UDC 004.8:519.673

DEVELOPMENT OF SOFTWARE FOR AUTOMATIC GENERATION OF MATHEMATICAL PROBLEMS

Stepanov D.E., student

Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus

Kniaziuk N.V. – PhD in Physics and Mathematics, Associate Professor

Annotation. Local software to perform the process of generating test tasks in higher mathematics with the possibility of training. Developed considering machine learning practices and technologies.

Keywords. EdTech, Python scikit-learn, machine learning, HDBSCAN, AST, question generation, artificial intelligence, higher mathematics.

The task was set to implement a question generator, using the latest technologies in the field of machine learning and artificial intelligence, aim was reached. Sole purpose of software is to generate questions on relevant mathematical topics that are high-on-demand in modern higher education. The project allows to create mathematical problems in various popular formats (Moodle XML - specifically for integration with LMS BSUIR; Tex; Word; Text file). Also while being completely sustainable for local work, no connection of third-party providers or systems is required, considering defined accuracy is achieved. The models are trained and data being stored in a local database, which in the future, picking up the concept of a certain subtype of tasks, are able to generate the required number of correct, solvable and customizable tasks.

The software development was carried out in the Python programming language, proceeded with SOLID standards and the principles of system design. Tho algorithms and data structures bases were used to optimize the work of algorithms, which allowed to significantly optimize the course of actions. For example, the transition to parallelism was made, which allowed to reduce the generation time of large amounts of data by more than 10 times, as shown in Figure 1:

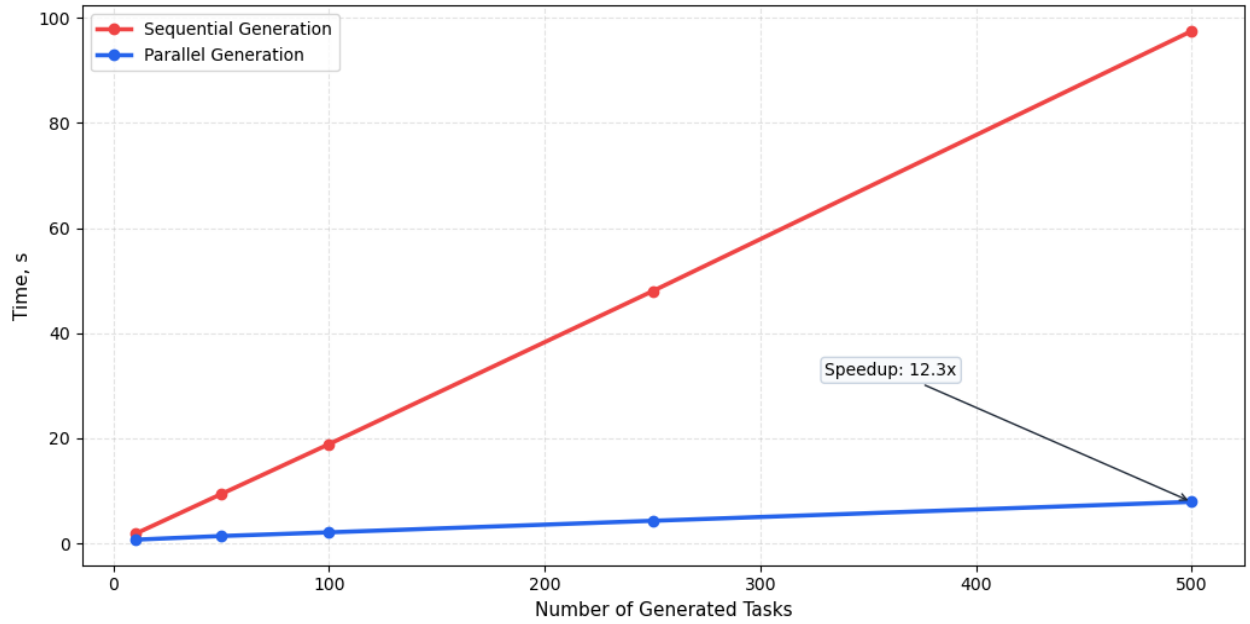

Figure 1 - Illustration of the comparison of the generation time rate of sequential(1) and optimized(2)

Before the introduction of the system, the algorithm of the entire generation process was time-heavy complex:

$$T_n = O(cn), \tag{1}$$

where T_n – generation time, c – average costs for processing one task, n – the number of generated tasks.

Now, after optimization, the estimated execution time became:

$$T_n = O(\frac{cn}{p}), \quad (2)$$

where T_n – generation time, c – average costs for processing one task, n – the number of generated tasks, p – number of parallel processes.

It follows from the expression (1) that with sequential processing, the execution time of the generation algorithm increases linearly with an increase in the number of tasks. After the implementation of parallel processing, the load is distributed between p workflows, as a result of which the estimated execution time is reduced and described by the expression (2). At the same time, the theoretical nature of the dependence on the volume of input data remains linear, but the actual operating time of the system decreases in the first approximation in proportion to the number of processes involved (taking into account the overhead costs for synchronization and data exchange)

Additional acceleration that allowed to slightly reduce the total overhead of the system. was achieved by caching validation results, reducing the number of generation retries and optimizing the data processing pipeline.

If we talk about the architecture of the software tool itself, it is recreated as a modular pipeline, each divided into submodules, within, which are responsible for data conversion, analysis and validation, input-output, etc. Each significant core module has implemented appropriate auxiliary modules and methods that perform certain tasks within the framework of the entire, seam pipeline. An important aspect of the design of the initial architecture, taking into account the modern trends in the continuous development of capabilities and widespread integration of machine learning and artificial intelligence systems, the idea of subsequent use, integration and applications of the PS (PS — Programmed solution) in the initial test, and with the successful effectiveness of the project and functioning training environments, the output interfaces of integration with popular technologies using API [7] (API -– Application Programming Interface) and endpoint simulations were set. As one of them, the development of a well structured API of the request node to the local LLM (LLM -– Large Language Model) for additional consolidation of the quality check of the training module in the context of the PS, the interface with access to the updated local database was recreated.

In the process of recreation, author got brainstormed with modern scientific articles and methods of working with artificial intelligence and machine learning in general [8]. The knowledge gained was applied and the relevant technologies were implemented. RAG [5] (RAG — Retrieval Augmented Generation) was recreated — a subsystem within the PS, allowing local data storage using a relational database with JSON (JSON — Data Format) fields. The approach allowed to develop a combination of combinations of both the advantages of the classical scheme for metadata and the local relational storage, ensuring consistency between the data and the context of the models. The purpose of the subsystem is to save information about the processed tasks, form their internal representations and further extract the closest examples when generating new tasks.

Returning back, if we consider the architecture of the software tool as a whole, it includes the stages of loading a sample of tasks, normalizing input data, segmenting the condition, parsing the mathematical expression, analyzing features, generating, validating and subsequent exporting the results. This organization allows us to divide the common task into a number of interrelated stages, (that's basically the pipeline concept) each of which is responsible for its own subtask within a single processing process, a detailed description is shown in Figure 2.

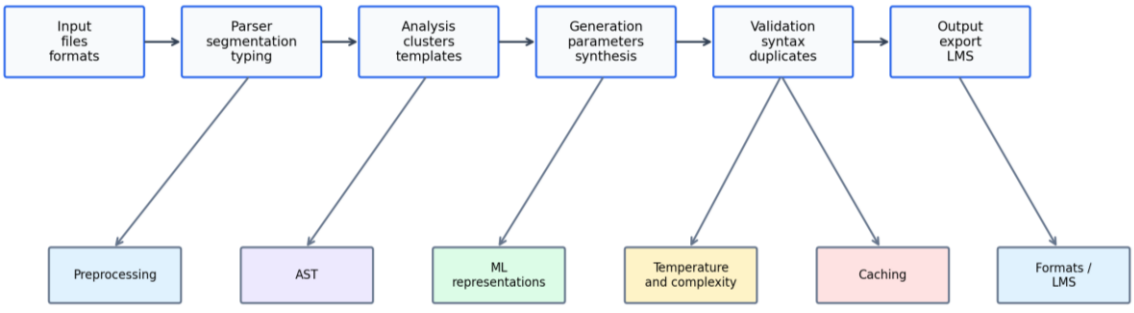

Figure 2 - Illustration of the stages of the pipeline and its auxiliary mechanisms

In addition to the main pipeline, the system provides a set of supporting components that provide data storage, structural analysis of expressions, prototyping, search for relevant examples and interaction with additional intelligent services. In the assembly, the components form an internal circuit of data storage, analysis and extraction, on which this solution, the system uses the accumulated ideas about previously processed examples.

A detailed description is given in Figure 3.

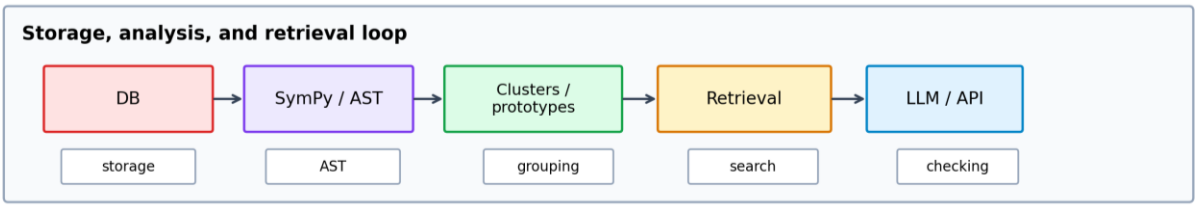

Figure 3 - Illustration of supporting components

Summarizing about architecture of the PS, three major factors were considered and applied.

1. Setting up a proper pipeline for the task. Strict principles were set about modules that should be used. Only the essential functionality is on. That ensures codebase was made clean and flexible for improvements and refactoring.

2. Keeping PS architecture maintainable and scalable for future additions. Implemented patterns, examples of which were explained up above, allow us to grow and scale.

3. Usage of modern principles in the machine learning field. We did our research to be certain that's the technologies retrieved is the pinnacle and up to date.

Now, slowly descending into the deeper end, let us talk about the processes within the pipeline itself.

It is useful to consider the operation of the system in two modes: with a trained and with an untrained model. In the untrained state, the generator mainly relies on predefined templates, configuration parameters, heuristic rules and standard mechanisms of symbol analysis. Such mode allows the system to function even without special adjustment to a specific data body, but the variety and objective accuracy of the created tasks are limited in case, and the untrained mode itself is not recommended for production use.

But after learning on a set of real examples, we stumble upon a trained model, which is much reliable and qualified for our tasks. The system forms a more meaningful internal representation of the subject area. Text templates, equation templates, parameter distributions, vector representations of tasks, class centroids, prototypes and generation indices are extracted from the sample of the original tasks, all data is arranged according to optimized data structures. Thanks to this, the trained model begins to take into account real formulations, typical expression structures and statistical features of parameters. As a result, new tasks become closer to the style of the training sample, better preserve the thematic focus and reflect real methodological patterns to a greater extent.

On the next graph we can witness the comparison between two. Summarizing, the untrained system works on the principle of generation based on rules and basic templates, while the trained system switches to the generation mode based on the studied examples and their structural characteristics. It is this difference that makes model training an important stage in improving the quality of automatic generation.

The comparison is shown in Figure 4.

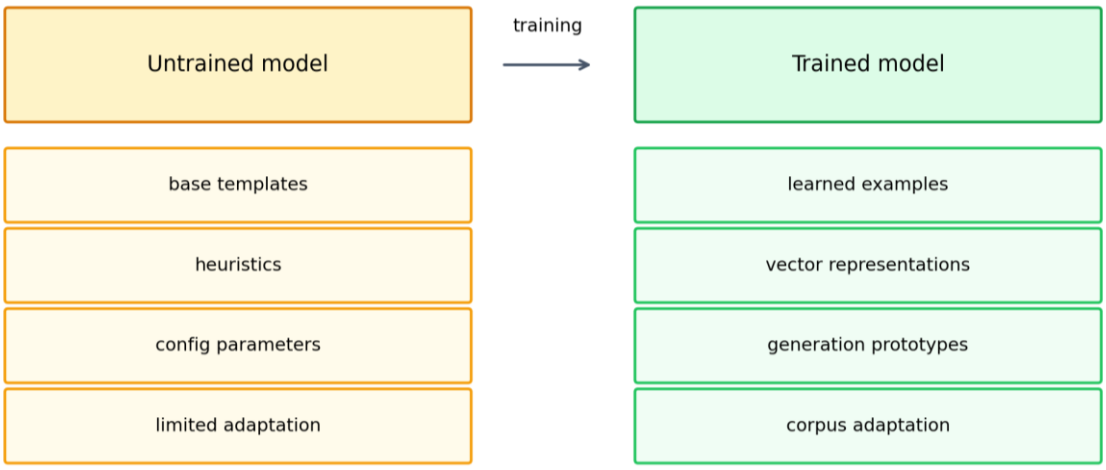

Figure 4 - Comparison of a trained and untrained generator.

Speaking of trained models, the latest technologies in the field of mathematics and machine learning are used to train models. It is worth distinguishing the essence of their work, that's because we could just "arrange the GUI for a LLM wrapper", instead, we stick to ours' own decisions; even thought PS uses local LLM for additional layer of validation, we do not rely on it. Components involved in the processes are described below. Note that the entire system is operational locally.

SymPy [1] and Ollama [6] components are used in the project and perform various functions and are not interchangeable. SymPy is a Python library used as a tool for strict symbolic analysis of mathematical

expressions. With its help, the system converts formulas into an internal representation, allocates variables, operations, functions, determines the structural properties of the expression and forms features based on the analysis tree. Its use allows you not to prescribe mathematical logic, but to use a ready-made component to speed up development.

Ollama, on the contrary, acts as an additional intellectual module focused on processing the text side of the task and supporting classification in cases where strict rules are not enough. By building an entry point, the generator is able to train the model in completely new types of tasks that have not been encountered before, access takes place in cases where additional verification of the task type or the possibility of its solution is required.

If SymPy can be considered as a mathematical formula analyzer, Ollama acts as a language assistant capable of working with formulations, context and non-rigid text descriptions. Consequently, the first component is responsible for strict formalization, and the second is responsible for interpretation flexibility and extensibility. Details are shown in Figure 5.

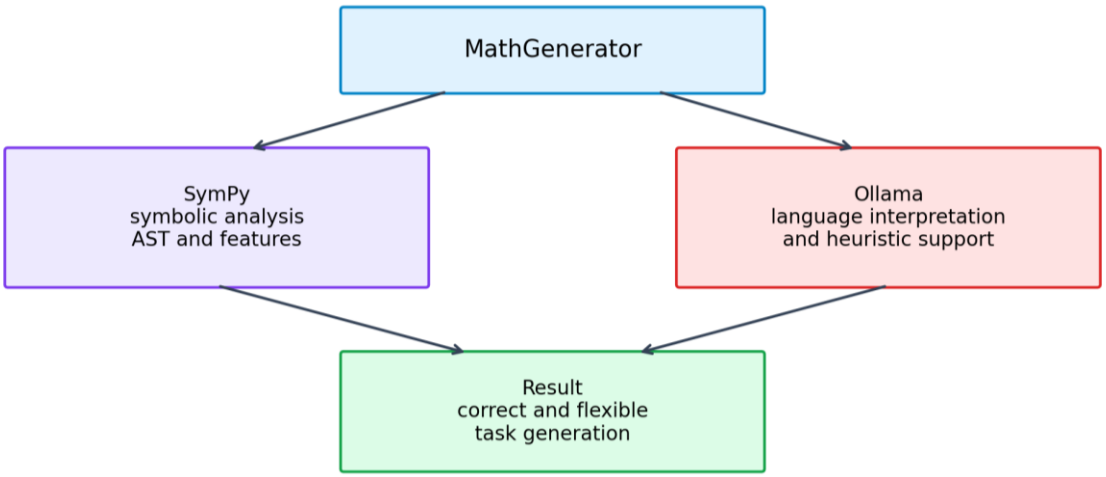

Figure 5 - Illustration of the use of SymPy, Ollama in the project architecture

For distinguishing the maths, we use AST [2] (AST - Abstract Syntactic Trees). The analysis of abstract syntactic trees, or AST, plays an essential role in the project. After converting a mathematical expression into a symbolic form, the system receives a parsing tree in which the nodes correspond to operations, functions, numbers and variables. Based on this tree, you can determine the depth of the expression, the number of operations, the number of variables, the presence of degrees, functions and other structural features. Such analysis is much more informative than a simple comparison of strings, as it allows you to take into account the internal organization of a mathematical object.

The use of AST is especially important in situations where different-looking records actually describe tasks that are similar in structure. For example, two expressions may differ in specific coefficients or form of record, but have the same architecture of the operations tree. In this case, the system is able to recognize them as close in terms of generation, classification and training. Working with parsing trees increases the content of the feature description and makes generation more meaningful, the use of parsing trees was justified by their widespread use when working with grammars, in particular in compiler technologies. With addition of AST's alone, we, of course, unable to build things right straight away. That's where we implement machine learning.

Machine learning methods in the project are used as part of a general system of analysis and generation of mathematical problems. The main idea is that the program receives a set of ready-made examples(which retrieved via AST's) , extracts textual, numerical and structural features from them, and then forms a representation of tasks in the general feature space. This allows not only to distinguish the types of tasks, but also to identify similarities between them, find characteristic templates and use them when building new options. This approach is especially important in educational systems, where the quality of generation is determined not by random selection of coefficients, but by the preservation of semantic and structural proximity to the original examples.

Among the methods used are the vector representation of tasks (we're talking about clustering, more of HDBSCAN information can be found here [3],[4]), the search for the nearest similar examples, the construction of class prototypes and groupings. Vectorization is based on the combination of text features, structural tokens and numerical characteristics of the expression. This allows us to take into account several levels of task description at once: the formulation of the condition, the mathematical structure of the expression and quantitative parameters. As a result, each task is converted into a set of features suitable for further comparison, grouping and classification.

Clustering in the project is used as a way to group tasks with a similar structure and similar features. The system provides usage of the HDBSCAN algorithm [3], which is convenient because it is able to allocate groups of objects without rigidly setting the exact number of clusters in advance. This is especially useful for the educational task corps, because real samples often contain uneven families of examples: some types are represented by a large number of samples, others are rare, and some of the tasks can be intermediate or mixed in structure. In such conditions, density-based clustering helps to detect natural groups of examples and are perfectly suited for use in building generation patterns.

In practice, this means that the system is able to distinguish individual groups of tasks not only by explicit label, but also by the similarity of the internal representation. Due to this, the stability of the generator increases: it is based not on a single example, but on a whole family of close tasks, which makes the created options more diverse and at the same time more consistent in structure [4]. We can see the logic used behind clustering in Figure 6.

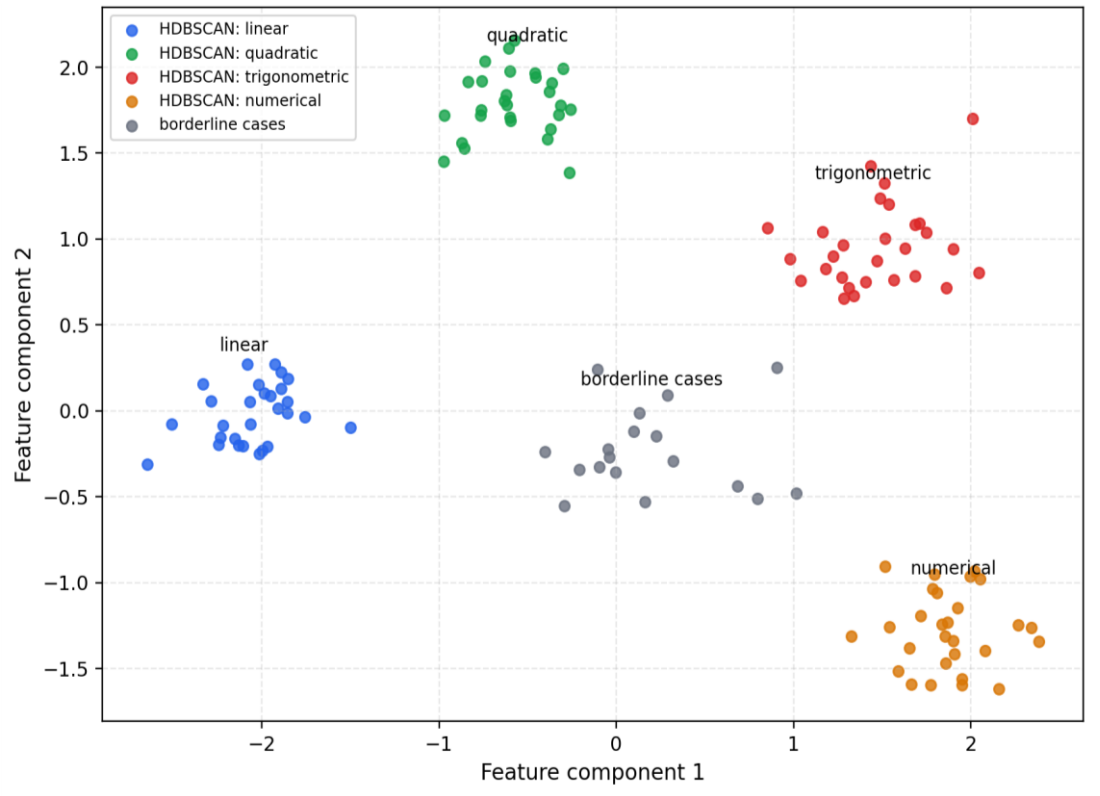

Figure 6 - Illustration of the principle of clustering in the context of software

Going through some additional clarity improving parameters, we can look at temperature and complicity:

A separate parameter affecting the generation process is temperature [9]. It is now used everywhere in modern AI-systems. It determines the degree of variability of the created results. At low values, the temperature makes generation more stable and predictable, because the system more often chooses the most typical and safe options for constructing a problem. As the temperature “rises”, the variety of results increases, but at the same time the proportion of correct and stable examples may decrease. The choice of temperature is a compromise between variability and reliability of generation. The graph shown in Figure 7.

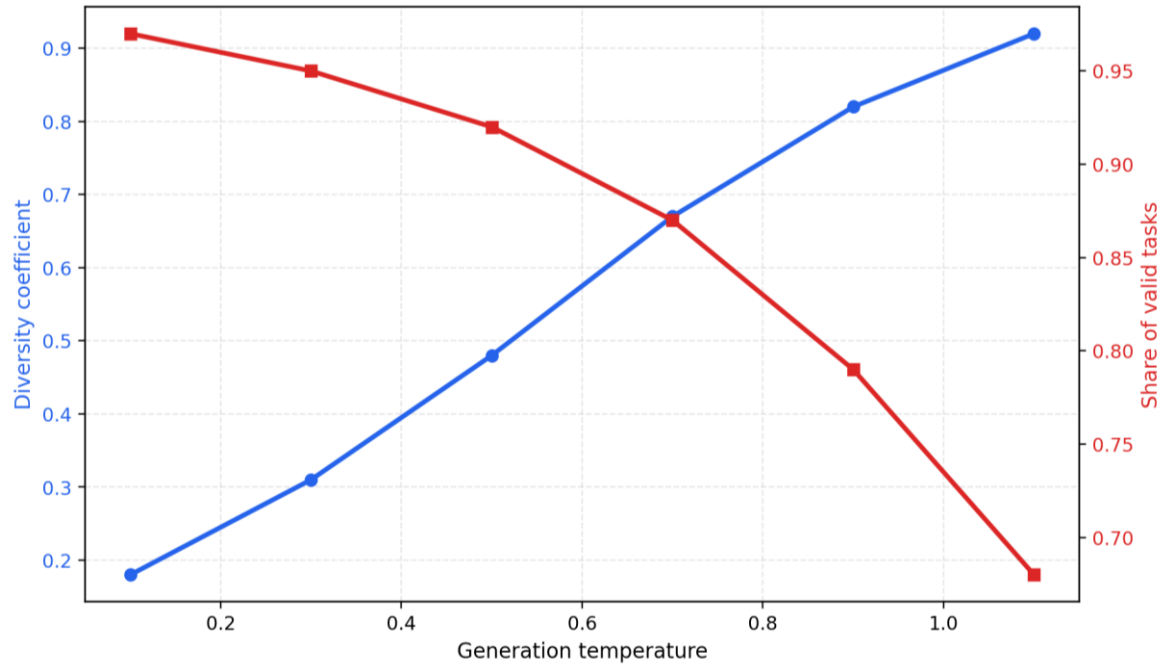

Figure 7 - Illustration of the principle of temperature influence on generation stability

No less important parameter is the target complexity of the task. The very concept of complexity was introduced as an additional factor affecting the flexibility of the final result, which made it possible to successfully select the required flexibility [10]. As it increases, the number of restrictions that must be taken into account when building a task increases, and the probability of rejected unsuccessful options also increases. This leads to an increase in generation time and increases the requirements for verification mechanisms, but as a result, the model successfully copes with the task. Although the complexity and significantly affects the cost of generation time, optimization processes have reduced the expected divergence as much as possible. The dependency graph is shown in Figure 8.

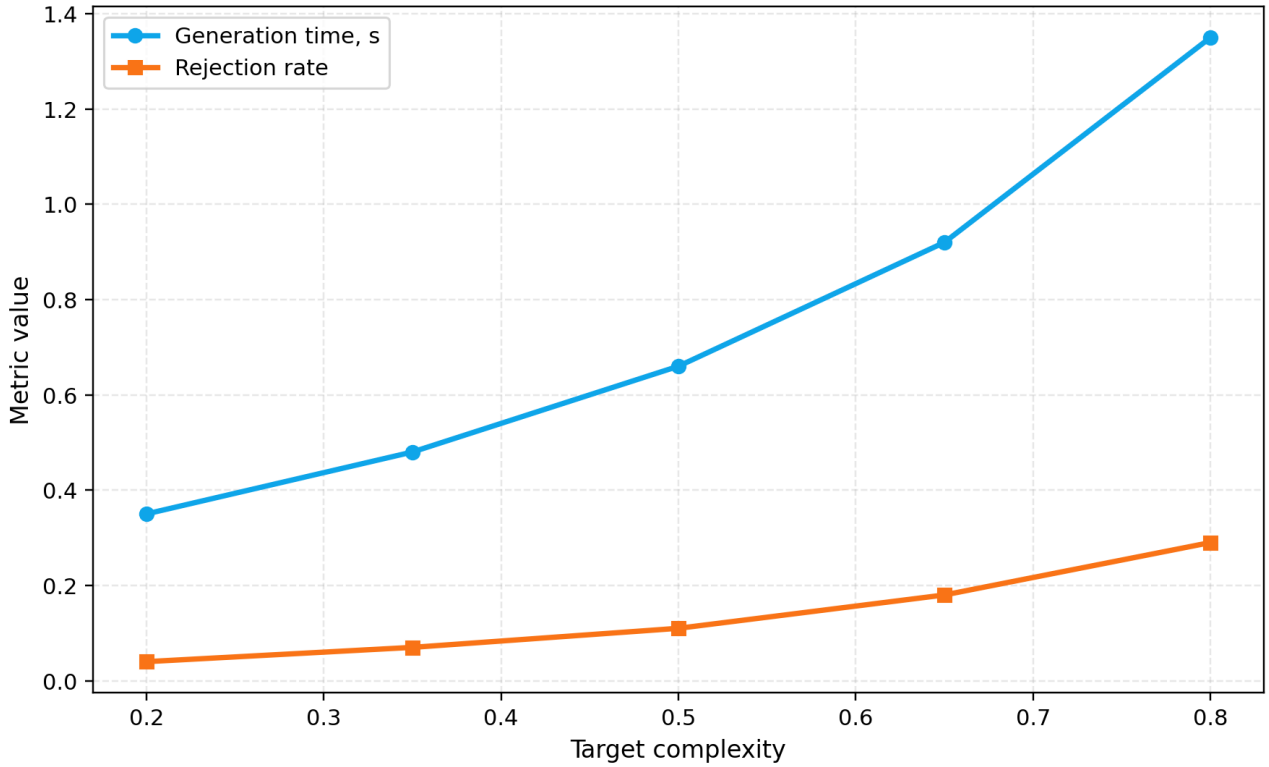

Figure 8 - Illustration of the principle of the impact of complexity on generation

During the development, of course a graphical implementation of the application functions was provided even though that's not the best authors solution in terms of UI, still it suits as a great interaction tool and provides with all the information needed. the entire process can be monitored through the user interface

created to perform all the required tasks without the need to access the console or code. For detailed debugging of the system, a logging system was implemented, which allows you to track all the logic of the passage, analysis results, errors and intermediate artifacts of generation, which makes the behavior of the system and its further support as transparent as possible.

A detailed representation is shown in Figures 9 and 10.

Figure 9 - Illustration of the graphical interface of the application

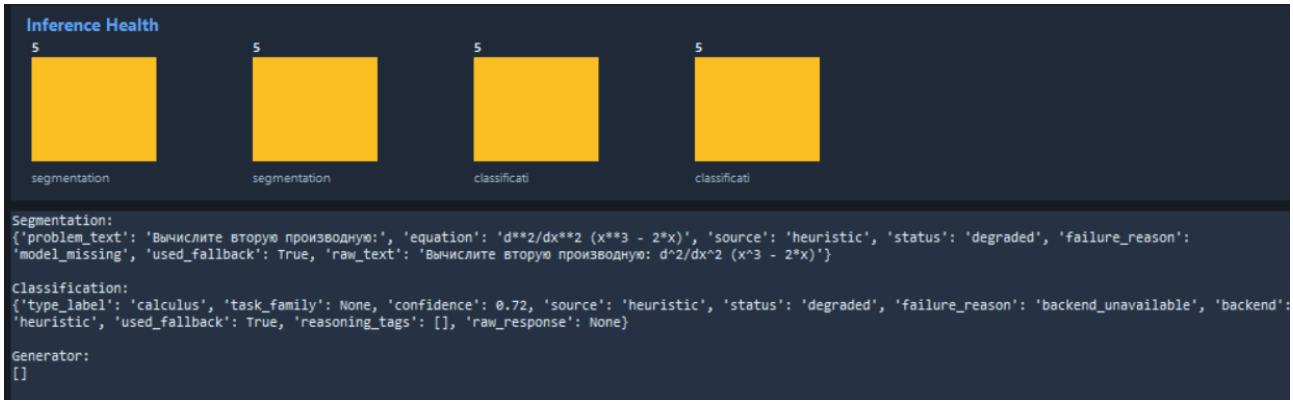

Figure 10 - Illustration of the error debugging system inside the application

The PS we've gone through in the work demonstrates that the automated generation of mathematical problems can be implemented as a full-fledged intellectual system that combines methods of symbolic analysis, machine learning, structural data representation and local context storage. Approach shown is especially relevant in the context of the development of digital education, where the need for scalable tools for the preparation of training and control materials is increasing, without the use of third-party tools.

The practical value of the developed solution lies in the possibility of local formation of problems in higher mathematics, taking into account the structure of the training sample, generation parameters, required complexity and available export formats. This creates the prerequisites for the use of the software not only within the experimental environment, but also as part of real educational platforms, electronic courses, distance learning systems and internal teaching tools for task preparation, as well as personal use. At the moment, the program has already been used for the successful preparation of several test courses in the Moodle training system.

The prospects for further development of the project are primarily related to the expansion of the number of supported topics, increasing the accuracy of classification and improving generation mechanisms. In the future, the system can be supplemented with a more developed adaptation to the subject area, improved algorithms for ranking close examples, mechanisms for automatic explanation of the results obtained, as well as closer integration with existing training services.

Of particular interest is the further application of the developed approach in functioning educational environments. In subsequent development, the system can be used as a tool for generating individual task options, preparing training kits, automated filling of LMS systems, as well as an auxiliary module in intelligent learning platforms. The presented solution can be considered not only as a research and training project, but also as a basis for the subsequent construction of applied EdTech systems.

References:

1. SymPy Development Team. SymPy Documentation [Electronic resource]. – Mode of access: https://docs.sympy.org/latest/index.html. – Date of access: 13.03.2026.

2. Python Software Foundation. AST — Abstract Syntax Trees. Python Documentation [Electronic resource]. – Mode of access: https://docs.python.org/3/library/ast.html. – Date of access: 10.03.2026.

3. McInnes, L., Healy, J., Astels, S. hdbscan: Hierarchical density based clustering // Journal of Open Source Software. – 2017. – Vol. 2, No. 11. – P. 205. – DOI: 10.21105/joss.00205.

4. Campello, R. J. G. B., Moulavi, D., Sander, J. Density-Based Clustering Based on Hierarchical Density Estimates // Advances in Knowledge Discovery and Data Mining. – 2013. – P. 160–172.

5. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., Kiela, D. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // NeurIPS. – 2020. – Mode of access: https://nlp.cs.ucl.ac.uk/publications/2020-05-retrieval-augmented-generation-for-knowledge-intensive-nlp-tasks/. – Date of access: 19.03.2026.

6. Ollama. Official Documentation [Electronic resource]. – Mode of access: https://docs.ollama.com/. – Date of access: 15.03.2026.

7. Ollama. API Reference / Introduction [Electronic resource]. – Mode of access: https://docs.ollama.com/api/introduction. – Date of access: 15.03.2026.

8. Scikit-learn developers. TfidfVectorizer — scikit-learn documentation [Electronic resource]. – Mode of access: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. – Date of access: 14.03.2026.

9. Bishop, C. M. Pattern Recognition and Machine Learning. – New York : Springer, 2006. – 738 p.

10. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. – Cambridge : MIT Press, 2016. – 785 p.