Feedback-Copilot: Cheatsheet für VW-Präsentation

🎯 Elevator Pitch (30 Sekunden)

> "Der Feedback-Copilot analysiert In-Car-Kundenfeedbacks mit KI. Er kombiniert Keyword- und semantische Suche, anonymisiert automatisch personenbezogene Daten und beantwortet Fragen mit Quellenangabe. Basiert auf 83 wissenschaftlichen Papers."

📚 Fachbegriffe erklärt

RAG (Retrieval-Augmented Generation)

Was es ist: Eine KI-Architektur die NICHT aus dem Gedächtnis antwortet, sondern zuerst relevante Dokumente sucht und dann basierend auf diesen antwortet.

Warum wichtig:

- Keine Halluzinationen (erfindet nichts)
- Jede Aussage ist nachvollziehbar
- Aktuelle Daten (nicht auf Trainingsdaten beschränkt)

Unsere Implementierung:

```
Frage → Dokumente suchen → Kontext aufbauen → GPT antworten lassen → Quellen zitieren
```

Hybrid Retrieval (BM25 + Vector)

Was es ist: Kombination von zwei Such-Methoden:

Methode	Stärke	Schwäche
BM25 (Keyword)	Findet exakte Wörter	Versteht keine Synonyme
Vector (Semantic)	Versteht Bedeutung	Verliert exakte Keywords
Hybrid (beide)	Best of both worlds	-

Beispiel:

- Suche: "Sprachassistent funktioniert nicht"
- BM25 findet: "Sprachassistent reagiert nicht"
- Vector findet: "Hey Volkswagen Befehl wird ignoriert"
- Hybrid: Beide Ergebnisse kombiniert

Code-Ausschnitt (`vectorstore.py`):

```
# Vector Search (Semantic)
vector_results = self.collection.query(
    query_texts=[query],
    n_results=top_k * 2
)

# BM25 Search (Keyword)
tokenized_query = query.lower().split()
bm25_scores = self._bm25_index.get_scores(tokenized_query)

# RRF (Reciprocal Rank Fusion) - Kombination
for doc_id in all_candidates:
    rrf_score = 0
    if doc_id in vector_rankings:
        rrf_score += 1 / (60 + vector_rankings[doc_id]["rank"])
    if doc_id in bm25_rankings:
        rrf_score += 1 / (60 + bm25_rankings[doc_id]["rank"])
```

RRF (Reciprocal Rank Fusion)

Was es ist: Mathematische Formel um Rankings zu kombinieren.

Formel:

```
score = 1/(k + rank_bm25) + 1/(k + rank_vector)
```

wobei k=60 (Standard)

Beispiel:

- Dokument A: Rang 1 bei BM25, Rang 5 bei Vector
- Score = 1/(60+1) + 1/(60+5) = 0.0164 + 0.0154 = 0.0318
- → Hoher Score = gutes Ergebnis

PII (Personally Identifiable Information)

Was es ist: Personenbezogene Daten die anonymisiert werden müssen.

Unsere erkannten PII-Typen:

Typ	Regex-Pattern	Ersetzt durch
E-Mail	`[A-Za-z0-9._%+-]+@...`	`[EMAIL]`
Telefon	`(+49|0049|0)...`	`[TELEFON]`
Kennzeichen	`B-AB 1234`	`[KENNZEICHEN]`
VIN	17-stellige Nummer	`[VIN]`
Datum	`01.01.2026`	`[DATUM]`

Code-Ausschnitt (`pii.py`):

```
self.patterns = {
    "email": r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',
    "phone_de": r'\b(?:\+49|0049|0)[\s\-]?(?:\d{2,4})[\s\-]?\d{3,8}\b',
    "plate_de": r'\b[A-ZÄÖÜ]{1,3}[\s\-]?[A-Z]{1,2}[\s\-]?\d{1,4}[EH]?\b',
    "vin": r'\b[A-HJ-NPR-Z0-9]{17}\b',
}
```

Guardrails

Was es ist: Sicherheitsmechanismen die Halluzinationen verhindern.

Unsere Guardrails:

1. **Zitationspflicht:** Jede Aussage muss Quelle angeben
2. **Unanswerable-Guard:** "Keine Information verfügbar" statt zu raten
3. **Confliction Detection:** Widersprüchliche Quellen markieren

Code-Ausschnitt (`rag.py`):

```
# Check: Gibt es relevante Quellen?
if not sources:
    return {
        "answer": "Zu dieser Frage liegen keine Informationen vor.",
        "answerable": False
    }
```

ChromaDB (VectorStore)

Was es ist: Datenbank die Texte als Vektoren (Zahlenreihen) speichert.

Warum:

- Semantische Ähnlichkeitssuche
- Lokal (keine Cloud nötig)

- Persistent (Daten bleiben bei Restart)

Code-Ausschnitt:

```
# Persistente Initialisierung
self.client = chromadb.PersistentClient(path="./chroma_db")
self.collection = self.client.get_or_create_collection(name="feedback")
```

Embeddings

Was es ist: Texte werden in Zahlen umgewandelt (Vektoren), um Ähnlichkeiten zu berechnen.

Modell: `all-MiniLM-L6-v2` (Standard in ChromaDB)

Beispiel:

```
"Auto fährt nicht" → [0.12, -0.34, 0.78, ...]  (384 Zahlen)
"Fahrzeug startet nicht" → [0.11, -0.32, 0.76, ...]  (ähnliche Zahlen!)
```

📅 Ausblick: Was kommt in 3 Wochen?

1. Evaluation

Was: Metriken messen um Qualität zu beweisen

Plan:

```
# Testdatensatz erstellen
test_questions = [
    {"question": "Welche Probleme gibt es mit dem Sprachassistenten?",
     "expected_ids": ["FB-2026-001", "FB-2026-015"]}
]

# Metriken berechnen
recall_at_5 = count(relevante in top_5) / count(alle_relevanten)
ndcg = normalized_discounted_cumulative_gain(rankings)
latency = time(response) - time(request)
```

Ziele:

Metrik	Beschreibung	Ziel
Recall@5	Sind relevante Docs in Top-5?	> 0.8
nDCG	Ranking-Qualität	> 0.7
Citation Coverage	% Aussagen mit Quelle	> 90%
Latenz	Zeit bis Antwort	< 3s

📖 Fachbegriffe Evaluation erklärt:

Recall@K (Trefferquote bei K Ergebnissen):

- Misst: "Wie viele relevante Dokumente finde ich in den Top-K Ergebnissen?"
- Formel: `Recall@K = Anzahl relevanter Docs in Top-K / Gesamtanzahl relevanter Docs`
- Beispiel: 4 relevante Docs existieren, 3 davon in Top-5 → Recall@5 = 3/4 = 0.75
- Warum wichtig: Zeigt ob das System alle wichtigen Infos findet

nDCG (Normalized Discounted Cumulative Gain):

- Misst: "Sind die besten Ergebnisse auch an den besten Positionen?"
- Formel: berücksichtigt Ranking-Position (frühere Treffer zählen mehr)
- Beispiel: Relevantes Doc auf Platz 1 → hoher Score, auf Platz 10 → niedriger Score
- Warum wichtig: Nicht nur OB gefunden, sondern WO im Ranking

- Wertebereich: 0.0 (schlecht) bis 1.0 (perfekt)

Citation Coverage (Zitationsabdeckung):

- Misst: "Wie viele Aussagen in der Antwort sind mit Quelle belegt?"
- Formel: `Citations / Aussagen * 100%`
- Warum wichtig: Verhindert Halluzinationen, macht nachvollziehbar

Latenz (Response Time):

- Misst: Zeit zwischen Frage und vollständiger Antwort
- Komponenten: Retrieval + LLM-Generierung + Netzwerk
- Ziel <3s: Nutzer erwartet schnelle Antworten

---

2. NER für Personennamen

Was: spaCy-Integration für Namen-Erkennung

Plan:

```
import spacy
nlp = spacy.load("de_core_news_lg")

def detect_names(text):
    doc = nlp(text)
    return [ent.text for ent in doc.ents if ent.label_ == "PER"]
```

Warum: Regex erkennt keine Namen wie "Max Mustermann"

📖 Fachbegriffe NER erklärt:

NER (Named Entity Recognition):

- Was: KI-basierte Erkennung von benannten Entitäten in Texten
- Entitäten: Personen (PER), Orte (LOC), Organisationen (ORG), Daten, etc.
- Unterschied zu Regex: NER versteht Kontext, Regex nur Muster

- Beispiel: "Herr Müller aus München" → PER: "Müller", LOC: "München"

spaCy:

- Was: Open-Source NLP-Bibliothek für Python
- Modelle: `de_core_news_sm` (klein), `de_core_news_md` (mittel), `de_core_news_lg` (groß)
- Größer = genauer aber langsamer
- `lg` -Modell: ~500MB, beste Genauigkeit für deutsche Texte

Entity Labels:

Label	Bedeutung	Beispiel
PER	Person	"Max Mustermann"
LOC	Ort	"München"
ORG	Organisation	"Volkswagen AG"
MISC	Sonstiges	Produkte, Events

---

3. PDF-Export

Was: Professionelle Berichte als PDF generieren

Plan:

```
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas

def generate_pdf_report(feedbacks, stats):
    c = canvas.Canvas("report.pdf", pagesize=A4)
    c.drawString(100, 800, "Feedback-Analyse Bericht")
    # ... Tabellen, Charts
    c.save()
```

📖 Fachbegriffe PDF-Export erklärt:

ReportLab:

- Was: Python-Bibliothek zur programmatischen PDF-Erstellung
- Vorteil: Volle Kontrolle über Layout, keine externen Tools nötig
- Komponenten: Canvas (Zeichenfläche), Platypus (High-Level Layouts)

Alternativen:

Library	Stärke	Schwäche
ReportLab	Volle Kontrolle	Mehr Code
WeasyPrint	HTML→PDF	Braucht HTML
FPDF	Einfach	Weniger Features

Canvas-Koordinaten:

- Ursprung (0,0) = unten links
- A4: 595 x 842 Punkte (72 Punkte = 1 Zoll)
- `drawString(100, 800, "Text")` = 100 von links, 800 von unten

---

4. Docker-Compose

Was: One-Click Deployment für VW

Plan (docker-compose.yml):

```
version: '3.8'
services:
  backend:
    build: ./backend
    ports:
      - "8000:8000"
    volumes:
      - ./chroma_db:/app/chroma_db
```

```
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend
```

Befehl: `docker-compose up` → Alles läuft

📖 Fachbegriffe Docker-Compose erklärt:

Docker:

- Was: Container-Plattform zur Isolierung von Anwendungen
- Vorteil: "Works on my machine" → Works everywhere
- Image: Vorlage/Blaupause einer Anwendung
- Container: Laufende Instanz eines Images

Docker-Compose:

- Was: Tool zur Definition und Verwaltung von Multi-Container-Anwendungen
- Eine YAML-Datei definiert alle Services, Netzwerke, Volumes
- Startet/stoppt alle Container mit einem Befehl

docker-compose.yml Elemente:

Element	Bedeutung
`services`	Liste aller Container (backend, frontend)
`build`	Pfad zum Dockerfile
`ports`	Port-Mapping (Host:Container)
`volumes`	Persistente Datenspeicherung
`depends_on`	Startreihenfolge (frontend wartet auf backend)

Volumes (`./chroma_db:/app/chroma_db`):

- Warum: Container sind stateless (Daten gehen bei Neustart verloren)
- Volume: Verbindet Host-Ordner mit Container-Ordner
- Hier: ChromaDB-Daten bleiben bei Container-Neustart erhalten

Wichtige Befehle:

```
docker-compose up        # Alle Services starten
docker-compose up -d     # Im Hintergrund starten (detached)
docker-compose down      # Alle Services stoppen
docker-compose logs -f   # Logs anzeigen (live)
docker-compose build     # Images neu bauen
```

🔑 Key Talking Points für VWler

1. **"End-to-End funktional"**
   - Kein Mockup, alles funktioniert wirklich
2. **"83 wissenschaftliche Papers analysiert"**
   - Jede Entscheidung ist literaturbasiert
3. **"Forschungslücke identifiziert"**
   - PII + RAG = Niemand hat das vorher kombiniert
4. **"In 3 Wochen produktionsreif"**
   - Nur noch Evaluation, PDF-Export, Docker
5. **"Das System erkennt automatisch Probleme"**
   - Top-Probleme Feature auf Dashboard

🚀 Demo-Reihenfolge

1. Dashboard öffnen → "40 Feedbacks, automatische Problem-Erkennung"
2. Top-Problem klicken → "Sprachassistent, 5 Meldungen, betrifft ID.4"
3. Chat öffnen → "Welche Probleme gibt es?" → Antwort mit Quellen
4. Quelle klicken → "Jede Aussage ist nachvollziehbar"
5. Settings öffnen → "Hier kann man das Modell wechseln"

Cheatsheet für VW-Präsentation | Januar 2026