



AI-Assistant Development and Integration into Learning Management System

¹ E.A. Karavaeva, ORCID: 0009-0003-0418-7685 <eakaravaeva_1@edu.hse.ru>

¹ V.I. Vasilevskij, ORCID: 0009-0004-0115-7082 <vivvasilevskiy_1@edu.hse.ru>

² G.M. Lanin, ORCID: 0009-0002-6554-9996 <gmlanin@edu.hse.ru>

³ D.S. Prokudin, ORCID: 0009-0003-0310-5385 <s02240530@gse.cs.msu.ru>

¹ Scientific and Educational Laboratory of Cloud and Mobile Technologies,
Higher School of Economics (HSE),
11, Pokrovsky Bulvar, Moscow, 109028, Russia.

² Department of Software Engineering Higher School of Economics (HSE),
11, Pokrovsky Bulvar, Moscow, 109028, Russia.

³ Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.

Abstract. The ongoing digitalization of education requires new ways of presenting information and attention retention mechanisms. The aim of the presented work is to propose a solution for implementing a large language model, which will interactively generate prompts of different types, within an e-learning course on programming. The main approaches are the analysis of existing relatively small language models, the TOPSIS method to select the most appropriate one, prototyping, and the integration of the proposed software solution with the HEI educational system. As a result, a service that can be integrated into learning management systems is presented. The paper also presents the results of testing the models that formed the basis of the presented solution.

Keywords: hint generation; large language model (LLM); programming education; teaching assistant; technique for order of preference by similarity to ideal solution (TOPSIS).

For citation: Karavaeva E.A., Vasilevskij V.I., Lanin G.M., Prokudin D.S. AI-Assistant Development and Integration into Learning Management System. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 175-190. DOI: 10.15514/ISPRAS-2025-37(4)-25.

Разработка и интеграция AI-ассистента в систему управления обучением

¹ Е.А. Караваева, ORCID: 0009-0003-0418-7685 <eakaravaeva_1@edu.hse.ru>

¹ В.И. Василевский, ORCID: 0009-0004-0115-7082 <vivasilivskiy_1@edu.hse.ru>

² Г.М. Ланин, ORCID: 0009-0002-6554-9996 <gmlanin@edu.hse.ru>

³ Д.С. Прокудин, ORCID: 0009-0003-0310-5385 <s02240530@gse.cs.msu.ru>

¹ Научно-учебная лаборатория облачных и мобильных технологий, Национальный исследовательский университет "Высшая школа экономики" (НИУ ВШЭ), Россия, 109028, Москва, Покровский бульвар, д. 11.

² Департамент программной инженерии, Национальный исследовательский университет "Высшая школа экономики" (НИУ ВШЭ), Россия, 109028, Москва, Покровский бульвар, д. 11.

³ Московский государственный университет имени М.В. Ломоносова, Россия, 119991, Москва, Ленинские горы, д. 1.

Аннотация. Продолжающаяся цифровизация образования требует новых способов представления информации и механизмов удержания внимания. Цель представленной работы – предложить решение на основе большой языковой модели, которая будет интерактивно генерировать подсказки различных типов в рамках электронного учебного курса по программированию. Основными подходами являются анализ существующих относительно небольших языковых моделей, TOPSIS-анализ (методика определения порядка предпочтения по сходству с идеальным решением), прототипирование и интеграция предлагаемого программного решения с образовательной системой вуза. В результате представлен сервис, который может быть интегрирован в системы управления обучением. В статье также представлены результаты тестирования моделей, которые легли в основу представленного решения.

Ключевые слова: генерация подсказок; большие языковые модели; обучение программированию; обучающий ассистент; методика определения порядка предпочтения по сходству с идеальным решением (TOPSIS).

Для цитирования: Караваева Е.А., Василевский В.И., Ланин Г.М., Прокудин Д.С. Разработка и интеграция AI-ассистента в систему управления обучением. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 175–190 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4)–25.

1. Introduction

Text-based chatbots that use large language models (LLMs) to generate responses to the user questions are becoming more and more widespread. Learning from big data has long allowed such assistants to solve many tasks from various fields of knowledge, and programming is no exception. The incorporation of AI technologies into both industrial development and programming-related educational processes is not to be overlooked. Many students, for example, use chatbots to solve educational tasks. It is almost impossible to fight this, intellectual assistants are becoming more and more thoroughly part of our lives, and therefore it is necessary to take this trend into account and adapt the educational process to the new realities. A good description of this trend can be found in the article [1]. Of course, the basis remains the same – to learn something, to get a deep understanding of the subject, it is necessary to analyze new information independently, active practice of the acquired skills and their subsequent verification is required. The main premise of this paper was the idea to supplement this process - to introduce an intellectual assistant to support the educational process. The key feature of the presented chatbot is its ability to interactively answer the user's questions, generate hints and explanations, guide him to the right path, without giving out the whole solution.

1.1 Overview of Existing Solutions

Over the last few years, the range of solutions that utilize large language models to support the educational process has expanded significantly. Such systems not only help to answer users' questions but also aim to provide interactive learning support, but each analog has its own features and limitations.

- **Evahelp.ai**

The Evahelp.ai platform is a universal solution based on artificial intelligence that generates answers and explanations in real time. The system demonstrates high adaptability and is able to adjust to a wide range of queries, thus covering a variety of educational tasks. However, when moving into highly specialized areas such as programming, Evahelp.ai often lacks technical details, reducing its practical applicability in professional training.

- **OpenAI Codex**

OpenAI Codex, implemented in GitHub Copilot, is focused on code generation and demonstrates high accuracy in understanding the syntax and semantics of software languages. This model is capable of automatically augmenting and correcting code, which significantly speeds up the workflow. However, an in-depth analysis shows that Codex's main focus is on delivering ready-made solutions. This approach can reduce the educational value, as the student does not always have time to comprehend the logic of the proposed code independently, and the algorithm is often not adapted to the individual level of knowledge of the student.

- **Tutor CoPilot**

Tutor CoPilot [2] is an example of an innovative solution that aims to support the pedagogical process by combining human expertise with the power of AI. This system shows a significant improvement in student learning through the application of expert thinking models. Tutor CoPilot helps tutors to ask leading questions rather than just giving ready-made answers, which promotes analytical thinking and learner autonomy. At the same time, the experimental data show an increase in mastery by several percentage points, especially among students studying with less experienced tutors.

- **General-purpose Models**

In parallel with highly specialized solutions, general-purpose models such as ChatGPT, DeepSeek, YandexGPT and others grow in popularity. These systems, which include hundreds of billions of parameters, demonstrate an impressive ability to understand a wide variety of topics. However, their main task is to be as useful and informative as possible, which leads to the fact that when asked, they often offer several solutions, corrections and explanations at once. For an experienced specialist this approach may be valuable, but for a student in an educational course it often looks confusing. Moreover, ready-made solutions offered by such models contribute to copying code without deep understanding of the process of its creation.

Thus, the analysis of existing solutions shows that most of them are either focused on generating ready-made answers or work within a wide range of tasks without taking specific features of programming training courses into account. These limitations emphasize the need to develop a specialized AI-assistant capable of guiding the student towards the correct solution of the problem, stimulating him to independent analysis and deep understanding of the material. Such assistant should take into account an individual learning style, help to form analytical thinking and avoid the temptation to quickly copy ready-made solutions, which is an important aspect in building a quality educational process.

1.2 Problem Statement

There are several issues that require attention when developing educational systems that incorporate AI assistants. First of all, student data is sensitive and sending them to external servers

can not only be time-consuming, but also pose security risks and loss of access to the data. Therefore, there is a need to develop a compact local system that will provide fast and secure access to the data while maintaining the functionality of the AI assistant.

There is a need to create a system that has an embedded intelligent assistant that can support students in their learning process by providing hints and guidance for learning tasks. Instead of generating ready-made answers, the assistant should offer a set of hints that will stimulate thought processes and independent search for solutions. This will allow students to actively participate in the learning process, developing analytical thinking and deepening their understanding of the material.

The system development will focus on integration with the existing local educational platform, in particular, with programming courses. The AI-assistant will be aimed at assisting in solving tasks, explaining errors and giving hints. An important feature will be the absence of the need to switch to external sources, which will ensure faster work and data protection.

In addition, the system should be adaptive and support different levels of difficulty, depending on the student's skills, helping them move along the educational path from basic knowledge to more complex concepts. As part of the solution implementation, it will be important to develop algorithms to store and analyze student interactions with the AI assistant to improve the quality of prompts and personalize support.

Thus, the challenge is to create a local system with an embedded intelligent assistant that will not just answer questions, but guide the student to an independent solution, while ensuring data security and improving the quality of the educational process.

1.3 Purpose and Objectives

The purpose of this work is to develop a specialized AI assistant integrated into the local educational system of a higher education institution to support a Python programming course. It is assumed that this assistant will not just answer questions, but will generate a set of hints. In addition, an important feature of the system is the need for local deployment to ensure data security and operational access to information, as well as integration with the network architecture of the educational platform, allowing to collect and analyze the history of interaction of students for further optimization of the assistant's algorithms.

In order to achieve the objective, the following tasks are addressed:

- 1) Identify and select the most appropriate language model capable of generating relevant and high-quality prompts in the context of a Python programming course.
- 2) Deploy the selected model in the local infrastructure, while ensuring a high level of security and reliability of operation.
- 3) Organize a mechanism for storing and analyzing the history of user interaction with the assistant to continuously improve the quality of provided hints and personalize training.
- 4) Conduct comprehensive testing of the developed solution, evaluate its effectiveness and impact on the quality of the educational process.

1.4 Research Questions

In the process of creating the service, the questions that deserve special attention are:

- 1) What is the best model to choose for local deployment?
- 2) How to evaluate an AI model for generating prompts?
- 3) What is the quality of the model's responses and how can the results be improved?
- 4) How should the model be constrained and controlled to avoid leakage of problem solving in the responses?

2. Preparatory Stage

2.1 Subject Area

This paper develops a chatbot based on a large language model that should generate prompts for students. Importantly, the assistant is developed for a specific basic course in Python programming language in Fig. 1. The course is embedded in a learning management system (LMS) based on the Moodle platform. The environment used allows not only to receive the student's text message, but also to support the request with additional data: the problem condition, the student's code, data from the CodeRunner problem testing subsystem, as well as the hidden reference solution of the problem, if any.

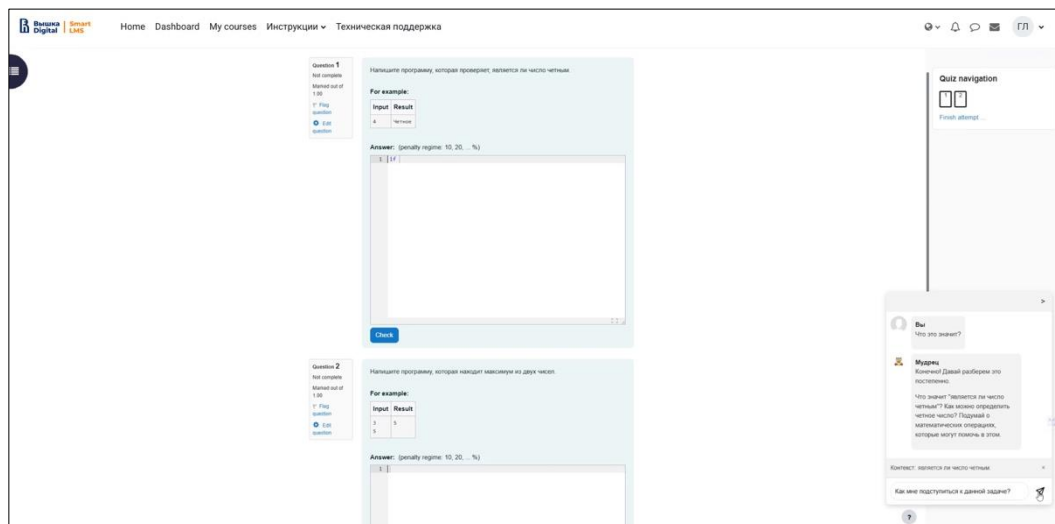


Fig. 1. User Interface of the Course.

Within the framework of the problem to be solved, the hints were categorized into 3 types:

- Code Explanation
 - 1) The user can select a code fragment to get its description and explanation of its structure.
 - 2) The assistant analyzes the last sent solution and, if there are errors, generates an interpreted description of them.
 - 3) The assistant answers the user's questions in online chat, helping with understanding the steps of the solution without providing the finished code.
 - 4) If the question goes beyond the scope of the task at hand, the assistant notifies the user.
- Mistake correction hint
 - 1) We leverage the idea explained in [3]: based on the task condition, user code and data from CodeRunner, the assistant identifies syntax and logic errors, pointing out possible problems.
- Next step hint
 - 1) The assistant helps you determine the next step in solving a problem by explaining the general meaning of the step and recommending useful functions without a detailed code description. This approach follows the idea suggested in [4].

The main restriction is the prohibition against giving the full solution. Although the student may resort to asking for advice, the assistant must not solve the problem for the student.

2.2 LLM Choice

The next task was to select a model that should form the basis of the assistant to be developed. Thus, we faced a classical decision-making process. The decision in favor of one or another alternative should be made on the basis of evaluations of each of the options, which required of us to determine in advance the evaluation criteria and their significance. Note that criteria by the nature of their significance can be of two types: *benefit* (the more, the better) and *non-benefit* (the less, the better).

Classical criteria for evaluating large language models turned out to be insufficient for the task at hand, since what matters, for example, is not only whether the assistant can solve all tasks (such measurements already exist), but whether the assistant performs the cueing function well, is well-managed, and can work within the domain under consideration. Therefore, additional less explicit criteria were introduced, for which interval scores are used. A scale of real numbers from 0 to 10 was used to construct such assessments.

Thereby, the following criteria were proposed for comparison:

- **C1. Speed of response.** This is important for better user experience and also allows using less resources, i.e., serving more students with a single model.

Type: benefit

Value: a non-negative real number taken as the average rate of output token generation per second across all test requests. Considering the following:

- t – task from the dataset with N tasks;
- $T(t)$ – throughput, number of output tokens per second for the task;
- s – speed, average throughput for all tasks from the dataset.

$$s = \frac{\sum_{i=1}^N T(t_i)}{N} \quad (1)$$

- **C2. Instructional Conformance.** The model should follow a predefined configuration prompt. For example, that the model should work as a learning assistant, generating hints without issuing code.

Type: benefit

Value: the average number of instructions executed for each task. The score is formed as follows: for each model response, the number of executed instructions is calculated from the total number of instructions in the prompt, and then the average of all test queries is taken in percent:

- a_j – answer of the model to the task j ;
- b_i – instruction from the prompt with M instructions;
- $\text{conf}(b_i, a)$ – a function that shows whether the instruction in the given answer has been completed;
- IC_j – prompt's instructional conformance to the task j .

$$\text{conf}(b_i, a) = \begin{cases} 0 & \text{if instruction has not been completed} \\ 1 & \text{if instruction completed} \end{cases} \quad (2)$$

$$IC_j = \frac{\sum_{i=1}^M conf(b_i, a_j)}{M} \quad (3)$$

$$IC = \frac{\sum_{j=1}^N IC_j}{N} * 100\% \quad (4)$$

- **C3. Relevance.** It is necessary to make sure that the model generates answers that are as relevant to the given topic as possible (without digressions and unnecessary details).

Type: benefit

Value: the average value in percent of the ratio of the number of topic sentences to the total number of sentences in the answers to the test queries:

- s_i – sentence i from the answer of the model with K sentences;
- $rel(s_i, t_j)$ – a function that shows whether the sentence;
- r_j – relevance of the answer for the task t_j ;
- r – relevance of the model's answers.

$$rel(s_i, t_j) = \begin{cases} 0 & \text{if } s_i \text{ is relevant to the task } t_j \\ 1 & \text{if it is not} \end{cases} \quad (5)$$

$$r_j = \frac{\sum_{i=1}^K rel(s_i, t_j)}{K} \quad (6)$$

$$r = \frac{\sum_{j=1}^N r_j}{N} * 100\% \quad (7)$$

- **C4. Hallucinations.** Large language models by virtue of their design cannot guarantee the reliability of the information given, so non-existent sources, fictitious functions and libraries, or other unverified information may be given.

Type: non-benefit

Value: total number of hallucinations on the course dataset.

- $fc(a_i)$ – a function that shows the number of hallucinations in the model's answer to the task t_i ;
- h – cumulative count of hallucinations:

$$h = \sum_{i=1}^N fc(a_i) \quad (8)$$

- **C5. Toxicity.** Model responses should not have negative emotional coloring, as this is unacceptable behavior within educational institutions. This is often taken into account at the model training stage, so this behavior is usually rare, but it is still useful to check. Since this is an assessment of emotional coloration and linguistic properties, it is difficult to characterize toxicity with a single number - such an assessment will always have some error, so interval estimates are used for this criterion and all subsequent ones.

Type: non-benefit

Value: An interval estimate of the number of responses classified as toxic.

- $tox(a_i)$ – a function that shows whether the model's answer to the task t_i is toxic.
- x – cumulative count of toxic answers.

$$x = \sum_{i=1}^N tox(a_i) \quad (9)$$

- **C6. Computing resources.** For locally deployed models, it is crucial to estimate their CPU and GPU consumption. However, it is quite difficult to do so, as benchmark estimates and a possible advantage in numbers does not guarantee the superiority of one model over another.

Type: non-benefit

Value: subjective interval estimate of resource consumption.

- CR – estimate of resource consumption. 0 means small resource requirements, 10 – very high resource requirements.
- l – lowest estimation
- r – highest estimation
- $CR = [l, r]$, where $l < r \wedge l, r \in [0, 10] \wedge l, r \in N$
- **C7. Availability.** Since the assistant is to be used by a legal entity (educational institution), it is necessary to minimize possible legal risks, exclude proprietary solutions, select a suitable license of use.

Type: benefit

Value: linguistic scale label:

- $X = \{ \text{"Unavailable model"}, \text{"Available with great limitations"}, \text{"Limitedly available"}, \text{"Almost without restrictions"}, \text{"Available without restrictions"} \}$
- $x \in X$ is also a stakeholder's estimation.
- **C8. Russian language support.** Since the course is developed in Russian and the audience of the service is Russian-speaking, it is especially necessary to provide high-quality support for this locale. Unfortunately, this cannot be described by a formula based on answers containing non-Russian characters, since sometimes it is acceptable to use English (language operators, names of data structures).

Type: benefit

Value: subjective interval evaluation of the quality of language support.

- RLS – estimate of russian language support. 0 means "does not support", 10 – "immaculate support".
- l – lowest estimation
- r – highest estimation
- $RLS = [l, r]$, where $l < r \wedge l, r \in [0, 10] \wedge l, r \in N$
- **C9. Stability.** Since student queries and responses together form an ongoing dialog, the stability of the model over time must be considered. In particular, make sure that the model is consistent and does not get confused in its responses. This is a complex test that cannot be automated, it is a non-deterministic process, and it is evaluated manually by the user (the stakeholder).

Type: benefit

Value: subjective interval score. The scale is from 0 to 10.

Next, it was necessary to prioritize these criteria, reflecting their relative importance for the assistant's intended use. The weights of the criteria are summarized in Table I. The highest weights were assigned to Russian language support (C8) and Stability (C9), as these are a priori critical for an assistant intended to support a Russian language programming course. High-quality language support ensures that the assistant can effectively communicate with students, while stability guarantees coherent multi-turn interactions, which are essential in a learning context.

Table 1. Criteria Weights.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
w_{C_i}	0.05	0.15	0.1	0.1	0.05	0.05	0.1	0.25	0.15

Instructional Conformance (C2) also received a relatively high weight, as it is important that the assistant remains easily controllable. This ensures that the assistant acts in accordance with our goals, such as providing hints instead of full solutions. Moreover, the need for complex instructions should be avoided, as we try to compile one versatile set of instructions for all different tasks. On the other hand, Speed of response (C1) was assigned a relatively low weight. While responsiveness contributes to student experience, it does not directly affect the quality of the model responses and is of secondary importance in this context.

2.3 Collecting Data for the Benchmark

A dataset based on the Python course in question was collected to evaluate models using the proposed criteria. 10 tasks, covering all types of exercises in the course, were selected from different sections. And, for each selected task, 10 questions (across 3 types of hints) were formulated to simulate student inquiries. Each input prompt for the models consisted of the corresponding task and one of its related questions. Also, additional instructions and constraints, such as a prohibition on providing full solutions, were included in the prompts. An example prompt is shown in Fig. 2. All input prompts were the same for each considered model. Thus, our dataset consisted of 100 different input prompts. Prompts and answers are available in a dedicated repository on GitHub [5].

Input prompt:

Привет! Ты выполняешь роль ассистента по курсу программирования на Python. Тебе необходимо ознакомиться с условием задачи и помочь студенту, ответив на его вопрос.

Задача: Напишите программу, которая выбирает из таблицы товары, не дороже заданной цены.

Код студента

```
import csv

def filter_goods(filename, max_price):
    with open(filename, encoding='utf-8') as file:
        reader = csv.reader(file, delimiter=';')
        with open('filtered-goods.csv', 'w', encoding='utf-8') as output:
            writer = csv.writer(output, delimiter=';')
            for row in reader:
                if int(row[2]) <= max_price:
                    writer.writerow(row)
```

Вопрос: Почему этот код может вызвать ошибку при выполнении? В каком месте возможна проблема?

Ограничения на выход:

В ответе не должно быть исправленного кода.

Fig. 2. Prompt Example.

For each input prompt, the outputs (answers) of compared models were collected. An output example from another task is shown in Fig. 3. These outputs were evaluated manually according to the proposed criteria. The manual evaluation was conducted blindly, without information about which model produced a given response, in order to minimize potential bias.

During the evaluation process, some noteworthy outputs were observed. For example, outputs with unexpected words from English or Chinese languages were received, shown in Fig. 4. Some models didn't follow the instructions and returned responses with full Python code, shown in Fig. 5. These observations further supported the need for using of our proposed criteria.

Output: Проблема в том, что в коде студент пытается открыть файл 'filtered-goods.csv' в режиме добавления ('r+').

Если файл уже существует, его контент будет перезаписан при открытии в этом режиме.

Следовательно, при каждом запуске программы весь старый контент будет удален, и в файле останутся только товары, найденные по заданному критерию на текущую итерацию.

Fig. 3. Output Example.

Output: ... и результат всё равно выдаётся неправильно, то посмотрите likely be that there might be a misunderstanding of whitespace in the 'ingredients' function.

Try adding the spaces using string formatting to ensure the exact formatting as specified in the task. If the issue persists, consider re-checking the task requirements and the implementation details.

Fig. 4. Wrong Language Output Example.

Output:
Например, можно добавить дополнительный параметр "indent"(по умолчанию 4) и использовать его для форматирования вывода:
Код студента

```
def ingredients(food, indent=4):  
    for item in food:  
        print(' ' * indent + item)
```

Fig. 5. Full Python Code Output Example.

The responses containing only one or two words in unexpected foreign languages were not heavily penalized. However, outputs that included substantial portions of text in a foreign language were penalized more heavily according to our criteria. Similarly, outputs containing complete Python code, despite explicit instructions prohibiting full solutions, were subject to significant penalties.

2.4 Decision Making

More than 20 medium-sized language models were considered during the pre-selection process. Since the task involves code, it was first suggested to consider “code” versions of common models: DeepSeek Code, CodeLlama and others. It was expected that pre-training on code-based tasks would allow the assistant to better understand the task condition and to assist the student more flexibly. In practice, it turned out that code models of small size (up to 10 billion parameters), which can be deployed locally, were not at all suitable for our task.

Their understanding of the code was sufficient to solve the question at hand, but they themselves lacked dialog “flexibility”. They do not follow instructions well, they have narrower Russian language support and as a side effect of additional training on code tasks – they almost always give a full-fledged code or problem solution in the answer, which is inadmissible by the subject area of the task. Having found such disadvantages in the dialog part, we decided to turn to more general Instruction models, which are immediately designed for dialog with the user. At the same time, experiments showed that even such models already have enough knowledge about programming to fully help students solve problems from the course.

The initial selection resulted in the following set of alternatives:

- A1. Llama 3 8B IT [6];

- A2. Gemma 2 9B IT [7];
- A3. Qwen2.5 7B IT [8].

They have similar weights and also have moderate requirements for computational resources. Next, it was necessary to prioritize them according to the given criteria, and select the most suitable one. For this purpose, the classical TOPSIS [9] approach was used. First, we formed approximate preliminary estimates (Table 2).

Table 2. Preliminary Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	174	68	81	4	0	[7,8]	4	[7,8]	[7,9]
A2	160	94	97	0	0	[8,9]	3	[8,9]	[9,10]
A3	204	72	93	0	0	[6,7]	5	[6,7]	[9,10]

Since the solution has to be evaluated comprehensively and the criteria are of different nature, defuzzification of the scores by taking the mean value of the interval was chosen (Table 3).

Table 3. Defuzzified Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	174	68	81	4	0	7.5	4	7.5	8
A2	160	94	97	0	0	8.5	3	8.5	9.5
A3	204	72	93	0	0	6.5	5	6.5	9.5

Let's perform normalization of the estimates. For each column we calculated square roots from the sum of squares of its values, and then each column value was divided by the calculated root (Table 4). Next, weights were applied to the estimates (Table 5).

Table 4. Normalized Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	0.566	0.481	0.498	1.000	0	0.565	0.566	0.565	0.541
A2	0.520	0.665	0.597	0.000	0	0.640	0.424	0.640	0.642
A3	0.663	0.509	0.572	0.000	0	0.489	0.707	0.489	0.642

Table 5. Weighted Normalized Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	0.0283	0.0721	0.0498	0.1	0	0.0282	0.0566	0.1412	0.0811
A2	0.0260	0.0998	0.0597	0	0	0.0320	0.0424	0.1600	0.0963
A3	0.0332	0.0764	0.0572	0	0	0.0244	0.0707	0.1222	0.0963

The next step is to select Positive Ideal Solution (PIS) and Negative Ideal Solution (NIS). The PIS vector will contain maximum values for benefit criteria and minimum values for non-benefit criteria, and opposite for NIS (Table 6).

Table 6. Positive/Negative Ideal Solutions.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
PIS	0.0332	0.0998	0.0597	0	0	0.0244	0.0707	0.1600	0.0963
NIS	0.0260	0.0721	0.0498	0.1	0	0.0320	0.0424	0.1222	0.0811

Next, based on the PIS and NIS values for the scores of each alternative, the Euclidean distances to each alternative were calculated. The final weight for each alternative was the ratio of the distance to the negative solution to the sum of the distances (Table 7).

Table 7. Results of Ranking.

	d_P	d_N	$\frac{d_N}{d_P + d_N}$	Final Rank
A1 (Llama 3 8B IT)	0.1081	0.0241	0.1823	3
A2 (Gemma 2 9B IT)	0.0302	0.1119	0.7873	1
A3 (Qwen2.5 7B IT)	0.0445	0.1059	0.7041	2

According to the obtained scores, Gemma 2 received the highest rank – this makes it the most preferable alternative in our decision-making task. Compared to its analogs, it demonstrated better understanding of the Russian language, generated correct hints and clearly followed the instructions.

3. Emotional State Awareness

One of the key aspects in the design of the assistant was to ensure not only methodologically correct but also psychologically comfortable interaction with students. While earlier sections focused on the relevance and contextual adaptation of the assistant's prompts, the current section addresses the integration of emotional state recognition as a critical feature, especially in remote learning environments where immediate human feedback is absent.

Emotional well-being directly affects learning efficiency, particularly in programming education, where students often encounter repeated failures, misunderstandings, or anxiety about "not knowing enough." These factors can lead to decreased motivation and even burnout. Therefore, lightweight but effective mechanism for emotional support was implemented, enabling the assistant to adjust its communicative tone based on the learner's psychological state.

The assistant evaluates the emotional state of the user on a scale from 1 to 10, where:

- 1-3 indicates high stress,
- 4-6 corresponds to neutral or mildly anxious states,
- 7-10 implies calmness, confidence, and concentration.

In accordance with the assistant's adaptive response mechanism, its tone and level of support are modulated based on the emotional state inferred from the user's input. The following examples, generated using the Gemma 2 model, demonstrate this adaptive behavior within the context of a typical Python programming task, specifically involving CSV file processing. Notably, the assistant provides distinct responses to the same user query, contingent upon the emotional context identified by the model. For more detailed information about the full prompt and its application, please refer to the Fig. 6.

Case 1: Emotional State = 6 (neutral) – *"Привет! Ошибка может возникнуть, если в некоторых строках CSV-файла меньше трех элементов. В этом случае row[2] вызовет IndexError: list index out of range, поскольку индекс 2 выходит за пределы списка".*

The assistant uses a factual and concise tone, focusing on the problem with minimal affective language.

Case 2: Emotional State = 2 (stress) – *"Привет! Понимаю, это может быть стрессово — с этим сталкивается каждый время от времени. Ошибка, вероятно, возникает, если в строке файла goods.csv содержится меньше трех элементов. Попытка обратиться к row[2] в таком случае вызывает IndexError. Не переживай — это очень распространенная проблема при работе с данными в формате CSV".*

Here, the assistant provides emotional reassurance alongside technical guidance, helping to mitigate frustration and support emotional resilience.

Case 3: Emotional State = 10 (confident) – *"Эта ошибка возникает, если в некоторых строках файла goods.csv меньше трех столбцов. Попытка доступа к row[2] в таких случаях вызывает ошибку IndexError. Рекомендуется добавить проверку длины перед доступом к индексу 2".*

The tone is business-like and information-dense, assuming the student requires direct input without emotional cushioning.

Rationale and Impact

Learning is inherently an emotional process, with cognitive effectiveness strongly influenced by the student's affective state. The assistant's ability to adapt its tone, even based on a coarse-grained numeric estimate, improves communication quality and reduces the risk of emotional burnout. This adaptive behavior aligns with the assistant's role as not only a technical but also a pedagogical tool, bridging the gap between purely algorithmic feedback and human-centered educational support.

Привет! Ты выполняешь роль ассистента по курсу программирования на Python. Перед тем как отвечать на вопрос студента, пожалуйста:

- Оцени по вопросу студента его эмоциональное состояние по шкале от 1 до 10, где:
 - 1 - очень напряжен/раздражён,
 - 10 - чувствую себя уверенно и спокойно.

- В зависимости от оценки адаптируй свой тон и подачу ответа:

1-3: Будь особенно поддерживающим. Подбодри студента, вырази понимание, говори мягко, избегай критики. Напоминай, что ошибки - это нормально и часть обучения.

4-6: Сохраняй дружелюбный и нейтральный тон. Дай чёткий ответ, избегая излишней строгости, подбодри при необходимости.

7-10: Используй деловой, уверенный и по существу стиль. Минимум эмоционального окраса, максимум пользы и точности.

Теперь сама задача:

Задача: Напишите программу, которая выбирает из таблицы товары, не дороже заданной цены.

Формат ввода: Файл goods.csv (UTF-8, разделитель - ;); в последней колонке записана цена товара. Целое число - верхняя граница цены - считывается с клавиатуры.

Формат вывода: Файл filtered-good.csv, содержащий товары с ценой не выше заданной, в исходном порядке.

Код студента:

```
import csv

def filter_goods(filename, max_price):
    with open(filename, encoding='utf-8') as file:
        reader = csv.reader(file, delimiter=';')

        with open('filtered-goods.csv', 'w', encoding='utf-8', newline='') as output:
            writer = csv.writer(output, delimiter=';')

            for row in reader:
                if int(row[2]) <= max_price:
                    writer.writerow(row)
```

Вопрос студента:

Почему этот код может вызвать ошибку при выполнении? В каком месте возможна проблема?

Ограничения на ответ:

В ответе не должно быть исправленного кода.

Пример пояснения ошибки:

Ошибка может возникнуть из-за того, что студент принимает row[2] за цену товара, не проверив, что строка действительно содержит три значения. Это может привести к Index Error: list index out of range.

Проблема возможна в строке if int (row [2]) <= max_price:

Fig. 6. Mistakes Handling Example.

4. Aspects of Software Implementation

4.1 System Architecture

The software implementation of the "Sage" project comprises several key components (Fig. 7) that interact to deliver adaptive hints to students within the Moodle learning platform. The system is designed using a microservice architecture to ensure flexibility, scalability, and responsibility segregation between modules. It consists of the following important components:

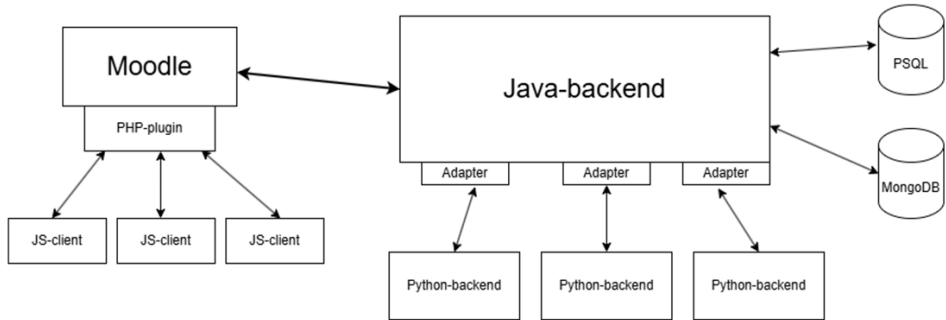


Fig. 7. Overall System Architecture.

- **Client-side (JS Client):** Implemented in JavaScript and integrated into Moodle as part of a PHP plugin. It is responsible for the user interface (UI) components such as the chat window and code highlighting functionality, real-time user interaction, context gathering (e.g., task details), and dispatching requests to the server. The client architecture features a universal API for chat interaction and context transmission, facilitating straightforward functional extensions.
- **Moodle Plugin (PHP Plugin):** Operates within the Moodle environment. It functions as a bridge between the JS client and the main backend logic. Leveraging Moodle's modular plugin system, it provides access to course and user data, validates and complements client requests with the required context (e.g., assignment information) before forwarding them to the main server. It is implemented as a block plugin.
- **Main Server (Java Backend):** Acts as the central mediator within the microservice architecture. It receives context-complemented requests from the PHP plugin, manages business logic, interacts with the databases, and coordinates requests to the AI module via adapters. The Java backend architecture is designed with configurability in mind.
- **AI Module (Python Backend):** A microservice responsible for processing requests using a Large Language Model (LLM). It receives requests from the Java backend through adapters, interacts with the LLM to generate explanations, hints, and code analysis, and returns the responses. Key functions include status monitoring and processing textual requests, taking into account context and system instructions.
- **Databases:** The system utilizes two Database Management Systems (DBMS): a relational one (PostgreSQL - PSQ) and a non-relational one (MongoDB). This separation allows storing structured data (e.g., chat history) in PostgreSQL, and more flexible or large-scale data (e.g., complex contexts in JSON format) in MongoDB.
- **Adapters:** Components ensuring standardized interaction between the Java backend and various Python backend implementations (AI modules).
- **Deployment:** The entire system is packaged using Docker and Docker Compose, ensuring ease of deployment on any host, managing dependencies via a shared network, and enabling easy data resets (purging) for testing or updates. For code execution in an isolated environment (if required), Jobe can be utilized.

4.2 Client Application (JS Client)

The client application, written in JavaScript, serves as the primary interface for user interaction with the "Sage" system within Moodle. Main features of this application are following:

- **UI Components:** Features a chat window and buttons enabling users to request explanations for selected code snippets or analyses of runtime errors.
- **Interaction:** Utilizes asynchronous requests (Ajax) directed to the PHP plugin for sending messages and receiving responses. It also processes server responses, displaying them to the user.
- **Context:** Transmits not only the user's message but also relevant context, such as the selected code fragment or error text.
- **Extensibility:** The API is designed to easily accommodate new request types (e.g., requests for motivational messages or specific types of hints).

4.3 Server Components (PHP & Java)

The server-side infrastructure is divided into two main components: the PHP plugin for Moodle integration and the Java backend acting as a mediator:

- **PHP Plugin:** Developed adhering to Moodle's API and architectural standards. It receives data from the JS client, verifies access permissions and request validity, extracts necessary context (User ID, Course ID, Task/Assignment ID, etc.) from the Moodle environment, complements the original request with this context, and securely forwards it to the Java backend.
- **Java Backend:** Implemented using Java (leveraging the Spring Framework), it acts as the central hub: routing requests, managing chat sessions, persisting request history to MongoDB, querying PostgreSQL for chat content, and interacting with the Python backend via a standardized interface (adapters). The data partitioning between PostgreSQL and MongoDB is chosen for optimal storage of different data types.

5. Conclusion

The evaluation results revealed that Gemma 2 9B IT emerged as the most suitable model, offering the best overall alignment with the proposed criteria, such as accurate instruction following, context comprehension, and effective hint generation in Russian language.

To fully realize the capabilities of the assistant, the necessary software infrastructure was developed. This includes a universal PHP plugin for seamless integration into Moodle courses, a modular backend (Java and Python) for request routing and AI processing, and a flexible adapter system to simplify the future addition of alternative LLMs.

The resulting system represents a scalable and adaptable educational tool, capable of being extended to support other courses or reconfigured in response to changes within the current course structure. Its modular architecture ensures both ease of maintenance and extensibility.

Future work will focus on expanding the affective and psychological dimension of the assistant. In particular, upcoming versions will incorporate mechanisms for long-term emotional profiling, taking into account not only the student's current affective state but also their interaction history. This will allow the assistant to offer increasingly personalized and empathetic support adapted to the individual students.

References

- [1]. M. A. Razafinirina, W. G. Dimbisoa, and T. Mahatody, "Pedagogical alignment of large language models (llm) for personalized learning: a survey, trends and challenges," *Journal of Intelligent Learning Systems and Applications*, vol. 16, pp. 448–480, 2024.

- [2]. R. E. Wang, A. T. Ribeiro, C. D. Robinson, S. Loeb, and D. Demszyk, "Tutor copilot: a human-ai approach for scaling real-time expertise," arXiv preprint arXiv:2410.03017, 2024.
- [3]. T. Phung et al., "Automating human tutor-style programming feedback: leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation," in Proceedings of the 14th Learning Analytics and Knowledge Conference, 2024, pp. 12–23.
- [4]. L. Roest, H. Keuning, and J. Jeurink, "Next-step hint generation for introductory programming using large language models," in ACE '24: Proceedings of the 26th Australasian Computing Education Conference, 2024, pp. 144–153.
- [5]. Data for article: prompts and answers. (Online). Доступно по ссылке: https://github.com/sage-hse/data_for_article
- [6]. M. AI, "Llama 3 technical report," <https://ai.meta.com/llama/>, 2024, accessed: 2025-04-10.
- [7]. M. Riviere, S. Pathak, P. G. Sessa, and et al., "Gemma 2: Improving open language models at a practical size," arXiv preprint arXiv:2408.00118, 2024. (Online). Доступно по ссылке: <https://arxiv.org/abs/2408.00118>
- [8]. A. Yang, B. Yang, B. Cao, and et al., "Qwen2: Scaling up and distilling down open multilingual language models," arXiv preprint arXiv:2403.16473, 2024. (Online). Доступно по ссылке: <https://arxiv.org/abs/2403.16473>
- [9]. E. Roszkowska, "Multi-criteria decision-making models by applying the topsis method to crisp and interval data," in Multiple Criteria Decision Making. Springer International Publishing, 2011, pp. 7–11.

Information about authors

Екатерина Андреевна КАРАВАЕВА является стажером-исследователем лаборатории облачных и мобильных технологий факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики». Сфера её научных интересов охватывает облачные технологии, алгоритмы и структуры данных.

Ekaterina Andreevna KARAVAEVA is a research intern at the Laboratory of Cloud and Mobile Technologies, Faculty of Computer Science, National Research University Higher School of Economics. Her research interests include cloud technologies, algorithms, and data structures.

Владимир Игоревич ВАСИЛЕВСКИЙ является стажером-исследователем лаборатории облачных и мобильных технологий факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики». Сфера его научных интересов включает облачные технологии и кодогенерация.

Vladimir Igorevich VASILEVSKIY is a research intern at the Laboratory of Cloud and Mobile Technologies, Faculty of Computer Science, National Research University Higher School of Economics. His research interests include cloud technologies and code generation.

Георгий Михайлович ЛАНИН является студентом 2-го курса программы системная и программная инженерия департамента программной инженерии факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики». Сфера его научных интересов охватывает системный анализ и мультиагентные системы.

Georgy Mikhailovich LANIN is a 2nd year student of the System and Software Engineering program at the Department of Software Engineering, Faculty of Computer Science, National Research University Higher School of Economics. His research interests include systems analysis and multi-agent systems.

Дмитрий Сергеевич ПРОКУДИН является специалистом кафедры Математических методов прогнозирования Московского государственного университета имени М.В. Ломоносова. Его научные интересы включают распознавание образов.

Dmitrii Sergeevich PROKUDIN – specialist of the Department of mathematical methods of forecasting of CMC of Lomonosov Moscow State University. His research interests include pattern recognition.