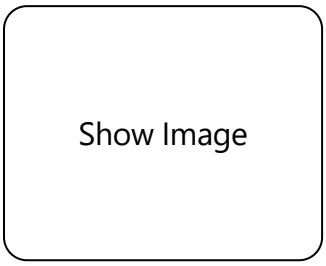
Wake Word Detector

A lightweight, efficient wake word detection system designed for easy integration into other applications. This module provides a complete pipeline for audio capture, feature extraction, and wake word recognition using a pre-trained neural network model.



Features

- Simple, clean API for easy integration
- Both blocking and non-blocking detection modes
- Event-based callback system
- Thread-safe operation
- Command-line interface for testing
- Efficient resource management
- Configurable detection sensitivity

Installation

Prerequisites

The wake word detector requires the following libraries:

bash Copy

```
pip install torch pyaudio librosa numpy
```

Setup

1. Download the wake word detector module:

   bash Copy

   ```
   # Place wake_word_detector.py in your project directory
   ```

2. Obtain a trained wake word model (.pth file) - this module is compatible with models trained using the standard Io wake word architecture.

Basic Usage

Standalone Testing

You can test the wake word detector directly from the command line:

bash Copy

```bash
python wake_word_detector.py --model path/to/model.pth
```

Additional options:

bash Copy

```bash
python wake_word_detector.py --model path/to/model.pth --threshold 0.8 --continuous
python wake_word_detector.py --list-devices  # View available audio devices
python wake_word_detector.py --model path/to/model.pth --device 1  # Specify device
```

Python Module Import

python Copy

```python
from wake_word_detector import WakeWordDetector

# Create detector with your model
detector = WakeWordDetector(model_path="path/to/model.pth")

# Simple blocking detection
if detector.listen_for_wake_word(timeout=5):
    print("Wake word detected within 5 seconds!")
```

Integration Guide

1. Import and Initialize

python Copy

```python
from wake_word_detector import WakeWordDetector

class MyApplication:
    def __init__(self):
        # Initialize wake word detector
        self.detector = WakeWordDetector(
            model_path="path/to/model.pth",  # Required
            threshold=0.85,                  # Optional (default: 0.85)
            device_index=None,               # Optional (default: system default)
            sample_rate=16000                # Optional (default: 16000 Hz)
        )

        # Register detection callback
        self.detector.register_detection_callback(self.on_wake_word_detected)
```

2. Choose an Integration Method

The wake word detector offers three methods of integration:

Option 1: Blocking Detection

Best for simple applications where you want to wait for a wake word before proceeding.

python Copy

```python
def run_blocking(self):
    print("Listening for wake word...")
    if self.detector.listen_for_wake_word(timeout=10):
        print("Wake word detected!")
        self.process_user_request()
    else:
        print("No wake word detected within timeout period")
```

Option 2: Non-Blocking Polling

Best for GUI applications or systems that need to remain responsive.

python Copy

```python
def run_polling(self):
    # Start detector in background
    self.detector.start()

    try:
        while self.running:
            # Check for detection (non-blocking)
            if self.detector.is_detected():
                print("Wake word detected!")
                self.process_user_request()

            # Do other work while checking periodically
            time.sleep(0.1)
    finally:
        # Clean up when done
        self.detector.stop()
```

Option 3: Event-Driven Callbacks

Best for reactive systems that should respond immediately to detections.

python Copy

```python
def run_event_driven(self):
    # Define callback function
    def on_wake_word_detected(confidence):
        print(f"Wake word detected with {confidence:.2f} confidence!")
        self.process_user_request()

    # Register callback and start detector
    self.detector.register_detection_callback(on_wake_word_detected)
    self.detector.start()

    # Your application can continue doing other things
    # The callback will be invoked whenever the wake word is detected

    # When finished:
    # self.detector.stop()
```

3. Continuous Listening

For systems that should continuously listen for multiple wake word activations:

python Copy

```python
def run_continuous(self):
    def on_wake_word(confidence):
        print(f"Wake word detected! ({confidence:.2f})")
        self.process_user_request()
        print("Listening for next wake word...")

    print("Starting continuous wake word detection...")
    self.detector.listen_for_wake_word(
        timeout=None,       # Listen indefinitely
        continuous=True,    # Enable continuous mode
        on_detect=on_wake_word
    )
```

4. Integration with Master Program Lifecycle

python Copy

```python
class MyVoiceAssistant:
    def __init__(self):
        self.wake_detector = WakeWordDetector("model.pth")
        self.speech_recognizer = MySpeechRecognizer()
        self.nlp_engine = MyNLPEngine()
        self.tts_engine = MyTTSEngine()
        self.running = False

    def start(self):
        self.running = True
        self.wake_detector.register_detection_callback(self.on_wake_word)
        self.wake_detector.start()
        print("Voice assistant is listening...")

    def stop(self):
        self.running = False
        self.wake_detector.stop()
        print("Voice assistant stopped")

    def on_wake_word(self, confidence):
        print(f"Wake word detected ({confidence:.2f})")

        # Play acknowledgment tone
        self.play_tone()

        # Capture voice command (using a different audio processing pipeline)
        audio = self.speech_recognizer.listen(timeout=5)

        # Process command
        if audio:
            text = self.speech_recognizer.recognize(audio)
            if text:
                response = self.nlp_engine.process(text)
                self.tts_engine.speak(response)
```

API Reference

WakeWordDetector

python Copy

```python
detector = WakeWordDetector(
    model_path="path/to/model.pth",  # Path to PyTorch model
    threshold=0.85,                  # Detection threshold (0.0-1.0)
    device_index=None,               # Audio device index (None for default)
    sample_rate=16000                # Audio sample rate in Hz
)
```

Methods

- `start()` - Start listening for wake words in the background
- `stop()` - Stop listening and free resources
- `is_detected()` - Non-blocking check if wake word was detected (returns boolean)
- `register_detection_callback(callback_function)` - Register function to call when wake word is detected
- `listen_for_wake_word(timeout=None, continuous=False, on_detect=None)` - Block until wake word is detected or timeout occurs

Advanced Configuration

Adjusting Detection Sensitivity

The detection threshold controls the trade-off between false positives and false negatives:

- Lower values (e.g., 0.7) increase sensitivity but may cause more false positives
- Higher values (e.g., 0.9) reduce false positives but might miss some detections

python Copy

```python
# More sensitive detection
detector = WakeWordDetector(model_path="model.pth", threshold=0.7)

# Less sensitive detection
detector = WakeWordDetector(model_path="model.pth", threshold=0.9)
```

Listing and Selecting Audio Devices

python Copy

```python
from wake_word_detector import AudioCapture

# List available devices
audio = AudioCapture()
devices = audio.list_devices()
for device in devices:
    print(f"Device {device['index']}: {device['name']}")

# Specify device for wake word detector
detector = WakeWordDetector(model_path="model.pth", device_index=2)
```

Troubleshooting

Common Issues

1. **No wake word detections**
   - Check audio input levels
   - Lower the detection threshold
   - Ensure model is compatible
2. **Too many false detections**
   - Increase detection threshold
   - Check for background noise
3. **Audio device errors**
   - Use `--list-devices` to find correct device index
   - Check microphone permissions

Debug Logging

Enable debug logging for more detailed information:

python Copy

```python
import logging
logging.getLogger("WakeWordDetector").setLevel(logging.DEBUG)
```

Or when using the command-line interface:

bash Copy

```bash
python wake_word_detector.py --model model.pth --debug
```

Performance Considerations

- The detector runs in separate threads to avoid blocking the main application
- CPU usage typically ranges from 5-15% on a modern system
- Memory usage is approximately 50-100MB including model
- For resource-constrained systems, consider:
  - Increasing frame size (reduces processing frequency)
  - Using a smaller model architecture