# citu-speech-evaluation

**luijip JustinVijar**

**Aug 18, 2025**

# CONTENTS:

Welcome to the CITU Speech Evaluation System documentation. This system provides automated evaluation of spoken English responses using AI-powered transcription and assessment tools.

CHAPTER

ONE

# BACKEND MODULES

## 1.1 app module

app.**evaluate**()

POST endpoint to evaluate a student's audio answer.

Receives a question, keywords, and audio file from the frontend, processes the audio through transcription and evaluation pipeline, and returns comprehensive results.

Expected form data: - question: The question that was asked - keywords: List of expected keywords for evaluation - audio: Audio file containing the student's answer (webm/ogg format)

**Returns**
JSON response containing transcript, audio metrics, evaluation scores, and feedback

**Return type**
flask.Response

Response format: {

"transcript": str, "audio_metrics": dict, "evaluation": dict, "comment": str

}

Error responses: - 400: Missing required fields (question, keywords, or audio) - 500: Audio conversion failure or evaluation pipeline failure

app.**get_current_question**()

GET endpoint to retrieve the current question.

**Returns**
JSON response containing the current question text

**Return type**
flask.Response

app.**next_question**()

POST endpoint to advance to the next question in the sequence.

Advances to the next question in the predefined list. If at the end of the list, loops back to the first question. Also triggers text-to-speech for the new question.

**Returns**
JSON response with success status and the new question

**Return type**
flask.Response

app.**speak**(*text*)

Convert text to speech using pyttsx3 engine with female voice preference.

**Parameters**
**text** (*str*) – The text to be spoken aloud

**Returns**
None

**Return type**
None

## 1.2 test_eval module

**class** test_eval.**CategoryScores**(** (Keyword-only parameters separator (PEP 3102))*, *task_relevance: int*, *grammar_lexis: int*, *discourse_management: int*, *pronunciation_fluency: int*, *coherence_appropriateness: int*)

Bases: BaseModel

**coherence_appropriateness: int**

**discourse_management: int**

**grammar_lexis: int**

**model_config: ClassVar[ConfigDict] = {}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

**pronunciation_fluency: int**

**task_relevance: int**

**class** test_eval.**Verdict**(***, *score: int*, *category_scores:* CategoryScores, *comment: str*)

Bases: BaseModel

**category_scores: *CategoryScores***

**comment: str**

**model_config: ClassVar[ConfigDict] = {}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

**score: int**

test_eval.**analyze_audio**(*file_path*)

Analyze audio file for pitch, duration, and estimated speaking rate.

**Parameters**
**file_path** (*str*) – Path to the audio file to analyze

**Returns**
Dictionary containing duration, average pitch, and estimated words per minute

**Return type**
dict

test_eval.**check_grammar**(*text*)

Check grammar and language issues in the provided text.

**Parameters**
**text** (*str*) – The text to check for grammar issues

**Returns**
List of grammar issues found by LanguageTool

**Return type**
list

test_eval.**create_error_verdict**(*error_message: str*) → dict

Create a default error verdict.

**Parameters**
**error_message** (*str*) – The error message to include in the verdict

**Returns**
Dictionary containing default error verdict with zero scores

**Return type**
dict

test_eval.**evaluate_answer**(*transcript*, *audio_metrics*, *expected_keywords*)

Basic evaluation function that returns transcript and audio metrics.

**Parameters**

- **transcript** (*str*) – The transcribed text from audio
- **audio_metrics** (*dict*) – Dictionary containing audio analysis results
- **expected_keywords** (*list*) – List of keywords expected in the answer

**Returns**
Dictionary containing transcript and audio metrics

**Return type**
dict

test_eval.**judge_answer**(*question*, *answer*)

Evaluate an answer using basic GPT criteria.

**Parameters**

- **question** (*str*) – The question that was asked
- **answer** (*str*) – The student's answer to evaluate

**Returns**
JSON string containing score (1-10) and feedback comment

**Return type**
str

test_eval.**judge_answer_detailed**(*question*, *answer*, *scores=None*)

Evaluate an answer using detailed GPT criteria with category scores.

**Parameters**

- **question** (*str*) – The question that was asked
- **answer** (*str*) – The student's answer to evaluate

- **scores** (*dict or None*) – Optional system scores for reference

**Returns**
JSON string containing overall score, category scores, and feedback comment

**Return type**
str

test_eval.**judge_answer_gemini**(*question*, *answer*, *scores=None*) → dict

Evaluate an answer using Google Gemini AI with structured response format.

**Parameters**

- **question** (*str*) – The question that was asked
- **answer** (*str*) – The student's answer to evaluate
- **scores** (*dict or None*) – Optional system scores for reference

**Returns**
Dictionary containing score, category scores, and feedback comment

**Return type**
dict

test_eval.**record_audio**(*file_name*, *duration=10*)

Record audio from microphone and save to file.

**Parameters**

- **file_name** (*str*) – The filename to save the recorded audio
- **duration** (*int*) – Duration of recording in seconds

**Returns**
None

**Return type**
None

test_eval.**run_full_evaluation**(*question*, *keywords*, *audio_file*, *use_deepgram=True*)

Main evaluation function that processes audio and returns complete results.

**Parameters**

- **question** (*str*) – The question that was asked to the student
- **keywords** (*list*) – Expected keywords for the answer evaluation
- **audio_file** (*str*) – Path to the audio file containing the student's answer
- **use_deepgram** (*bool*) – Whether to use Deepgram for transcription (default: True)

**Returns**
Complete evaluation results including transcript, metrics, and scores, or None if failed

**Return type**
dict or None

test_eval.**transcribe_audio_deepgram**(*audio_path*)

Transcribe audio using Deepgram API with filler word detection.

**Parameters**
**audio_path** (*str*) – Path to the audio file to transcribe

**Returns**
Dictionary containing transcript, fillers, and word data, or None if failed

**Return type**
dict or None

CHAPTER

TWO

# API REFERENCE

## 2.1 Backend Modules

### 2.1.1 Flask Application (app.py)

app.**evaluate**()

POST endpoint to evaluate a student's audio answer.

Receives a question, keywords, and audio file from the frontend, processes the audio through transcription and evaluation pipeline, and returns comprehensive results.

Expected form data: - question: The question that was asked - keywords: List of expected keywords for evaluation - audio: Audio file containing the student's answer (webm/ogg format)

**Returns**
JSON response containing transcript, audio metrics, evaluation scores, and feedback

**Return type**
flask.Response

Response format: {

"transcript": str, "audio_metrics": dict, "evaluation": dict, "comment": str

}

Error responses: - 400: Missing required fields (question, keywords, or audio) - 500: Audio conversion failure or evaluation pipeline failure

app.**get_current_question**()

GET endpoint to retrieve the current question.

**Returns**
JSON response containing the current question text

**Return type**
flask.Response

app.**next_question**()

POST endpoint to advance to the next question in the sequence.

Advances to the next question in the predefined list. If at the end of the list, loops back to the first question. Also triggers text-to-speech for the new question.

**Returns**
JSON response with success status and the new question

**Return type**
flask.Response

app.**speak**(*text*)

Convert text to speech using pyttsx3 engine with female voice preference.

**Parameters**
**text** (*str*) – The text to be spoken aloud

**Returns**
None

**Return type**
None

### 2.1.2 Evaluation Engine (test_eval.py)

**class** test_eval.**CategoryScores**(*, *task_relevance: int*, *grammar_lexis: int*, *discourse_management: int*, *pronunciation_fluency: int*, *coherence_appropriateness: int*)

Bases: BaseModel

**coherence_appropriateness: int**

**discourse_management: int**

**grammar_lexis: int**

**model_config: ClassVar[ConfigDict] = {}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

**pronunciation_fluency: int**

**task_relevance: int**

**class** test_eval.**Verdict**(*, *score: int*, *category_scores:* CategoryScores, *comment: str*)

Bases: BaseModel

**category_scores: *CategoryScores***

**comment: str**

**model_config: ClassVar[ConfigDict] = {}**

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

**score: int**

test_eval.**analyze_audio**(*file_path*)

Analyze audio file for pitch, duration, and estimated speaking rate.

**Parameters**
**file_path** (*str*) – Path to the audio file to analyze

**Returns**
Dictionary containing duration, average pitch, and estimated words per minute

**Return type**
dict

test_eval.**check_grammar**(*text*)

Check grammar and language issues in the provided text.

**Parameters**
**text** (*str*) – The text to check for grammar issues

**Returns**
List of grammar issues found by LanguageTool

**Return type**
list

test_eval.**create_error_verdict**(*error_message: str*) → dict

Create a default error verdict.

**Parameters**
**error_message** (*str*) – The error message to include in the verdict

**Returns**
Dictionary containing default error verdict with zero scores

**Return type**
dict

test_eval.**evaluate_answer**(*transcript*, *audio_metrics*, *expected_keywords*)

Basic evaluation function that returns transcript and audio metrics.

**Parameters**

- **transcript** (*str*) – The transcribed text from audio
- **audio_metrics** (*dict*) – Dictionary containing audio analysis results
- **expected_keywords** (*list*) – List of keywords expected in the answer

**Returns**
Dictionary containing transcript and audio metrics

**Return type**
dict

test_eval.**judge_answer**(*question*, *answer*)

Evaluate an answer using basic GPT criteria.

**Parameters**

- **question** (*str*) – The question that was asked
- **answer** (*str*) – The student's answer to evaluate

**Returns**
JSON string containing score (1-10) and feedback comment

**Return type**
str

test_eval.**judge_answer_detailed**(*question*, *answer*, *scores=None*)

Evaluate an answer using detailed GPT criteria with category scores.

**Parameters**

- **question** (*str*) – The question that was asked
- **answer** (*str*) – The student's answer to evaluate

- **scores** (*dict or None*) – Optional system scores for reference

**Returns**
JSON string containing overall score, category scores, and feedback comment

**Return type**
str

test_eval.**judge_answer_gemini**(*question*, *answer*, *scores=None*) → dict

Evaluate an answer using Google Gemini AI with structured response format.

**Parameters**

- **question** (*str*) – The question that was asked
- **answer** (*str*) – The student's answer to evaluate
- **scores** (*dict or None*) – Optional system scores for reference

**Returns**
Dictionary containing score, category scores, and feedback comment

**Return type**
dict

test_eval.**record_audio**(*file_name*, *duration=10*)

Record audio from microphone and save to file.

**Parameters**

- **file_name** (*str*) – The filename to save the recorded audio
- **duration** (*int*) – Duration of recording in seconds

**Returns**
None

**Return type**
None

test_eval.**run_full_evaluation**(*question*, *keywords*, *audio_file*, *use_deepgram=True*)

Main evaluation function that processes audio and returns complete results.

**Parameters**

- **question** (*str*) – The question that was asked to the student
- **keywords** (*list*) – Expected keywords for the answer evaluation
- **audio_file** (*str*) – Path to the audio file containing the student's answer
- **use_deepgram** (*bool*) – Whether to use Deepgram for transcription (default: True)

**Returns**
Complete evaluation results including transcript, metrics, and scores, or None if failed

**Return type**
dict or None

test_eval.**transcribe_audio_deepgram**(*audio_path*)

Transcribe audio using Deepgram API with filler word detection.

**Parameters**
**audio_path** (*str*) – Path to the audio file to transcribe

**Returns**
Dictionary containing transcript, fillers, and word data, or None if failed

**Return type**
dict or None

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX