AI Voice Detection API - Evaluation Guide

Overview

This document explains how your submitted API endpoint will be evaluated in the AI Voice Detection hackathon challenge. Understanding this evaluation process will help you build an API that meets all requirements and maximizes your score.

Platform Submission Process

How to Submit Your API

Follow these steps to submit your solution on the hackathon platform:

Step 1: Navigate to Timeline Page

- Go to the **Timeline** page in the hackathon platform
- Look for the card titled **"Final Submission: API Endpoints"**

Step 2: Wait for Submission Window

- The **Submit** button will become active only when the level starts
- You cannot submit before the designated start time
- Make sure to complete your submission before the deadline

Step 3: Submit Your Details Once the submission button is active, you will need to provide:

1. **Deployment URL**: Your publicly accessible API endpoint
    - Example: `https://your-api.example.com/detect`
    - Must be live and accessible from the internet
2. **API Key**: Your authentication key
    - This is the key your API expects in the `x-api-key` header
    - Example: `abc123xyz789`
3. **GitHub URL**: Link to your source code repository
    - Example: `https://github.com/username/voice-detection-api`
    - Repository must be public or accessible to evaluators

Important Notes:

- Double-check all URLs before submitting

- Ensure your API is live and responding correctly
- Test your endpoint using the self-evaluation tool (provided below)
- You may have limited submission attempts, so verify everything first

Submission Requirements

Your submission must include:

1. **Deployed URL**: A publicly accessible endpoint URL
2. **API Key**: Authentication key for your API (optional but recommended)
3. **GitHub URL**: Your complete source code must be uploaded to GitHub and the repository URL submitted

Code Review

Important: In certain cases, our evaluation team will conduct a manual review of your submitted GitHub repository. This may occur when:

- Your API achieves exceptionally high scores (to verify authenticity)
- There are suspicious patterns in the evaluation results
- We need to verify compliance with hackathon rules and guidelines
- Random quality checks for fairness

GitHub Repository Requirements:

- Must be public or accessible to evaluators
- Should contain complete, runnable source code
- Must include a README with setup instructions
- Should document model architecture and approach
- Include requirements.txt or equivalent dependency file

What we look for in code review:

- Original implementation (not copied solutions)
- Proper attribution for any third-party libraries or models
- Code quality and documentation
- Alignment between submitted API behavior and source code
- Compliance with competition rules

Evaluation Process

1. Request Format

Your API will receive POST requests with the following structure:

Headers:

```json
{
  "Content-Type": "application/json",
  "x-api-key": "your-submitted-api-key"
}
```

Request Body:

```json
{
  "language": "English",
  "audioFormat": "mp3",
  "audioBase64": "base64_encoded_audio_data"
}
```

Fields:

- `language`: Language of the audio (e.g., "English", "Spanish", "French")
- `audioFormat`: Format of audio file (will be "mp3")
- `audioBase64`: Base64-encoded audio file content

2. Required Response Format

Your API **must** return a JSON response with the following structure:

```json
{
  "status": "success",
  "classification": "HUMAN",
  "confidenceScore": 0.85
}
```

Required Fields:

Field	Type	Valid Values	Description
`status`	string	`"success"` or `"error"`	Indicates if processing was successful
`classification`	string	`"HUMAN"` or `"AI_GENERATED"`	The predicted classification
`confidenceScore`	number	0.0 to 1.0	Confidence level of the prediction

3. Validation Rules

Your response will be validated against these strict criteria:

✅ **Must Pass All Checks:**

1. **HTTP Status Code**: Must return `200 OK`
2. **Response Type**: Must be a valid JSON object
3. **Required Fields**: All three fields (`status`, `classification`, `confidenceScore`) must be present
4. **Status Value**: Must be exactly `"success"` (case-sensitive)
5. **Classification Value**: Must be exactly `"HUMAN"` or `"AI_GENERATED"` (case-sensitive)
6. **Confidence Score Range**: Must be a number between 0 and 1 (inclusive)

❌ **Common Failures:**

- Missing any required field
- `confidenceScore` outside 0-1 range
- Invalid classification values (e.g., "Human", "human", "AI", "ARTIFICIAL")

- Non-200 HTTP status codes
- Response timeout (>30 seconds)
- Connection errors

Scoring System

Score Calculation

Each test audio file is worth an equal portion of 100 points:

```
None
Score per file = 100 / Total number of test files
```

Per-File Scoring Rules

Your score for each file depends on:

1. **Correct Classification** (Required)
    - Classification must match the expected value
    - Wrong classification = 0 points for that file
2. **Confidence Score** (Affects final points)
    - The confidence score scales your earned points:

Confidence Score	Points Awarded
≥ 0.8	100% of file score
0.6 - 0.79	75% of file score
0.4 - 0.59	50% of file score
< 0.4	25% of file score

Example Scoring

Scenario: 10 test files (each worth 10 points)

File	Expected	Your Response	Confidence	Points Earned
1	HUMAN	HUMAN	0.92	10.0 (100%)
2	AI_GENERATED	AI_GENERATED	0.75	7.5 (75%)
3	HUMAN	HUMAN	0.55	5.0 (50%)
4	AI_GENERATED	HUMAN	0.80	0.0 (wrong)
5	HUMAN	HUMAN	0.35	2.5 (25%)

Final Score: Sum of all points earned, rounded to nearest integer

Test Cases

Your API will be tested with:

- **Multiple audio samples** in various languages
- **Mix of human and AI-generated voices**
- **Different audio qualities and durations**
- Each test file has a known correct classification

You won't know the specific test files in advance, so ensure your model is robust across various scenarios.

Example Implementation

Sample API Endpoint (Python/Flask)

```python
from flask import Flask, request, jsonify
import base64
import io

app = Flask(__name__)
```

```python
@app.route('/detect-voice', methods=['POST'])
def detect_voice():
    try:
        # Validate API key
        api_key = request.headers.get('x-api-key')
        if api_key != 'your-expected-key':
            return jsonify({
                'status': 'error',
                'message': 'Invalid API key'
            }), 401

        # Parse request
        data = request.json
        language = data.get('language')
        audio_format = data.get('audioFormat')
        audio_base64 = data.get('audioBase64')

        # Decode audio
        audio_bytes = base64.b64decode(audio_base64)

        # YOUR MODEL INFERENCE HERE
        # classification, confidence =
your_model.predict(audio_bytes, language)

        # Example response
        return jsonify({
            'status': 'success',
            'classification': 'HUMAN',  # or 'AI_GENERATED'
            'confidenceScore': 0.85
        }), 200

    except Exception as e:
        return jsonify({
            'status': 'error',
            'message': str(e)
```

```
        }), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Self-Evaluation Tool

Before submitting your API, you can use this evaluation script to test it exactly as our system will. This helps you identify and fix issues before official submission.

Evaluation Script (Python)

Save this as `test_my_api.py`:

Python

```
import requests
import base64
import json

def evaluate_voice_detection_api(endpoint_url, api_key,
test_files):
    """
    Evaluate your voice detection API locally using the same
logic as the official evaluator.

    Args:
        endpoint_url: Your deployed API endpoint URL
        api_key: Your API key
        test_files: List of test cases with format:
                  [
                      {
                          'language': 'English',
                          'file_path': 'path/to/audio.mp3',
```

```
                        'expected_classification': 'HUMAN'  #
or 'AI_GENERATED'
                    }
                ]
    """

    if not endpoint_url:
        print("❌ Error: Endpoint URL is required")
        return False

    if not test_files or len(test_files) == 0:
        print("❌ Error: No test files provided")
        return False

    total_files = len(test_files)
    score_per_file = 100 / total_files
    total_score = 0
    file_results = []

    print(f"\n{'='*60}")
    print(f"🚀 Starting Evaluation")
    print(f"{'='*60}")
    print(f"Endpoint: {endpoint_url}")
    print(f"Total Test Files: {total_files}")
    print(f"Score per File: {score_per_file:.2f}")
    print(f"{'='*60}\n")

    for idx, file_data in enumerate(test_files):
        language = file_data.get('language', 'English')
        file_path = file_data.get('file_path', '')
        expected_classification =
file_data.get('expected_classification', '')

        print(f"📝 Test {idx + 1}/{total_files}: {file_path}")

        if not file_path or not expected_classification:
```

```
            result = {
                'fileIndex': idx,
                'language': language,
                'expectedClassification':
expected_classification,
                'status': 'skipped',
                'message': 'Missing file path or expected
classification',
                'score': 0
            }
            file_results.append(result)
            print(f"   ⚠️  Skipped: Missing file path or expected
classification\n")
            continue

        # Read and encode audio file
        try:
            with open(file_path, 'rb') as audio_file:
                audio_base64 =
base64.b64encode(audio_file.read()).decode('utf-8')
        except Exception as e:
            result = {
                'fileIndex': idx,
                'language': language,
                'expectedClassification':
expected_classification,
                'status': 'failed',
                'message': f'Failed to read audio file:
{str(e)}',
                'score': 0
            }
            file_results.append(result)
            print(f"   ❌ Failed to read file: {str(e)}\n")
            continue

        # Prepare request
```

```
        headers = {
            'Content-Type': 'application/json',
            'x-api-key': api_key
        }

        request_body = {
            'language': language,
            'audioFormat': 'mp3',
            'audioBase64': audio_base64
        }

        try:
            # Send request
            response = requests.post(endpoint_url,
headers=headers, json=request_body, timeout=30)

            # Check HTTP status code
            if response.status_code != 200:
                result = {
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'status': 'failed',
                    'message': f'API returned status
{response.status_code}',
                    'score': 0
                }
                file_results.append(result)
                print(f"   ❌ HTTP Status:
{response.status_code}")
                print(f"   Response: {response.text}\n")
                continue

            response_data = response.json()
```

```
            # Validate response is a JSON object
            if not isinstance(response_data, dict):
                result = {
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'status': 'failed',
                    'message': 'Response is not a valid JSON
object',
                    'score': 0
                }
                file_results.append(result)
                print(f"   ❌ Invalid response type (not a JSON
object)\n")
                continue

            response_status = response_data.get('status', '')
            response_classification =
response_data.get('classification', '')
            confidence_score =
response_data.get('confidenceScore', None)

            # Validate required fields exist
            if not response_status or not response_classification
or confidence_score is None:
                result = {
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'status': 'failed',
                    'message': 'Response missing required fields
(status, classification, confidenceScore)',
                    'responseData': response_data,
                    'score': 0
```

```
                }
                file_results.append(result)
                print(f"   ❌ Missing required fields")
                print(f"   Response: {json.dumps(response_data,
indent=2)}\n")
                continue

            # Validate status is success
            if response_status != 'success':
                result = {
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'status': 'failed',
                    'message': f'API returned status:
{response_status}',
                    'responseData': response_data,
                    'score': 0
                }
                file_results.append(result)
                print(f"   ❌ Status not 'success':
{response_status}\n")
                continue

            # Validate confidenceScore is a number between 0 and
1
            if not isinstance(confidence_score, (int, float)) or
confidence_score < 0 or confidence_score > 1:
                result = {
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'status': 'failed',
```

```
                    'message': f'Invalid confidenceScore:
{confidence_score}. Must be between 0 and 1',
                    'responseData': response_data,
                    'score': 0
                }
                file_results.append(result)
                print(f"   ❌ Invalid confidence score:
{confidence_score} (must be 0-1)\n")
                continue

            # Validate classification value
            valid_classifications = ['HUMAN', 'AI_GENERATED']
            if response_classification not in
valid_classifications:
                result = {
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'status': 'failed',
                    'message': f'Invalid classification:
{response_classification}. Must be HUMAN or AI_GENERATED',
                    'responseData': response_data,
                    'score': 0
                }
                file_results.append(result)
                print(f"   ❌ Invalid classification:
{response_classification}\n")
                continue

            # Score calculation
            file_score = 0
            if response_classification ==
expected_classification:
                # Scale score by confidence
                if confidence_score >= 0.8:
```

```
                    file_score = score_per_file
                    confidence_tier = "100%"
                elif confidence_score >= 0.6:
                    file_score = score_per_file * 0.75
                    confidence_tier = "75%"
                elif confidence_score >= 0.4:
                    file_score = score_per_file * 0.5
                    confidence_tier = "50%"
                else:
                    file_score = score_per_file * 0.25
                    confidence_tier = "25%"

                total_score += file_score

                result = {
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'actualClassification':
response_classification,
                    'confidenceScore': confidence_score,
                    'status': 'success',
                    'matched': True,
                    'score': round(file_score, 2),
                    'responseData': response_data
                }
                file_results.append(result)
                print(f"   ✅ Classification:
{response_classification} (Correct!)")
                print(f"   📊 Confidence: {confidence_score:.2f}
→ {confidence_tier} of points")
                print(f"   🎯 Score:
{file_score:.2f}/{score_per_file:.2f}\n")
            else:
                result = {
```

```
                    'fileIndex': idx,
                    'language': language,
                    'expectedClassification':
expected_classification,
                    'actualClassification':
response_classification,
                    'confidenceScore': confidence_score,
                    'status': 'success',
                    'matched': False,
                    'score': 0,
                    'responseData': response_data
                }
                file_results.append(result)
                print(f"   ❌ Classification:
{response_classification} (Expected: {expected_classification})")
                print(f"   📊 Confidence:
{confidence_score:.2f}")
                print(f"   🎯 Score: 0/{score_per_file:.2f}\n")

        except requests.exceptions.Timeout:
            result = {
                'fileIndex': idx,
                'language': language,
                'expectedClassification':
expected_classification,
                'status': 'failed',
                'message': 'Request timed out (>30 seconds)',
                'score': 0
            }
            file_results.append(result)
            print(f"   ⏱️  Timeout: Request took longer than 30
seconds\n")

        except requests.exceptions.ConnectionError:
            result = {
                'fileIndex': idx,
```

```
                'language': language,
                'expectedClassification':
expected_classification,
                'status': 'failed',
                'message': 'Connection error - unable to reach
endpoint',
                'score': 0
            }
            file_results.append(result)
            print(f"   🔌 Connection Error: Unable to reach
endpoint\n")

        except Exception as e:
            result = {
                'fileIndex': idx,
                'language': language,
                'expectedClassification':
expected_classification,
                'status': 'failed',
                'message': str(e),
                'score': 0
            }
            file_results.append(result)
            print(f"   ❌ Error: {str(e)}\n")

    # Calculate final score
    final_score = round(total_score)

    # Print summary
    print(f"{'='*60}")
    print(f"📊 EVALUATION SUMMARY")
    print(f"{'='*60}")
    print(f"Total Files Tested: {total_files}")
    print(f"Final Score: {final_score}/100")
    print(f"{'='*60}\n")
```

```
    # Print detailed results
    successful = sum(1 for r in file_results if r.get('matched',
False))
    failed = sum(1 for r in file_results if r['status'] ==
'failed')
    wrong = sum(1 for r in file_results if r['status'] ==
'success' and not r.get('matched', False))

    print(f"✅ Correct Classifications:
{successful}/{total_files}")
    print(f"❌ Wrong Classifications: {wrong}/{total_files}")
    print(f"⚠️  Failed/Errors: {failed}/{total_files}\n")

    # Save detailed results to JSON
    results_file = 'evaluation_results.json'
    with open(results_file, 'w') as f:
        json.dump({
            'finalScore': final_score,
            'totalFiles': total_files,
            'scorePerFile': round(score_per_file, 2),
            'successfulClassifications': successful,
            'wrongClassifications': wrong,
            'failedTests': failed,
            'fileResults': file_results
        }, f, indent=2)

    print(f"💾 Detailed results saved to: {results_file}\n")

    return True


# Example usage
if __name__ == '__main__':
    # Configure your API details
    ENDPOINT_URL = 'https://your-api-url.com/detect'
    API_KEY = 'your-api-key-here'
```

```
    # Define your test cases
    TEST_FILES = [
        {
            'language': 'English',
            'file_path': 'test_data/human_voice_1.mp3',
            'expected_classification': 'HUMAN'
        },
        {
            'language': 'English',
            'file_path': 'test_data/ai_voice_1.mp3',
            'expected_classification': 'AI_GENERATED'
        },
        {
            'language': 'Spanish',
            'file_path': 'test_data/human_voice_spanish.mp3',
            'expected_classification': 'HUMAN'
        },
        {
            'language': 'French',
            'file_path': 'test_data/ai_voice_french.mp3',
            'expected_classification': 'AI_GENERATED'
        }
    ]

    # Run evaluation
    evaluate_voice_detection_api(ENDPOINT_URL, API_KEY,
TEST_FILES)
```

How to Use the Evaluation Script

Step 1: Install Required Library

```
Shell
pip install requests
```

Step 2: Prepare Test Data

Create a folder structure with your test audio files:

```
None
your_project/
├── test_my_api.py
└── test_data/
    ├── human_voice_1.mp3
    ├── ai_voice_1.mp3
    ├── human_voice_spanish.mp3
    └── ai_voice_french.mp3
```

Step 3: Configure the Script

Edit the script's main section with your details:

```python
Python
ENDPOINT_URL = 'https://your-actual-endpoint.com/detect'
API_KEY = 'your-actual-api-key'

TEST_FILES = [
    {
        'language': 'English',
        'file_path': 'test_data/human_voice_1.mp3',
        'expected_classification': 'HUMAN'
    },
    # Add more test cases...
]
```

Step 4: Run the Evaluation

```
Shell
python test_my_api.py
```

Understanding the Output

The script will show real-time progress:

```
None
============================================================
🚀 Starting Evaluation
============================================================
Endpoint: https://your-api.com/detect
Total Test Files: 4
Score per File: 25.00
============================================================

📝 Test 1/4: test_data/human_voice_1.mp3
   ✅ Classification: HUMAN (Correct!)
   📊 Confidence: 0.92 → 100% of points
   🎯 Score: 25.00/25.00

📝 Test 2/4: test_data/ai_voice_1.mp3
   ❌ Classification: HUMAN (Expected: AI_GENERATED)
   📊 Confidence: 0.65
   🎯 Score: 0/25.00

============================================================
📊 EVALUATION SUMMARY
============================================================
Total Files Tested: 4
Final Score: 75/100
============================================================

✅ Correct Classifications: 3/4
❌ Wrong Classifications: 1/4
⚠️  Failed/Errors: 0/4

💾 Detailed results saved to: evaluation_results.json
```

Detailed Results File

The script generates `evaluation_results.json` with complete details:

JSON


```json
{
  "finalScore": 75,
  "totalFiles": 4,
  "scorePerFile": 25.0,
  "successfulClassifications": 3,
  "wrongClassifications": 1,
  "failedTests": 0,
  "fileResults": [
    {
      "fileIndex": 0,
      "language": "English",
      "expectedClassification": "HUMAN",
      "actualClassification": "HUMAN",
      "confidenceScore": 0.92,
      "status": "success",
      "matched": true,
      "score": 25.0,
      "responseData": {
        "status": "success",
        "classification": "HUMAN",
        "confidenceScore": 0.92
      }
    }
  ]
}
```


Quick Testing with cURL

For quick manual tests, use this curl command:

Shell

```shell
curl -X POST https://your-api-url.com/detect \
  -H "Content-Type: application/json" \
  -H "x-api-key: your-api-key" \
```

```
  -d '{
    "language": "English",
    "audioFormat": "mp3",
    "audioBase64": "your_base64_encoded_audio"
  }'
```

Expected Response:

```JSON
{
  "status": "success",
  "classification": "HUMAN",
  "confidenceScore": 0.85
}
```

Evaluation Results

After evaluation, you'll receive detailed results including:

- **Final Score**: Your total score out of 100
- **Per-File Results**: Breakdown for each test case
    - Expected vs. actual classification
    - Confidence score
    - Points earned
    - Success/failure status
- **Error Messages**: Details on any failures

Tips for Success

1. **Validate Input**: Ensure your API handles all input fields correctly
2. **Error Handling**: Return proper error responses with `status: "error"`
3. **Response Time**: Optimize to respond within 30 seconds
4. **Exact Format**: Match the response format exactly (case-sensitive)

5. **Test Thoroughly**: Test with various audio samples before submission
6. **High Confidence**: Aim for confidence scores ≥ 0.8 for maximum points
7. **Edge Cases**: Handle different languages, audio qualities, and edge cases
8. **Clean Code**: Maintain clean, well-documented code in your GitHub repository
9. **README Documentation**: Include clear setup and deployment instructions
10. **Original Work**: Ensure your solution is your original work with proper attributions

Common Issues & Solutions

Issue	Solution
Timeout errors	Optimize model inference time or use async processing
Wrong classification format	Use exactly `"HUMAN"` or `"AI_GENERATED"` (uppercase)
Confidence score issues	Ensure it's a float/int between 0.0 and 1.0
Missing fields	Include all three required fields in every response
API key errors	Ensure you're reading from `x-api-key` header

Support

If you have questions about the evaluation process:

1. Review this documentation thoroughly
2. Test your API with sample requests
3. Check the detailed error messages in evaluation results
4. Contact hackathon support with specific questions

Good luck with your submission!