

---

## Voice Assistant - Computer Wake-Word Project

Dokument: Computer Wake-Word Training Guide

Datum: 05. Dezember 2025

---

Seite: {page}

---

# Computer Wake-Word Training Guide

---

## Schritt-für-Schritt-Anleitung für “Computer” Wake-Word Implementation

---

### Inhaltsverzeichnis

---

1. [Voraussetzungen](#)
  2. [Methode A: Porcupine \(Empfohlen - 30 Minuten\)](#)
  3. [Methode B: OpenWakeWord \(Alternative - 4-8 Stunden\)](#)
  4. [Integration in Voice Assistant](#)
  5. [Testing & Optimierung](#)
  6. [Troubleshooting](#)
- 

### Teil 1: Voraussetzungen

---

#### System-Anforderungen

Hardware:

- Windows 11 Mini PC (bereits vorhanden 

- Mikrofon (bereits vorhanden ✓)
- Mindestens 8GB RAM
- 2GB freier Festplattenspeicher
- Internetverbindung für Training

## Software:

- Python 3.11 (bereits installiert ✓)
- Git (bereits installiert ✓)
- Virtual Environment .venv (bereits konfiguriert ✓)

## Software-Installation

### Schritt 1.1: Virtual Environment aktivieren

Öffne PowerShell oder Command Prompt im Projektordner:

```
cd C:\Users\ModBot\ki-sprachsteuerung  
.venv\Scripts\activate
```

### Erwartete Ausgabe:

```
(.venv) C:\Users\ModBot\ki-sprachsteuerung>
```

### Schritt 1.2: Bestehende Pakete überprüfen

```
pip list
```

### Wichtige Pakete (sollten bereits installiert sein):

- openwakeword
- vosk
- edge-tts
- sounddevice
- numpy

- webrtcvad

## Schritt 1.3: Mikrofon-Setup & Test

### Mikrofon testen mit Python:

```
# Speichere als test_microphone.py
import sounddevice as sd
import numpy as np

print("Verfügbare Audio-Geräte:")
print(sd.query_devices())

print("\nMikrofon-Test startet in 3 Sekunden...")
print("Sprich etwas ins Mikrofon!")

duration = 5 # Sekunden
sample_rate = 16000

recording = sd.rec(int(duration * sample_rate),
                   samplerate=sample_rate,
                   channels=1,
                   dtype=np.int16)
sd.wait()

print(f"Aufnahme abgeschlossen!")
print(f"Max Amplitude: {np.max(np.abs(recording))}")
print(f"Durchschnitt: {np.mean(np.abs(recording))}")

if np.max(np.abs(recording)) < 100:
    print("⚠ WARNUNG: Mikrofon zu leise! Überprüfe Einstellungen.")
else:
    print("✅ Mikrofon funktioniert korrekt!")
```

### Ausführen:

```
python test_microphone.py
```

### Windows Mikrofon-Einstellungen überprüfen:

1. Windows-Taste + I → System → Sound
2. Eingabegerät auswählen

3. Lautstärke auf 80-100% einstellen
  4. "Mikrofon testen" klicken und sprechen
  5. Balken sollte sich bewegen
- 

## Teil 2: Methode A - Porcupine (Empfohlen)

---

### Warum Porcupine?

Porcupine ist die empfohlene Methode, weil:

- Kein Aufnahme-Prozess notwendig
- Training in Sekunden statt Stunden
- Produktionsreife Qualität
- Kostenlos für persönliche Projekte
- Perfekt für Windows 11

**Geschätzte Gesamtzeit:** 30 Minuten

---

### Schritt 2A.1: Picovoice Account erstellen

1. Öffne Browser und navigiere zu: <https://console.picovoice.ai/>
2. Klicke auf "Start Free" oder "Sign Up"
3. Registriere dich mit E-Mail: kommuniverse@gmail.com
4. Bestätige E-Mail-Adresse
5. Logge dich ein

**Wichtig:** Notiere dir den **AccessKey** nach dem Login!

- Zu finden unter: Account → AccessKey
  - Format: xxxxxxxxxxxxxxxxxxxxxxxxx
  - **Geheim halten!** Nicht in GitHub committen!
-

## Schritt 2A.2: “Computer” Wake-Word trainieren

### 1. In Picovoice Console:

- Klicke auf “Porcupine” in der Navigation
- Oder direkt: <https://console.picovoice.ai/porcupine>

### 2. Sprache auswählen:

- Language: **English** (für “Computer”)

### 3. Wake-Word eingeben:

- Textfeld: Tippe **“Computer”** ein
- System validiert automatisch die Phrase

### 4. Testen (Optional aber empfohlen):

- Klicke auf Mikrofon-Button
- Sage mehrmals “Computer”
- Beobachte Detection-Rate
- Teste auch ähnliche Wörter (“Comuter” , “Komputer”) → sollten NICHT erkannt werden

### 5. Trainieren:

- Klicke auf “Train” Button unten
  - Warte 5-10 Sekunden
  - Training ist abgeschlossen!
- 

## Schritt 2A.3: Modell herunterladen

### 1. Platform auswählen:

- Wähle **“Windows”** aus der Liste

### 2. Download:

- Klicke “Download”

- Datei wird heruntergeladen: `computer_windows.ppn` (ca. 50KB)

3. **Datei verschieben:** ``powershell

## Erstelle models Ordner falls nicht vorhanden

---

```
mkdir C:\Users\ModBot\ki-sprachsteuerung\models
```

```
# Verschiebe .ppn Datei aus Downloads move  
C:\Users\ModBot\Downloads\computer_windows.ppn C:\Users\ModBot\ki-  
sprachsteuerung\models\computer.ppn
```

```
---
```

```
### Schritt 2A.4: Porcupine Python SDK installieren  
  
```powershell  
# Virtual Environment sollte aktiviert sein  
pip install pvpocupine
```

**Erwartete Ausgabe:**

```
Collecting pvpocupine  
  Downloading pvpocupine-x.x.x-py3-none-any.whl  
Installing collected packages: pvpocupine  
Successfully installed pvpocupine-x.x.x
```

**Schritt 2A.5: AccessKey sicher speichern**

**Erstelle .env Datei:**

```
# Im Projektordner  
notepad .env
```

### Inhalt der `.env` Datei:

```
PICOVOICE_ACCESS_KEY=dein_access_key_hier
```

### Installiere `python-dotenv`:

```
pip install python-dotenv
```

### Aktualisiere `.gitignore`:

```
notepad .gitignore
```

### Füge hinzu:

```
# Environment variables  
.env  
  
# Porcupine models  
*.ppn  
models/
```

---

## Schritt 2A.6: Porcupine testen

### Erstelle `test_porcupine.py`:

```
import pvpocupine
import sounddevice as sd
import numpy as np
from dotenv import load_dotenv
import os

# Lade AccessKey aus .env
load_dotenv()
ACCESS_KEY = os.getenv('PICOVOICE_ACCESS_KEY')

# Pfad zum Computer Wake-Word Modell
KEYWORD_PATH = 'models/computer.ppn'

print("Initialisiere Porcupine...")
porcupine = pvpocupine.create(
    access_key=ACCESS_KEY,
    keyword_paths=[KEYWORD_PATH]
)

print(f"Porcupine Version: {porcupine.version}")
print(f"Sample Rate: {porcupine.sample_rate}")
print(f"Frame Length: {porcupine.frame_length}")
print("\n🎤 Höre auf 'Computer' Wake-Word...")
print("Drücke Ctrl+C zum Beenden\n")

def audio_callback(indata, frames, time, status):
    if status:
        print(f"Status: {status}")

    # Konvertiere zu int16
    audio_frame = (indata[:, 0] * 32767).astype(np.int16)

    # Verarbeite Frame
    keyword_index = porcupine.process(audio_frame)

    if keyword_index >= 0:
        print("🚀 'Computer' Wake-Word erkannt!")

try:
    with sd.InputStream(
        channels=1,
        samplerate=porcupine.sample_rate,
        blocksize=porcupine.frame_length,
        dtype=np.float32,
        callback=audio_callback
```

```
):
    while True:
        sd.sleep(100)

    except KeyboardInterrupt:
        print("\n\nBeende...")
    finally:
        porcupine.delete()
        print("Porcupine beendet.")
```

## Ausführen:

```
python test_porcupine.py
```

## Erwartetes Verhalten:

- Programm startet und wartet
- Sage “Computer”
- Ausgabe: “ ‘Computer’ Wake-Word erkannt!”
- Teste mehrmals mit verschiedenen Tonlagen

## Erfolgs-Kriterien:

- “Computer” wird zuverlässig erkannt (>90%)
- Keine Falsch-Positive bei anderen Wörtern
- Keine Doppel-Erkennungen

---

## Teil 3: Methode B - OpenWakeWord (Alternative)

---

### Wann OpenWakeWord nutzen?

Nutze OpenWakeWord wenn:

- Du vollständige Kontrolle über das Modell möchtest
- Du Zeit für Aufnahmen und Training hast (4-8 Stunden)
- Du das Modell iterativ verbessern möchtest

- Du keine Abhängigkeit von kommerziellen Services willst

**Geschätzte Gesamtzeit:** 4-8 Stunden

---

## Teil 3.1: Aufnahme-Prozess

### Wie viele Aufnahmen?

**Minimum:** 100 positive Samples **Empfohlen:** 200-500 positive Samples **Optimal:** 500+ positive Samples

### Negative Samples:

- Mindestens 1000 Samples von anderen Wörtern
- Hintergrundgeräusche (Musik, TV, Gespräche)
- Ähnlich klingende Wörter

### Aufnahme-Länge

**Pro Aufnahme:** 2 Sekunden

- 0.5s Stille vor dem Wort
- 1s für “Computer”
- 0.5s Stille nach dem Wort

### Variationen erforderlich

#### Tonlage:

- 40% normale Stimme
- 20% höhere Tonlage
- 20% tiefere Tonlage
- 10% geflüstert
- 10% laut

#### Geschwindigkeit:

- 50% normale Geschwindigkeit

- 25% schneller
- 25% langsamer

### Betonung:

- 50% normale Betonung
- 25% erste Silbe betont (“COMputer”)
- 25% zweite Silbe betont (“comPUter”)

### Umgebungs-Bedingungen

#### Aufnahme-Umgebungen:

- 60% leise Umgebung (Büro, Zimmer)
- 20% mit leichten Hintergrundgeräuschen (Musik leise)
- 10% mit mittleren Hintergrundgeräuschen (TV, Gespräche)
- 10% mit lauten Hintergrundgeräuschen (Straßenlärm)

#### Aufnahme-Format & Qualität

**Format:** WAV **Sample Rate:** 16000 Hz (16 kHz) **Channels:** Mono (1 Kanal) **Bit Depth:** 16-bit **Codec:** PCM

---

## Teil 3.2: Daten-Vorbereitung

### Ordner-Struktur

```
C:\Users\ModBot\ki-sprachsteuerung\  
└── wake_word_data/  
    ├── positive/  
    │   ├── computer_001.wav  
    │   ├── computer_002.wav  
    │   └── ... (bis computer_500.wav)  
    ├── negative/  
    │   ├── other_001.wav  
    │   ├── other_002.wav  
    │   └── ... (1000+ Dateien)  
    └── background/  
        ├── noise_001.wav  
        ├── noise_002.wav  
        └── ... (100+ Dateien)
```

### Datei-Benennung

#### Positive Samples:

```
computer_[nummer]_[variation].wav
```

Beispiele:

- computer\_001\_normal.wav
- computer\_002\_loud.wav
- computer\_003\_whisper.wav
- computer\_004\_fast.wav
- computer\_005\_slow.wav

#### Negative Samples:

negative\_[kategorie]\_[nummer].wav

Beispiele:

- negative\_speech\_001.wav
- negative\_music\_001.wav
- negative\_similar\_001.wav (z.B. "commuter")

## Daten-Augmentation

OpenWakeWord unterstützt automatische Daten-Augmentation:

- Pitch Shifting (Tonhöhen-Änderung)
- Time Stretching (Geschwindigkeits-Änderung)
- Background Noise Addition (Hintergrundgeräusche)
- Reverb (Hall-Effekt)

**Nicht manuell notwendig** - wird beim Training automatisch angewendet!

---

## Teil 3.3: Training mit OpenWakeWord

### Schritt 3B.1: Google Colab öffnen

1. Öffne Browser
2. Navigiere zu: <https://colab.research.google.com/>
3. Suche nach “OpenWakeWord Training” oder nutze direkten Link:  
<https://colab.research.google.com/drive/1q1oe2zOyZp7UsB3jJiQ1IFn8z5YfjwEb>

### Schritt 3B.2: Daten hochladen

1. In Colab: Klicke auf Ordner-Symbol links (Files)
2. Erstelle Ordner-Struktur:

```
/content/  
└── positive/  
└── negative/
```

3. Lade positive Samples in /content/positive/ hoch
4. Lade negative Samples in /content/negative/ hoch

**Tipp:** Für große Datenmengen, nutze Google Drive:

```
from google.colab import drive  
drive.mount('/content/drive')
```

### Schritt 3B.3: Training konfigurieren

Im Colab Notebook, ändere Parameter:

```
# Wake-Word Konfiguration  
WAKE_WORD = "computer"  
POSITIVE_SAMPLES_DIR = "/content/positive"  
NEGATIVE_SAMPLES_DIR = "/content/negative"  
  
# Training Parameter  
EPOCHS = 50 # Anzahl Training-Durchläufe  
BATCH_SIZE = 32  
LEARNING_RATE = 0.001  
  
# Augmentation  
USE_AUGMENTATION = True  
AUGMENTATION_FACTOR = 3 # Jedes Sample wird 3x augmentiert
```

### Schritt 3B.4: Training starten

Führe Colab Zellen aus:

1. Runtime → Run all
2. Oder Zelle für Zelle mit Shift+Enter

**Erwartete Dauer:** 30 Minuten - 2 Stunden (abhängig von Datenmenge)

### Training-Ausgabe beobachten:

```
Epoch 1/50
loss: 0.4523 - accuracy: 0.7821
Epoch 2/50
loss: 0.3214 - accuracy: 0.8534
...
Epoch 50/50
loss: 0.0821 - accuracy: 0.9723
```

### Erfolgs-Kriterien:

- Accuracy > 95%
- Loss < 0.1
- Validation Accuracy > 90%

### Schritt 3B.5: Wie erkenne ich erfolgreiches Training?

#### Gute Indikatoren:

- Accuracy steigt kontinuierlich
- Loss sinkt kontinuierlich
- Validation Accuracy nahe Training Accuracy (kein Overfitting)
- Confusion Matrix zeigt hohe True Positives

#### Schlechte Indikatoren:

- Accuracy stagniert bei <80%
- Loss bleibt hoch (>0.3)
- Validation Accuracy viel niedriger als Training Accuracy (Overfitting)
- Viele False Positives in Tests

#### Bei schlechten Ergebnissen:

- Mehr Daten sammeln (besonders negative Samples)
- Epochs erhöhen

- Learning Rate anpassen
  - Augmentation-Factor erhöhen
- 

## Teil 3.4: Model-Export

### Wo finde ich das fertige Modell?

Nach erfolgreichem Training:

```
/content/models/  
└── computer.onnx (oder computer.tflite)
```

### Welches Format?

**ONNX (.onnx):** Empfohlen für Windows/Desktop

- Bessere Performance auf CPU
- Größere Datei (~5-10 MB)

**TFLite (.tflite):** Für Edge Devices

- Kleinere Datei (~1-3 MB)
- Optimiert für mobile/embedded

**Für dieses Projekt:** Nutze .onnx

### Modell herunterladen

**In Colab:**

```
from google.colab import files  
files.download('/content/models/computer.onnx')
```

**Oder:** Rechtsklick auf Datei → Download

**Speichern unter:**

```
C:\Users\ModBot\ki-sprachsteuerung\models\computer.onnx
```

## Modell testen (in Colab)

```
from openwakeword.model import Model

# Lade Modell
owwModel = Model(wakeword_models=["./models/computer.onnx"])

# Test mit Audio-Datei
import librosa
audio, sr = librosa.load("test_audio.wav", sr=16000)

# Verarbeite Audio
predictions = owwModel.predict(audio)
print(predictions)

# Erwartete Ausgabe:
# {'computer': 0.95} # Hoher Score = Erkannt
```

## Teil 4: Integration in Voice Assistant

### Schritt 4.1: Code-Backup erstellen

Vor Änderungen:

```
cd C:\Users\ModBot\ki-sprachsteuerung
copy voice_assistant_edge_ultimate.py voice_assistant_edge_ultimate_backup.py
```

### Schritt 4.2: Neue Datei erstellen

Erstelle `voice_assistant_computer.py`:

```
copy voice_assistant_edge_ultimate.py voice_assistant_computer.py
```

## Schritt 4.3: Code-Änderungen für Porcupine

Öffne `voice_assistant_computer.py` in Editor:

```

# =====
# Voice Assistant v3.0 - "Computer" Wake-Word Edition
# Trainiertes Custom Wake-Word: "Computer" via Porcupine
# Modell-Pfad: ./models/computer.ppn
# =====

import pporcupine # NEU
from dotenv import load_dotenv # NEU
import os # NEU

# ... (andere imports bleiben gleich)

# =====
# KONFIGURATION
# =====

# Lade Environment Variables
load_dotenv() # NEU

# Wake-Word Konfiguration
WAKE_WORD = "computer" # GEÄNDERT von "hey jarvis"
PICOVOICE_ACCESS_KEY = os.getenv('PICOVOICE_ACCESS_KEY') # NEU
PORCUPINE_MODEL_PATH = "models/computer.ppn" # NEU

# Audio Konfiguration (Porcupine-spezifisch)
# SAMPLE_RATE wird von Porcupine bestimmt
# FRAME_LENGTH wird von Porcupine bestimmt

# ... (restliche Konfiguration bleibt gleich)

# =====
# WAKE-WORD DETECTION
# =====

def initialize_wake_word_detector():
    """Initialisiert Porcupine Wake-Word Detector."""
    try:
        porcupine = pporcupine.create(
            access_key=PICOVOICE_ACCESS_KEY,
            keyword_paths=[PORCUPINE_MODEL_PATH]
        )
        print(f"✅ Porcupine initialisiert (Version: {porcupine.version})")
        print(f"  Sample Rate: {porcupine.sample_rate} Hz")
        print(f"  Frame Length: {porcupine.frame_length}")
        return porcupine
    
```

```

except Exception as e:
    print(f"❌ Fehler beim Initialisieren von Porcupine: {e}")
    return None

# =====
# MAIN LOOP
# =====

def main():
    print("=" * 60)
    print("🤖 VOICE ASSISTANT v3.0 - COMPUTER WAKE-WORD")
    print("=" * 60)

    # Initialisiere Komponenten
    porcupine = initialize_wake_word_detector()
    if not porcupine:
        print("❌ Konnte Porcupine nicht initialisieren. Beende.")
        return

    recognizer = initialize_speech_recognizer()
    vad = webrtcvad.Vad(3)

    print(f"\n🎤 Höre auf Wake-Word: '{WAKE_WORD}'")
    print("Drücke Ctrl+C zum Beenden\n")

    try:
        with sd.InputStream(
            channels=1,
            samplerate=porcupine.sample_rate,
            blocksize=porcupine.frame_length,
            dtype=np.float32
        ) as stream:

            while True:
                # Lese Audio Frame
                audio_frame, overflowed = stream.read(porcupine.frame_length)

                if overflowed:
                    print("⚠️ Audio buffer overflow!")

                # Konvertiere zu int16 für Porcupine
                audio_int16 = (audio_frame[:, 0] * 32767).astype(np.int16)

                # Wake-Word Detection
                keyword_index = porcupine.process(audio_int16)

```

```

        if keyword_index >= 0:
            print(f"\n🚀 '{WAKE_WORD.upper()}' erkannt!")
            play_sound("wake_detected.wav") # Optional

            # Warte kurz
            time.sleep(0.5)

            # Starte Spracherkennung
            command = listen_for_command(recognizer, vad,
porcupine.sample_rate)

            if command:
                print(f"📝 Befehl: {command}")
                execute_command(command)
            else:
                speak("Befehl nicht verstanden.")

            # Zurück zum Wake-Word Listening
            print(f"\n🎙 Höre auf Wake-Word: '{WAKE_WORD}'")

except KeyboardInterrupt:
    print("\n\n👋 Beende Voice Assistant...")

finally:
    porcupine.delete()
    print("✅ Porcupine beendet.")

if __name__ == "__main__":
    main()

```

## Schritt 4.4: Code-Änderungen für OpenWakeWord

Falls OpenWakeWord genutzt wird:

```

# Imports
from openwakeword.model import Model

# Konfiguration
WAKE_WORD = "computer"
OWW_MODEL_PATH = "models/computer.onnx"

# Initialisierung
def initialize_wake_word_detector():
    oww_model = Model(wakeword_models=[OWW_MODEL_PATH])
    print(f"✓ OpenWakeWord initialisiert")
    return oww_model

# In Main Loop
predictions = oww_model.predict(audio_frame)
if predictions["computer"] > 0.5: # Threshold anpassbar
    print("🚀 'COMPUTER' erkannt!")
    # ... rest bleibt gleich

```

## Teil 5: Testing & Optimierung

### Test-Szenarien

#### 1. Basis-Funktionalität

##### Test: Wake-Word Erkennung

- ✓ Sage "Computer" 10x → Sollte 10x erkannt werden
- ✓ Sage "Computer" leise → Sollte erkannt werden
- ✓ Sage "Computer" laut → Sollte erkannt werden
- ✓ Sage "Computer" schnell → Sollte erkannt werden
- ✓ Sage "Computer" langsam → Sollte erkannt werden

#### 2. Falsch-Positive Tests

##### Test: Ähnliche Wörter

- Sage "Commuter" 10x → Sollte NICHT erkannt werden
- Sage "Komputer" 10x → Sollte NICHT erkannt werden
- Sage "Puter" 10x → Sollte NICHT erkannt werden
- Normale Konversation → Keine Fehl-Erkennungen

### 3. Umgebungs-Tests

**Test:** Verschiedene Umgebungen

- Leise Umgebung → Funktioniert
- Mit Musik im Hintergrund → Funktioniert
- Mit TV im Hintergrund → Funktioniert
- Mit anderen Personen im Raum → Funktioniert
- Aus verschiedenen Entfernungen (1m, 2m, 3m) → Funktioniert

### 4. Stress-Tests

**Test:** Langzeit-Stabilität

- 100x hintereinander → Keine Doppel-Erkennungen
- 1 Stunde Dauerbetrieb → Stabil, kein Memory Leak
- Schnelle Wiederholungen → Keine Verzögerungen

## Performance-Metriken

Messe folgende Werte:

```
import time

# Latenz messen
start_time = time.time()
keyword_index = porcupine.process(audio_frame)
latency = time.time() - start_time
print(f"Latenz: {latency*1000:.2f} ms")

# CPU-Auslastung messen
import psutil
cpu_percent = psutil.cpu_percent(interval=1)
print(f"CPU: {cpu_percent}%")

# Memory-Auslastung messen
import psutil
process = psutil.Process()
memory_mb = process.memory_info().rss / 1024 / 1024
print(f"Memory: {memory_mb:.2f} MB")
```

## Zielwerte:

- Latenz: < 50 ms
- CPU: < 20%
- Memory: < 200 MB

# Optimierung

## Porcupine Sensitivity anpassen

```
# Höhere Sensitivity = mehr Erkennungen, mehr False Positives
# Niedrigere Sensitivity = weniger Erkennungen, weniger False Positives

porcupine = pvp_porcupine.create(
    access_key=PICOVOICE_ACCESS_KEY,
    keyword_paths=[PORCUPINE_MODEL_PATH],
    sensitivities=[0.5] # Wert zwischen 0.0 und 1.0
)

# Empfohlene Werte:
# 0.3 - Sehr konservativ (wenig False Positives)
# 0.5 - Balanced (Standard)
# 0.7 - Aggressiv (hohe Erkennungsrate)
```

## OpenWakeWord Threshold anpassen

```
# Threshold für Erkennung
THRESHOLD = 0.5 # Standard

if predictions["computer"] > THRESHOLD:
    # Erkannt

# Empfohlene Werte:
# 0.3 - Aggressiv
# 0.5 - Balanced
# 0.7 - Konservativ
```

## Cooldown-Period hinzufügen

```
import time

last_detection_time = 0
COOLDOWN_SECONDS = 2

if keyword_index >= 0:
    current_time = time.time()
    if current_time - last_detection_time > COOLDOWN_SECONDS:
        print("Wake-Word erkannt!")
        last_detection_time = current_time
        # ... process command
    else:
        # Ignoriere (zu kurz nach letzter Erkennung)
        pass
```

## Teil 6: Troubleshooting

### Problem 1: “AccessKey invalid” (Porcupine)

#### Symptome:

```
Error: Invalid AccessKey
```

#### Lösungen:

1. Überprüfe .env Datei:

```
PICOVOICE_ACCESS_KEY=dein_key_hier
```

2. Kein Leerzeichen vor/nach dem Key
3. Keine Anführungszeichen um den Key
4. Key von Console kopieren: <https://console.picovoice.ai/>

## Problem 2: “Model file not found”

### Symptome:

```
FileNotFoundException: models/computer.ppn
```

### Lösungen:

1. Überprüfe Pfad:

```
dir models\computer.ppn
```

2. Falls nicht vorhanden, erneut von Console downloaden

3. Überprüfe Dateiname (exakt computer.ppn )

## Problem 3: Wake-Word wird nicht erkannt

### Symptome:

- Sage “Computer” aber keine Reaktion

### Lösungen:

1. Mikrofon-Test:

```
python test_microphone.py
```

2. Sensitivity erhöhen:

```
sensitivities=[0.7] # statt 0.5
```

3. Deutlicher sprechen

4. Näher ans Mikrofon

5. Hintergrundgeräusche reduzieren

## Problem 4: Zu viele False Positives

### Symptome:

- Wake-Word wird erkannt ohne dass es gesagt wurde

### Lösungen:

#### 1. Sensitivity senken:

```
sensitivities=[0.3] # statt 0.5
```

#### 2. Cooldown hinzufügen (siehe Optimierung)

#### 3. Threshold erhöhen (OpenWakeWord):

```
THRESHOLD = 0.7 # statt 0.5
```

## Problem 5: Doppel-Erkennungen

### Symptome:

- “Computer” wird 2-3x hintereinander erkannt

### Lösungen:

#### 1. Cooldown implementieren (siehe Optimierung)

#### 2. Audio-Buffer nach Erkennung leeren:

```
if keyword_index >= 0:  
    # Leere Buffer  
    stream.read(porcupine.frame_length * 10)  
    # ... process command
```

## Problem 6: Hohe CPU-Auslastung

### Symptome:

- CPU bei 50%+ konstant

### Lösungen:

#### 1. Frame-Processing optimieren:

```
# Nur jedes 2. Frame verarbeiten
frame_counter = 0
if frame_counter % 2 == 0:
    keyword_index = porcupine.process(audio_frame)
    frame_counter += 1
```

#### 2. Andere Programme schließen

#### 3. Python-Version aktualisieren

## Problem 7: Import-Fehler

### Symptome:

```
ModuleNotFoundError: No module named 'pvpocupine'
```

### Lösungen:

#### 1. Virtual Environment aktiviert?

```
.venv\Scripts\activate
```

#### 2. Paket installieren:

```
pip install pvpocupine
```

#### 3. Überprüfen:

```
pip list | findstr porcupine
```

# Checkliste: Fertig für Production

---

## Vor dem Deployment

- Wake-Word wird zuverlässig erkannt (>95%)
- Keine Falsch-Positive (% in 1h Test)
- Keine Doppel-Erkennungen
- Funktioniert in allen Ziel-Umgebungen
- CPU-Auslastung akzeptabel (<20%)
- Memory-Auslastung stabil (<200MB)
- Code dokumentiert und kommentiert
- `.env` Datei in `.gitignore`
- `.ppn` / `.onnx` Dateien in `.gitignore`
- README.md aktualisiert
- Tests dokumentiert
- Backup des alten Codes erstellt

## Nach dem Deployment

- Monitoring für 24h
  - User-Feedback sammeln
  - Performance-Metriken loggen
  - Optimierungen basierend auf Daten
-

# Nächste Schritte

---

## Sofort (heute):

1. Entscheide: Porcupine oder OpenWakeWord
2. Folge entsprechender Anleitung
3. Teste Basis-Funktionalität

## Diese Woche:

1. Umfangreiche Tests in verschiedenen Umgebungen
2. Optimierung basierend auf Testergebnissen
3. Dokumentation vervollständigen

## Nächste 2 Wochen:

1. LLM-Integration vorbereiten
  2. Weitere Befehle hinzufügen
  3. Performance-Monitoring
- 

## Dokumentende

---

Seite {page}