Blending Language Models and Domain-Specific Languages in Computer Science Education. A Case Study on API RESTFul

Francisco Jurado[1], Francy D. Rodríguez[2], Enrique Chavarriaga[3], Luis Rojas[4]

[1] Department of Computer Engineering, Universidad Autónoma de Madrid, Madrid (Spain)
[2] Computer Engineering Department, Universidad Politécnica de Madrid, Madrid (Spain)
[3] UGround Global S.L., Madrid (Spain)
[4] Facultad de Ingeniería, Universidad San Sebastián, Santiago (Chile)

* Corresponding author: francisco.jurado@uam.es (F. Jurado), francydiomar.rodriguez@upm.es (F. D. Rodriguez), echavarriaga@uground.com (E. Chavarriaga), luis.rojasp@uss.cl (L. Rojas).

Received 19 September 2024 | Accepted 4 May 2025 | Early Access 29 September 2025

Abstract

Since Computer Science students are used to applying both General Purpose Programming Languages (GPPLs) and Domain-Specific Languages (DSLs), Generative Artificial Intelligence based on Language Models (LMs) can help them on automatic tasks, allowing them to focus on more creative tasks and higher skills. However, the teaching and evaluation of technical tasks in Computer Science can be inefficient and prone to errors. Thus, the main objective of this article is to explore the performance of LMs compared to that of undergraduate Computer Science students in a specific case study: designing and implementing RESTful APIs DSLs. This research aims to determine if LMs can enhance the efficiency and accuracy of these processes. Our case study involved 39 students and 5 different LMs that must use the two DSLs we also designed for their task assignment. To evaluate performance, we applied uniform criteria to student and LMs-generated solutions, enabling a comparative analysis of accuracy and effectiveness. With a case study comparing performance between students and LMs, this article contributes to checking to what extent LMs are able to carry out software development tasks involving the use of new DSLs specially designed for highly specific settings in a similar way as well-qualified Computer Science students are able to. The results underscore the importance of well-defined DSLs and effective prompting processes for optimal LM performance. Specifically, LMs demonstrated high variability in task execution, with two GPT-based LMs achieving similar grades to those scored by the best of the students for every task, obtaining 0.78 and 0.92 on a normalized scale $[0, 1]$, with 0.23 and 0.14 Standard Deviation for ChatGPT-4 and ChatGPT-4o respectively. After the experience, we can conclude that a well-defined DSL and a proper prompting process, providing the LM with metadata, persistent prompts, and a good knowledge base, are crucial for good LM performance. When LMs receive the right prompts, both large and small LMs can achieve excellent results depending on the task.

Keywords

Case Study, Computer Science Education, Domain-Specific Languages, Language Models, Student Assessment.

DOI: 10.9781/ijimai.2025.09.005

I. Introduction

Since the irruption of Generative Artificial Intelligence (GenAI) and particularly the emergence of IA-powered chat-bots that serve as front-ends to trained Language Models (LMs), either Large Language Models (LLMs) or Short Language Models (SLMs) —the scaled-down versions—, there is a kind of sprint race in the adoption, adaptation and analysis of the weaknesses, threats, strengths and opportunities that these technologies can offer in educational contexts. To estimate a benchmark of its performance, there are reports and research on how well these LLMs pass well-known examinations [1], [2], [3], generate quality texts [4], [5], create content educational, improve the students engagement, or personalize learning experiences [6].

It is remarkable that, with the achievements in the field of LMs, there is a great opportunity to automatize the assessment and feedback of text-based responses in educational environments, enhancing the efficiency and effectiveness of the learning process [7]. Nevertheless,

Please cite this article as:
F. Jurado, F. D. Rodriguez, E. Chavarriaga, L. Rojas. Blending Language Models and Domain-Specific Languages in Computer Science Education. A Case Study on API RESTFul, International Journal of Interactive Multimedia and Artificial Intelligence, (2025), http://dx.doi.org/10.9781/ijimai.2025.09.005

not all educational contexts have been explored. There is still work to do, and one of these contexts is Computer Science, where, to the best of our knowledge, there are very few research works. Students in this discipline used to work with end users' specifications taken in Natural Language (NL), which they must formalise in the corresponding Domain-Specific Language (DSL) and use other DSLs (such as SQL, HTML, CSS, etc.) and General Purpose Programming Languages (GPPL) that follow contexts-free grammars, to implement and deploy the solution to the problem.

Therefore, due to its nature with the use of NL and context-free grammar, Computer Science is an ideal scenario where to apply LMs. Not in vain, there are several LMs specifically fine-tuned for programming [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], but little research works to test the performance of LMs on easing the tasks of both, teachers and students [19], [20], [21].

Thus, because Computer Science students are familiar with the use of GPPL and DSLs to design specific solutions, the assumption made in this article is that the use of LMs to generate solutions based on such GPPL and DSLs, should allow students to better focus on formalizing the description of requirements in natural language, thus focusing on more relevant tasks, leaving the automatic ones to the LMs, and providing better results for the solutions. But before getting there, it will be necessary to see to what extent the performance of the LMs is equivalent to that of the learners. Therefore, to gather initial results for future work in this direction, we conducted a case study on the design and implementation of Representational State Transfer (RESTful) APIs by undergraduate Computer Science students.

In this sense, the research questions set for this article were stated as follows:

- RQ1. Is it possible to use current LMs together with DSLs for assignments that involve the design and implementation of RESTful APIs?
- RQ2. How do LMs perform against real students for assignments on designing and implementing RESTful APIs using DSLs?

To answer these research questions, the methodology used to evaluate the performance of the LMs can be summarized as follows: 1) two DSLs were designed to be used for Computer Science undergraduate students to perform assignments; 2) then, a group of students was taken to perform the assignment and, in turn, the corresponding LMs were prompted; 3) both, the students' proposed solutions and the LMs output, were collected and scored by the teachers of the subject; 4) finally, the scores from all students as well as LMs were compared as if LMs was just another student.

The study carried out, allows us to state that the use of DSLs and LMs in Computer Science education presents a significant opportunity to improve efficiency and accuracy in teaching and evaluating technical tasks. The results indicate that, although LMs are valuable tools, specific prompting is required to maximize their potential.

As a consequence, the main contribution of this article is the comparative analysis carried out on the performance of various real students against different LMs, contrasting different versions of them, large and small ones, and general purpose versus specially tailored for programming, all this, specifying a complex prompting that includes metadata, persistent Prompts and knowledge corpus.

Accordingly, the rest of the article is structured as follows: section II shows some related works; section III details the methodology developed to answer the research question; section IV withdraws the obtained results; then, section V answers the research question and provides a discussion on the obtained results; and finally, section VI summarizes the concluding remarks and spot some future works.

II. Related Works

The following subsections will review recent studies on how LMs have been applied in education across various disciplines, focusing on Computer Science education and code generation.

A. LMs in Education

In their technical report, OpenIA [1] shows the results they obtained when GPT-4 tries to pass exams of different disciplines. However, out of the box of this report, we can find limited research works on how LMs perform in specific disciplines, and even less in Computer Sciences related fields.

As an illustration, within the scope of medical education, Gilson et al. [2] evaluated the performance of ChatGPT for the 1st and 2nd steps of the United States Medical Licensing Examination (USMLE), a three-step examination required for medical licensure in the United States. They concluded that, for medical question answering, ChatGPT achieves the equivalent of a passing score for a third-year medical student, and highlight the ChatGPT's capacity to provide logic and informational context across the majority of answers. Thus, the authors see the potential applications of ChatGPT as an interactive medical education tool to support learning. Similarly, in the context of law education, Katz et al. [3] analysed how GPT-4 performs the Uniform Bar Exam, a standardized examination that aspiring lawyers must pass to be admitted to the bar and become licensed to practice law in a specific jurisdiction in the United States. These authors find GPT-4 being graded in the same way as a human scores above the passing threshold for all components of the test.

The work performed by Parra et al. [22], tried to understand the strengths and weaknesses of ChatGPT 3.5, Bing Chat, and Bard in the field of Geometry. To this end, they analyzed their ability to provide correct solutions, and categorize the errors found in the described reasoning processes. In terms of the correctness of the obtained solutions, LMs had what the authors qualified as a disappointing performance. The responses given by the different LLMs contained several types of errors, which the authors categorize as construction, conceptual, and contradiction.

Nevertheless, the teachers' concerns go beyond whether LMs pass the exams. Thus, for instance, since LMs can generate original text, one question that many teachers wonder about is whether LMs can be used by students to assist them in their academic tasks, i.e. to use them as learning tools. Thus, focusing on the application of ChatGPT in engineering higher education Bernabei et al.[4] investigate whether students can generate high-quality university essays with the assistance of LMs, whether existing LM detection systems can identify LMs and how students perceive the usefulness of LMs in their learning process. As a result, the authors suggest avoiding banning LMs.

Likewise, supporting students in assignments regarding writing skills, Liu et al. [5] conducted a qualitative study examining whether LMs can facilitate students' multimodal writing process for learning English as a foreign language. In their experiment, students from the control group developed a PPT presentation while being assisted with generated texts and images. The authors concluded that students were able to focus on text production and generate high-quality texts.

Rather than assisting students in the assignment development process, other authors seek to provide feedback on the students' submissions. In this regard, for those tasks specially designed to improve writing skills in secondary students, Meyer et al. [23] propose the use of LMs to generate automated evidence-based feedback for the students' text revision. Thus, they conducted a study comparing the effectiveness of the feedback produced by GPT-3.5-turbo to English as a foreign language students' essays, compared to no feedback at all. The results they obtained indicate that LM-generated feedback increased

revision performance and task motivation. In their study, Meyer et al. [23] assessed the improvement in the revision using automated essay scoring by training an algorithm using a Support Vector Machine. Similarly, Rigaud et al.[24] analyzed the performance on the feedback of ChatGPT in Linear Algebra problems. In their analysis, they examined and evaluated the feedback provided to engineering students in their solutions, both from teachers and ChatGPT. The results revealed potentialities and challenges in improving feedback on graduate-level mathematical problems, identifying deficiencies in reasoning, proofs, and model construction, among other areas.

Besides that, LMs can also be used for the automatic scoring task, and in this regard, it is remarkable the work performed by Mizumoto et al. [25] explored the potential use of GPT-3 text-davinci-003 LM in automated essay scoring for non-native speakers of English. They concluded that students, teachers and researchers can obtain valuable insights by mastering effective strategies exploiting GPT in their work, because it can significantly improve their language skills.

For their part, Latif et al. [26] argue that the direct use of pre-trained GPT-3.5 is not enough for automatic scoring, and that contextual information is necessary for accurate scoring. As a consequence, in their study, they fine-tuned GPT-3.5 using data from six assessment tasks for student-written responses in science education. The results showed that when trained on in-domain training corpora, GPT-3.5 demonstrated a remarkable performance in automatic scoring accuracy.

In their study, Urban et al. [27] explored the impact of ChatGPT on university students' performance in complex creative problem-solving tasks. The results showed that, although ChatGPT did not increase task interest, students who used ChatGPT had significantly higher self-efficacy in task resolution and produced higher quality, more elaborate, and original solutions. Also, the students found the task easier and required less mental effort.

The above clearly shows that there is a need to further explore the goodness, strengths and threats that Generative AI brings to different educational contexts, specific subjects and particular skills. Particularly, the goal of this article is on the competencies and skills that relate to Computer Science to understand their specific impact on this educational area. Thus, the next subsection will delve into how these LMs are being used to support both students and teachers in this field.

B. LMs in Computer Science Education

Focusing on Computer Science Education, Prather et al. [19] conducted a study on how novice students interact with Github Copilot [14] during their learning progress in an introductory Programming course (CS1) assignments. In their study, most of the students considered that Copilot could enhance their coding speed, although they expressed apprehensions regarding comprehending the auto-generated code, and becoming dependent on such tools. Also, the study reveals two interaction patterns: on the one hand, some students guide Copilot by leveraging its auto-generated code towards a solution rather than starting from scratch and incorporating Copilot's recommendations; on the other hand, some students go through some of Copilot's inaccurate suggestions, moving from one to the next and consequently becoming lost.

Likewise, Haindl et al. [20] explained the students' experience using ChatGPT in an undergraduate Programming course. The information gathered using anonymous surveys about the use from the students, shows that students feel confidence using ChatGPT to support them with learning programming concepts, but not for the implementation tasks. Some students refrained from using the tool due to concerns about insufficient development of programming skills, the risk of receiving incorrect code, or scepticism regarding its benefits. Also, the authors offer guidelines to enhance the use of LMs in education like design of engaging student-centred projects, guidance on prompt engineering, and strategies for assessing the accuracy of generated responses.

Moving towards later courses, Kozov et al. [21] use LMs that generate images and code in their workshops with students in the subjects of Analysis of Software Requirements and Specifications, and Artificial Intelligence. The results showed a positive and motivational impact during and after the workshops. Although the authors do not provide results in other subjects, they spot some uses of LMs in subjects like Introduction to Programming or Computer Graphics.

Despite the above-mentioned research, so far, we can see evidence that there is still a lot of work to assess the performance of LMs in educational contexts, both assisting students and teachers in the teaching-learning process, where it is necessary to create good practices and guidelines for both roles and in the tasks of automatic guidance and assessment, where, at present, train and fine-tuning of LMs seems to be necessary. However, these fine-tuning processes often require time, computing resources and training corpus. Therefore, as previously stated in Section I, this research work asks whether existing LMs be used in combination with DSLs for these tasks, and for this purpose, we analyse the performance of LMs in Computer Science tasks that use DSLs.

As seen, research in Computer Science education indicates that LMs can be valuable tools, but they also present challenges. Next, we will explore how some of these LMs are being specifically applied for the particular task of code generation.

C. LMs for Code Generation

As far as we know, the use of DSL together with AI techniques and Natural Language Processing (NLP) in code generation contexts is something worth noting. Thus, before the irruption of LMs, there were interesting approaches that attempted to generate DSL code from descriptions written in natural language (NL). Particularly, Desai et al. [28] propose a framework for building program synthesizers that can interpret NL inputs and generate expressions in a designated DSL. The framework takes as input a DSL definition and training data of NL/DSL pairs, and thus, it constructs a synthesizer that prioritizes the outputs of a keyword-programming-based translation.

Similarly, also without using an LM-based approach but instead using techniques from NLP, Kolthoff et al. [29] proposed a methodology for prototyping a Graphical User Interface (GUI) by processing requirements described in NL, and translating them into a DSL describing the GUI and its navigational schema. Then, with the generated DSL, the corresponding target platform prototypes can be generated. Also, the users' feedback about the GUI proposal can be provided in NL to fine-tune the generated prototypes.

Only two years ago, Kolahdouz-Rahimi et al. [30] conducted a review on existing studies on NLP and Machine Learning (ML) for Requirement Formalisation. Their study found that heuristic NLP techniques are the most frequently used for this specific problem and that classical ML techniques are more prevalent rather than Deep Learning. However, at that moment, there was not even any mention of LMs in this regard.

The best-suited approaches relied on DSL as a mechanism close to NL to facilitate the specification and development of solutions by end-users. Thus, for instance, Vernotte et al. [31] present a DSL to prevent air traffic control cyberattacks by allowing air traffic control experts to design tests where attackers modify, block, or emit fake messages to dupe controllers and surveillance systems. Their work demonstrates the design capabilities and a productivity gain. For their part, the works performed by Chavarriaga et al. [32], [33], [34] show several approaches that allow specifying custom DSL for both textual and

visual, facilitating end-users the specification of solutions in a way closer to the domain, and automatically generate JavaScript code that implement those solutions with high quality.

Once the LMs were raised, interesting research proposals emerged. Thus, in a blended way between DSL and NLP, Singh et al. [35] used LMs with an approach that has demonstrated state-of-the-art success rates with the use of program-like specifications to prompt LMs to generate the code of plans for situated robots, with specific recommendations on instruction structure and generation constraints.

The review conducted by Wang et al. [36] on code generation with LMs, concluded that, due to the powerful code understanding and writing ability LMs provide, they can be applied in software engineering tasks to boost the productivity of developers. However, the work also pointed out that the evaluation of the quality of the generated code receives less attention from researchers. In this last regard, for evaluating the code generation ability of LMs, Yeo et al. [37] propose a framework and a new metric to measure the accuracy according to the pass rate of test cases.

Going on to other tasks of the software development process, Schafer et al. [38] present an approach for software testing where the LMs are provided with prompts that include the signature and implementation of a function under test, and then, the tool generates the JavaScript unit tests. Their evaluation concluded that the effectiveness of the approach is influenced by the size and training set of the LM, but does not depend on the specific LM itself. Regarding databases, Zhou et al. [39] propose a framework that includes automatic prompt generation, as well as LMs fine-tuning, designing and pretraining specifically for database management. Their preliminary results indicate that the framework performs relatively well in tasks such as query rewriting and index adjustment.

As can be seen so far, in the software engineering domain, some researchers have theorised about the potential of Generative AI technologies and LMs, as well as their main scenarios, and an example of it is the work presented by Sauvola et al. [40]. However, the fact is that the potential of what can be achieved can only be discerned through a few early research efforts within the last two years.

So far, code generation using LMs has opened new possibilities in software development, but it also raises questions about performance. In the next subsection, we will discuss the metrics and techniques used to measure the effectiveness of these LMs in programming tasks.

D. Measuring the LMs Performance

Analyzing and comparing the performance of each LM against human students in programming tasks may involve several techniques and measures [41, 42]. Hence, to benchmark the performance of LMs, we can measure text similarity, coherence, relevance, and precision. In the following, we will describe some of the most common strategies.

The first approach is to measure Similarity, that is, to score how similar the content of two texts is. There are several metrics to evaluate Similarity. Thus, the Bilingual Evaluation Understudy (BLEU) [43] is a commonly used metric for benchmarking the quality of automatic translations, but it can also be applied to compare texts in the same language. It calculates similarity based on the match of n-grams (sequences of n words). Based in the BLEU metric but code oriented, CodeBLEU [44] is built upon the n-gram matching strength of BLEU, and further integrates code syntax using abstract syntax trees and code semantics via data-flow, resulting in a more robust approach. Likewise to BLEU, the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [45] focuses on evaluating the quality of summaries, and therefore, it can be useful for comparing short answers or summaries.

Another simple approach is the Cosine similarity, which measures the similarity between two text vectors, transforming each text into a vector in a high-dimensional space like Term frequency – Inverse document frequency (Tf-idf), and then, the similarity is the cosine of the angle between these vectors. For its part, Jaccard similarity measures similarity based on the set of unique words present in both texts, which is helpful for more general text comparisons. With results near Jaccard similarity, MinHash technique [46] is usually helpful for plagiarism detection tasks in text documents. However, its applicability can extend to code snippets, enabling the identification of similarities in code fragments. Analogous, the Wagner-Fischer algorithm is a dynamic programming algorithm that measures the Levenshtein distance between two strings of characters. This distance calculates how similar the two strings are, based on the number of deletions, insertions and substitutions needed to transform one string into the other. This algorithm would be useful for strings comparing, i.e., in our case, code fragments.

For medium to long texts, another relevant benchmark is Coherence to evaluate the ability of LMs to maintain a line of thought in the responses, that is, to score how well the ideas within a text connect. To analyze the Coherence, we can use NLP techniques that help analyse the structure and fluency of the text, like identifying the topics and checking thematic and logical coherence.

To determine the significance of the generated text, Relevance assesses the capability of LMs to address the question's topic. To measure Relevance, we can use Human Evaluation to rate it based on their knowledge, or Information Search tools to verify if the text provides relevant and up-to-date information on the topic. Likewise, Precision evaluates the accuracy and correctness of the information provided by LMs in terms of meeting the acceptance criteria. It refers to how correctly the details in the text are. To measure Precision, Human Evaluation and even Fact-Checking are used to verify the statements made in the text.

Each of these metrics and techniques has advantages and limitations, and the choice of tools and methods will depend on the specific context of the study and the objectives of the evaluation.

As final remarks, along with this and previous subsection, after analyzing the applications and evaluations of LMs in education, particularly in Computer Science, and for the specific task of code generation, it is evident that these technologies have a significant and evolving impact that can be measured with specific metrics. Consequently, the next section will detail the case study performed.

III. Methodology

This section will provide details on how we have compared the performance between students and LMs, to check to what extent LMs can carry out coding tasks involving the use of new DSLs specially designed for highly specific settings.

A. Experimental Process Definition

This study adopts an exploratory case study approach, which is suitable for examining the performance of LMs and undergraduate Computer Science students in a controlled educational environment. Rather than aiming for broad generalization, this exploratory design focuses on gaining initial insights into how LMs compare to students when working with DSLs and RESTful APIs, providing a foundation for future, more extensive studies.

To evaluate the LMs to answer the defined research questions, we defined and organized the experimental process in this way:

1. Two DSLs were created to specify RESTful APIs within a structured programming environment, designed to be used both by undergraduate Computer Science students and LMs in performing their assignments (subsection III.B).

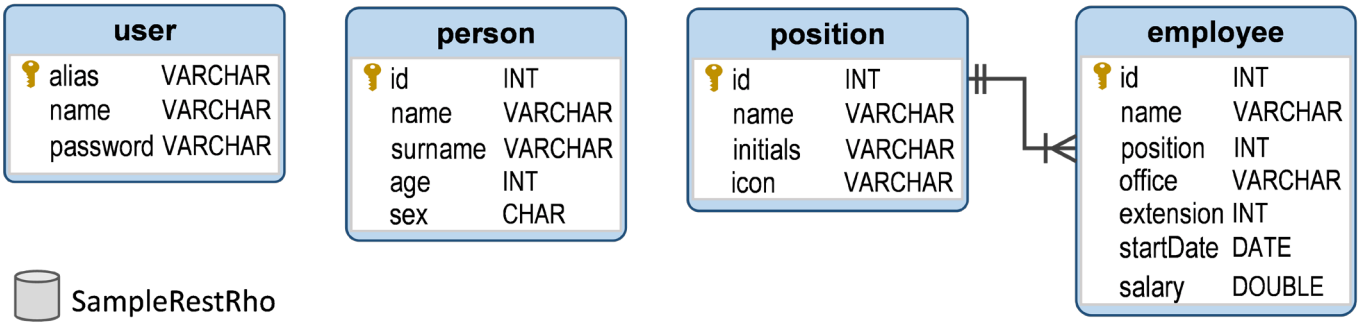

Fig. 1. Database Diagram for Sample RestRho. The database has four tables: two with non-related data (user and person), and two with a one-to-many relationship (position and employee).

2. An assignment was designed to evaluate participants' abilities in using these DSLs, incorporating tasks with varying complexities to ensure a balanced assessment (subsection III.C).
3. A group of 39 undergraduate Computer Science students was selected for the experiment, ensuring a homogeneous skill set and sufficient expertise to engage with the tasks (subsection III.D).
4. A diverse range of existing LMs was studied and carefully selected to ensure broad representation, covering both general-purpose and programming-specific capabilities (subsection III.F).
5. Evaluation scenarios were defined to ensure consistency between students and LMs. These included uniform task assignment, structured prompt engineering, and validation of outputs using a standardized rubric (subsection III.G).
6. The selected LMs were configured with tailored prompts and contextual metadata to simulate real-world scenarios and maximize their performance in the experimental setup (subsection III.H).
7. Both the solutions provided by students and the outputs generated by LMs were collected and assessed by the course instructors using predefined evaluation metrics. Grades were assigned and analyzed to compare performance across both groups, with results summarized and analyzed in Section IV.

The next subsections will cover all the details about how the experimental process was performed.

B. Environment Used for the Assignments

This section describes RestRho, a RESTful API server based on JSON-DSL designed and implemented using RhoArchitecture [33]. By engaging with this example, students can gain hands-on experience in working with JSON-DSL and RESTful API development. For short, a DSL is a specialized programming language tailored to a specific application domain, offering a concise and efficient way to express domain-specific rules and algorithms. Unlike GPPLs, DSLs are optimized for particular tasks, enhancing productivity and precision in software development [47], [48], [49].

Within this context, RhoArchitecture provides a framework for building and executing JSON-based DSLs (JSON-DSLs). This framework includes the Rho Programming Model (RhoModel), which integrates JSON-DSL specification, JavaScript classes for grammar functionality, JavaScript and Web Components, the Handlebars Template Engine, and data source connections (JSON, XML, Text) [33], [34]. The Rho JSON-DSL Evaluation Engine (RhoEngine) is a JavaScript component that runs JSON-DSL programs, evaluates grammar symbols, and supports connections, template usage, component interactions, security policies, and good programming practices.

The Web Integrated Development Environment for Rho (WebIDERho) uses RhoModel and RhoEngine, allowing project definition for server and client-side, class diagram visualization, automatic documentation generation, and deployment of NodeJS applications.

Before deploying the DSLs (DBRestRho and SQLRho) in the experiment, we conducted an internal evaluation phase. This included: (i) defining the DSL grammar and syntax rules based on RESTful API design principles; (ii) reviewing the DSL specifications with two domain experts; and (iii) testing the DSLs with sample queries and templates to ensure their correctness, clarity, and ease of use. These steps helped validate that the DSLs were well-prepared and robust for the study.

1. RestRho: API RESTful Server Based on JSON-DSL

RestRho aims to create JSON-DSLs within the RhoArchitecture framework, facilitating the specification, implementation, and deployment of RESTful APIs. These APIs are designed to enable efficient and scalable web application interactions using standard HTTP/HTTPS verbs such as GET, POST, PUT, and DELETE [50], [51], [52]. RestRho serves as a JavaScript component to define and dynamically deploy these HTTP requests on a RESTful NodeJS server, known as ServerRestRho [52], [53]. The server and its requests are implemented using Express, a fast and minimalist web framework for Node.js [50], [54].

Also, RestRho facilitates resource access through two RhoLanguages:

1. **DBRestRho** is the JSON-DSL in charge of (i) defining the list of HTTP requests and associating a template with a SQL statement to each operation; (ii) executing SQLRho programs; and (iii) helping to create prompts based on the information from a database, its tables, and its operations, so that it eases the task of creating the knowledge corpus necessary for prompting the LLMs. The definition of the DBRestRho Grammar can be found in Appendix A.
2. **SQLRho** is the JSON-DSL that allows configuring DB connections (MariaDB or MySQL) and defines a set of operations on existing tables based on the list of requests defined in DBRestRho. There are two kinds of operation associated with a table: (i) basic operations at record level (Insert, Update, Delete, and Select), and at table level (Where and SelectAll); and, (ii) custom operations, which support configuration, and the application of transformation and output templates to the results. The definition of the SQLRho Grammar can be found in Appendix B.

2. A Sample of the Use of RestRho

The purpose of this section is to create an illustrative example to show how to write DBRestRho and SQLRho code and run them on ServerRestRho. This same example was the one used as a reference when prompted the LMs. Fig. 1 shows the Sample RestRho database model as called *SampleRestRho*, which is made up of four tables: *user* (with the login information), *person* (an isolated table with some attributes to store information about specific persons), and *position* and *employee* (two joined tables to manage a one-to-many relationship).

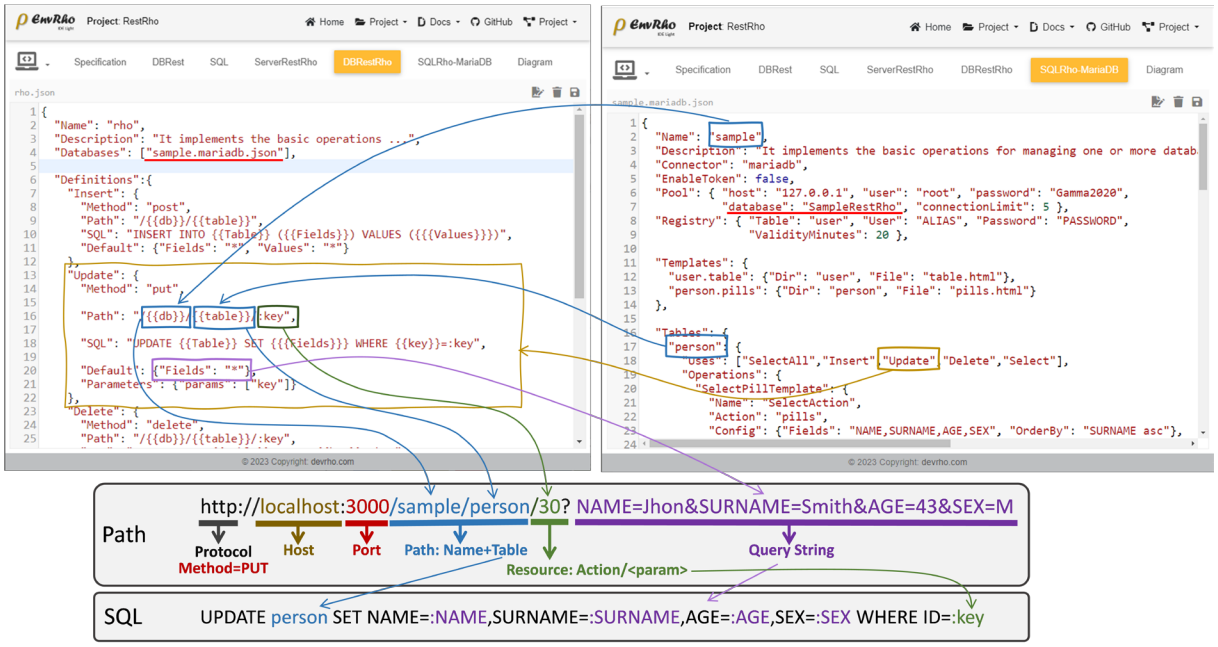

Fig. 2. DBRestRho and SQLRho example programs, and how a HTTP path request (REST endpoint) and SQL statement are created.

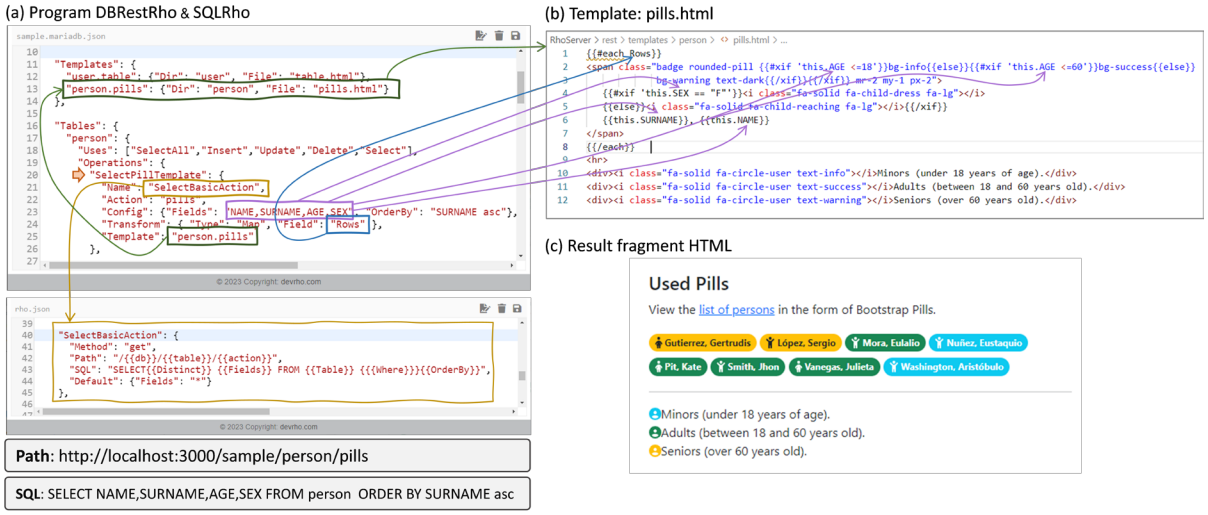

Fig. 3. Example of DBRestRho and SQLRho to create an operation and link it to a specific template to render the result in the browser.

For its part, Fig. 2 shows the program DBRestRho "rho.json"[1] (on the left), denoted by *PrgDBRest*. In that part of the figure, we can see the "Databases" attribute with a list of the database the JSON-DSL code is using, in this case, the "sample.mariadb.json"[2]. We can also observe how the program defines the functionalities (Insert, Update, Delete), that can be applied to those "Databases". Looking the "Update" definition, we can see the HTTP method it uses ("put"), the path templated that specifies the REST endpoint, the SQL template statement to perform, as well as other default values and parameters.

Fig. 2 also shows SQLRho program (on the right, and connected to the database SampleRestRho), denoted by *PrgSQL*. This code contains database specific information, like the database name, the tables information (with the attributes they has, the operations that can be performed on each one, etc.). Using the information contained in the specific SQLRho code, the DBRestRho templates fills the endpoint and SQL templates (at the bottom of the figure). With this two DSLs, the students work in an example that isolates REST-API definition from the database definition, but must link all together.

In addition, in *PrgSQL* the register table is user. Also, the figure shows how the HTTP Path request is generated, as well as the SQL sentence of the Update operation (for table "person"). For the Path, RestRho dynamically creates the PUT request as follows: (i) the *Host and Port* where ServerRestRho is deployed, the *URL Path* taken from the *PrgSQL* program name, concatenated with the name of the table person, the Resource primary key of table person, e.i., the record id=30, and the Query String fields defined in the table: NAME, SURNAME, AGE and SEX. We can also see the SQL statement, the table name ("person"), the values of each field (those that come in the Query String), and the primary key ("id" field).

For its part, Fig. 3 details a custom SelectPillTemplate operation in the program *PrgSQL* for the "person" table, which is the way to specify an HTML template to visualize information provided by the REST. Particularly, Fig. 3(a) details the specification of *SelectBasicAction* definition in the program *PrgDBRest* using DBRestRho JSON-DSL

[1] https://www.devrho.com/restrho/rho.json

[2] https://www.devrho.com/restrho/sample.mariadb.json

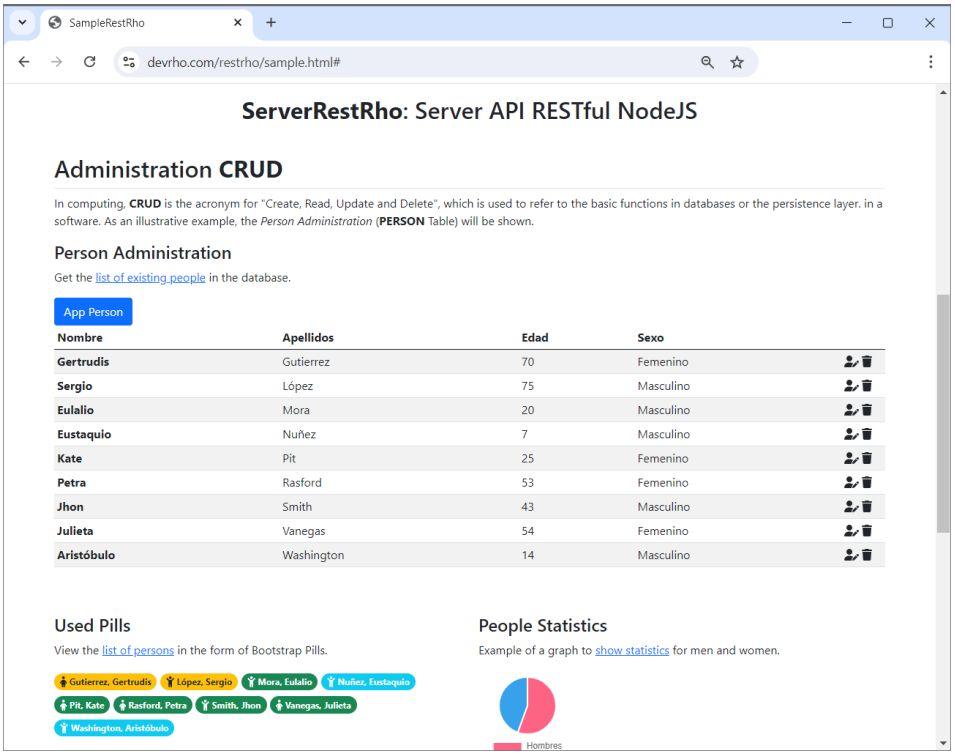

Fig. 4. Example of how a Handlebar template is finally rendered in the browser.

in the way explained in the previous paragraph. However, this code shows the Handlebars templates that will be used to render and display the information in the browser. In particular, the code shows two templates, namely: "user.table" and "persol.pill" (at the top of Fig. 3(a). As an example, the "person.pill" defines that, to render the table "person", the template "pills.html" must be used (at the top of Fig. 3(b)). The "pill.html" template is based on the Bootstrap framework v.5.3[3], and what we see in the code is how it goes through all the records and for each one, it creates a coloured Pill according to the age range, and an icon if male or female. The result when executed by RhoEngine and rendered in the browser is displayed in Fig. 3(c).

The running example of Sample RestRho is shown in Fig. 4 and available at http://www.devrho.com/restrho/sam ple.html. That figure shows an example of the CRUD administration of the "person" table, the list of people in Pills, and the graph of percentages of men and women. Also included in the example is Login, the "person" table where the list of users is shown, and the 'employee' table that shows the "employees" in the form of Cards.

C. Assignment Description

Table I offers an in-depth description of the tasks assigned to participants. These assignments are meticulously formulated based on the Employees Sample Database from MariaDB, which is freely accessible and serves as a solid practical basis for the experimental tasks. The goal of these tasks is to design and effectively implement a RESTful API server using the previously described JSON-DSL. Each of these tasks is assigned a unique identification code and is described with clarity and brevity in the "Tasks" column of the table.

To ensure a balanced evaluation across different skill levels and task types, the tasks were divided into categories, such as configuration, operations, and usage, representing a range of complexities from simple setup tasks to advanced operations requiring template creation and data visualization. Table I includes a "Task Complexity" column, which quantifies the relative difficulty of each task based on its design and expected effort. These complexity values were normalized to ensure their sum equals 1, offering a clear view of the progression in difficulty.

The tasks are systematically arranged under relevant main activities, which provides further insights into the specific purposes or applications of each task within the experimental setup. This arrangement ensures a logical flow from foundational tasks to more advanced challenges, facilitating both learning and assessment. Additionally, the tasks were intentionally designed with increasing complexity to evaluate participants' performance at different levels of expertise.

By including a quantifiable measure of task complexity and aligning tasks with specific types and activities, we aimed to balance the experimental design and ensure fair evaluation of both students and LMs. Each task falls into specific types, such as configuration, operations, and usage, offering a clear classification that underscores the functional nature and requirements of the tasks. This structured presentation not only enhances the comprehension of the tasks' goals and methodologies but also assists in evaluating their impact and relevance to the overall experimental goals.

D. Selecting the Students

The subjects involved in this case study were students enrolled in a Software Project Management course during their final year of the Computer Engineering program. This specific course was chosen because it aligns with the experiment's requirements: according to the curriculum, these students had already mastered essential skills in database management, web application development, and RESTful API design. Thus, no additional initial assessments were necessary. The relatively homogeneous skill set of these 39 students, while not large enough for broad generalization, is appropriate for an exploratory case study aiming to derive preliminary insights.

To ensure the ethics of the research, all participants provided their written informed consent, voluntarily agreeing to participate in the study. In total, 39 students participated, all of whom were voluntarily selected from the aforementioned course, ensuring a representative sample of students with the requisite background. These students had

[3] https://getbootstrap.com/

TABLE I. Comprehensive Summary and Categorization of Tasks in the RestRho Experimental Framework

ID	Tasks	Task Complexity (*)	Task Type	Main Activity
1A	Setting up a file to launch an SQLRho program.	0.0750	TT1. Configure the SQLRho program with the employees database	MA1. SQLRho program
1B1	Query the current number of employees in each department,sorted alphabetically by department name.	0.0450	TT2. Operations on table 'departments'	
1B2	List all managers who have worked in a department in chronological order.	0.0450		
1B3	List the current managers of each department, sorted by last name and first name.	0.0450		
1B4	List the current employees in a department, including their current age.	0.0450		
1B5	Query the current number of men and women in each department, sorted by the number of employees in descending order.	0.0450		
1B6	Query the current minimum, maximum, and average salaries y department, sorted by average in ascending order.	0.0600		
1C1	Query the current number of men and women in the company.	0.0600	TT3. Operations on table 'employees'	
1C2	Query the titles that an employee has held in chronological order.	0.0300		
1C3	Query the departments where an employee has worked in chronological order.	0.0450		
1C4	Query the current number of men and women in the Company within the following age brackets [0-17, 18-40, 40-65, 65+].	0.0450		
1C5	Query an employee´s salary history in chronological order.	0.0450		
2A	Create an HTML template for managing CRUD operations on he "departments" table.	0.0600	TT4. Usage of Handlebars templates	MA2. Company Website
2B	Create an HTML template for managing CRUD operations on he table of employees who are department managers.	0.0444		
2C	Use "SelectManagersDepto (T4)" with its fields to create a emplate for cards of current managers in each department.	0.0889		
2D	Use "SelectEmployeesListDepto (T5)" with its fields to créate template for pills of current employees in a department.	0.0889		
2E	Create a diagram to display the current number of men and women in the company.	0.0889		
2F	Create a diagram to display the current number of employees in each department.	0.0444		

(*) The values in the Task Complexity column are normalized such that their sum equals 1.

an average age of 23.5 years, with age variations ranging from 22 to 29 years and a standard deviation of 1.8 years, noting that 90% of the participants were male.

Their technical skills were consistent with the expected outcomes of their training. They had developed skills in programming in multiple programming languages, which allowed them to effectively address complex software problems. Additionally, they possessed practical knowledge in the design and management of databases, an essential competence for the efficient handling of large volumes of data. They also had experience in web application development, which included both front-end and back-end, preparing them to face the challenges of creating comprehensive web solutions.

E. Rubric Metrics

To ensure rigorous assessment of both LMs and student performance, specific metrics and criteria were applied to evaluate task success. The metrics used include:

1. Correctness: The extent to which the solutions meet the task requirements. For SQLRho tasks, this involved syntactically correct SQL queries that returned the expected results. For DBRestRho tasks, this required correctly implemented RESTful endpoints that adhered to the specified API schema.
2. Completeness: Whether all components of the task were addressed (e.g., handling all fields in a database table or completing all CRUD operations).
3. Execution Success: The ability of the solution to run without errors (e.g., SQL queries executing successfully or APIs passing validation checks).
4. Error Analysis: The number and type of errors (e.g., syntactic errors in SQL queries or missing endpoints in API configurations).

Additionally, the criteria for determining task success were as follows:

- For SQLRho tasks: Success was defined as writing queries that retrieved accurate and complete data as specified in the task description. Queries were tested against a predefined dataset to ensure validity.

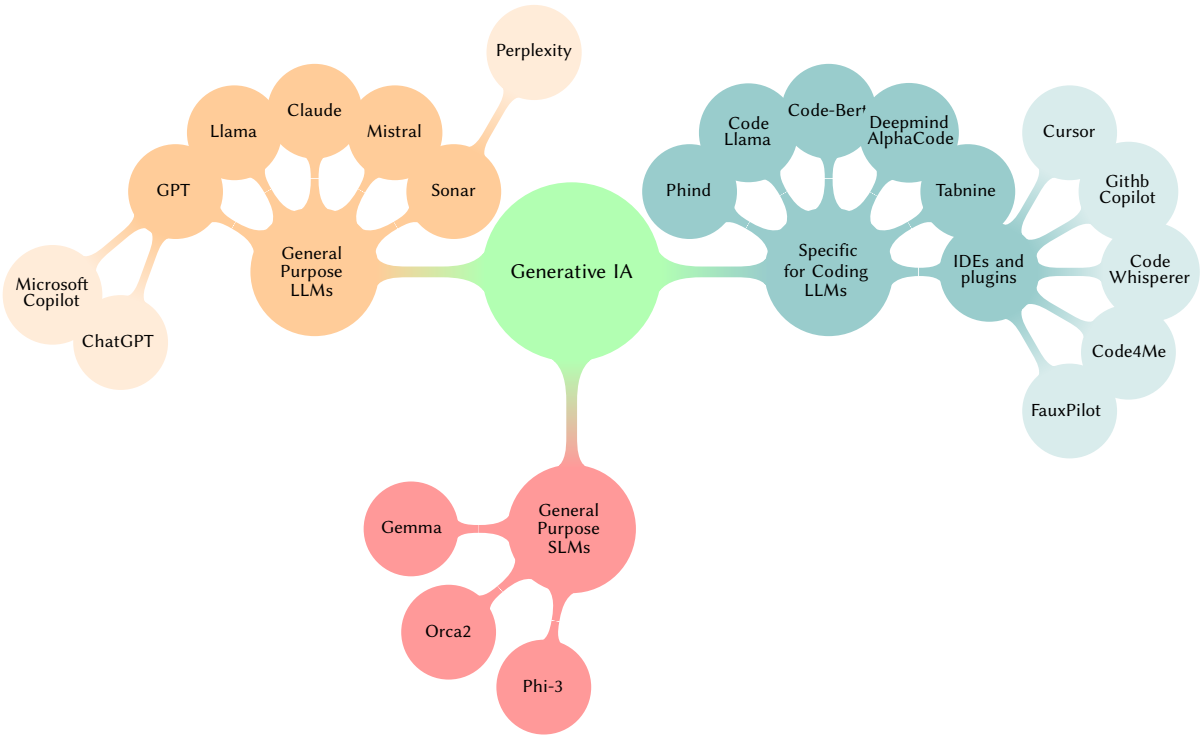

Fig. 5. Mind map with some of the most well-known LLMs and SLMs that students and teachers can use for programming assignments. On the left hand, the general purpose LLMs; on the right hand, those LLMs fine-tuned for programmingspecific purposes; at the bottom, some general purpose SLMs.

- For DBRestRho tasks: Success was defined as implementing endpoints that followed the JSON-DSL requirements and passed a series of automated integration tests.

Human evaluation was conducted using a standardized rubric that included predefined solutions for all tasks. Evaluators compared the solutions provided by participants against these expected answers to assess correctness, completeness, and adherence to task requirements. This rubric ensured consistency and objectivity in evaluating both LMs and students.

These metrics and criteria provided a comprehensive framework to compare the performance of LMs and students across all tasks, ensuring fairness and reproducibility.

F. Outline of Language Models

Before conducting the case study, it is necessary to decide which LMs to use among all available. To better follow this section, Fig. 5 summarizes some of the LMs programmers can use to assist their work. In the figure, we can see three big clusters: general purpose, fine-tuned for coding, and those that can be integrated as plugins or are an IDE themselves.

In the first instance, we can find generalist LMs. These LMs are designed to understand and generate human-like text across a wide range of topics. Among them, the most widely used and well-known is ChatGPT [55] and those based on it, like Microsoft Copilot [56] (also known as Bing Chat). For its part, Perplexity [57] supposes another good approach conceived as a chat-like search engine. Although it works with several LLMs, Perplexity has a LM called Sonar [58]. Llama [59], the LM developed by Meta, Claude [60] and Mistral [61], are other great examples of generalist LLMs.

In addition to the LLMs, it is worth noting the good results obtained from the SLMs, like Phi3 [62] and Orca2 [63] developed by Microsoft, or Gemma [64] developed by Google and based on Gemini [65].

Specifically for programming, Phind [8] offers tailored support for coding tasks. Phind has a web interface, but it is also available as a plugin for popular Integrated Development Environments (IDEs) like Visual Studio Code, enhancing developers' workflow. Code Llama [9], a variant based on Llama, focuses on assisting programmers with code-related queries and tasks. Deepmind AlphaCode [10], CodeBert [11], and Tabnine [12] are other specialized LLMs tailored for programming contexts.

Furthermore, we can find more IDE plugins designed to improve programmers' capabilities using LM. Thus, for instance, Cursor is an IDE that uses LM to enhance code navigation and code-editing [13]. Similarly, Copilot [14] is a well-known plugin built on top of Codex [15], a GPT LLM fine-tuned on GitHub code repositories, that provides intelligent code suggestions and completions. CodeWhisperer[16], which is the approach developed by Amazon, Code4Me [17] and FauxPilot [18], are other examples of plugins aiming to improve coding workflows and boost productivity.

To enable interfacing with the different LMs, standard tools are emerging to interact with them and to allow using them in the research and development processes. Among these tools, we can highlight Ollama[4], LM-Studio[5] or Jan[6], three front-ends to search, load, and interact with LMs in an easy to use way.

For our case study, we selected a set of seven different LMs aiming to cover a broad spectrum of capabilities. Among them, we included three GPT-based LLMs, three nonGPT-based LLMs, one programming-focused LLMs, and one SLM. This selection allowed us to analyze the performance across general-purpose and programming-specific LMs, as well as LLMs and SLMs approaches, ensuring that the chosen LMs were relevant to the experiment's objectives and diverse enough to provide meaningful insights into their capabilities and limitations. The details on the chosen LMs will be provided in Subsection III.H.

[4] https://ollama.com

[5] https://lmstudio.ai/

[6] https://jan.ai/

G. Evaluation Issues for LMs

To ensure a fair and comprehensive evaluation of the LMs, the next issues were addressed:

- Uniform Task Assignment: All LMs were presented with the same set of tasks outlined in Table I, ensuring consistency with the assignments given to students. These tasks included both basic (e.g., SQLRho configuration) and advanced operations (e.g., template-based queries).
- Prompt Engineering: Each LM received structured prompts created with a standard template that included metadata, task descriptions, and expected behaviors. This approach was designed to simulate a consistent and controlled interaction framework for all LMs.
- Diverse Model Capabilities: The selected LMs included both general-purpose and programming-specific LMs. Each LM was evaluated under its optimal configuration (e.g., GPT-4 with temperature 0.1 and enabled browsing).
- Output Validation: LM outputs were collected and evaluated using a standardized rubric (see subsection III.E), ensuring uniformity in how task success was assessed across all scenarios.

H. LMs Selection and Setting-Up

For our exploratory analysis, we selected a set of LMs covering a wide range of possible configurations, from LLMs and SLMs, generative LMs based and not based on GPT, and generalist and specially trained for programming tasks. The LMs set selected were:

- **GPT-4, GPT-4o**[7], and **Microsoft Copilot pro**[8] as three GPT-based LLMs. For these LMs, the temperature parameter was set to 0.1, and both, the internet browsing and code interpreter functions were enabled. Microsoft Copilot Pro, which lacks a code interpreter function, was configured with only the internet browsing function activated and operated in GPT-Balanced mode.
- **Perplexity**[9] as front-end for the Sonar LLM. For this LM, we used the default configuration.
- **LLama3** version with 7 billions of tokens and Q4_K_M quantification[10] loaded in LM-Studio.
- **Phi3** as a SLM, with 3 billion of tokens and Q4_K_M quantification[11] loaded in LM-Studio.
- **Phind** as a code-specific LLM, for which we selected the *instant* version through the available Visual Studio Code plugin[12].

After the LMs selection, Prompt engineering was applied to the LMs, so we asked the them to solve the same assignment presented to the students (see Table I). Specifically, they were required to provide an explanation of **RestRho** and its JSON-DSL, DBRestRho and SQLRho, with the same examples used to train the students (one of these examples is detailed in Subsection III.B.2). It is worth noting that DBRestRho has the capability to generate structured information (see the "Prompts" tag in Appendix A). Accordingly, the prompts include the metadata and persistent prompts (subsection III.H.1), as well as the knowledge corpus (subsections III.H.2, III.H.3 and III.H.4). The prompt outline was as follows:

```
RestRho APIRESTful

# Description
{Description prompts goes here}

# Instruction
{Restrictions prompts goes here}{Clarifications prompts goes here}
{Purposes prompts goes}
{Guidelines prompts goes here}
{Personalization prompts goes here}

# The RestRho Languages
{The description of the languages goes here}

## Sample DBRestRho
An example of DBRestRho (program JSON) is shown between the ```
↪ characters:
``` {The whole text with the DBRestRho program goes here} ```

## Sample SQLRho
An example of SQLRho (program JSON) is shown between the ``` characters:
``` {The whole text with the SQLRho program goes here} ```

# Database named **sample**

## Template Definition
A Handlebars 'template' is a kind of HTML skeleton that includes
↪ placeholders and logic blocks, allowing specific data and logic to be
↪ directly injected into the web page. The following Handlebars
↪ templates are defined:
1. Template: **user.table**
``` {The whole text with the template goes here} ```
2. Template: **person.table**
``` {The whole text with the template goes here} ```
3. Template: **person.edit**
``` {The whole text with the template goes here} ```
4. Template: **person.pills**
``` {The whole text with the template goes here} ```
5. Template: **person.piesex**
``` {The whole text with the template goes here} ```
6. Template: **employee.cards**
``` {The whole text with the template goes here} ```

## Table Definition
In the **sample** database, the tables are defined with SQL statements:
{The SQL statements go here}
```

After applying the same prompt to each LM, we prompted them for each assignment task in the same order as posed to the students. This process ended up with the creation of the file we called "prompt.txt"[13] which serves as a structured input to ensure the LMs could interpret and address the tasks correctly.

The methodology behind the construction of this prompt is thoroughly explained in the subsequent subsections, which will detail the text with the metadata used to contextualize the tasks, the domain-specific knowledge included to enhance the LMs' understanding, and the structured database-related information provided for accuracy. Later, with this prompt in place, the selected LMs were instructed to perform the same tasks previously presented to the students.

1. Metadata and Persistent Prompts

In addition to specific knowledge, it is necessary to provide additional documentation that is key for guiding, contextualizing and optimizing the use of the LM: description, specific purpose, restrictions, guidelines, clarifications, personalization, and conversation starters. This information is also commonly referred to as Corpus Metadata.

[7] https://chatgpt.com

[8] https://copilot.microsoft.com/

[9] https://perplexity.ai

[10] https://huggingface.co/lmstudio-community/Meta-Llama-3-8B-Instruct-GGUF

[11] https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf

[12] https://marketplace.visualstudio.com/items?itemName=phind.phind

[13] https://www.devrho.com/restrho/prompt.txt

Therefore, the first step is to provide a general description, which we defined as follows:

```
RestRho includes JSON-DSLs DBRestRho and SQLRho to deploy a RESTful API
↪ server, supporting HTTP methods like GET, POST, PUT, DELETE. It
↪ transforms JSON and uses Handlebars to separate logic from design in
↪ web content.
```

Restrictions define the LM's limitations, such as prohibited topics, technical limitations, or content guidelines. In our case:

```
The model must focus on technical accuracy and clarity, ensuring that its
↪ guidance is directly applicable to the SQLRho and DBRestRho languages.
↪ It should avoid giving advice on unrelated technologies or straying
↪ off topic. It must also avoid executing or suggesting any harmful
↪ code.
```

Clarifications can include possible assumptions that the LM can make if the information is missing and the expected behaviour of the LM in specific situations, such as explaining the technical solution or when to ask the user for more details. In our case, the clarifications were:

```
The model should lean towards providing a response based on the expected
↪ behavior of SQLRho and DBRestRho, filling in missing details with
↪ assumptions that are standard practice for SQLRho and DBRestRho
↪ implementations.
```

Another key aspect is to clearly indicate the LM's purpose:

```
The model is designed to be a software engineer with expertise in using
↪ the RestRho specification and its JSON-DSL (domain-specific language
↪ with JSON grammar): DBRestRho and SQLRho. Its main function is to help
↪ users understand, implement, and troubleshoot issues related
↪ RestRho, providing information, code examples, and explanations
↪ tailored to this technology, especially in generating JSON code.
```

To also establish guidelines that the LM must follow when interacting with users, the following guidelines have been defined:

```
When interacting with users, the model should ask clarifying questions to
↪ understand the specific context or problem the user is facing with
↪ SQLRho and DBRestRho. It should aim to provide concise yet
↪ comprehensive responses that are immediately useful for the user's
↪ inquiry.
```

Regarding personalization, it refers to interaction preferences that can include adjusting the level of formality, the technical complexity of the responses, or the communication style. For our corpus, the following has been defined:

```
The model should maintain a professional tone, akin to a highly
↪ knowledgeable software engineer, but also be approachable and willing
↪ to explain complex concepts in simpler terms when needed.
```

2. RestRho Knowledge Corpus

The RestRho knowledge corpus included the following: a description of RestRho, and a summary of its JSON-DSLs: SQLRho and DBRestRho. Next, the programs *PrgDBRest* (code DBRestRho) and *PrgSQL* (code SQLRho) are added, defined and explained in Section III.B.2. Each program is delimited by the triple quotation mark "```".

3. DB Sample Knowledge Corpus

The DB Sample knowledge corpus provides a structured set of texts to train LMs. Its purpose is to facilitate learning how to program in *SQLRho*. The first part of the corpus involves setting up the available tables by defining CREATE TABLE statements. The second part establishes the rules for defining an operation on a table. Finally, for each table, a list of operations is set up, including, for each operation, the equivalence between an SQL statement and an SQLRho statement (in JSON-format).

The DBRestRho includes the creation of **Prompts** based on the information from a database, its tables, operations, and templates, to construct structured SQLRho program text. A prompt is built using a Handlebar template.

To dynamically build the RestRho knowledge corpus, in the program *PrgDBRest* the "Sample" entry is set to Prompts, which uses the "sample.han"[14] template. To illustrate this template, the following segment allows you to write the list of operations for a table:

```
### Table '{{this.Name}}'
For the table **{{this.Name}}** there is the following list of JSON
↪ operations:
    {{#each this.Operations}}
    {{this.Index}}. Name: '{{this.Name}}'
      - Sentence SQL: {{{this.SQL}}}
      - Sentence SQLRho: {{{this.Body}}}
{{/each}}
```

The result for the operation in Fig. 3 is shown below:

```
Name: 'SelectPillTemplate'
     - Sentence SQL: SELECT NAME,SURNAME,AGE,SEX FROM person ORDER BY
    ↪ SURNAME asc
    - Sentence SQLRho: {"Definition": "SelectAction", "Action": "pills",
    ↪ "Clauses":{"Fields": "NAME,SURNAME,AGE,SEX", "OrderBy": "SURNAME
    ↪ asc"}, "Transform":{"Type": "Map", "Field": "Rows"}, "Template":
    ↪ "person.pills", "Table": "person"}
```

4. DB Employees Knowledge Corpus

To initiate the LLMs evaluation assignment, it is necessary to establish the DB Employees knowledge corpus. This involves providing the LM with information about the available tables in the DB Employees (*departments*, *employees*, *salaries*, *titles*, *dept_manager*, and *dept_emp*), that is, the same database definition given to the students.

To create this corpus, the program *PrgDBRest* sets the "Employees" entry in Prompts, where the "employees.han"[15] template was used. Initially, the available tables were configured by defining CREATE TABLE statements. Subsequently, metadata and persistent prompts were added to commence the evaluation of the LLMs.

IV. Results

To measure the LMs performance in our case study, we used Human Evaluation to rate it based on their knowledge to verify if the solution that LMs and student provides are relevant (see Section II.D).

Thus, to assess the performance of participants and LMs in the case study, we relied on both quantitative and qualitative measures. Using a standardized rubric (described in Section III.E), human evaluators assessed task success based on correctness, completeness,

[14] https://www.devrho.com/restrho/sample.han

[15] https://www.devrho.com/restrho/employees.han

TABLE II. GRADES OBTAINED FOR EACH STUDENT AND LMS

	1A	1B1	1B2	1B3	1B4	1B5	1B6	1C1	1C2	1C3	1C4	1C5	2A	2B	2C	2D	2E	2F	Avg.	SD
P1	1.00	0.67	0.33	0.67	0.67	0.50	0.50	0.50	0.67	0.67	0.33	0.75	0.00	0.00	0.00	0.00	0.00	0.00	0.40	0.33
P2	1.00	0.67	0.33	0.33	0.33	0.50	0.50	0.50	0.33	0.00	0.33	0.25	0.00	0.00	0.40	0.10	0.00	0.00	0.31	0.27
P3	1.00	0.33	1.00	0.67	0.67	0.50	0.50	0.50	0.67	0.67	0.33	0.75	0.60	0.70	0.60	0.30	0.80	0.80	0.63	0.20
P4	1.00	0.33	0.33	0.33	0.00	0.25	0.25	0.50	0.33	0.33	0.00	0.25	0.80	0.70	0.70	0.80	0.40	0.40	0.43	0.27
P5	1.00	0.67	0.33	0.33	0.33	0.25	0.50	0.50	0.67	0.33	0.33	1.00	0.20	0.40	0.10	0.10	0.20	0.20	0.41	0.27
P6	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	1.00	0.75	1.00	0.90	0.90	1.00	1.00	1.00	0.95	0.10
P7	1.00	0.00	0.67	0.67	0.67	0.50	0.75	1.00	0.67	0.67	0.00	0.50	0.60	0.70	0.50	0.40	0.40	0.00	0.54	0.30
P8	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.80	1.00	1.00	0.95	0.10
P9	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.90	0.80	1.00	0.94	0.10
P10	1.00	0.00	1.00	0.67	0.67	0.50	0.50	0.50	0.67	0.67	0.33	0.50	0.60	0.60	0.60	0.10	0.80	0.80	0.58	0.26
P11	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	1.00	0.75	1.00	0.90	0.90	0.90	0.80	0.80	0.92	0.10
P12	1.00	0.67	0.33	0.33	0.33	0.50	0.25	0.50	0.33	0.33	0.33	0.25	0.60	0.70	0.40	0.40	0.60	0.00	0.44	0.22
P13	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	1.00	0.75	1.00	1.00	0.90	0.90	1.00	1.00	0.95	0.10
P14	1.00	1.00	0.67	0.67	0.67	0.50	0.50	1.00	1.00	0.67	0.67	0.75	0.20	0.00	0.00	0.00	0.00	0.00	0.52	0.39
P15	1.00	1.00	0.67	0.67	0.33	0.75	0.50	1.00	0.67	0.67	0.00	0.50	1.00	0.90	0.00	0.50	0.80	0.00	0.61	0.34
P16	1.00	0.67	0.33	0.67	0.33	0.50	0.50	0.50	0.33	0.33	0.33	0.50	0.60	0.70	0.30	0.30	0.80	0.60	0.52	0.20
P17	1.00	0.67	0.33	0.33	0.33	0.25	0.25	0.50	0.67	0.33	0.33	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.33
P18	1.00	1.00	0.67	1.00	0.67	0.50	0.75	1.00	0.67	0.67	0.67	0.50	1.00	0.50	0.50	0.00	0.40	1.00	0.69	0.28
P19	1.00	1.00	1.00	0.67	1.00	0.75	0.75	1.00	1.00	0.67	0.00	0.75	1.00	0.00	0.70	0.50	1.00	0.00	0.71	0.36
P20	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	1.00	0.75	1.00	0.80	0.80	0.80	0.60	0.60	0.88	0.15
P21	1.00	0.33	0.33	0.67	0.67	0.50	0.50	0.50	0.67	0.67	0.33	1.00	0.20	0.50	0.20	0.10	0.20	0.00	0.46	0.28
P22	0.60	0.33	0.33	0.33	0.33	0.25	0.25	0.50	0.33	0.33	0.33	0.50	0.80	0.50	0.00	0.60	0.40	0.40	0.40	0.17
P23	1.00	1.00	0.67	0.67	0.67	0.75	0.50	1.00	0.67	0.67	0.67	0.50	0.60	0.60	0.80	0.00	0.00	0.00	0.60	0.31
P24	1.00	0.67	0.67	0.67	0.67	0.25	0.50	0.50	0.67	0.67	0.33	0.25	0.60	0.70	0.00	0.20	0.60	0.20	0.51	0.25
P25	1.00	1.00	1.00	0.67	0.67	0.75	0.75	1.00	0.67	0.67	1.00	0.50	0.60	0.00	0.70	0.60	0.20	0.20	0.66	0.30
P26	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	1.00	0.75	1.00	0.90	0.90	0.90	0.80	0.80	0.92	0.10
P27	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.67	0.67	1.00	0.50	0.40	0.60	0.70	0.30	0.40	0.40	0.42	0.37
P28	1.00	0.67	0.33	0.67	0.67	0.25	0.50	0.50	0.67	0.00	0.00	0.75	0.60	0.70	0.30	0.10	0.60	0.40	0.48	0.27
P29	1.00	0.33	1.00	0.67	0.67	0.50	0.50	0.50	0.67	0.67	0.33	0.75	0.60	0.70	0.60	0.10	0.80	0.80	0.62	0.23
P30	1.00	0.67	0.67	0.67	0.67	0.50	0.50	0.50	1.00	0.67	0.33	0.75	1.00	0.90	0.40	0.30	1.00	1.00	0.70	0.24
P31	1.00	0.67	1.00	0.67	0.67	0.50	0.50	0.50	1.00	0.67	0.33	0.75	0.60	0.70	0.40	0.30	1.00	0.00	0.63	0.27
P32	1.00	1.00	0.67	1.00	0.67	0.50	0.75	1.00	0.67	0.67	0.67	0.50	0.80	0.50	0.50	0.00	0.40	0.40	0.65	0.26
P33	1.00	0.33	0.33	0.33	0.33	0.25	0.25	0.50	0.33	0.33	0.33	0.25	0.40	0.60	0.20	0.10	0.20	0.20	0.35	0.20
P34	1.00	0.33	0.67	0.67	0.33	0.25	0.50	0.50	0.67	0.67	0.33	0.75	0.20	0.50	0.40	0.10	0.20	0.00	0.45	0.26
P35	1.00	0.33	0.33	0.33	0.00	0.25	0.25	0.50	0.33	0.33	0.00	0.25	1.00	0.90	0.50	0.70	0.40	0.40	0.43	0.29
P36	1.00	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	1.00	0.00	0.75	1.00	1.00	0.80	0.80	0.40	0.40	0.81	0.28
P37	1.00	1.00	0.67	1.00	0.67	0.75	0.50	0.50	1.00	0.67	0.67	1.00	0.60	0.70	0.40	0.30	0.60	0.40	0.69	0.23
P38	1.00	0.67	0.33	0.33	0.33	0.50	0.50	0.50	0.67	0.67	0.33	0.75	0.00	0.00	0.00	0.00	0.00	0.00	0.37	0.31
P39	1.00	0.67	0.67	0.67	0.33	0.50	0.50	0.50	0.67	0.33	0.33	0.25	0.60	0.70	0.50	0.30	1.00	1.00	0.58	0.24
ChatGPT-4	1.00	1.00	0.67	0.33	0.33	0.75	0.75	0.50	0.67	0.67	0.67	1.00	1.00	0.90	0.90	0.90	1.00	1.00	0.78	0.23
ChatGPT-4o	1.00	1.00	1.00	0.67	0.67	0.75	0.75	1.00	1.00	1.00	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.92	0.14
Perplexity	1.00	0.67	0.33	0.67	0.67	0.50	0.25	0.50	0.67	0.67	0.67	0.75	0.80	0.90	0.90	0.90	1.00	1.00	0.71	0.22
Copilot	1.00	0.33	0.00	0.00	0.00	0.00	0.25	0.00	0.00	0.33	0.00	0.75	0.80	0.70	0.70	0.70	0.80	0.60	0.39	0.36
Phind	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.80	0.90	0.90	0.90	0.80	0.60	0.33	0.43
Llama3	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.80	0.90	0.90	0.90	0.80	0.20	0.31	0.42
Phi-3	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.80	0.90	0.90	0.80	0.00	0.00	0.24	0.41

and adherence to task requirements. This ensured consistency and objectivity in the evaluation process. Furthermore, the alignment of results with the research questions was verified by analyzing trends and insights presented in Table II and Table III. This approach demonstrated the feasibility and limitations of using LMs alongside DSLs for complex assignments, offering valuable insights for future work.

Consequently, Table II presents the grades normalized in per cent per one (from 0 to 1) obtained by the participants of the experiment. The table shows two different groups, namely: students (P1 to P39) and the selected LMs (both LLMs and SLM). The results are organised in columns corresponding to the 18 tasks (see Table I) that the participants performed. In addition, columns are included showing the average task success (Avg.), and the standard deviation (SD). For its part, Table III provides the average and standard deviation for each task, presenting the results for all participants combined (students and LMs), as well as separately for students only and for LMs only.

The overall performance results revealed clear trends. The 39 students achieved an average task success rate of 0.6% (SD: 0.14%) across all tasks, demonstrating consistent performance in the initial tasks (e.g., 1A and 1C1). Conversely, LMs presented a broader range of variability, with an average success rate of 0.52% (SD: 0.28%). Notably, students excelled in SQLRho-related tasks (1A–1C5), while LMs performed

TABLE III. AVERAGE AND STANDARD DEVIATION FOR THE GRADES IN EACH TASK

	1A	1B1	1B2	1B3	1B4	1B5	1B6	1C1	1C2	1C3	1C4	1C5	2A	2B	2C	2D	2E	2F	Avg.	SD
Avg Total	1.08	0.63	0.59	0.59	0.53	0.47	0.48	0.64	0.65	0.57	0.43	0.59	0.74	0.82	0.72	0.61	0.55	0.44	0.62	0.15
SD Total	0.06	0.35	0.34	0.31	0.32	0.25	0.23	0.31	0.30	0.30	0.35	0.28	0.32	0.32	0.33	0.35	0.35	0.39	0.30	0.07
Avg. Students	0.99	0.68	0.66	0.67	0.60	0.51	0.53	0.72	0.72	0.62	0.47	0.62	0.64	0.59	0.48	0.37	0.53	0.42	0.60	0.14
SD Students	0.06	0.32	0.30	0.26	0.30	0.21	0.19	0.25	0.24	0.27	0.35	0.23	0.34	0.32	0.32	0.33	0.34	0.38	0.28	0.08
Avg. LM	1.00	0.43	0.29	0.24	0.24	0.29	0.29	0.29	0.33	0.38	0.29	0.50	0.86	0.89	0.89	0.87	0.77	0.63	0.52	0.28
SD LM	0.00	0.46	0.40	0.32	0.32	0.37	0.34	0.39	0.43	0.40	0.36	0.48	0.10	0.09	0.09	0.10	0.35	0.41	0.30	0.15

better in Handlebars-related tasks (2A–2F), particularly with GPT-4o achieving results comparable to the highest-performing students.

Grades for the students reflect considerable variability in their performance concerning specific task completion. Tasks 1A, 1C1, and 1C2 emerged with notably high average success. The low standard deviation in task 1A indicates significant consistency in success achievement among the majority of participants, except for participant P22. This suggests a solid understanding and robust skills in the initial configuration of SQLRho and query operations within the 'employees' table.

Conversely, tasks 2D, 2F, and 2C revealed lower success rates in execution. The high standard deviations, particularly due to extremely low scores from over 10 participants, point to significant challenges for most individuals in handling Handlebars templates (TT4), associated with operations within the 'departments' and 'employees' tables. These tasks may require a more detailed approach and additional support in future iterations of the experiment.

On the other hand, LMs showed a distinct behaviour. Moreover, it is remarkable the difference in performance among GPT-based LLMs and the others, being GPT-4o the one that best performs, obtaining similar grades to those we can see for the best of the students. It is surprising to see how Copilot, which after all is a front-end to ChatGPT platform for Microsoft's Office suite, has achieved such different results.

For their part, Llama3 together with Phind and Phi-3, have obtained very poor performance for the first tasks. Maybe, the case of Phind is the most remarkable of all, since it is specially trained for programming tasks. However, as will be discussed later, far from making use of the designed DSLs, this LM used main-stream GPPLs and DSLs such as SQL, which may be comprehensible given its training process. The grades of Phi-3 are also interesting, as although it is a SLM, it performed similarly to the previous ones.

Analysing the grades obtained by LMs generally, they presented significantly lower success rates for the initial tasks (1A to 1C5), with some LMs obtaining extremely low results. This indicates that LMs struggled in the first tasks of the experiment, involving basic configurations and operations of SQLRho and queries in the 'departments' and 'employees' tables. However, in tasks 2A to 2F, they showed notable improvement in their results, with high average success rates in most tasks and lower standard deviations, reflecting better consistency in their performance due to the training of the LLMs in programming tasks.

To show some examples of answers, Appendix presents the responses of the LMs to question 1B1. These responses reveal an error pattern among the lower-performing LMs (grade 0): they generate an element containing a complete query using standard SQL rather than employing the required JSON structure. The reason for these meagre results is that they failed to use the basic JSON structure of an SQLRho operation and rapidly tended to reply with an SQL query. Another feasible cause is that they did not differentiate between the grammar of DBRestRho and SQLRho; in the former, there is an 'SQL' tag in the definition for managing the template of an SQL instruction, while in a SQLRho operation on a table, there is a JSON structure for the SQL clauses.

Despite working with the same data, GPT-4o demonstrated a superior ability to correctly interpret SQLRho grammar and query operations, demonstrating a greater aptitude for distinguishing DSLs from mainstream GPPLs and DSLs such as SQL. This suggests a better capacity for generalization and processing of complex structures such as JSON. In contrast, the other LMs appeared to prioritize the handling of standard SQL, which limited their adaptability to the DSLs. The use of a common training corpus underscores the intrinsic characteristics of each LM as key determinants of observed performance variations.

The comparative analysis between students and LMs reveals key differences. Students showed greater consistency and success in the initial tasks of the experiment, while LMs presented more variable performance, significantly improving in the later tasks. Tasks 2D, 2F, and 2C were challenging for students and LMs, although LMs achieved better results on average in tasks 2A to 2F. In terms of grades and evaluations, students tended to obtain higher and more consistent grades in the initial tasks, while LMs improved their grades in the later tasks.

Finally, as a consequence of analysing the result, at this stage, we can answer the research questions set out in Section I.

- **RQ1. Is it possible to use current LMs together with DSLs for assignments that involve the design and implementation of RESTful APIs?** The answer to this research question is affirmative. The main requirement is a good definition of the DSL and a proper prompting process to provide the LM with metadata, persistent prompting and a solid knowledge corpus. This step is vital for a good performance of the LM.
- **RQ2. How do LMs perform against real students for assignments on designing and implementing RESTful APIs using DSLs?** For this research question, the answer is that both LLMs and SLMs can achieve excellent results depending on the task and the prompt provided. Some LMs outperform others, and this performance must be evaluated for each specific problem.

V. DISCUSSION

The related works section of this article has evidenced the growing importance of integrating LMs and DSLs into Computer Science education. This integration not only facilitates the learning of fundamental concepts but also allows students to experiment with technologies that are transforming the industry. With an engineering process in the use and customization of LM for specific applications, LMs can provide instant and personalized student feedback, adapting to their knowledge levels and learning styles.

Based on the results we obtained with our case of study, at this stage it is possible to point out some issues that should be addressed in future experimentation and use cases. First, as previously stated, the combination of DSLs with LMs implies engineering the metadata,

persistent prompts, and knowledge corpus so that the latter can use the DSLs with guarantees. Moreover, this adds complexity, since not only the DSL itself must be validated, but also the completeness, clarity and disambiguation of the information provided through prompts to the LMs to allow them for optimal performance. That means opening up a whole new interesting research line, that is, the design of tests to validate prompts to maximise the performance of the LM.

Likewise, another important issue that should be taken into account in the engineering process when applying and using LMs (not only in education but in any domain), is to identify and define the best metrics that allow the validation of different LMs as well as to explore multiple outputs according to the hyperparameterization each LM provides. The methodology applied in this case study to compare the performance of LMs on specific JSON-DSL tasks in a reduced learning context, against the solutions provided by the 39 students, has facilitated the identification of areas where LMs need improvement (for instance the SQLRest operations in our case study) and where they already excel (like the use of HTML template creation in our case study). This reveals that LM better performs on tasks where they can use the knowledge they were trained for, but not on tasks where new knowledge is necessary (like new JSON-DSLs designed in our case study). Therefore, better-prompting LMs to handle complex SQLRho operations is crucial in developing RestRho-based solutions.

While the sample size of 39 students provided valuable insights into the comparative performance of LMs and human participants, it represents a limitation in terms of generalizability. As this study is exploratory in nature, future work should aim to validate findings with larger and more diverse participant groups to strengthen the robustness of the conclusions. Expanding the sample would also allow for a more granular analysis of factors such as prior programming experience, familiarity with DSLs, and demographic variability.

This study acknowledges potential biases inherent in its design. The student participants, all in their final year of a Computer Engineering program, represent a relatively homogeneous group, which may not reflect the broader diversity of learners in Computer Science. Similarly, the selected LMs were among the most prominent general-purpose and programming-specific at the time of the study, potentially excluding emerging LMs or alternative paradigms. Additionally, the standardized rubric used for evaluation, while ensuring consistency, may have limited the recognition of unconventional but valid approaches in task completion.

VI. Conclusion and Future Works

Since Computer Science students use GPPLs and DSLs to solve problems, we need to first to determine how well LMs perform compared to students. To investigate this, we conducted a case study on the design and implementation of RESTful APIs by undergraduate Computer Science students.

In this article, we have detailed a case study with Computer Science undergraduates, designing, implementing and deploying two DSLs and some specific task assignments. Up to 39 students and 5 different LMs completed the assignment, and instructors graded both sets of solutions so that the LMs scores were compared to those of the students to measure the LMs performance.

Implementing evaluation criteria uniformly was essential to obtain more accurate and useful comparisons between different LLM and their possible applications. This allowed for a better understanding of each LM's capabilities and limitations, thus facilitating the identification of areas for improvement and making informed decisions about the use of these LMs in specific contexts. Adopting these uniform criteria can lead to developing standards for LLM evaluation, allowing clear expectations and goals for future LM development.

The results highlight that well-defined DSLs and an effective prompting process are essential for LMs to perform well. This involves providing the LM with metadata, consistent prompts, and a robust knowledge base. With the right prompts, both LLMs and SLMs can achieve excellent results, depending on the task. In the presented case study, LMs must be rigorously prompted to handle complex operations. While some LMs can manage the tasks with relative accuracy, others struggle to generate correct and coherent results. This is evidenced by the high variability in scores. Among the tested LMs, it is worth mentioning that GPT-based LMs performed these tasks almost perfectly.

Although the findings provide valuable insights, the relatively small sample size limits the generalizability of the results. Expanding the participant pool in future studies, incorporating students from diverse backgrounds and skill levels, will enable more robust statistical analyses and uncover potential nuances in LMs performance. Additionally, addressing potential biases introduced by the specific selection of LMs and configurations will be crucial. Broader experimentation with various LMs, including those optimized for different purposes, will ensure that results are representative and applicable to diverse scenarios.

Given that our research questions delimited the prompt context, in future works, this must be better studied and improved. LMs need to be provided and customized with a greater number of examples that include more complex and detailed operations while using the defined DSLs. Additionally, these examples must be provided in a systematically way to cover a wide range of scenarios and variations in queries and database structures, ensuring that models can generalize their learning in different contexts and provide solutions to specific programming problems.

In this same concern, the automatic generation of prompts by the tools is an interesting strategy for building a robust and effective corpus for LMs. These autogenerated prompts can help and guide the LMs in understanding and executing specific tasks, significantly improving their ability to handle complex operations (in the case study presented in this article, to create specific Handlebars templates for database information). A well-structured corpus ensures that LMs can learn from clear and well-defined examples, reducing ambiguity and improving the accuracy of generated responses. This autogeneration, in the future, allows for continuous improvement of the training corpus. As new needs are identified or areas of difficulty are discovered, prompts can be modified, expanded, or added to address these aspects, enriching the corpus and improving LM performance.

Another future work regards on analyzing how to make each LM better perform for each specific tasks. The uniform evaluation carried out in our case study may have penalized some LMs against others, due to each LM follows different approaches and goals. Therefore, we are willing to design specific prompts and hyperparameter exploration for each LM to find the best performance for each one.

As a challenge at a technical level, we want to explore developing and evaluating the performance of LMs in creating templates that use specific frameworks based on Material Design principles. This effort would include implementing and testing complex design tasks that require frameworks such as Material Design for Bootstrap, Material-UI, Angular Material, and Semantic UI. Thus, it is necessary to improve the prompting of LMs in managing and creating templates. Additionally, specific prompts must be included to automatically generate these templates and integrate them as outputs in the RestRho system.

Appendix A. DBRestRho Grammar

This appendix briefly shows the **DBRestRho** and its associated classes as shown in the CodeRho 1.

This JSON-DSL has been defined to specify what SQLRho definitions the RESTful API can use, the templates for the HTTP requests (endpoints) and related SQL statement to manage the data. Also, to easy the labour of prompting LMs, the JSON-DSL allows provide a prompt template.

Consequently, a DBRestRho program has five attributes:

- **Name**: name of the Server/Program.
- **Description**: brief description of the definitions.
- **Databases**: array of SQLRho programs, where each *SQLRhoCodeI* follows the format:
  - "*alias.connector.json*",
  - alias is the program alias, and
  - textitconnector is either "mariadb" or "mysql"
- **Definitions**: map of HTTP requests, each linked to a SQL statement, a transformation result, and an associated Handlebars template. Each definition (*definitionI*) includes:
  - **Method** (HTTP request method "get", "post", "put" or "delete"),
  - **Parameters** (list of parameters for the HTTP request and SQL statement, i.e.: {"params": ["key"]}),
  - **Default** (default options for a SQLRho program, i.e.:{"Fields": "*"}),
  - **Path** (URL structure of the HTTP request with Handlebars). Each Path (templatePath) defined as:
    - /{{db}}/{{table}}/{{action}}: where {{db}} is the database name, {{table}} is the table name, and {{action}} is action defined in a SQLRho table.
    - /{{db}}/{{table}}/*resource*: a fixed resource. For example, *resource*=/all.
    - /{{db}}/{{table}}/*params*: the list of parameters defined in **Params**. For example, *params*=/:key.
  - **SQL** (SQL template in Handlebars). Each SQL template (*templateSLQ*) in Handlebars with predefined tags: Fields, Distinct, Join, InnerJoin, LeftJoin, RightJoin, FullJoin, Where, GroupBy, Having, OrderBy, Limit, are added to attributes. For example:

    templateSql = "UPDATE {{Table}} SET {{{Fields}}} WHERE {{key}}=:key".
- **Prompts**: map for creating Prompts from a database, based on Handlebar template. This is really useful for creating the corresponding part of the prompt for the LMs. Each prompt (promptI) includes:
  - **Method** (HTTP request method "get", "post", "put" or "delete"),
  - **Action** (action of the petition),
  - **Database** (reference to a database and information available about tables and operations, where "*" applies all available operations and [..."*operationJ*", ...] specifies a particular list),
  - **Template** (Handlebars template that allows specifying the content of promptI with the information available from the database), and,
  - **File** (output to a text file when the template is applied to the database information).

RhoCode 1. Grammar Structure DBRestRho

```
{ "Name": "name",
  "Description": "description",
  "Databases": ["SQLRhoCode1", "...", "SQLRhoCodeI", "...",
   ↪ "SQLRhoCodeN"],
  "Definitions": {
    "..."
    "definitionI": {
      "Method": "method",
      "Parameters": {"params": ["param1", "...", "paramM"]},
      "Default": {"option1": "*", "..."},
      "Path": "templatePath",
      "SQL": "templateSLQ"
    },
    "..."
  },
  "Prompts": {
    "...",
    "promptI": {
      "Method": "method", "Action": "action",
      "Database": {
        "Name": "sample",
        "Templates": "*",
        "Tables": {
          "...",
          "tableI" : "*",
          "...",
          "tableJ": ["...", "operationJ", "..."],
          "..."
        }
      },
      "Template": "templateFile (*.han)",
      "File": "outputFile (*.txt)"
    },
    "..."
}
```

Appendix B. SQLRho Grammar

This appendix provides a brief description for the **SQLRho** grammar and associated classes are established, RhoCode 2 shows a summary explanation of a *SQLRho program*.

SQLRho is the JSON-DSL designed to configure DB connections (particularly MariaDB or MySQL) and define the set of operations for the database tables necessary to manage the requests defined in DBRestRho.

In SQLRho we can specify basic operations at record level (Insert, Update, Delete, and Select) and at table level (Where and SelectAll); as well as custom operations to support configuration, transformation and the application of output templates to render the results in the web browser.

Consequenlty, SQLRho has eight attributes:

- **Name**: part of the Path defined for the HTTP request
- **Description**: a brief description of the program
- **Connector**: the database type, as *mariadb* or *mysql*
- **EnableToken**: enables token use for HTTP requests, *true* or *false*
- **Pool**: defines the standard connection pool to a MariaDB or MySQL database
- **Registry**: defines the database record table
- **Templates**: defines Handlebars templates for transforming table operation results. Each template includes the name, directory, and filename
- **Tables**: sets the tables targeted by HTTP request. For each table (*tableI*), four attributes are defined:
  - **Encrypt** (list of fields to be encrypted using AES-256 in CBC mode), **Decrypt** (list of fields to be decrypted on the server using AES-256 in CBC mode),

- **Uses** (list of basic operations enabled for the table: Insert, Update, Delete, Select, Where, SelectAll), and
- **Operations** (list of custom operations including configuration, parameters, transformations, and templates). Each operation (operationI) includes:
  - **Name** (name of the definition specified in the DBRestRho program);
  - **Action**: action of the petition.
  - **Parameters**: list of parameters to be used in the request.
  - **Clauses**: list of elements to build the SQL statement: Fields, Distinct, Join, InnerJoin, LeftJoin, RightJoin, FullJoin, Where, GroupBy, Having, OrderBy, Limit.
  - **Transform**: where **Type** is the type mandatory of transformation, **Map** is return a map with the result, **Group** is group the result by a field, or **Position** it returns an object from the array.
  - **Template**: refers to a name nameTemplateI the list of Templates available.

RhoCode 2. Grammar Structure SQLRho

```
{
  "Name": "name",
  "Description": "description",
  "Connector": "connector",
  "EnableToken": "true | false",
  "Pool": { "host":"…", "user":"…", "password":"…", "database":"…",
   ↪ "connectionLimit": "<int>" },
  "Registry": { "Table":"…", "User":"…", "Password":"…",
   ↪ "ValidityMinutes": "<int>" },

  "Templates": { "...", "nameTemplateI": {"Dir":"dirI",
   ↪ "File":"nameFileI"}, "..." },

  "Tables": {
    "...",
    "tableI": {
      "Encrypt": ["field1", "...", "fieldK", "..."],
      "Decrypt": ["field1", "...", "fieldJ", "..."],
      "Uses": ["use1", "...", "useI", "...", "useN"],
      "Operations": {
        "...",
        "operationI": {
          "Name": "name operation",
          "Action": "action",
          "Parameters": { "query": ["param1", "...", "paramJ"] },
          "Clauses": { "Fields": "…", "Where": "…", "OrderBy": "…", "..."
           ↪ },
          "Transform": { "Type": "type", "Map": "...", "Group": "..." },
          "Template": "nameTemplateI"
        }
      }
    },
    "..."
  }
}
```

Appendix C. LMs Results for Task 1B1 From Table I

This appendix presents the fragment of the results from the LMs for question **1B1** from Table I:

```
1B1. Query the current number of employees in each department, sorted
↪ alphabetically by department name.
```

Fragments C1 and C2 display the responses of ChatGPT-4 and ChatGPT-4o, respectively, to question 1B1. Both LMs correctly structured and resolved the query as required in SQLRho.

Fragment C1. ChatGPT-4. Grade: 1.00

```
...
  "Operations": {
    "SelectEmployeesNumbers": {
      "Definition": "SelectAction",
      "Action": "employeeNumbers",
      "Clauses": {
        "Fields": "D.dept_name, COUNT(E.emp_no) AS num_employees",
        "InnerJoin": {"dept_emp X": "D.dept_no=X.dept_no", "employees
         ↪ E": "X.emp_no=E.emp_no"},
        "GroupBy": "D.dept_name",
        "OrderBy": "D.dept_name ASC"
      }
    },
...
```

Fragment C2. ChatGPT-4o. Grade: 1.00

```
...
      "Operations": {
        "SelectEmployeesNumbers": {
          "Definition": "SelectAction",
          "Action": "employeeNumbers",
          "Clauses": {
            "Fields": "D.dept_name, COUNT(E.emp_no) AS num_employees",
            "InnerJoin": {"dept_emp X": "D.dept_no=X.dept_no", "employees
             ↪ E": "X.emp_no=E.emp_no"},
            "GroupBy": "D.dept_name",
            "OrderBy": "D.dept_name ASC"
          }
        },
...
```

Fragment C3 presents Perplexity's response to question 1B1. While it did not construct the complete solution, the structure proposed by the LM is correct; however, it included an additional field, "Table."

Fragment C3. Perplexity. Grade: 0.67

```
...
B1: SelectEmployeesNumbers
Descripción: Consulta del número de empleados actuales de cada
 ↪ departamento ordenados por el nombre del departamento de forma
 ↪ ascendente.
SQLRho:
{
    "Definition": "SelectAction",
    "Action": "employeesNumbers",
    "Clauses": {
        "Fields": "D.dept_name, COUNT(E.emp_no) AS employee_count",
        "GroupBy": "D.dept_name",
        "OrderBy": "D.dept_name ASC"
    },
    "Table": "departaments"
}
...
```

Fragment C4 presents Copilot's response to question 1B1. It did not construct the complete solution, only added the action, and created two tags, "type" and "select," which do not belong to the SQLRho grammar.

Fragment C4. Copilot. Grade: 0.33

```
...
    "Operations": {
    "SelectEmployeesNumbers": {
      "action": "employeesNumbers",
      "type": "select",
      "query": "SELECT dept_name, COUNT(*) as employee_count FROM
       ↪ departaments INNER JOIN dept_emp ON departaments.dept_no =
       ↪ dept_emp.dept_no GROUP BY dept_name ORDER BY dept_name ASC"
    },
...
```

Fragments C5, C6, and C7 present the responses of Phind, Llama3, and Phi-3, respectively, to question 1B1. These LMs did not understand

the structure of the SQLRho grammar and provided a response based on their knowledge of SQL.

Fragment C5. Phind. Grade: 0.00

```
...
    "Operations": {
        "SelectEmployeesNumbers": "SELECT D.dept_name, COUNT(E.emp_no) AS
         ↪ employees_count FROM departaments D INNER JOIN dept_emp X ON
         ↪ D.dept_no = X.dept_no INNER JOIN employees E ON X.emp_no =
         ↪ E.emp_no WHERE X.to_date = '9999-01-01' GROUP BY D.dept_name
         ↪ ORDER BY D.dept_name ASC",
...
```

Fragment C6. Llama3. Grade: 0.00

```
...
  "Operations": [
    {
      "Name": "SelectEmployeesNumbers",
      "SQL": "SELECT D.dept_name, COUNT(E.emp_no) AS num_employees FROM
       ↪ departaments D INNER JOIN dept_emp X ON D.dept_no = X.dept_no
       ↪ INNER JOIN employees E ON X.emp_no = E.emp_no GROUP BY
       ↪ D.dept_name ORDER BY D.dept_name ASC",
      "Type": "SELECT"
    },
...
```

Fragment C7. Phi-3. Grade: 0.00

```
...
  "Operations": [
    {
      "B1": {
        "Description": "Consulta del número de employees actuales de cada
         ↪ departament ordenados por dept_name de forma ascendente.",
        "SQL": "SELECT D.dept_no, COUNT(E.emp_no) AS employee_count FROM
         ↪ departaments D INNER JOIN dept_emp X ON D.dept_no = X.dept_no
         ↪ GROUP BY D.dept_no ORDER BY D.dept_name ASC;"
      }
    },
...
```

References

[1] OpenAI, "GPT-4 technical report," OpenAI, 2023. [Online]. Available: https://cdn.openai.com/papers/gpt-4.pdf, doi: 10.48550/arxiv.2303.08774.

[2] A. Gilson, C. W. Safranek, T. Huang, V. Socrates, L. Chi, R. A. Taylor, D. Chartash, "How does ChatGPT perform on the united states medical licensing examination? the implications of large language models for medical education and knowledge assessment," *JMIR Medical Education*, vol. 9, p. e45312, Feb 2023, doi: 10.2196/45312.

[3] D. M. Katz, M. J. Bommarito, S. Gao, P. Arredondo, "Gpt-4 passes the bar exam," *382 Philosophical Transactions of the Royal Society A*, 2024, doi: 10.2139/ssrn.4389233.

[4] M. Bernabei, S. Colabianchi, A. Falegnami, F. Costantino, "Students' use of large language models in engineering education: A case study on technology acceptance, perceptions, efficacy, and detection chances," *Computers and Education: Artificial Intelligence*, vol. 5, p. 100172, 2023, doi: https://doi.org/10.1016/j.caeai.2023.100172.

[5] M. Liu, L. J. Zhang, C. Biebricher, "Investigating students' cognitive processes in generative ai-assisted digital multimodal composing and traditional writing," *Computers & Education*, vol. 211, p. 104977, 2024, doi: https://doi.org/10.1016/j.compedu.2023.104977.

[6] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn, G. Kasneci, "ChatGPT for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, 2023, doi: 10.1016/j.lindif.2023.102274.

[7] R. Gao, H. E. Merzdorf, S. Anwar, M. C. Hipwell, A. R. Srinivasa, "Automatic assessment of text-based responses in post-secondary education: A systematic review," *Computers and Education: Artificial Intelligence*, vol. 6, p. 100206, 2024, doi: https://doi.org/10.1016/j.caeai.2024.100206.

[8] Phind, "Introducing phind-70b – closing the code quality gap with gpt-4 turbo while running 4x faster," 2024. [Online]. Available: https://www.phind.com/blog/introducing-phind-70b, Accessed on April 23, 2024.

[9] Meta, "Don't settle for less. build your interview skills and take your career to the next level," 2024. [Online]. Available: https://codellama.dev, Accessed on April 23, 2024.

[10] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. Sutherland Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, O. Vinyals, "Competition-level code generation with AlphaCode," *Science (New York, N.Y.)*, vol. 378, p. 1092—1097, December 2022, doi: 10.1126/science.abq1158.

[11] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," 2020. doi: https://doi.org/10.18653/v1/2020.findings- emnlp.139.

[12] Tabnine, "Tabnine | the AI coding assistant that you control," 2023. [Online]. Available: https://www.tabnine. com/, Accessed on April 23, 2024.

[13] Anysphere, "Build software faster in an ide designed for pair-programming with ai," 2024. [Online]. Available: https://cursor.sh/, Accessed on April 23, 2024.

[14] GitHub, "Copylot," 2024. [Online]. Available: https://github.com/features/copilot, Accessed on April 23, 2024.

[15] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, W. Zaremba, "Evaluating large language models trained on code," 2021. [Online]. Available: https://arxiv.org/abs/2107.03374, doi: https://doi.org/10.48550/arxiv.2107.03374.

[16] D. Ankur, D. Atul, "Introducing amazon CodeWhisperer, the ml-powered coding companion," 2022. [Online]. Available: https://aws.amazon.com/es/blogs/machine- learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/, Accessed on April 23, 2024.

[17] Code4Me, "Code4Me," 2022. [Online]. Available: https://code4me.me/, Accessed on April 23, 2024.

[18] FauxPilot, "FauxPilot - an open-source alternative to GitHub copilot server," 2023. [Online]. Available: https://github.com/fauxpilot/fauxpilot, Accessed on April 23, 2024.

[19] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie- Ansley, E. A. Santos, ""it's weird that it knows what i want": Usability and interactions with Copilot for novice programmers," *ACM Transactions on Computer-Human Interaction*, vol. 31, pp. 1–31, nov 2023, doi: 10.1145/3617367.

[20] P. Haindl, G. Weinberger, "Students' experiences of using ChatGPT in an undergraduate programming course," *IEEE Access*, vol. 12, pp. 43519–43529, 2024, doi: 10.1109/ACCESS.2024.3380909.

[21] V. Kozov, G. Ivanova, D. Atanasova, "Practical application of ai and large language models in software engineering education," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 1, 2024, doi: 10.14569/IJACSA.2024.0150168.

[22] V. Parra, P. Sureda, A. Corica, S. Schiaffino, D. Godoy, "Can generative ai solve geometry problems? strengths and weaknesses of llms for geometric reasoning in spanish," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 8, no. 5, pp. 65–74, 2024, doi: https://doi.org/10.9781/ijimai.2024.02.009.

[23] J. Meyer, T. Jansen, R. Schiller, L. W. Liebenow, M. Steinbach, A. Horbach, J. Fleckenstein, "Using LLMs to bring evidence-based feedback into the classroom: AI-Generated feedback increases secondary students' text revision, motivation, and positive emotions," *Computers and Education: Artificial Intelligence*, vol. 6, p. 100199, 2024, doi: 10.1016/j.caeai.2023.100199.

[24] N. Rigaud Téllez, P. Rayón Villela, R. Blanco Bautista, "Evaluating chatgpt-generated linear algebra formative assessments," *International*

Journal of Interactive Multimedia and Artificial Intelligence, vol. 8, no. 5, pp. 75–82, 2024, doi: https://doi.org/10.9781/ijimai.2024.02.004.

[25] A. Mizumoto, M. Eguchi, "Exploring the potential of using an ai language model for automated essay scoring," *Research Methods in Applied Linguistics*, vol. 2, no. 2, p. 100050, 2023, doi: https://doi.org/10.1016/j.rmal.2023.100050.

[26] E. Latif, X. Zhai, "Fine-tuning ChatGPT for automatic scoring," *Computers and Education: Artificial Intelligence*, vol. 6, p. 100210, 2024, doi: 10.1016/j.caeai.2024.100210.

[27] M. Urban, F. Děchtěrenko, J. Lukavský, V. Hrabalová, F. Svacha, C. Brom, K. Urban, "ChatGPT improves creative problem-solving performance in university students: An experimental study," *Computers & Education*, vol. 215, p. 105031, 2024, doi: 10.1016/j.compedu.2024.105031.

[28] A. Desai, S. Gulwani, V. Hingorani, N. Jain, Karkare, M. Marron, S. R, S. Roy, "Program synthesis using natural language," in *Proceedings of the 38Th International Conference on Software Engineering*, ICSE '16, New York, NY, USA, 2016, p. 345–356, Association for Computing Machinery.

[29] K. Kolthoff, "Automatic generation of graphical user interface prototypes from unrestricted natural language requirements," in *2019 34Th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Los Alamitos, CA, USA, nov 2019, pp. 1234–1237, IEEE Computer Society.

[30] S. Kolahdouz-Rahimi, K. Lano, C. Lin, "Requirement formalisation using natural language processing and machine learning: A systematic review," in *Proceedings of the 11Th International Conference on Model-Based Software and Systems Engineering - Volume 1: MODELSWARD,*, 2023, pp. 237–244, INSTICC, SciTePress.

[31] A. Vernotte, A. Cretin, B. Legeard, F. Peureux, "A domain-specific language to design false data injection tests for air traffic control systems," *International Journal on Software Tools for Technology Transfer*, vol. 24, no. 2, pp. 127–158, 2022, doi: 10.1007/S10009-021-00604-4.

[32] E. Chavarriaga, F. Jurado, F. Díez, "An Approach to Build XML-Based Domain Specific Languages Solutions for Client-Side Web Applications," *Computer Languages, Systems & Structures*, vol. 49, pp. 133–151, 2017, doi: https://doi.org/10.1016/j.cl.2017.04.002.

[33] E. Chavarriaga, F. Jurado, F. D. Rodríguez, "An Approach to Build JSON-Based Domain Specific Languages Solutions for Web Applications," *Journal of Computer Languages*, vol. 75, p. 101203, 2023, doi: https://doi.org/10.1016/j.cola.2023.101203.

[34] E. Chavarriaga, L. A. Rojas, K. Sorbello, F. D. Rodríguez, F. Jurado, "JSON-based domain-specific language: A case study using rhoarchitecture in designing and developing API restful," 2024. [Online]. Available: https://ssrn.com/abstract=4740783, doi: http://dx.doi.org/10.2139/ssrn.4740783.

[35] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, A. Garg, "Progprompt: Program generation for situated robot task planning using large language models," *Autonomous Robots*, vol. 47, pp. 999–1012, 2023, doi: 10.1007/s10514-023-10135-3.

[36] J. Wang, Y. Chen, "A review on code generation with llms: Application and evaluation," in *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*, 2023, pp. 284–289.

[37] S. Yeo, Y.-S. Ma, S. C. Kim, H. Jun, T. Kim, "Framework for evaluating code generation ability of large language models," *Electronics and Telecommunications Research Institute (ETRI) Journal*, vol. 46, no. 1, pp. 106–117, 2024, doi: 10.4218/etrij.2023-0357.

[38] M. Schäfer, S. Nadi, A. Eghbali, F. Tip, "An empirical evaluation of using large language models for automated unit test generation," *IEEE Transactions on Software Engineering*, vol. 50, no. 1, pp. 85–105, 2024, doi: 10.1109/TSE.2023.3334955.

[39] X. Zhou, Z. Sun, G. Li, "Db-gpt: Large language model meets database," 2024. doi: 10.1007/s41019- 023-00235-6.

[40] J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekki, D. Doermann, "Future of software development with generative ai," *Automated Software Engineering*, vol. 31, 2024, doi: 10.1007/s10515-024-00426-zDO.

[41] Z. Guo, R. Jin, C. Liu, Y. Huang, D. Shi, Supryadi, L. Yu, Y. Liu, J. Li, B. Xiong, D. Xiong, "Evaluating large language models: A comprehensive survey," 2023. doi: https://doi.org/10.48550/arXiv.2310.19736.

[42] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, X. Xie, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, pp. 1–45, mar 2024, doi: 10.1145/3641289.

[43] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proceedings of the 40Th Annual Meeting on Association for Computational Linguistics*, ACL '02, USA, 2002, p. 311–318, Association for Computational Linguistics.

[44] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, S. Ma, "Codebleu: a method for automatic evaluation of code synthesis," *CoRR*, vol. abs/2009.10297, 2020.

[45] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain, jul 2004, pp. 74–81, Association for Computational Linguistics.

[46] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, 1997, pp. 21–29, IEEE.

[47] M. Voelter, *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. Springer, 2013.

[48] M. Fowler, T. White, *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[49] J. Smith, *DSLs in Practice*. 2019.

[50] V. Bojinov, *RESTful Web API Design with Node.js 10, Third Edition: Learn to create robust RESTful web services with Node.js, MongoDB, and Express.js, 3rd Edition*. Packt Publishing, 2018.

[51] H. Subramanian, P. Raj, *Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs*. Packt Publishing Ltd, 2019.

[52] J. Wexler, *Get Programming with Node. js*. Simon and Schuster, 2019.

[53] D. Flanagan, *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language*. 7th editio ed., 2020.

[54] O. Foundation, "Express: Fast, unopinionated, minimalist web framework for node.js," 2023. [Online]. Available: https://expressjs.com/.

[55] OpenAI, "Introducing ChatGPT," 2022. [Online]. Available: https://openai.com/blog/chatgpt, Accessed on April 23, 2024.

[56] Microsoft, "Microsoft copilot," 2024. [Online]. Available: https://copilot.microsoft.com, Accessed on April 23, 2024.

[57] Perplexity AI, "Perplexity | where knowledge begins," 2024. [Online]. Available: https://www.perplexity.ai/, Accessed on April 23, 2024.

[58] Perplexity AI, "Sonar," 2024. [Online]. Available: https://docs.perplexity.ai/docs/model-cards, Accessed on April 23, 2024.

[59] Meta, "Getting started with meta llama," 2024. [Online]. Available: https://llama.meta.com/, Accessed on April 23, 2024.

[60] Anthropic, "Claude," 2024. [Online]. Available: https://claude.ai, Accessed on April 23, 2024.

[61] Mistral IA, "Mistral technology," 2024. [Online]. Available: https://mistral.ai/technology/, Accessed on April 23, 2024.

[62] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, H. Behl, A. Benhaim, M. Bilenko, J. Bjorck, S. Bubeck, M. Cai, C. C. T. Mendes, W. Chen, V. Chaudhary, P. Chopra, A. D. Giorno, G. de Rosa, M. Dixon, R. Eldan, D. Iter, A. Garg, A. Goswami, S. Gunasekar, E. Haider, J. Hao, R. J. Hewett, J. Huynh, M. Javaheripi, X. Jin, P. Kauffmann, N. Karampatziakis, D. Kim, M. Khademi, L. Kurilenko, J. R. Lee, Y. T. Lee, Y. Li, C. Liang, W. Liu, E. Lin, Z. Lin, P. Madan, A. Mitra, H. Modi, A. Nguyen, B. Norick, B. Patra, D. Perez-Becker, T. Portet, R. Pryzant, H. Qin, M. Radmilac, C. Rosset, S. Roy, O. Ruwase, O. Saarikivi, A. Saied, A. Salim, M. Santacroce, S. Shah, N. Shang, H. Sharma, X. Song, M. Tanaka, X. Wang, R. Ward, G. Wang, P. Witte, M. Wyatt, C. Xu, J. Xu, S. Yadav, F. Yang, Z. Yang, D. Yu, C. Zhang, C. Zhang, J. Zhang, L. L. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, X. Zhou, "Phi-3 technical report: A highly capable language model locally on your phone," 2024. doi: https://doi.org/10.48550/arXiv.2404.14219.

[63] A. Mitra, L. D. Corro, S. Mahajan, A. Codas, C. Simoes, S. Agarwal, X. Chen, A. Razdaibiedina, E. Jones, K. Aggarwal, H. Palangi, G. Zheng, C. Rosset, H. Khanpour, A. Awadallah, "Orca 2: Teaching small language models how to reason," 2023. doi: https://doi.org/10.48550/arXiv.2311.11045.

[64] Gemma Team, "Gemma: Open models based on gemini research and technology," 2024. doi: https://doi.org/10.48550/arXiv.2403.08295.

[65] Gemini Team, "Gemini: A family of highly capable multimodal models," 2024. doi: https://doi.org/10.48550/arXiv.2312.11805.

Francisco Jurado

He received the Ph.D. degree with honours in Computer Science from the University of Castilla-La Mancha in 2010. He is Associated Professor in the Computer Engineering Department, Universidad Autónoma de Madrid, Spain. He has (co)authored more than 70 research articles in journals and conferences, participated in more than 15 competitive researches projects, and used to served as reviewer in indexed journals and international conferences. His main research areas include Intelligent Tutoring Systems, Heterogeneous Distributed eLearning systems, and Natural Language Processing.

Francy Rodriguez

She received her PhD degree from the Universidad Politécnica de Madrid (UPM) in 2015. She is currently a full-time Assistant Professor in the Computer Engineering Department at UPM, Spain. Her research interests include software development, design and programming patterns, software usability, and applications of artificial intelligence.

Enrique Chavarriaga

He received his PhD in Computer Science and Telecommunications Engineering from the Autonomous University of Madrid in 2017. He currently works as a research engineer in the I+D+i Department at UGROUND GLOBAL S.L., Spain. His research interests include domain-specific visual and textual languages, low-code development environments, and artificial intelligence.

Luis Rojas

He received his M.S. and Ph.D. in Computer Science from Universidad Autónoma de Madrid in 2017. He is currently a Lecturer in the Universidad San Sebastian, Santiago, Chile. He has been a software engineer at Chilean Nuclear Energy Commission for 10 years. He has also served as a reviewer and chair at different conferences on Software Engineering and Human-Computer Interaction. He is the author of several research articles, book chapters, and has also served as a book editor. His research interests include artificial intelligent, human-computer interaction, software engineering and learning analytics