Language Learning Sandbox

Aya Khattab

Mayar Elmenisy

Yousra Okasha

Yassin Zakaria

Youssef yasser

Youssef ehab

Arab academy for science and technology



Submitted to Dr. Mohammed Khalaf

**Abstract**

This project builds a system that checks how well a student understands the syntax and semantics of a written answer. Many students memorize answers without understanding structure or meaning. Teachers cannot review every answer carefully when the number of students is large. This system solves that problem by offering an automated way to evaluate responses.

The system reads the student’s answer, compares it with the expected structure, and checks the core meaning. It focuses on the key parts of the sentence that show understanding. It gives a score and brief feedback so the student knows what needs improvement. The goal is to help students learn how to write correct and meaningful sentences.

The project uses HTML, CSS, and JavaScript for the interface. It uses Python and Flask for backend processing. It stores questions and answers in JSON files. These tools make the system simple to use and easy to expand.

The final outcome is a working platform that evaluates answers instantly. It supports teachers by reducing manual checking and helps students improve their language skills through direct feedback.

**Introduction**

Learning a language requires practice with sentence structure and meaning. Many students struggle to form correct sentences even when they know vocabulary. They often repeat memorized patterns without understanding why a sentence is correct. This leads to weak writing skills and slow progress.

Teachers try to correct every answer, but this takes time. Large classes make it hard to give quick feedback. Students wait days before knowing what they did wrong. They lose the chance to fix mistakes immediately. A system that checks answers in real time can solve this problem.

This project builds a platform that evaluates how well a student understands syntax and semantics. Syntax refers to the order of words. Semantics refers to the meaning behind the words. When both are correct, the answer shows real understanding. When one of them is wrong, the student needs more practice.

The system shows a question, takes the student’s answer, and checks if the structure and meaning match the expected response. It gives a score and a simple explanation. The goal is to help students improve through fast and clear feedback. The system is easy to use and can support different levels of language learners.

**Problem Definition**

Students often answer language questions without knowing if their structure or meaning is correct. They guess. They repeat patterns. They rely on memorized phrases. They do not get feedback at the moment they need it. This slows learning.

Teachers cannot check every answer instantly. Manual correction takes time. Students forget what they were thinking when they made the mistake. They do not learn why their sentence is wrong. They keep repeating the same errors.

There is no simple tool that checks both syntax and semantics in a single response. Many tools only check grammar or spelling. Others check meaning but ignore structure. Learners need something that looks at both sides. Syntax shows how the sentence is built. Semantics shows what the sentence means. Mixing both gives a more accurate evaluation.

This project solves the gap by creating a system that evaluates student answers in real time. It detects incorrect structure, incorrect meaning, and unclear sentences. It helps learners understand their mistakes and improve faster.

**Project Goals**

This project aims to achieve the following:

• Build a simple environment where students answer questions and receive instant feedback  
• Evaluate syntax and semantics together to reflect true understanding  
• Provide clear messages that explain whether the answer is correct  
• Support different levels of questions to track progress  
• Create a user-friendly interface that beginners can use without training  
• Allow teachers to add or modify questions easily  
• Ensure responses are checked quickly without long processing time  
• Help students practice consistently and improve their accuracy

The final system should work smoothly, give reliable feedback, and support real learning through repetition and correction.

**Background on Syntax and Semantics**

Syntax refers to sentence structure. It shows how words are arranged. If the order is wrong, the meaning becomes unclear. Example:  
You say “Runs he fast” instead of “He runs fast.”  
The words are correct, but the structure breaks the sentence.

Semantics refers to meaning. A sentence can be grammatically correct but still wrong in meaning. Example:  
You answer “The sky is green” when the expected meaning is different.  
The structure is fine, but the meaning does not match the intent.

Language learning depends on both. A student must know how to place words and how to express the correct idea. Many learners get one right and miss the other. A good evaluation system checks both sides. Syntax shows technical correctness. Semantics shows understanding. When combined, they give a complete picture of the student’s skills.

**Related Work**

Several tools help students learn language structure, but each one covers only part of the problem. Grammar checkers highlight spelling and punctuation but cannot judge meaning. They correct surface errors without understanding the student’s intention.

Educational platforms provide quizzes, but most of them check answers by matching text exactly. If a student writes a correct sentence with different wording, the system marks it wrong. This discourages creativity and makes students depend on memorization.

Some research systems analyze meaning, but they are complex and hard to use in a classroom. They require advanced setup. They are not friendly for beginners. They also focus on long texts instead of short answers.

No tool combines simple design, instant feedback, syntax checking, and semantic checking in one place. This project fills that gap by creating a lightweight, accessible system that evaluates both structure and meaning for short student responses.

**Technology Stack**

The project uses a small stack to keep the system simple and maintainable.

**Frontend**  
• HTML  
• CSS  
• JavaScript  
These tools build an interface that displays questions, takes user answers, and shows feedback instantly.

**Backend**  
• Python  
• Flask  
The backend checks syntax and semantics. Flask handles communication between the page and the evaluation logic.

**Data**  
• JSON  
All questions and correct answers are stored in JSON files. This makes it easy to update or add new content.

**Environment**  
• Local server using Flask  
• Browser-based interface  
This allows the system to run on any computer without extra installation.

The stack is lightweight and approachable for beginners while still supporting meaningful analysis.

**System Architecture**

The system follows a simple flow so students and teachers can use it without technical knowledge.

**1. User Interface**  
The student sees the question on the screen. The interface includes a text box for typing the answer and a button to submit it.

**2. Request Handling**  
When the student submits the answer, JavaScript sends the text to the backend through a POST request.

**3. Processing Layer**  
The backend receives the answer.  
The system checks two things:

• Syntax  
It looks at word order and structure by comparing patterns.

• Semantics  
It matches meaning by checking keywords, expected ideas, and allowed variations.

Both checks happen within Python functions.

**4. Evaluation Logic**  
The system compares the student’s answer to the correct models stored in JSON.  
It decides if the meaning matches and if the structure follows the expected pattern.  
It returns a result with a score and a short explanation.

**5. Response to Frontend**  
The backend sends the evaluation back to the browser.  
The interface shows the result in real time.  
If the answer is wrong, the student sees what part needs improvement.

**6. Data Layer**  
All questions, accepted answers, difficulty levels, and patterns exist in JSON files.  
This separates content from the code and makes updates easy.

The architecture stays simple, clear, and functional. Each part serves one purpose. The system runs smoothly even with basic hardware.

A diagram of a process flow

Description automatically generated

**Flow Diagrams**

**1.Use Case Diagram**

This diagram shows how students and teachers interact with the system. Students answer questions and receive feedback. Teachers manage the question content. The diagram helps clarify the main functions and the users who depend on them.

A diagram of a student

Description automatically generated

**2.Activity Diagram**

This diagram explains the steps a student follows when responding to a question. It shows how the answer is submitted, evaluated, and returned. It also shows the decision point where the system checks if the answer is correct. The diagram summarizes the full path of user interaction.

A flowchart of a program

Description automatically generated

**3.Sequence Diagram**

This diagram describes how different parts of the system communicate when a student submits an answer. It shows the order of messages between the browser, backend, evaluation logic, and data storage. It clarifies how the system processes input and returns a result in real time.

A diagram of a process

Description automatically generated

**4. Data Flow Diagram**

This diagram shows how data moves through the system. The user enters an answer. The backend processes it. The evaluation logic checks structure and meaning. The system retrieves expected answers from JSON files and returns feedback to the user. This view explains how information flows from start to end.

A diagram of a process

Description automatically generated

**User Interface**

The system uses a simple interface so students can focus on learning. The layout guides the user through each step without confusion.

**Home page**  
You see a landing page with a start button. It directs you to select the level you want to practice. The page keeps the options clear and avoids extra elements.A screenshot of a computer

Description automatically generated

**Choose exam**  
You choose between beginner and advanced levels. Each level contains questions that match its difficulty. This step helps students move at the pace that fits their understanding.

A screenshot of a blue screen

Description automatically generated

A screenshot of a screen

Description automatically generated

**Display questions**  
The system shows one question at a time. This prevents distractions and keeps the student focused on the current task. The question appears in a clean box at the top of the page.

A screenshot of a quiz

Description automatically generated

**Input field for answer**  
There is a text box where the student writes the answer. The field is large enough for clear visibility. It accepts any text so the system can evaluate structure and meaning.



**Submit button**  
The submit button sends the student’s answer to the backend. The user receives feedback instantly based on syntax and semantics. The student can then move to the next question.

A yellow and blue rectangle with black text

Description automatically generated

**Hint Button**

**The system includes a hint button to support students who need guidance without giving them the full answer. The button appears under the question. You can choose to use it when you feel unsure about how to structure your response.**

**A blue and pink rectangular box with a light bulb and white text

Description automatically generated**

**Score page**  
When all questions are done, the system displays the student’s score. The page shows how many answers were correct. It helps students see their progress and identify areas to improve.

A screenshot of a quiz

Description automatically generated

**Implementation**

The system runs on a Flask backend with a browser-based interface. The backend receives the student’s answer, evaluates syntax and semantics, and sends feedback to the interface. The implementation focuses on short response time and clear logic.

The frontend uses HTML and CSS to display questions and collect answers. JavaScript sends requests to the backend and updates the page with the result. This keeps each interaction fast without reloading the entire page.

The backend uses Python functions to compare the student’s answer with the expected response. Syntax evaluation checks order and structure. Semantic evaluation checks meaning. Both checks produce a short message that guides the student. The backend stores questions and answers inside JSON files for simple access and easy editing.

The system follows a modular design. Each part handles one job. This avoids complexity and makes the project easy to maintain and update.

Code Structure A black screen with white text

Description automatically generated

**Backend Logic**

**app.py**

* loads questions from JSON
* displays the first question
* receives the student’s answer
* sends the answer to the evaluation module
* returns feedback to the frontend

**Frontend Logic**

**index.html**

* shows a question
* shows input field
* shows submit and hint buttons
* displays feedback

**script.js**

* sends the student’s input to the backend using fetch
* receives JSON feedback
* clears old messages
* loads the next question
* displays the hint when requested

**style.css**

* controls layout
* sets colors and spacing
* keeps the interface clean and easy to read

**How Questions Are Processed**

The system loads all questions from a JSON file when the application starts. Each question includes the text, the expected answer, and the hint. The backend sends one question at a time to the browser.

When the user submits a response, the backend splits the text into words. The system checks the structure of the sentence to see if it follows the expected order. It checks meaning by looking for key words that match the intended concept of the question.

This process keeps evaluation simple and fast while giving the student clear feedback.

**How Scoring Works**

The system uses two checks to determine the result. The syntax score evaluates the order of the words. The semantic score evaluates the meaning. Each category contributes to the total score through weighted scoring.

Syntax score helps see if the student understood how to form the structure. Semantic score confirms if the student captured the idea of the answer.

Examples of scoring outcomes:

• correct syntax and correct meaning gives a full score  
• correct meaning but weak structure gives a partial score  
• wrong structure and wrong meaning gives a zero score

This method gives balanced evaluation.

**JavaScript Logic**

The frontend uses JavaScript to manage the interaction between the user and the backend. The script loads questions, validates input, sends data to the backend, and updates the score at the end.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

**Flask Backend**

Route for receiving answers

* POST endpoint receives the user answer and question index.
* Endpoint reads JSON payload.
* Endpoint validates input.
* Endpoint calls the evaluation module.
* Endpoint returns a JSON response with scores and message.

Processing logic

* Load question set from JSON or session.
* Normalize user text (trim, lower case).
* Tokenize the text into words.
* Run syntax check function.
* Run semantic check function.
* Calculate weighted total score.
* Format feedback text.
* Update session score if used.
* Return JSON result.

Comparing expected vs actual answer

* Use normalized forms for both expected and actual.
* Compare key tokens and token order for structure.
* Check presence of required keywords for meaning.
* Allow small variations and synonyms via a whitelist.
* Mark partial matches and record which part failed.
* Produce two component scores: syntax and semantics.

Returning score

* Response fields:
  + syntax\_score numeric
  + semantic\_score numeric
  + total\_score numeric
  + correct\_answer optional string
  + explanation short text
* HTTP 200 for normal responses.
* HTTP 400 for malformed requests.

**Testing**

Unit tests

* Test syntax check with many small examples.
* Test semantic check with synonyms and paraphrases.
* Test scoring function for boundary cases.
* Test normalization utilities.
* Use pytest for small, fast tests.
* Mock file I/O where needed.

Integration tests

* Start Flask test client.
* Simulate POST requests from the frontend.
* Verify JSON structure and status codes.
* Verify session score updates across requests.

Testing with sample students

* Collect sample answers from a few users.
* Run them through the system.
* Compare automated feedback with teacher feedback.
* Measure agreement rate.

Testing for wrong answers

* Feed clearly wrong answers.
* Ensure system returns low scores and helpful explanation.
* Verify no crashes or exceptions.

Observing system behavior

* Log requests and results in a development log.
* Monitor common failure modes.
* Keep counters for types of errors.
* Use logs to refine hints and matching rules.

**Challenges**

Handling different synonyms

* Exact string match fails on synonyms.
* Solution: small synonym map for each question.
* Add accepted variations in the question JSON.

Handling partial answers

* Students may give partial correct content.
* System must give partial credit.
* Split scoring into components and weigh them.

Time constraints

* Response must be fast.
* Avoid heavy NLP models for the MVP.
* Keep checks rule-based and local.

Integration issues

* Session handling across requests can break if misconfigured.
* Cross-origin and browser caching issues may appear.
* Ensure consistent JSON schema between frontend and backend.

**Future Improvements**

Add machine learning

* Use a small classifier to detect semantic equivalence.
* Fine-tune embeddings for short-answer matching.
* Use ML only after collecting labeled data.

Support more languages

* Add localization for UI.
* Add language-specific tokenizers and rules.

Add dashboard for teachers

* Show class performance.
* Allow bulk question upload.
* Export student scores.

Add question difficulty levels

* Tag questions by difficulty.
* Use difficulty in scoring weight and progression logic.

**Conclusion**

What the project aims to solve

* Provide instant feedback on short written answers.
* Evaluate both structure and meaning.
* Reduce teacher grading load.

Benefits to students

* Faster correction cycles.
* Clear hints that guide learning.
* Measured progress through scores.

System contributions

* A compact, maintainable evaluation pipeline.
* Simple tools for teachers to add content.
* Balanced scoring that rewards meaning and form.

Final results

* A working web prototype that evaluates answers in real time.
* Clear feedback for students.
* A foundation that can evolve to ML and multi-language support.

**References**

**Websites**

* Flask official documentation  
  <https://flask.palletsprojects.com>
* MDN JavaScript Guide  
  <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
* JSON Specification  
  <https://www.json.org>
* Python documentation  
  <https://docs.python.org/3/>
* W3Schools JavaScript tutorial  
  <https://www.w3schools.com/js>