🤖 COMPUTER ASSI - ULTIMATE OVERNIGHT HANDOVER

🎯 MISSION: "Computer" Wake-Word Training + Projekt-Vorbereitung

Zeitrahmen: Über Nacht (6-8 Stunden autonome Arbeit)
Ziel: Morgen früh komplett vorbereitet für "Computer" Wake-Word Implementation

---

📋 VOLLSTÄNDIGER PROJEKT-KONTEXT

Projekt-Info

- **Name:** Voice Assistant - KI Sprachsteuerung
- **GitHub:** https://github.com/KoMMb0t/voice_assi
- **Nutzer:** ModBot / KoMMb0t (kommuniverse@gmail.com)
- **System:** Windows 11 Mini PC, Python 3.11
- **Projektordner:** C:\Users\ModBot\ki-sprachsteuerung
- **Virtual Environment:** .venv

Aktueller Status ✅

- Funktionsfähiger Voice Assistant mit Wake-Word ("hey jarvis")
- Speech-to-Text: Vosk (deutsches Modell)
- Text-to-Speech: Edge TTS (Stimme: de-DE-KatjaNeural)

- Voice Activity Detection (VAD) für automatische Aufnahme-Beendigung
- Befehle: Programme öffnen, Webseiten, Datum/Uhrzeit, Höflichkeits-Befehle
- Alles auf GitHub hochgeladen mit README.md
- Desktop-Icon & Batch-Datei zum Starten

Aktuelles Problem ❌

- **Wake-Word-Doppelerkennung:** "hey jarvis" wird manchmal doppelt erkannt
- Score ist konstant sehr hoch (0.95-1.00)
- Verschiedene Fixes wurden versucht (Threshold, Cooldown, Buffer-Clearing)
- Problem besteht weiterhin → Lösung: Neues "Computer" Wake-Word trainieren

Roadmap 🗺️

- [🔄] Train a custom "Computer" wake word model (IN ARBEIT - DEINE AUFGABE!)
  - [ ] LLM integration (ChatGPT, Perplexity, Monica, Manus)
  - [ ] Expand to other devices (Jetson Nano, Raspberry Pi, Android)
  - [ ] Integrate with home automation systems
  - [ ] Enable secure remote access (VPN/Tailscale)

🌙 DEINE OVERNIGHT MISSION

HAUPTZIEL: "Computer" Wake-Word komplett vorbereiten

Arbeite **AUTONOM** und erstelle **FERTIGE, AUSFÜHRBARE** Dokumente, Code und Anleitungen.

🔟 10 PARALLELE MANUS-AUFGABEN

📊 AUFGABE 1: Wake-Word-Methoden-Vergleich (45 Min)

Ziel: Entscheidungsgrundlage für beste Trainings-Methode

Recherchiere und vergleiche:

1. **OpenWakeWord** (aktuell genutzt)
    - Training-Prozess
    - Anforderungen (Aufnahmen, Hardware)
    - Erfolgsquote
    - Community-Support
    - Kosten: Kostenlos
2. **Porcupine (Picovoice)**
    - Custom Wake-Word Service
    - Einfachheit
    - Kosten & Limits
    - Windows-Kompatibilität
3. **Snowboy** (veraltet?)
    - Noch nutzbar?
    - Alternativen?
4. **Andere Methoden**
    - Mycroft Precise
    - Tensorflow/PyTorch Custom Models

- Cloud-Services

Erstelle Datei: `01_wake_word_comparison.md`

Inhalt:

```
# Wake-Word Training Methoden Vergleich

## 1. OpenWakeWord
### Vorteile
- [Liste]
### Nachteile
- [Liste]
### Kosten
- [Details]
### Training-Aufwand
- [Details]
### Empfehlung für Windows 11
- [Ja/Nein + Begründung]

## 2. Porcupine
[Gleiche Struktur]

## 3. Snowboy
[Gleiche Struktur]

## 4. Andere
[Gleiche Struktur]

## FINALE EMPFEHLUNG
**Beste Methode:** [Name]
**Begründung:** [3-5 Sätze]
**Nächste Schritte:** [Bullet Points]
```

📖 AUFGABE 2: Detaillierte Trainings-Anleitung (90 Min)

Ziel: Schritt-für-Schritt-Anleitung, die ich morgen 1:1 umsetzen kann

Erstelle Datei: `02_computer_training_guide.md`

Inhalt (SEHR DETAILLIERT):

Teil 1: Voraussetzungen

- Software-Installation (mit exakten Befehlen)
- Hardware-Anforderungen
- Mikrofon-Setup & Test

Teil 2: Aufnahme-Prozess

- Wie viele Aufnahmen? (Positive + Negative Samples)
- Wie lange pro Aufnahme?
- Welche Variationen? (Tonlage, Lautstärke, Geschwindigkeit)
- Umgebungs-Bedingungen (leise, laut, Hintergrundgeräusche)
- Aufnahme-Format & Qualität

Teil 3: Daten-Vorbereitung

- Ordner-Struktur
- Datei-Benennung
- Daten-Augmentation (falls nötig)

Teil 4: Training

- Exakte Befehle zum Trainieren
- Parameter-Erklärungen
- Erwartete Dauer
- Wie erkenne ich, ob Training erfolgreich ist?

Teil 5: Model-Export

- Wo finde ich das fertige Modell?
- Welches Format?
- Wie teste ich es?

Teil 6: Integration in Code

- Welche Zeilen ändern?
- Wo speichere ich das Modell?
- Wie lade ich es?

Teil 7: Testing & Optimierung

- Test-Szenarien
- Performance-Metriken
- Troubleshooting häufiger Probleme

Format: Markdown mit Code-Blöcken, Screenshots-Platzhaltern, Checklisten

---

💻 AUFGABE 3: Aufnahme-Skript erstellen (60 Min)

Ziel: Python-Skript für einfache, automatisierte Aufnahmen

Erstelle Datei: `03_record_wake_word.py`

Features:

- Automatische Aufnahme von 200 "Computer" -Samples
- Countdown zwischen Aufnahmen (3-2-1-JETZT!)
- Verschiedene Modi:
  - Normal (100x)
  - Laut (30x)
  - Leise (30x)
  - Schnell (20x)
  - Langsam (20x)
- Progress-Anzeige (z.B. "$^{42}/_{200}$ aufgenommen")
- Automatisches Speichern mit Nummerierung (computer_001.wav, etc.)
- Pause-Funktion (bei Bedarf unterbrechen)

- Qualitäts-Check (zu leise? zu kurz?)
- Zusammenfassung am Ende

Code-Struktur:

```
import sounddevice as sd
import numpy as np
import wave
import time
import os

# Konfiguration
SAMPLE_RATE = 16000
DURATION = 2.0  # Sekunden pro Aufnahme
OUTPUT_DIR = "wake_word_recordings"

def record_sample(filename, countdown=True):
    """Nimmt ein Sample auf."""
    # [Implementierung]

def main():
    """Hauptfunktion."""
    # [Implementierung mit Menü]

if __name__ == "__main__":
    main()
```

Bonus: Auch Negative Samples aufnehmen (andere Wörter, Hintergrundgeräusche)

🔧 AUFGABE 4: Integrations-Code vorbereiten (60 Min)

Ziel: Fertiger Code, der morgen nur noch getestet werden muss

Erstelle Datei: `04_voice_assistant_computer.py`

Basis: Kopiere `voice_assistant_edge_ultimate.py` und passe an:

Änderungen:

1. **Zeile 16:** `WAKE_WORD = "computer"` (statt “hey jarvis”)

2. **Zeile 176:** `if prediction["computer"] > 0.5:` (statt "hey_jarvis")
3. **Zeile 239:** `oww_model = Model(wakeword_models=["computer"])`
4. **Zeile 1:** Kommentar hinzufügen:

```
# Voice Assistant v3.0 - "Computer" Wake-Word Edition
# Trainiertes Custom Wake-Word: "Computer"
# Modell-Pfad: ./models/computer.onnx (oder .tflite)
```

Zusätzlich:

- Füge Funktion hinzu: `load_custom_model(model_path)`
- Füge Konfiguration hinzu: `CUSTOM_MODEL_PATH = "./models/computer.onnx"`
- Kommentiere alle Änderungen mit `# CHANGED FOR COMPUTER WAKE-WORD`

📚 AUFGABE 5: GitHub-Dokumentation (45 Min)

Ziel: Professionelle Dokumentation für GitHub

Erstelle 3 Dateien:

5.1: `05_WAKE_WORD_TRAINING.md`

```
# Custom Wake-Word Training Guide

## Übersicht
Dieses Projekt nutzt ein custom-trainiertes "Computer" Wake-Word.

## Warum "Computer"?
- Star Trek Inspiration
- Kurz und prägnant
- Eindeutig erkennbar

## Training-Prozess
[Zusammenfassung aus Aufgabe 2]

## Verwendung
[Wie nutzt man das trainierte Modell]

## Eigenes Wake-Word trainieren
[Link zu detaillierter Anleitung]
```

5.2: `06_README_UPDATE.md`

````
# README.md Updates

## Roadmap Update
- [x] Train a custom "Computer" wake word model ✅
- [ ] LLM integration (ChatGPT, Perplexity, etc.)

## Features Update
* **Wake-Word Detection:** Custom-trained "Computer" wake word using OpenWakeWord

## Installation Update
5. (Optional) Train your own wake word:
   ```sh
   python record_wake_word.py
   python train_wake_word.py
````
````

```
### 5.3: `07_GITIGNORE_UPDATE.txt`
```

Wake-Word Training Data

wake_word_recordings/ *.wav *.onnx *.tflite models/temp/

🧪 AUFGABE 6: Testing-Framework (45 Min)

Ziel: Systematisches Testen des Wake-Words

Erstelle Datei: `08_wake_word_testing.md`

Inhalt:

Test-Kategorien

1. Erkennungsrate-Tests
- [ ] 100x "Computer" sagen → Erkennungsrate: ___%
- [ ] Verschiedene Tonlagen → Erkennungsrate: ___%
- [ ] Verschiedene Geschwindigkeiten → Erkennungsrate: ___%

2. Falsch-Positiv-Tests
- [ ] 100x andere Wörter sagen → Falsch-Positive: ___
- [ ] Hintergrundgeräusche → Falsch-Positive: ___
- [ ] Ähnliche Wörter ("Komputer", "Commuter") → Falsch-Positive: ___

3. Umgebungs-Tests
- [ ] Leise Umgebung → Funktioniert: Ja/Nein
- [ ] Laute Umgebung → Funktioniert: Ja/Nein
- [ ] Mit Musik → Funktioniert: Ja/Nein
- [ ] Mit TV → Funktioniert: Ja/Nein

4. Stress-Tests
- [ ] 1000x hintereinander → Stabil: Ja/Nein
- [ ] Über 1 Stunde → Stabil: Ja/Nein
- [ ] Mit anderen Personen → Funktioniert: Ja/Nein

Erfolgs-Kriterien
- ✅ Erkennungsrate > 95%
- ✅ Falsch-Positive < 1%
- ✅ Funktioniert in allen Umgebungen
- ✅ Keine Doppel-Erkennungen

Erstelle auch: `09_test_wake_word.py` (Automatisiertes Test-Skript)

🔧 AUFGABE 7: Troubleshooting-Guide (30 Min)

Erstelle Datei: `10_troubleshooting.md`

Inhalt:

Problem 1: Wake-Word wird nicht erkannt
Symptome: [...]
Mögliche Ursachen: [...]
Lösungen: [...]

Problem 2: Zu viele Falsch-Positive
Symptome: [...]
Lösungen: [...]

Problem 3: Doppel-Erkennungen
Symptome: [...]
Lösungen: [...]

Problem 4: Performance-Probleme
Symptome: [...]
Lösungen: [...]

Problem 5: Training schlägt fehl
Symptome: [...]
Lösungen: [...]

🎨 AUFGABE 8: Asset-Sammlung (30 Min)

Ziel: Visuelle Assets für Projekt

Erstelle Datei: `11_assets_list.md`

Recherchiere und liste auf:

Icons
- Mikrofon-Icon für Desktop (3 Optionen mit Links)
- Voice Assistant Logo (3 Optionen)
- "Computer" Wake-Word Visualisierung

Diagramme
- Architektur-Diagramm (Vorlage)
- Workflow-Diagramm (Wake-Word → STT → TTS → Command)

Screenshots

```
- Beispiel-Ausgaben
- Terminal-Logs
- GitHub-Preview

**Format:** Markdown mit Links, Lizenz-Info, Download-Anweisungen

---

## 📊 AUFGABE 9: LLM-Integration Architektur (60 Min)

**Ziel:** Vorbereitung für nächste Phase

**Erstelle Datei:** `12_llm_architecture.md`

**Inhalt:**

### Übersicht
- Warum LLM-Integration?
- Welche LLMs?
- Wie unterscheidet man "Befehl" vs. "Frage"?

### Architektur-Diagramm
```

User → Wake-Word → STT → Command Parser

```
                ↓
          [Ist es ein Befehl?]
          ↙              ↘
    Ja: Execute      Nein: LLM
          ↓               ↓
      TTS ← Response ← API
```

````
### API-Integration
#### ChatGPT
- API-Key Setup
- Anfrage-Format
- Response-Handling
- Kosten-Schätzung

#### Perplexity
- [Gleiche Struktur]

#### Manus
- [Gleiche Struktur]

### Code-Struktur
```python
def process_input(text):
    if is_command(text):
        return execute_command(text)
    else:
        return query_llm(text)

def is_command(text):
    # Logik zur Unterscheidung
    pass

def query_llm(text, llm="chatgpt"):
    # API-Aufruf
    pass
````

Fallback-Strategie

- ChatGPT nicht erreichbar → Perplexity
- Alle LLMs down → Offline-Antwort

🚀 AUFGABE 10: Projekt-Roadmap & Next Steps (30 Min)

Erstelle Datei: `13_next_steps.md`
````

Inhalt:

Sofort (Morgen)

- [ ] Aufnahmen machen (1h)
- [ ] Modell trainieren (2h)
- [ ] Code integrieren (30min)
- [ ] Testen (30min)
- [ ] Auf GitHub pushen (10min)

Diese Woche

- [ ] LLM-Integration (ChatGPT)
- [ ] Mehr Befehle hinzufügen
- [ ] Performance-Optimierung

Nächste 2 Wochen

- [ ] Perplexity-Integration
- [ ] Manus-Integration
- [ ] Home Automation (erste Schritte)

Langfristig (1-3 Monate)

- [ ] Raspberry Pi Port
- [ ] Android App
- [ ] VPN/Remote Access

🤝 PROMPTS FÜR ANDERE KIs

💬 ChatGPT Prompt

````
Hallo ChatGPT! Ich arbeite an einem Voice Assistant und brauche Code-Optimierung.

**Projekt:** https://github.com/KoMMb0t/voice_assi

**AUFGABE 1: Code-Review**
Analysiere meinen Code auf:
- Performance-Bottlenecks
- Memory Leaks
- Best Practices
- Error Handling

**AUFGABE 2: Refactoring**
Erstelle eine refactored Version mit:
- Klassen statt Funktionen (OOP)
- Type Hints
- Docstrings
- Unit Tests

**AUFGABE 3: LLM-Integration Code**
Schreibe fertigen Code für:
- ChatGPT API Integration
- Command vs. Question Detection
- Response Caching
- Error Handling

**AUFGABE 4: Command Pattern**
Implementiere Command Pattern für einfacheres Hinzufügen neuer Befehle:
```python
class Command:
    def execute(self): pass

class OpenCalculatorCommand(Command):
    def execute(self):
        subprocess.Popen("calc.exe")
````

Gib mir fertigen, ausführbaren Code mit Kommentaren!
````

```
---

## 🔍 Perplexity Prompt
```

Recherchiere bitte folgende Themen für mein Voice Assistant Projekt:

THEMA 1: Wake-Word Training State-of-the-Art 2025

- Welche Methoden sind aktuell am besten?
- Neue Frameworks seit 2024?
- Best Practices für kleine Datensätze?
- Wie viele Aufnahmen sind wirklich nötig?

THEMA 2: Voice Assistant Architektur-Patterns

- Wie strukturieren professionelle Projekte ihren Code?
- Design Patterns für Voice Assistants?
- Microservices vs. Monolith?

THEMA 3: Audio-Processing Optimierung

- Wie verhindert man Doppel-Erkennungen?
- Buffer-Management Best Practices?
- Latenz-Optimierung?

THEMA 4: LLM-Integration Patterns

- Wie integrieren andere Projekte LLMs?
- Kosten-Optimierung?
- Caching-Strategien?

THEMA 5: Erfolgreiche Open-Source Voice Assistants

- Top 10 GitHub-Projekte (2024-2025)
- Was macht sie erfolgreich?

- Welche Features sind am beliebtesten?

THEMA 6: Cross-Platform Voice Assistants

- Windows → Linux → Android Portierung
- Welche Frameworks sind plattformübergreifend?
- Herausforderungen & Lösungen?

Gib mir eine strukturierte Zusammenfassung mit Quellen!

```
---

## 🎨 Monica Prompt
```

Hilf mir bei der Asset-Sammlung für mein Voice Assistant Projekt!

AUFGABE 1: Icon-Suche Finde 10 kostenlose Icons für:

- Mikrofon (für Desktop)
- Voice Assistant (für GitHub)
- Wake-Word Visualisierung
- “Computer” Symbol

Quellen: icons8.com, flaticon.com, fontawesome

AUFGABE 2: Dokumentations-Inspiration Finde 5 GitHub-Repos mit exzellenter Dokumentation:

- README.md Beispiele
- Wiki-Strukturen
- Diagramme (Mermaid, etc.)
- Screenshots & GIFs

AUFGABE 3: Tutorial-Sammlung Finde YouTube-Tutorials zu:

- Wake-Word Training (Top 5)

- OpenWakeWord Usage (Top 3)
- Voice Assistant Development (Top 5)
- Python Audio Processing (Top 3)

AUFGABE 4: Competitor-Analyse Finde ähnliche Open-Source Projekte:

- Features-Vergleich
- Stars/Forks
- Aktive Entwicklung?
- Was können wir lernen?

Gib mir Links, Screenshots und Zusammenfassungen!

```
---

## 🤖 Claude/Copilot Prompt
```

Bitte führe ein umfassendes Code-Review für meinen Voice Assistant durch.

Projekt: https://github.com/KoMMb0t/voice_assi

REVIEW-KATEGORIEN:

1. Code-Qualität

- PEP 8 Compliance
- Naming Conventions
- Code Duplication
- Magic Numbers

2. Performance

- Bottlenecks identifizieren
- Memory Usage

- CPU Usage
- Optimierungs-Vorschläge

3. Architektur

- Separation of Concerns
- Modularity
- Scalability
- Testability

4. Sicherheit

- API-Key Handling
- Input Validation
- Error Handling
- Logging (keine sensiblen Daten)

5. Dokumentation

- Fehlende Docstrings
- Unklare Kommentare
- README-Verbesserungen

REFACTORING-AUFGABEN:

1. Erstelle eine OOP-Version mit Klassen:
    - `VoiceAssistant` (Main Class)
    - `WakeWordDetector`
    - `SpeechRecognizer`
    - `TextToSpeech`
    - `CommandExecutor`
2. Füge Type Hints überall hinzu

3. Schreibe Unit Tests für:
    - `execute_command()`
    - `is_command()`
    - `speak()`
4. Erstelle ein Config-File (YAML/JSON) statt hardcoded values
5. Implementiere Logging (statt print)

Gib mir den kompletten refactored Code!

📊 ZUSAMMENFASSUNG & DELIVERABLES

Am Ende der Nacht solltest du erstellt haben:

Dokumente (13 Dateien)
1. ✅ `01_wake_word_comparison.md` - Methoden-Vergleich
2. ✅ `02_computer_training_guide.md` - Detaillierte Anleitung
3. ✅ `03_record_wake_word.py` - Aufnahme-Skript
4. ✅ `04_voice_assistant_computer.py` - Integrations-Code
5. ✅ `05_WAKE_WORD_TRAINING.md` - GitHub-Doku
6. ✅ `06_README_UPDATE.md` - README-Updates
7. ✅ `07_GITIGNORE_UPDATE.txt` - .gitignore-Ergänzungen
8. ✅ `08_wake_word_testing.md` - Test-Framework
9. ✅ `09_test_wake_word.py` - Test-Skript
10. ✅ `10_troubleshooting.md` - Troubleshooting-Guide
11. ✅ `11_assets_list.md` - Asset-Sammlung
12. ✅ `12_llm_architecture.md` - LLM-Architektur
13. ✅ `13_next_steps.md` - Roadmap

Von anderen KIs
- ✅ ChatGPT: Refactored Code + LLM-Integration
- ✅ Perplexity: Recherche-Report (6 Themen)
- ✅ Monica: Asset-Links + Tutorial-Liste
- ✅ Claude: Code-Review + OOP-Version

🌅 MORGEN FRÜH: ACTION PLAN

Schritt 1: Review (30 Min)
- Alle 13 Dokumente durchlesen
- ChatGPT/Perplexity/Monica/Claude Ergebnisse checken
- Entscheidungen treffen (welche Methode, welche Tools)

Schritt 2: Setup (15 Min)
- Software installieren (falls nötig)
- Mikrofon testen
- Ordner-Struktur erstellen

Schritt 3: Aufnahmen (60 Min)
- `record_wake_word.py` ausführen
- 200x "Computer" sagen

```
- Negative Samples aufnehmen

## Schritt 4: Training (2 Std)
- Training-Skript ausführen
- Kaffee trinken ☕
- Modell validieren

## Schritt 5: Integration (30 Min)
- `voice_assistant_computer.py` anpassen
- Modell einbinden
- Erste Tests

## Schritt 6: Testing (30 Min)
- Test-Checklist abarbeiten
- Performance messen
- Optimieren

## Schritt 7: GitHub (15 Min)
```bash
git add .
git commit -m "feat: Add custom 'Computer' wake word"
git push
```

Schritt 8: Dokumentation (15 Min)

- README.md updaten
- WAKE_WORD_TRAINING.md hochladen
- Screenshots machen

FERTIG! 🎉

💡 WICHTIGE HINWEISE

Für Manus (Computer Assi Chat):

- Arbeite **AUTONOM** - erstelle fertige Dateien, nicht nur Outlines
```

- Nutze **echten Code** - nicht nur Pseudocode
- Sei **DETAILLIERT** - ich bin Anfänger, erkläre alles
- **TESTE** deine Vorschläge mental - funktioniert das wirklich?
- Erstelle **AUSFÜHRBARE** Anleitungen - Schritt für Schritt

Qualitäts-Kriterien:

- ✅ Jede Datei ist **vollständig** (nicht "TODO" oder "...")
- ✅ Jeder Code ist **ausführbar** (keine Syntax-Fehler)
- ✅ Jede Anleitung ist **nachvollziehbar** (für Anfänger)
- ✅ Alle Links sind **aktuell** (2024-2025)
- ✅ Alle Empfehlungen sind **begründet**

🚀 LOS GEHT'S!

Manus, du hast jetzt alle Informationen!

Deine Mission:

1. Lies dieses Dokument komplett
2. Verstehe den Kontext
3. Arbeite die 10 Aufgaben ab
4. Erstelle 13+ fertige Dateien
5. Gib mir am Ende eine Zusammenfassung

Ich gehe jetzt schlafen und freue mich morgen auf deine Ergebnisse! 😴

Viel Erfolg! 💪🔥

P.S.: Wenn du Fragen hast oder etwas unklar ist, dokumentiere das in einer `QUESTIONS.md` Datei, die ich morgen beantworten kann.

P.P.S.: Sei kreativ! Wenn du bessere Ideen hast als ich beschrieben habe, implementiere sie!

P.P.P.S.: Hab Spaß dabei! 🎉