Voice Assistant - Computer Wake-Word Project
Dokument: Roadmap & Next Steps
Datum: 05. Dezember 2025

Seite: {page}

Roadmap & Next Steps

Projekt-Planung und zukünftige Entwicklung

Inhaltsverzeichnis

Projekt-Status

✅ Abgeschlossen (v3.0)

Wake-Word Detection:

- ✅ Custom "Computer" Wake-Word mit Porcupine trainiert

- ✅ Cooldown-System gegen Doppel-Erkennungen implementiert
- ✅ Sensitivity-Anpassung möglich
- ✅ Funktioniert zuverlässig in ruhiger Umgebung

Speech-to-Text:

- ✅ Vosk deutsches Modell integriert
- ✅ Voice Activity Detection (VAD) für automatische Aufnahme-Beendigung
- ✅ Offline-Funktionalität

Text-to-Speech:

- ✅ Edge TTS mit natürlicher deutscher Stimme (Katja)
- ✅ Asynchrone Ausführung
- ✅ Temporäre Dateien werden korrekt gelöscht

Befehls-Ausführung:

- ✅ Programme öffnen (Taschenrechner, Notepad, Explorer, Firefox)
- ✅ Webseiten öffnen (YouTube, ChatGPT, Google, Gmail, GitHub, Wikipedia)
- ✅ Datum & Uhrzeit (mit deutscher Übersetzung)
- ✅ Höflichkeits-Befehle (Danke, Hallo, Abbrechen)
- ✅ Hilfe-Befehl

Projekt-Setup:

- ✅ Git Repository initialisiert
- ✅ GitHub Repository erstellt (Computer-Voice-Assi)
- ✅ Dokumentation (README, WAKE_WORD_TRAINING, etc.)
- ✅ .gitignore konfiguriert
- ✅ requirements.txt erstellt

🔄 In Arbeit (v3.1)

Dokumentation:

- 🔄 Overnight Work Deliverables (10 Aufgaben)

- 🔄 Testing-Framework
- 🔄 Troubleshooting-Guide
- 🔄 LLM-Architektur-Planung

Optimierung:

- 🔄 Performance-Testing
- 🔄 Fehlerbehandlung verbessern

❌ Noch nicht begonnen

LLM-Integration:

- ❌ ChatGPT API-Integration
- ❌ Perplexity API-Integration
- ❌ Command vs. Question Classification
- ❌ Konversations-History

Erweiterte Features:

- ❌ Multi-Device-Support (Raspberry Pi, Android)
- ❌ Home Automation Integration
- ❌ Remote Access (VPN/Tailscale)
- ❌ GUI/Dashboard

Kurzfristige Ziele (1-2 Wochen)

Woche 1: Finalisierung v3.0

Tag 1-2: Testing & Bugfixing

Aufgaben:

```
[ ] Führe alle Tests aus 08_wake_word_testing.md durch
[ ] Dokumentiere Ergebnisse
[ ] Fixe gefundene Bugs
[ ] Optimiere Sensitivity-Wert basierend auf Tests
[ ] Teste in verschiedenen Umgebungen (Küche, Büro, etc.)
```

Erfolgs-Kriterium: ⩾95% Erkennungsrate, Falsch-Positiv/Stunde

---

Tag 3-4: Dokumentation vervollständigen

Aufgaben:

```
[ ] Integriere alle Overnight Work Deliverables ins Hauptprojekt
[ ] Aktualisiere README.md mit allen Änderungen
[ ] Erstelle Screenshots für Dokumentation
[ ] Erstelle Demo-Video (30s)
[ ] Füge Assets zu GitHub hinzu
```

Erfolgs-Kriterium: Vollständige, professionelle Dokumentation

---

Tag 5-7: Code-Refactoring & Cleanup

Aufgaben:

```
[ ] Code-Kommentare verbessern
[ ] Funktionen modularisieren (separate Dateien)
[ ] Type Hints hinzufügen
[ ] Logging-System implementieren
[ ] Unit-Tests schreiben (pytest)
[ ] Performance-Profiling durchführen
```

Code-Struktur (neu):

```
voice_assi/
├── src/
│   ├── __init__.py
│   ├── wake_word.py          # Porcupine Integration
│   ├── speech_to_text.py     # Vosk Integration
│   ├── text_to_speech.py     # Edge TTS Integration
│   ├── command_engine.py     # Befehlsausführung
│   └── utils.py              # Helper-Funktionen
├── tests/
│   ├── test_wake_word.py
│   ├── test_stt.py
│   └── test_commands.py
├── docs/
│   ├── 01_wake_word_comparison.md
│   ├── 02_computer_training_guide.md
│   ├── ...
│   └── 13_roadmap_next_steps.md
├── assets/
│   ├── icons/
│   ├── logos/
│   └── screenshots/
├── models/
│   └── computer.ppn
├── .env.example
├── .gitignore
├── requirements.txt
├── README.md
├── LICENSE
└── voice_assistant_computer.py  # Main Entry Point
```

Erfolgs-Kriterium: Sauberer, wartbarer Code

Woche 2: Erste LLM-Integration

Tag 8-10: ChatGPT API Setup

Aufgaben:

```
[ ] OpenAI API-Key besorgen (https://platform.openai.com)
[ ] Erstelle llm_integration.py Modul
[ ] Implementiere ChatGPTAssistant Klasse
[ ] Teste mit einfachen Fragen
[ ] Implementiere Fehlerbehandlung & Fallback
```

Code-Beispiel:

```python
# llm_integration.py
from openai import OpenAI
import os

class ChatGPTAssistant:
    def __init__(self):
        self.client = OpenAI(api_key=os.getenv('OPENAI_API_KEY'))
        self.conversation_history = []

    def query(self, user_input):
        # ... (siehe 12_llm_architecture.md)
        pass
```

Test-Fragen:

```
1. "Computer, was ist 15 mal 23?"
2. "Computer, erkläre mir, was ein LLM ist"
3. "Computer, schreibe einen kurzen Witz"
4. "Computer, übersetze 'Guten Morgen' ins Englische"
5. "Computer, was ist die Hauptstadt von Frankreich?"
```

Erfolgs-Kriterium: $^{5}/_{5}$ Fragen korrekt beantwortet

Tag 11-14: Command vs. Question Classification

Aufgaben:

```
[ ] Implementiere Pattern-based Classifier
[ ] Implementiere LLM-based Classifier (Fallback)
[ ] Implementiere Hybrid-Classifier
[ ] Teste mit 50 Beispiel-Inputs
[ ] Optimiere Patterns basierend auf Fehlern
```

Test-Cases:

```
test_cases = [
    ("öffne YouTube", "COMMAND"),
    ("wie wird das Wetter?", "QUESTION"),
    ("danke", "CONVERSATION"),
    ("kannst du mir YouTube öffnen?", "COMMAND"),
    ("was ist die Uhrzeit?", "QUESTION"),
    ("Computer, starte den Taschenrechner", "COMMAND"),
    ("erkläre mir Quantenphysik", "QUESTION"),
    # ... 43 weitere
]
```

Erfolgs-Kriterium: ⩾90% korrekte Klassifizierung

Mittelfristige Ziele (1-3 Monate)

Monat 1: LLM-Integration vervollständigen

Woche 3-4: Perplexity Integration

```
[ ] Perplexity API-Key besorgen
[ ] Implementiere PerplexityAssistant Klasse
[ ] Implementiere LLM-Router (ChatGPT vs. Perplexity)
[ ] Teste Recherche-Fragen (Wetter, News, etc.)
[ ] Implementiere Caching für häufige Fragen
```

Woche 5-6: Konversations-History

```
[ ] Implementiere Session-Management
[ ] Speichere Konversations-History (letzte 10 Nachrichten)
[ ] Implementiere Kontext-Verständnis
[ ] Teste Folgefragen ("Und übermorgen?")
[ ] Implementiere History-Reset-Befehl
```

Woche 7-8: Optimierung & Testing

```
[ ] Latenz-Optimierung (Streaming-Responses)
[ ] Kosten-Monitoring implementieren
[ ] Erweiterte Fehlerbehandlung
[ ] Offline-Fallback (lokales LLM mit Ollama)
[ ] Performance-Tests durchführen
```

Monat 2: Multi-Device-Support

Raspberry Pi Port:

```
[ ] Raspberry Pi 4 besorgen (oder vorhandenes nutzen)
[ ] Porcupine ARM-Version testen
[ ] Vosk auf ARM kompilieren/testen
[ ] Audio-Setup auf Raspberry Pi
[ ] Performance-Optimierung für ARM
[ ] Dokumentation für Raspberry Pi Setup
```

Android App (Optional):

```
[ ] Evaluiere Android-Entwicklung (Kotlin/Flutter)
[ ] Porcupine Android SDK testen
[ ] Proof-of-Concept App erstellen
[ ] Wake-Word Detection auf Android
[ ] Entscheide: Native App vs. Web App
```

Monat 3: Home Automation Integration

Smart Home Platforms:

```
[ ] Evaluiere Plattformen (Home Assistant, openHAB, etc.)
[ ] Implementiere Home Assistant Integration
[ ] Teste Licht-Steuerung ("Computer, Licht an")
[ ] Teste Thermostat-Steuerung
[ ] Teste Musik-Steuerung (Spotify, etc.)
[ ] Dokumentiere Smart Home Setup
```

Beispiel-Befehle:

```
"Computer, mach das Licht im Wohnzimmer an"
"Computer, stelle die Heizung auf 22 Grad"
"Computer, spiele Musik ab"
"Computer, zeig mir die Kamera an der Haustür"
```

Langfristige Vision (6-12 Monate)

Vision: Universeller Voice Assistant

Ziel: Ein plattformübergreifender, intelligenter Voice Assistant, der auf allen Geräten funktioniert und komplexe Aufgaben automatisieren kann.

Features

Multi-Modal Interaction:

```
[ ] Bild-Verständnis (GPT-4 Vision)
[ ] Dokument-Verarbeitung (PDF, Word, etc.)
[ ] Screen-Sharing ("Computer, was siehst du auf meinem Bildschirm?")
[ ] Gestensteuerung (optional)
```

Advanced LLM Integration:

```
[ ] Manus Integration für komplexe Aufgaben
[ ] Claude Integration (Anthropic)
[ ] Gemini Integration (Google)
[ ] Lokales LLM als Haupt-Engine (Llama 3, Mistral)
[ ] Multi-LLM-Routing (bestes LLM für jeden Task)
```

Proaktive Assistenz:

```
[ ] Kalender-Integration (Erinnerungen)
[ ] E-Mail-Monitoring ("Du hast eine wichtige E-Mail")
[ ] News-Briefing ("Guten Morgen, hier sind die Nachrichten")
[ ] Kontext-Awareness (Zeit, Ort, Aktivität)
```

Personalisierung:

```
[ ] Nutzer-Profile (mehrere Nutzer)
[ ] Stimm-Erkennung (wer spricht?)
[ ] Lern-System (Präferenzen merken)
[ ] Anpassbare Persönlichkeit
```

Sicherheit & Datenschutz:

```
[ ] Ende-zu-Ende-Verschlüsselung
[ ] Lokale Verarbeitung (kein Cloud-Zwang)
[ ] Datenschutz-Dashboard
[ ] GDPR-Compliance
```

Feature-Priorisierung

Priorität 1 (Must-Have)

Funktionalität:

1. ✅ Zuverlässige Wake-Word-Erkennung

2. ✅ Offline STT & TTS
3. 🔄 LLM-Integration (ChatGPT)
4. ❌ Command vs. Question Classification
5. ❌ Fehlerbehandlung & Fallbacks

Begründung: Basis-Funktionalität muss robust sein, bevor erweiterte Features hinzugefügt werden.

Priorität 2 (Should-Have)

Erweiterungen:

1. ❌ Perplexity-Integration (Recherche)
2. ❌ Konversations-History
3. ❌ Raspberry Pi Support
4. ❌ Home Automation (Basis)
5. ❌ Performance-Optimierung

Begründung: Diese Features erhöhen den Nutzen signifikant und sind technisch machbar.

Priorität 3 (Nice-to-Have)

Extras:

1. ❌ Android App
2. ❌ GUI/Dashboard
3. ❌ Multi-Modal (Vision)
4. ❌ Proaktive Assistenz
5. ❌ Manus-Integration

Begründung: Diese Features sind cool, aber nicht essentiell. Können später hinzugefügt werden.

Technische Schulden

Bekannte Probleme

Problem 1: Monolithischer Code

- **Aktuell:** Alles in einer Datei (`voice_assistant_computer.py`)
- **Ziel:** Modularisierung in separate Dateien
- **Priorität:** Hoch
- **Aufwand:** 1-2 Tage

Problem 2: Keine Unit-Tests

- **Aktuell:** Nur manuelle Tests
- **Ziel:** Automatisierte Tests mit pytest
- **Priorität:** Mittel
- **Aufwand:** 2-3 Tage

Problem 3: Hardcoded Konfiguration

- **Aktuell:** Konfiguration im Code
- **Ziel:** Separate `config.yaml` Datei
- **Priorität:** Mittel
- **Aufwand:** 1 Tag

Problem 4: Keine Logging

- **Aktuell:** Nur `print()` Statements
- **Ziel:** Strukturiertes Logging mit `logging` Modul
- **Priorität:** Mittel
- **Aufwand:** 1 Tag

Problem 5: Keine CI/CD

- **Aktuell:** Manuelles Deployment
- **Ziel:** GitHub Actions für Tests & Releases

- **Priorität:** Niedrig
- **Aufwand:** 2-3 Tage

Community & Open Source

GitHub-Strategie

Ziel: Projekt als Open-Source-Community-Projekt etablieren

Maßnahmen:

```
[ ] Erstelle CONTRIBUTING.md
[ ] Erstelle CODE_OF_CONDUCT.md
[ ] Erstelle Issue-Templates
[ ] Erstelle Pull-Request-Templates
[ ] Erstelle GitHub Discussions
[ ] Erstelle GitHub Projects (Roadmap)
[ ] Erstelle Wiki
```

Community-Engagement

Marketing:

```
[ ] Reddit-Post in r/Python, r/homeautomation
[ ] Hacker News Submission
[ ] Dev.to Artikel schreiben
[ ] YouTube-Tutorial erstellen
[ ] Twitter/X Thread
```

Dokumentation:

```
[ ] Beginner-friendly Tutorial
[ ] Video-Tutorials (YouTube)
[ ] FAQ erstellen
[ ] Troubleshooting erweitern
[ ] Use-Cases dokumentieren
```

Support:

```
[ ] Discord-Server erstellen (optional)
[ ] GitHub Discussions aktivieren
[ ] Schnelle Issue-Responses (<24h)
[ ] Community-Beiträge würdigen
```

Milestones

v3.0 - "Computer Wake-Word" ✅

- Custom Wake-Word Training
- Basis-Funktionalität
- Dokumentation

Release: Dezember 2025

---

v4.0 - "LLM Integration" 🔄

- ChatGPT API
- Perplexity API
- Command vs. Question Classification
- Konversations-History

Geplant: Januar 2026

---

v5.0 - “Multi-Device”

- Raspberry Pi Support
- Android App (optional)
- Synchronisation zwischen Geräten

Geplant: März 2026

---

v6.0 - “Smart Home”

- Home Assistant Integration
- Licht-/Thermostat-Steuerung
- Musik-Steuerung
- Kamera-Integration

Geplant: Mai 2026

---

v7.0 - “Advanced AI”

- Multi-Modal (Vision)
- Manus Integration
- Proaktive Assistenz
- Personalisierung

Geplant: September 2026

Metriken & Erfolgs-Kriterien

Technische Metriken

Performance:

- Wake-Word Latenz: <50ms

- STT Latenz: <500ms
- LLM Latenz: <2s
- TTS Latenz: <500ms
- **Gesamt-Latenz: <3s** (Wake-Word bis Antwort)

Zuverlässigkeit:

- Wake-Word Erkennungsrate: ⩾95%
- Falsch-Positive Rate: /Stunde
- Uptime: ⩾99% (bei $^{24}/_{7}$ Betrieb)
- Error Rate: %

Ressourcen:

- CPU-Auslastung: <20% (Idle), <50% (Aktiv)
- Memory: <200MB (Idle), <500MB (Aktiv)
- Disk Space: <1GB (mit Modellen)

Community-Metriken

GitHub:

- Stars: 100+ (6 Monate)
- Forks: 20+ (6 Monate)
- Contributors: 5+ (12 Monate)
- Issues: <10 offene (kontinuierlich)

Nutzung:

- Downloads: 500+ (6 Monate)
- Aktive Nutzer: 50+ (12 Monate)
- Positive Feedback: ⩾80%

Nächste konkrete Schritte

Diese Woche (6.-12. Dezember 2025)

Montag:

```
[ ] Alle Overnight Work Deliverables zu GitHub pushen
[ ] README.md aktualisieren
[ ] Testing-Framework durchführen
```

Dienstag:

```
[ ] Bugs fixen (falls gefunden)
[ ] Screenshots erstellen
[ ] Demo-Video aufnehmen
```

Mittwoch:

```
[ ] Code refactoren (Modularisierung)
[ ] Type Hints hinzufügen
[ ] Logging implementieren
```

Donnerstag:

```
[ ] OpenAI API-Key besorgen
[ ] llm_integration.py erstellen
[ ] Erste ChatGPT-Tests
```

Freitag:

```
[ ] Classifier implementieren
[ ] Integration testen
[ ] Dokumentation aktualisieren
```

Wochenende:

```
[ ] Entspannen! 🎉
[ ] Projekt-Review
[ ] Nächste Woche planen
```

Zusammenfassung

Aktueller Stand: v3.0 - Computer Wake-Word funktioniert ✅

Nächster Meilenstein: v4.0 - LLM-Integration (Januar 2026)

Langfristige Vision: Universeller, intelligenter Voice Assistant auf allen Geräten

Erfolgs-Wahrscheinlichkeit: Hoch (solide Basis, klarer Plan, gute Dokumentation)

Let’ s build the future of voice assistants! 🚀

Dokumentende
