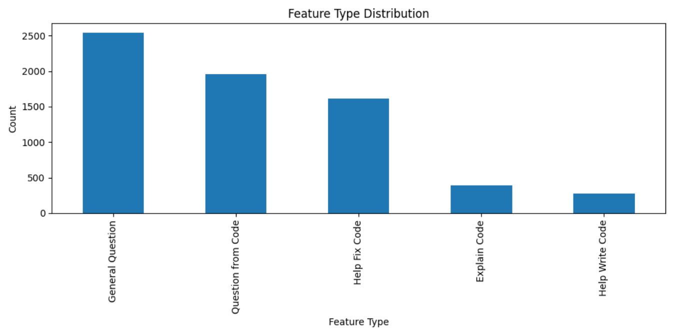


Figure 1 – Feature-type distribution in the data set

B. Models

Two ways to turn text into predictions were implemented and evaluated.

Text embedding + SVM. Texts were turned into X-dimensional vectors with the pre-trained small model - all-MiniLM-L6-v2 (X-384), and larger model - all-roberta-large-v1 (X-1024). Vectors were scaled and trained on a Support Vector Machine (SVM) with an RBF kernel.

BERT architecture. Fine-tuned bert-base-uncased by adding a small classification layer on top. Text was tokenised (max 512 tokens, with truncation and padding). Training ran for three epochs with a batch size of 16.

C. Evaluation

Data was split 80% training and 20 % for testing. For each experiment, the report had: accuracy, precision, recall, and macro F_1 — also, a confusion matrix and a t-SNE plot for a quick visual check of class separation.

IV. Results

A. MiniLM + SVM Classifier, balanced classes 250 each

The first experiment used text embeddings from all-MiniLM-L6-v2, classified with a Support Vector Machine. The model got an overall accuracy of 72 %. The highest-performing class was *Question from Code* with an F_1 score of 0.81. *Help Fix Code* was the most challenging class, probably because its most simmilar to other categories.

Table I shows the detailed classification report and figure 2 shows the confusion matrix.

Class	**Precision**	**Recall**	**F1-score**
Explain Code	0.73	0.76	0.75
General Question	0.64	0.76	0.70
Help Fix Code	0.75	0.54	0.63
Help Write Code	0.67	0.70	0.69
Question from Code	0.80	0.82	0.81
Accuracy		0.72	

Table I – Classification results for MiniLM + SVM model

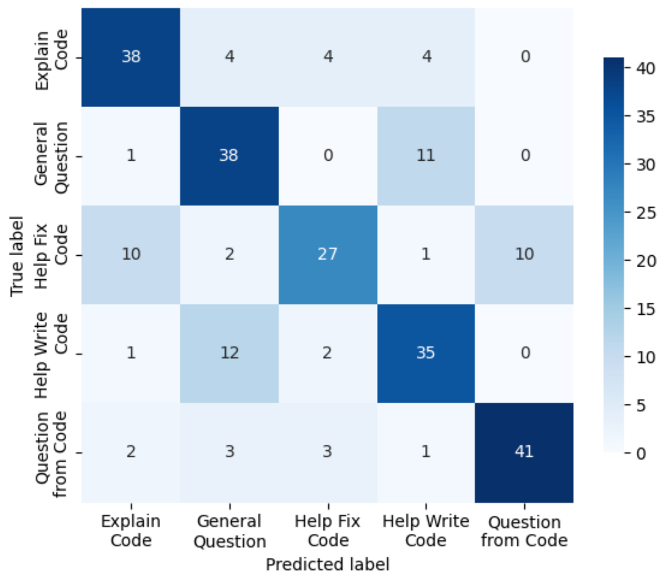


Figure 2 – Confusion matrix of MiniLM sentence embeddings

Figure 3 shows a two-dimensional t-SNE projection of the MiniLM sentence embeddings used in the SVM classifier. Each point represents one input example, colored by feature type.

Clusters are hardly visible, while there is some order, it doesn't help us much in determining feature types consistently. While some concentrated spots exist, the 2D view shows no clear clusters.

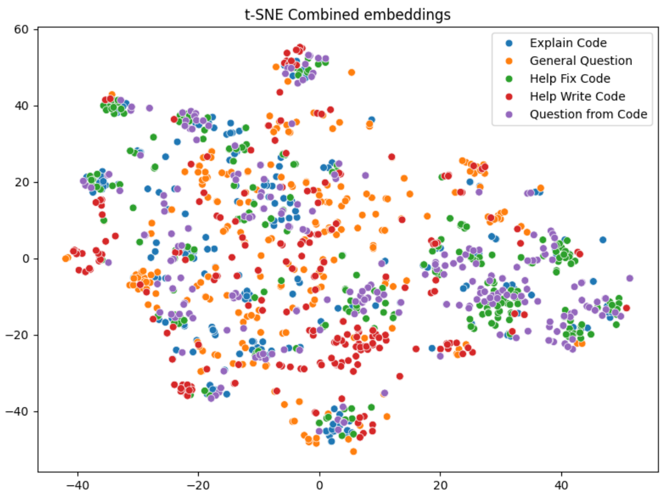


Figure 3 – t-SNE projection of MiniLM sentence embeddings colored by feature type

B. Roberta + SVM, balanced classes 250 each

Replaced the MiniLM embedder with a larger model: `all-roberta-large-v1`. This improved overall accuracy to 76 %.

The best performance was again on *Explain Code*, reaching an F_1 score of 0.86. *Help Fix Code* remained the most challenging class, with a lower recall of 0.64, likely due to similarity.

Table II shows the detailed classification report, and Figure 4 shows the confusion matrix.

Class	**Precision**	**Recall**	**F1-score**
Explain Code	0.91	0.82	0.86
General Question	0.71	0.82	0.76
Help Fix Code	0.74	0.64	0.69
Help Write Code	0.71	0.74	0.73
Question from Code	0.77	0.80	0.78
Accuracy		0.76	

Table II – Classification results using all-roberta-large-v1 + SVM

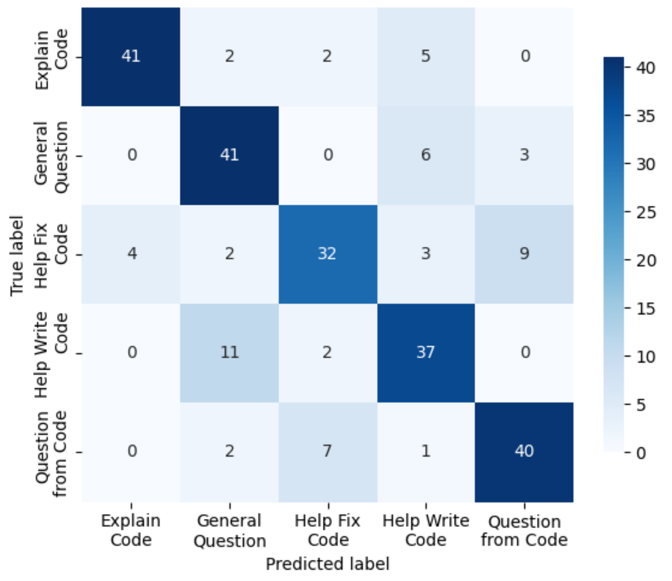


Figure 4 – Confusion matrix of Roberta sentence embeddings

Figure 5 shows a two-dimensional t-SNE projection of the Roberta sentence embeddings used in the SVM classifier. Each point represents one input example, colored by feature type.

Clusters are more visible than on a small model; there are some clear clusters, but still a lot of chaos.

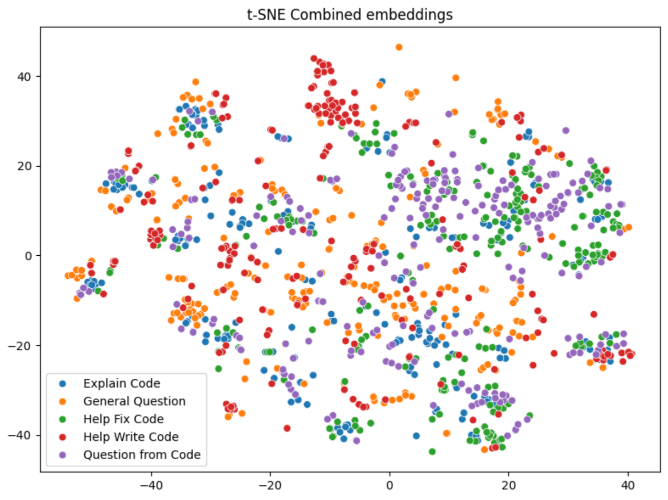


Figure 5 – t-SNE projection of Roberta sentence embeddings colored by feature type

C. Roberta + SVM, full data-set

The final experiment used `all-roberta-large-v1`. The model was evaluated on the full dataset without class balancing. It achieved an overall accuracy of 88 %, the highest among all tested configurations, probably due to the larger data size.

The best-performing class was *General Question*, with an F_1 score of 0.94. *Question from Code* and *Help Fix Code* also showed strong results with F_1 scores. The most challenging label remains *Help Write Code*, which had the lowest recall (0.52), likely due to the small sample size and similarity with other types.

Table III shows the complete classification metrics, and Figure 6 shows the confusion matrix.

Class	**Precision**	**Recall**	**F1-score**	**Entries**
Explain Code	0.75	0.77	0.76	78
General Question	0.92	0.96	0.94	509
Help Fix Code	0.87	0.86	0.87	322
Help Write Code	0.69	0.52	0.59	56
Question from Code	0.89	0.88	0.89	391
Accuracy		0.88		
Macro Avg	0.83	0.80	0.81	1356
Weighted Avg	0.88	0.88	0.88	1356

Table III – Classification results using all-roberta-large-v1 model

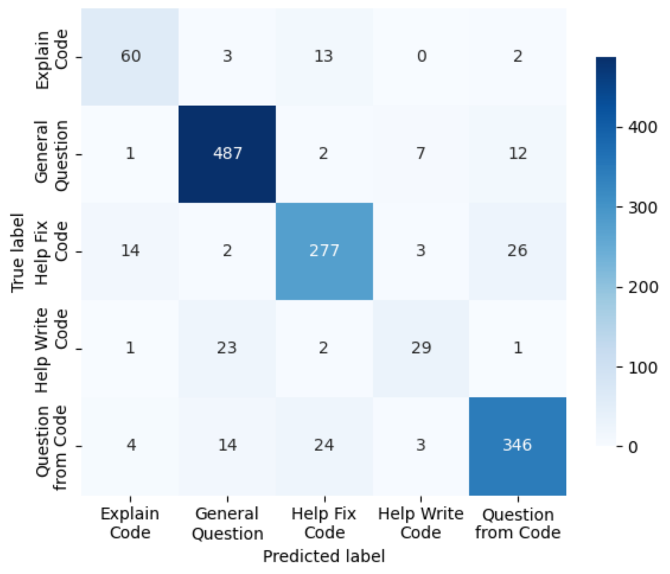


Figure 6 – Confusion matrix of Roberta sentence embeddings

Figure 7 shows a two-dimensional t-SNE projection of the Roberta sentence embeddings used in the SVM classifier. Each point represents one input example, colored by feature type.

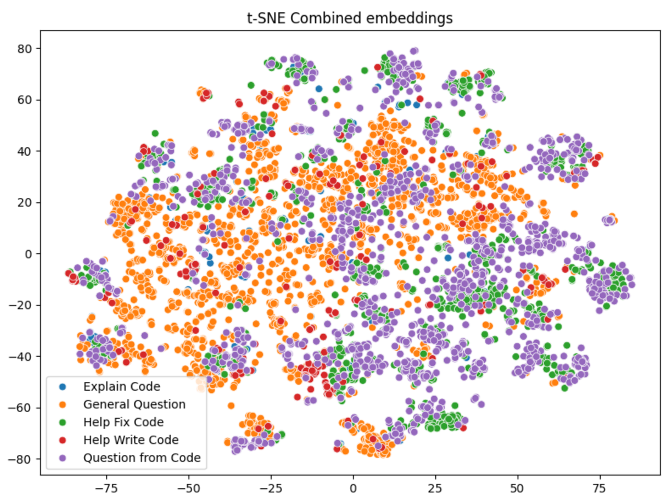


Figure 7 – t-SNE projection of Roberta sentence embeddings colored by feature type

D. BERT Classifier, balanced classes 250 each

This experiment fine-tuned the `bert-base-uncased` model directly on the classification task using HuggingFace's `Trainer` API. The input text is formatted in the same way as in previous models. A maximum of 250 samples per category was used to keep class distribution even.

The model was trained for 3 epochs on a GPU using a batch size 16. During training, performance steadily improved: the weighted F1-score increased from 0.58 in the first epoch to 0.89 in the third. Training and validation losses also decreased significantly, indicating effective convergence, see Table IV.

Epoch	Training Loss	Validation Loss	F1-score
1	0.919	0.837	0.581
2	0.661	0.600	0.800
3	0.443	0.438	0.888

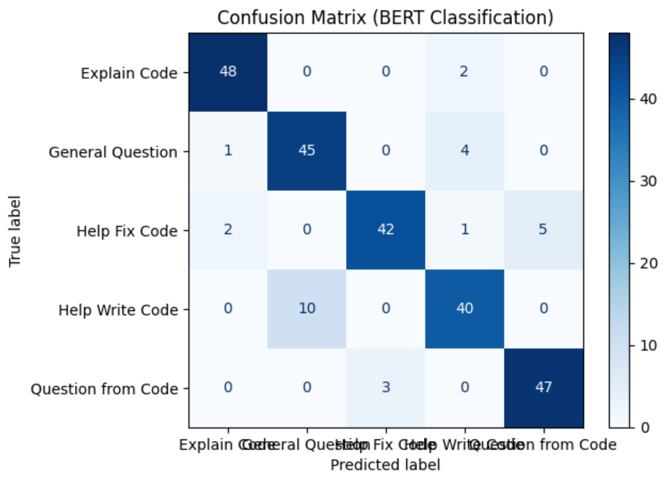
Table IV – Training progress of BERT-based classifier across 3 epochs

Compared to other models, the BERT architecture-based model achieved strong results with minimal feature engineering and could effectively separate classes. However, its training time was notably higher than that of the SVM-based alternatives.

The model's classification performance is further analyzed in Table V, and the confusion matrix in Figure 8.

Class	Precision	Recall	F1-score
Explain Code	0.94	0.96	0.95
General Question	0.82	0.90	0.86
Help Fix Code	0.93	0.84	0.88
Help Write Code	0.85	0.80	0.82
Question from Code	0.90	0.94	0.92
Accuracy		0.89	

Table V – Classification results of BERT fine-tuned model on balanced 5-class task



	Explain Code	General Question	Help Fix Code	Help Write Code	Question from Code
Explain Code	48	0	0	2	0
General Question	1	45	0	4	0
Help Fix Code	2	0	42	1	5
Help Write Code	0	10	0	40	0
Question from Code	0	0	3	0	47

Figure 8 – Confusion matrix of trained BERT model

Figure 9 shows a two-dimensional t-SNE projection of the trained BERT model.

Clear clusters are visible; some points are where they don't belong, but similarity between types is inescapable in this problem domain. Overall, the results are great.

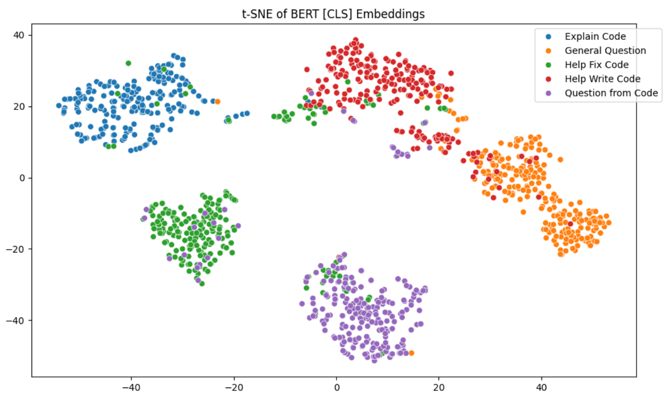


Figure 9 – t-SNE projection of trained BERT model, colored by feature type

E. BERT Classifier, full data set

The BERT model was also fine-tuned on the full dataset without class balancing. As shown in Table VI, the model converged quickly, achieving a validation F1-score of 0.91 by the third epoch. Training and validation losses steadily decreased, showing effective learning and no overfitting. Compared to previous smaller-scale experiments, performance improved across all metrics.

Epoch	Training Loss	Validation Loss	F1-score
1	0.242	0.344	0.884
2	0.211	0.311	0.908
3	0.176	0.301	0.910

Table VI – Training progress of BERT classifier on the full dataset

The model's classification performance is further analyzed in Table VII, and the confusion matrix in Figure 10.

Class	Precision	Recall	F1-score	Support
Explain Code	0.90	0.90	0.90	78
General Question	0.96	0.96	0.96	509
Help Fix Code	0.91	0.86	0.89	322
Help Write Code	0.59	0.62	0.61	56
Question from Code	0.90	0.93	0.92	391
Accuracy		0.91		
Macro Avg	0.85	0.85	0.85	1356
Weighted Avg	0.91	0.91	0.91	1356

Table VII – Classification report for fine-tuned BERT on full dataset

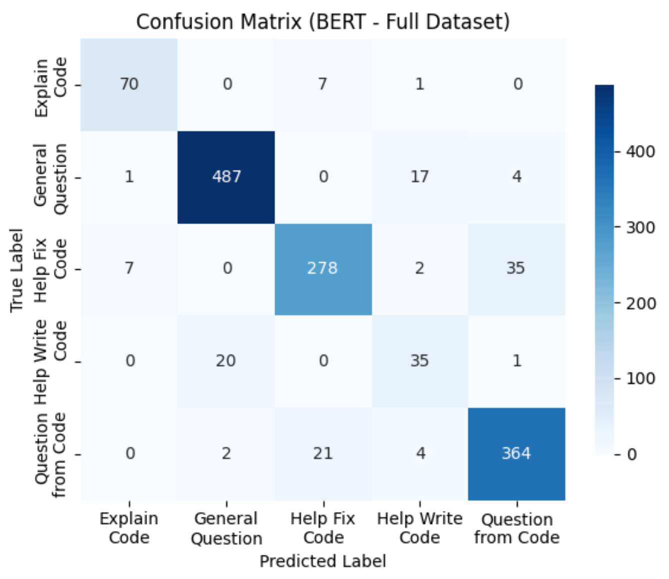


Figure 10 – Confusion matrix of trained BERT model

Figure 11 shows a two-dimensional t-SNE projection of the trained BERT model.

Clear cluster visibility, model performed well even with smaller and unrepresented classes.

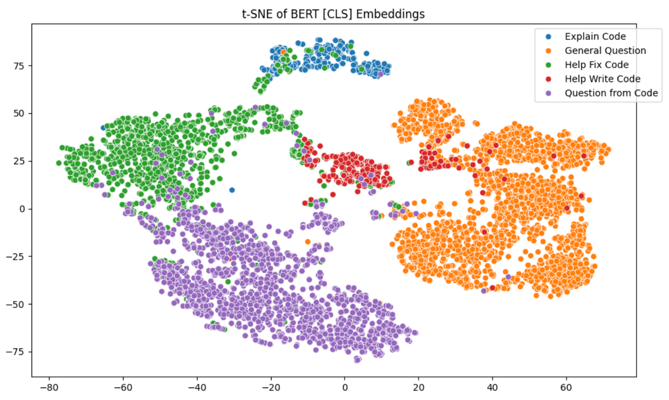


Figure 11 – t-SNE projection of trained BERT model, colored by feature type

V. Conclusion

The project explored the classification of programming-related questions using semantic embeddings and machine learning models. Several approaches were tested, including sentence embeddings combined with SVM classifiers, MiniLM and Roberta, and fine-tuning transformer models like BERT. The experiments showed that transformer-based models significantly outperform simpler setups. The fine-tuned BERT classifier on the full dataset achieved the best results, reaching an accuracy of 91%, and a macro F1-score of 0.85. Overall, this work demonstrates that deep learning models, with minimal feature engineering, can effectively categorize technical questions into meaningful types.

VI. Recommendations

- Use pre-trained transformer models like BERT to generate solid embeddings for input text.
- Combine question text and code snippets into a single input string to preserve context and improve classification relevance; the more context, the better.
- Apply sentence embedding models like all-MiniLM-L6-v2 for lightweight and fast classification with SVMs, for simple experimentations.
- Visualize sentence embedding spaces with t-SNE to understand class separation and potential overlap.
- Evaluate model performance using macro-averaged F1-score, not just accuracy, especially when class sizes differ.

References

[1] Rani Horev. Bert explained: State of the art language model for nlp. https://medium.com/towards-data-science/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270, 2018.
[2] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805, 2018.
[3] UPTU Khabar. Machine learning techniques old paper aktu b tech third year, 2023.
[4] Wikipedia. Word2vec. https://en.wikipedia.org/wiki/Word2vec.