System for Automatic Generation of Educational Quizzes Based on Ontology and SOLO Taxonomy

Seminar Paper

Abstract

This paper describes the development of a system that can automatically create educational quizzes. The system uses artificial intelligence running locally on the user's computer, combined with semantic web technology and the SOLO taxonomy to generate questions that match different levels of student understanding. Some of the main features include a chatbot that helps students learn, a tool for running SPARQL queries against the knowledge base, and the ability to export the course ontology as an OWL file that can be opened in Protégé. The paper also includes a hands-on evaluation where I manually reviewed the generated questions to check how good they are and whether the wrong answer choices are tricky enough to actually test student knowledge.

Table of Contents

1. Introduction

Education has always been important, and with technology becoming a bigger part of our lives, there is a growing need for tools that can help both teachers and students. One of the challenges teachers face is creating quizzes and tests. It takes a lot of time to come up with good questions, especially when you want to test students at different levels of understanding. A simple recall question is very different from a question that asks students to apply what they learned to a new situation.

This project tries to help with that problem. The idea is to build a system that can take educational content, like a PDF of lecture notes, and automatically generate quiz questions from it. But not just random questions - questions that are organized according to the SOLO taxonomy, which is a way of categorizing how deeply someone understands a topic. The system also includes a chatbot that students can talk to if they have questions about the material, and it stores all the knowledge in a structured way using ontologies so that connections between concepts can be explored.

The rest of this paper is organized as follows. Section 2 explains motivation. Section 3 looks at what other researchers and companies have done in this area. Section 4 describes how the system is built from a technical perspective. Section 5 goes into detail about the SOLO taxonomy and the ontology that stores the knowledge. Section 6 covers the implementation, meaning the actual code and how everything works together. Section 7 presents an evaluation where I checked the quality of the generated questions by hand. Finally, Section 8 wraps everything up with conclusions and ideas for future work.

2. Motivation

The first and most obvious one is that creating quizzes manually is time-consuming. Teachers already have a lot on their plate, and spending hours writing questions for every topic is not always realistic. According to some research, writing a single good multiple-choice question can take 15 to 30 minutes if you want to do it properly. Multiply that by the number of questions needed for an entire course, and it becomes clear why automation could help.

Another motivation is personalization. Students learn at different paces and have different strengths. A quiz that is too easy for one student might be too hard for another. By using the SOLO taxonomy, the system can generate questions at different cognitive levels, from simple recall to complex application. This makes it possible to create quizzes that are tailored to what each student needs to practice.

Project is not dependant on expensive cloud services. Many AI-powered tools today require API keys and charge money for every request. For a student project, or for schools with limited budgets, that is not always an option. By using Ollama to run a

language model locally, the system can work without internet access and without ongoing costs. This also means that student data stays on the local machine, which is better for privacy.

AI is also combined with knowledge representation. Ontologies are a way of formally describing the relationships between concepts, and I thought it would be interesting to see how they could be used together with modern language models to create a smarter educational tool.

3. Review of Existing Solutions

The field of Automatic Question Generation (AQG) has evolved significantly over the past two decades. One of the foundational works is by Mitkov and Ha (2003), who developed a system that used natural language processing rules to transform declarative sentences into questions. Their approach could take a sentence like "Photosynthesis occurs in chloroplasts" and generate "Where does photosynthesis occur?" While effective for simple factual content, these rule-based systems struggled with complex material and often produced grammatically awkward questions. You can find their paper here.

The challenge of generating good distractors has received special attention. Liang et al. (2018) proposed using a "learning to rank" approach where the system generates many candidate distractors and then ranks them by how plausible they are. Their key insight was that good distractors should be semantically related to the correct answer but clearly wrong upon careful reading. For example, if the correct answer is "mitochondria," good distractors might be "ribosomes" or "nucleus" rather than "banana." Their work is available here.

More recently, Kasneci et al. (2023) published a comprehensive review of using large language models like ChatGPT in education. They highlighted both the potential and the risks, noting that while these models can generate diverse questions quickly, they also raise concerns about accuracy, bias, and the need for teacher oversight. Importantly, they pointed out that cloud-based solutions create privacy issues when handling student data, which is one reason why local AI solutions like the one in this project are valuable. The paper can be accessed here.

Lister et al. (2006) examined how the SOLO (Structure of Observed Learning Outcomes) taxonomy could be applied to classify and evaluate programming questions. Their seminal work, "Not seeing the forest for the trees: novice programmers and the SOLO taxonomy," revealed a critical insight: many programming education assessments focus too heavily on low-level skills (identifying syntax errors, recognizing code fragments) while neglecting higher-order thinking skills (integrating concepts, designing solutions). They demonstrated that exam questions could be systematically classified into SOLO

levels—from uni-structural questions that test single concepts like "What does this variable store?" to extended abstract questions like "Design a program that solves this complex problem using multiple data structures." Their framework provides a structured way to ensure that questions across all complexity levels are included in assessments, which is directly applicable to your educational platform's question generation system. The paper can be accessed here.

4. System Architecture

The system is split into three main parts: the backend, the frontend, and the AI layer. Each part has a specific job, and they communicate with each other through a REST API. This kind of separation makes it easier to work on one part without breaking the others.

4.1 Backend

The backend is written in Python using Flask, which is a lightweight web framework. It handles all the business logic, like storing lessons in the database, generating questions, and managing quizzes. The database is SQLite, which stores everything in a single file. This makes the system easy to set up because you do not need to install a separate database server.

The backend is organized into several layers. The repository layer handles database operations using SQLAlchemy, which is an ORM that lets you work with the database using Python objects instead of writing raw SQL. The service layer contains the actual logic for things like parsing content, generating questions, and managing the ontology. There are separate services for lessons, questions, quizzes, the chatbot, and SPARQL queries.

4.2 Frontend

The frontend is a React application that provides the user interface. It has different components for different tasks. The CourseManager lets you create and organize courses. The LessonManager handles uploading and parsing PDF files. The QuestionGenerator triggers the AI to create questions. The QuestionBank shows all the generated questions and lets you edit or delete them. The QuizBuilder lets you put questions together into a quiz, and the QuizSolver is where students actually take the quiz.

There is also a ChatBot component that opens a chat window where students can ask questions about the course material. And the SPARQLQueryTool is a more advanced feature that lets users write their own queries to explore the ontology.

4.3 AI Layer

The AI layer uses Ollama, which is a tool for running large language models locally. The system is configured to use the Qwen 2.5 model with 14 billion parameters. This model is

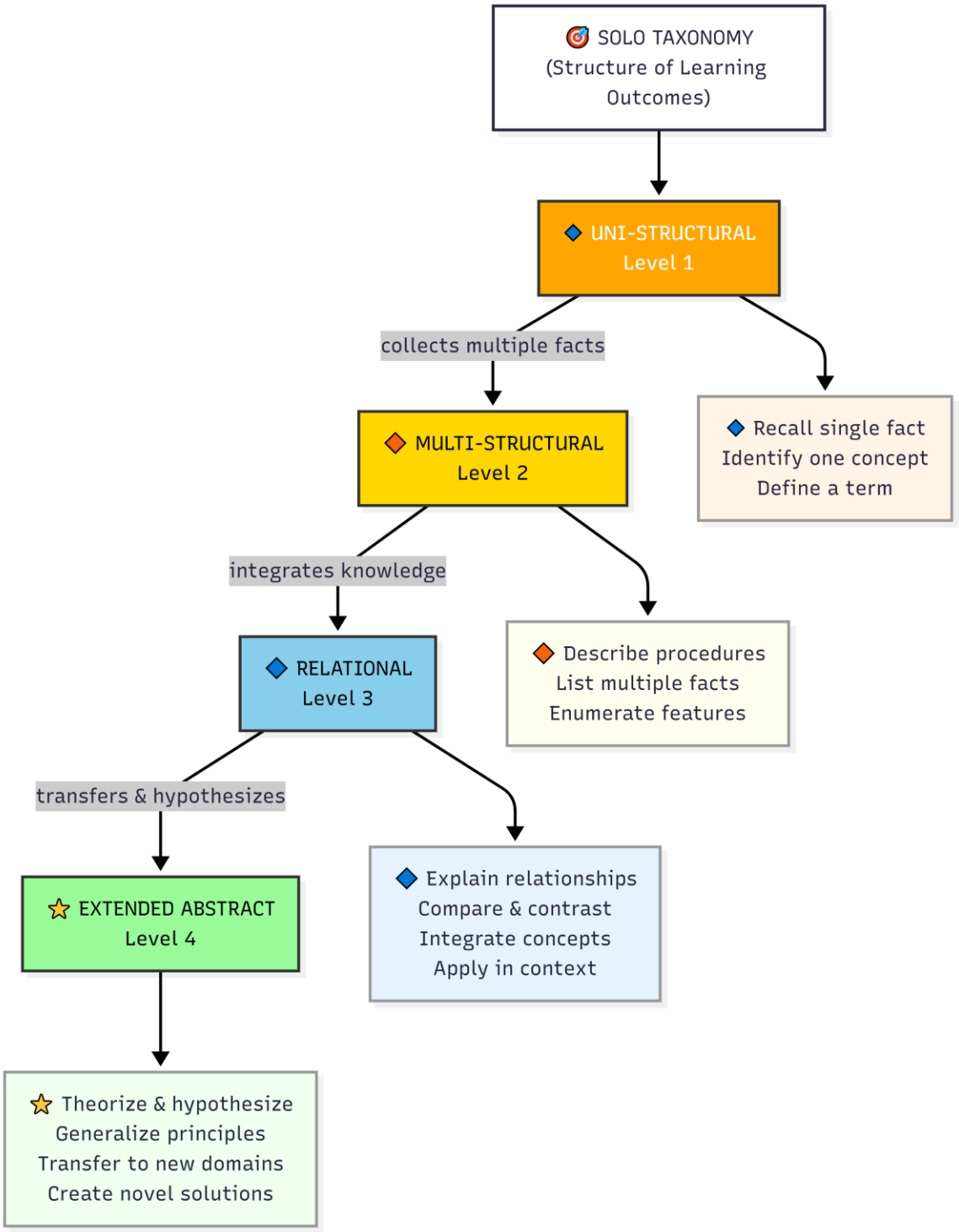
good at understanding and generating text, and it runs on the local machine without needing an internet connection or API keys. The backend communicates with Ollama through HTTP requests, sending prompts and receiving generated text.

5. SOLO Subsystem and Ontology

5.1 What is the SOLO Taxonomy

SOLO stands for Structure of Observed Learning Outcomes. It was developed by Biggs and Collis in 1982 as a way to describe how student understanding develops from simple to complex. There are five levels: Prestructural, where the student does not really understand the topic yet; Unistructural, where they understand one aspect; Multistructural, where they understand several aspects but do not see how they connect; Relational, where they can integrate different pieces into a coherent whole; and Extended Abstract, where they can generalize and apply the knowledge to new situations.

There are four levels: Unistructural, Multistructural, Relational, and Extended Abstract. Each level corresponds to a different type of question. Unistructural questions ask about a single fact. Multistructural questions ask about multiple facts. Relational questions ask how things connect. And Extended Abstract questions ask students to apply what they learned in a new context. You can see the question structure on the diagram on the next page.



5.1 SOLO question structure diagram

5.2 How SOLO is Used in Question Generation

When the system generates questions, it uses different prompts for each SOLO level. The prompts are carefully designed to guide the AI toward creating appropriate questions. For example, the prompt for Unistructural questions explicitly says to focus on a single fact and avoid asking about connections. The prompt also includes instructions for creating good distractors that are plausible but incorrect.

The prompts also tell the AI to avoid boring question formats like "Which of the following..." and to use more varied phrasing. This makes the generated questions feel less repetitive and more natural.

5.3 The Ontology

The system uses an ontology to represent the structure of knowledge. An ontology is basically a formal way of describing what concepts exist and how they relate to each other. The ontology is written in Turtle format, which is a syntax for RDF (Resource Description Framework). It defines classes like Course, Lesson, Section, LearningObject, and Question, and properties that link them together.

When a lesson is parsed, the system creates entries in the ontology for all the concepts it finds. It also tries to identify relationships between concepts, like prerequisites or related topics. This information can be used to generate better questions and to help students understand how different parts of the material connect.

5.4 SPARQL Queries and OWL Export

The system includes a SPARQL query interface. SPARQL is a query language for RDF data, kind of like SQL but for ontologies. Users can write queries to find specific information, like all questions at a certain SOLO level, or all learning objects that are related to a particular concept. The interface includes some example queries to help users get started.

There is also an option to export the ontology as an OWL file. OWL stands for Web Ontology Language, and it is a standard format that can be opened in tools like Protégé. Protégé is an ontology editor that can visualize the knowledge graph, showing nodes and edges that represent concepts and their relationships. This can be useful for teachers who want to see the structure of their course at a glance.

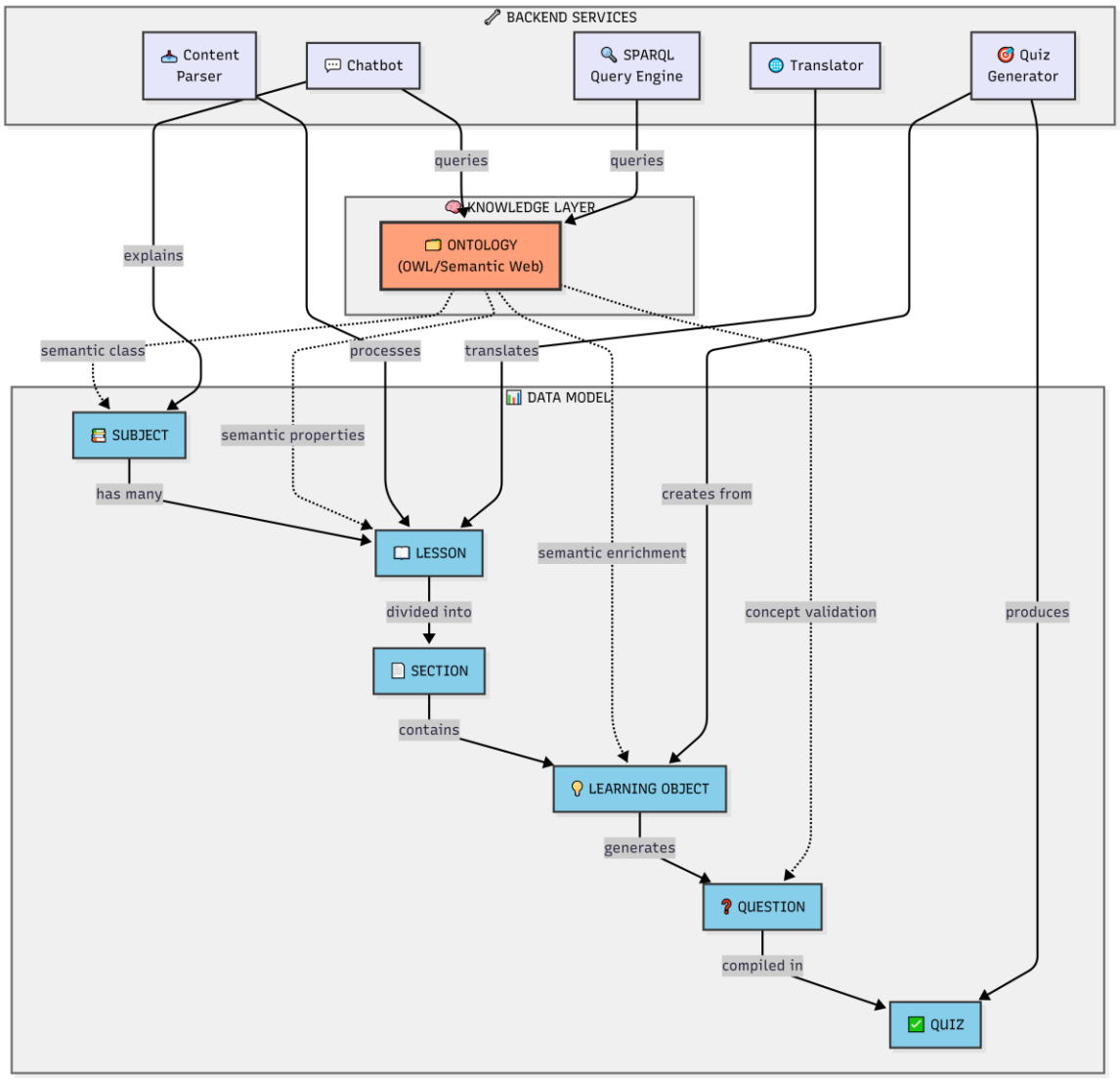
6. System Implementation

6.1 How the Workflow Works

Here is how a typical workflow goes: First, the teacher uploads a PDF file containing lecture notes or a textbook chapter. The backend extracts the text from the PDF using a library called PyPDF2. Then the text is sent to the AI model, which parses it into sections and learning objects. These are stored in the database with the hierarchical structure: Course contains Lessons, Lessons contain Sections, and Sections contain Learning Objects.

Next, the teacher can trigger question generation. The system goes through the learning objects and generates questions at different SOLO levels. Each question is saved to the database along with its options, correct answer, and explanation. The teacher can review the questions, edit them if needed, and then build quizzes by selecting which questions to include.

Students access the quizzes through the QuizSolver component. They answer the questions, and when they submit, they see their score and can review which questions they got right or wrong. The explanations help them understand why the correct answer is correct. You can see the structure in detail on the diagram shown in the picture below.



6.1 Image of the workflow diagram

6.2 The Chatbot

The chatbot is one of the newer features of the system. It uses a Retrieval-Augmented Generation (RAG) approach. When a student asks a question, the system first searches the database for relevant content. It looks through lessons, sections, and learning objects to find text that matches the question. This context is then included in the prompt sent to the language model, so the response is grounded in the actual course material rather than being made up.

6.3 Multi-Language Support

The system includes a translation manager that can translate content into different languages. This uses the same AI model but with prompts for translation. The interface remembers the user's language preference, so if a student switches to Serbian, for example, the questions will be shown in Serbian if translations are available.

7. Evaluation and Quality

To evaluate the system, I did a manual review of the generated questions. The goal was to check two things: first, whether the questions match their assigned SOLO level, and second, whether the distractors are good enough that students cannot just guess the right answer by eliminating obviously wrong options.

7.1 SOLO Level Accuracy

The questions generally matched their assigned SOLO levels. Unistructural questions asked about single facts. Multistructural questions asked about multiple aspects without requiring integration. Relational questions required understanding connections. And Extended Abstract questions presented new scenarios that required applying and synthesizing knowledge.

The prompts seem to be doing their job of guiding the AI toward the right type of question. The explicit definitions of each SOLO level in the prompts help the model understand what kind of question to generate.

7.2 Sample Questions Reviewed

Here are some questions from the database to analyze:

7.2.1 Unistructural question examples

Unistructural Example: "The Program Counter (PC) holds the memory address of which type of entity?" The options were: A) The next instruction to be executed, B) The current executing instruction, C) All instructions in memory, D) Data variables. The correct answer is A. This item successfully isolates a single, fundamental concept—the specific function of the Program Counter. The distractors show variable quality. Option B is particularly effective, exploiting temporal confusion between "next" and "current" that targets students with incomplete understanding of the fetch-execute cycle. Option C addresses a novice misconception about PC scope, though it may be somewhat obvious to students who understand that registers hold single values. Option D, while testing the distinction between instruction and data addresses, is the weakest distractor—students with even basic architecture knowledge would likely recognize that the PC relates to instructions rather than data variables, making this option easily eliminated.

Unistructural Example: "The term 'volatile memory' refers to:" The options were: A) Memory that maintains data even after the system restarts, B) Non-volatile storage used for long-term data retention, C) A type of computer storage that loses data when powered off, D) High-speed cache directly integrated into the CPU. The correct answer is C. Options A and B both describe non-volatile memory characteristics, directly testing whether students understand the actual meaning of "volatile" versus merely recognizing it as memory-related terminology. This antonym approach is pedagogically sound for

definition questions. However, having two distractors that essentially convey the same concept (persistence of data) represents some redundancy in the distractor set. Option D is more problematic—while cache memory is technically volatile, the distractor conflates two distinct concepts (volatility and cache architecture). Students may eliminate this option not because they understand volatility, but simply because the question asks about a memory type rather than a specific implementation or location.

7.2.2 Multistructural question examples

Multistructural Example: "Which of the following statements correctly describe both User-Level Threads (ULT) and Kernel-Level Threads (KLT)?" The options were: A) ULT are managed by application, KLT by OS; ULT can provide finer control but may be less efficient than KLT, B) Both ULT and KLT require explicit synchronization mechanisms to prevent race conditions, C) ULT provides more system resources allocation units compared to KLT, D) Kernel manages both ULT and KLT, providing equal efficiency and control. The correct answer is A. Option B is well-constructed, appearing plausible since synchronization is genuinely relevant to threading, though it incorrectly universalizes a concern that varies by implementation. Option D effectively targets the misconception that kernel management equalizes all thread characteristics. Option C, however, is the weakest distractor—the phrasing "provides more system resources allocation units" is awkwardly technical and vague enough that students might eliminate it based on unclear wording rather than actual knowledge of thread resource allocation.

Multistructural Example: "What are the key components of dynamic testing?" The options were: A) Loop coverage, statement coverage, design review, B) Integration testing, functional testing, system testing, C) Static analysis, positive testing, unit testing, D) Function execution, negative testing, code path coverage. The correct answer is D. Option A effectively mixes coverage metrics (legitimately part of dynamic testing) with "design review" (a static activity), testing whether students can distinguish between testing paradigms. Option C employs a similar strategy by embedding "static analysis" among otherwise plausible terms. However, both options B and D present significant issues. Option B lists valid testing levels rather than components, which represents a category confusion that may allow students to eliminate it through semantic reasoning rather than actual knowledge of dynamic testing mechanics.

7.2.3 Relational question examples

Relational Example: "How do conditional variables and mutexes work together to manage thread synchronization in multi-threaded environments?" The options were: A) Mutexes are used to manage thread blocking, whereas conditional variables handle independent thread execution, B) Conditional variables and mutexes both prevent race

conditions by ensuring mutual exclusion, C) Conditional variables wait for specific conditions, while mutexes prevent simultaneous access to shared resources, D) Mutexes signal waiting threads when a condition is met, allowing conditional variables to proceed. The correct answer is C. Option D is particularly effective, directly reversing the roles of mutexes and condition variables—this catches students who possess surface-level familiarity with both mechanisms but lack deep understanding of their interaction patterns. Option B presents a subtler challenge by offering a partially correct statement (both mechanisms do contribute to preventing race conditions) while obscuring the crucial distinction in how they accomplish this through different mechanisms. Option A is the weakest distractor. The phrase "independent thread execution" is vague and doesn't correspond to any common misconception about condition variables. Students might eliminate this option based on the awkward phrasing rather than genuine understanding of the mutex-condition variable relationship.

Relational Example: "How does test automation contribute to the effectiveness of regression testing?" The options were: A) Test automation only improves the speed of functional testing without impacting regression testing, B) Test automation replaces manual testing entirely, making regression testing unnecessary, C) Test automation allows for faster execution and identification of changes in functionality, enhancing the thoroughness of regression tests, D) Test automation focuses on creating new test cases rather than running existing ones efficiently. The correct answer is C. The question successfully requires students to connect two concepts—understanding both what regression testing entails (re-executing existing tests to detect unintended changes) and how automation specifically addresses its challenges. The distractors show reasonable quality. Option B captures an extreme misconception about automation replacing human judgment entirely, while Option A creates an artificial boundary between functional and regression testing that might appeal to students with lesser understanding. However, Option D is problematic. The claim that "test automation focuses on creating new test cases" doesn't align with any common misconception about automation—most students would recognize that automation executes tests rather than generates them. This makes the distractor too easily eliminated. Additionally, Option A's specificity about "only functional testing" may be overly obvious, as students familiar with basic automation concepts would recognize that automated tests can serve multiple purposes. Stronger distractors might address misconceptions about automation coverage limitations or the false belief that automated tests detect different types of defects than manual regression testing.

7.2.4 Extended abstract question examples

Extended Abstract Example: "In a distributed computing environment, if a main process spawns multiple child processes across different nodes, what mechanism ensures that all child processes receive the correct system resources and do not interfere with each

other?" The options were: A) Implementing inter-process communication (IPC) protocols to manage resource allocation and synchronization, B) Increasing the priority of the parent process over its children to control their execution flow, C) Allocating all child processes to a single node for centralized management of resources, D) Using thread-level synchronization techniques within each individual node. The correct answer is A. This question takes the concept of process management learned in a single-machine context and asks the student to extend it to a distributed environment. Option C effectively identifies a common misconception (centralizing defeats distribution), and Option D correctly recognizes that thread-level synchronization addresses only intra-node concerns. However, Option B is too obviously incorrect—"increasing priority" doesn't logically connect to resource allocation or interference prevention across nodes, making it easily eliminated

Extended Abstract Example: "In a hypothetical cloud computing environment, how should an OS manage process states to optimize resource allocation and reduce wait times?" The options were: A) Dynamically adjust the priority of processes based on their current state and system load, B) Keep all processes in the running state continuously for efficiency, C) Allocate fixed resources to each process regardless of its state transitions, D) Terminate waiting processes after a certain timeout period. The correct answer is A. First, the correct answer (A) is vague and doesn't specifically leverage cloud computing characteristics, dynamic priority adjustment is standard OS scheduling behavior even in non-cloud contexts, making this more of a general scheduling question than a genuine extension to new domains. True extended abstract thinking would require students to reason about cloud-specific concerns like elasticity, multi-tenancy, or distributed state management. Option B is far too obviously incorrect—any student familiar with basic CPU time-sharing would eliminate this immediately, as it violates fundamental operating system principles regardless of context. Option C is similarly weak; the notion of ignoring state transitions contradicts basic process management so fundamentally that it tests minimal competence rather than abstract reasoning. Option D is more interesting, as timeout-based process termination does exist in some systems, though it's typically applied to unresponsive rather than waiting processes. Overall, the distractors are too easily eliminated through basic knowledge rather than requiring sophisticated extension of concepts to new contexts.

7.3 Limitations Found

The evaluation also revealed some limitations. First, the PDF parsing sometimes struggles with complex layouts like tables or diagrams. Second, very long documents can lose context because the model has a limited context window. Third, the questions and distractors could use improvements.

The evaluation revealed several systematic limitations in the question generation process. A clear pattern emerged across cognitive levels: as the taxonomic complexity increased from unistructural to extended abstract, question quality deteriorated markedly. Lower-level questions (unistructural and multistructural) generally demonstrated stronger alignment with their intended cognitive levels and featured more pedagogically sound distractors. However, higher-level questions, particularly at the relational and extended abstract levels, exhibited significant validity concerns. The correct answers themselves sometimes contained technical inaccuracies or oversimplifications.

Distractor quality also declined at higher cognitive levels. While unistructural questions typically featured distractors targeting genuine misconceptions, extended abstract questions often included options that were either trivially eliminable (violating fundamental principles obvious to any minimally competent student) or failed to reflect realistic reasoning errors students might make when extending concepts to new domains. Additionally, distractor redundancy emerged as a recurring issue, with multiple options sometimes conveying essentially the same incorrect concept, reducing the discriminatory power of the assessment items.

8. Conclusion

This paper presented a system for automatically generating educational quizzes based on the SOLO taxonomy. The system combines local AI models, semantic web technology, and a user-friendly interface to help teachers create better assessments with less effort.

The main contributions include the integration of SOLO taxonomy into the question generation process, a chatbot that helps students learn by answering questions grounded in the course material, SPARQL query support for exploring the knowledge base, and the ability to export ontologies as OWL files for visualization in Protégé.

The manual evaluation showed that the generated questions generally match their intended SOLO levels and have distractors that are challenging enough to test real understanding. The system is not meant to replace teachers, but to be a tool that saves time and helps create more diverse assessments.

Future work could focus on improving the distractor generation, handling more complex document formats, and adding analytics to track student performance over time. The system could also be extended to support other question types beyond multiple choice, like short answer or essay questions.

Overall, this project shows that combining AI with educational theory can produce useful tools that make teaching and learning more effective. By running everything locally, the system is accessible to anyone without requiring expensive subscriptions or giving up control over their data.

References

Biggs, J. B., & Collis, K. F. (1982). Evaluating the quality of learning: The SOLO taxonomy. Academic Press.

Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., ... Kasneci, G. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 103, 102274.

Liang, C., Yang, X., Dave, N., Wham, D., Pursel, B., & Giles, C. L. (2018). Distractor generation for multiple choice questions using learning to rank. Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications, 284-290.

Mitkov, R., & Ha, L. A. (2003). Computer-aided generation of multiple-choice tests. Proceedings of the HLT-NAACL 2003 Workshop on Building Educational Applications Using Natural Language Processing, 17-22.