FlowAudit - Präsentation

KI-gestützte Rechnungsprüfung für Fördermittel

Inhalt

1. Prüfmerkmale (Features)

Vorgehen bei der Merkmalsprüfung

Jede Rechnung wird gegen ein **Regelwerk (Ruleset)** geprüft, das die steuerlichen Pflichtangaben definiert.

Verfügbare Regelwerke

Regelwerk	Rechtsgrundlage	Kernmerkmale
DE_USTG	§ 14 Abs. 4 UStG	11 Pflichtmerkmale + Zuwendungslogik
EU_VAT	Art. 226 MwSt-RL	Erweitert um USt-ID-Pflicht, Reverse Charge
UK_VAT	HMRC Notice 700	Tax Point, Unit Price

Die 11 Pflichtmerkmale (DE_USTG)

Identität

- **supplier_name_address** - Name und Anschrift des Lieferanten
- **customer_name_address** - Name und Anschrift des Kunden
- **supplier_tax_or_vat_id** - Steuernummer oder USt-ID
- **invoice_number** - Eindeutige Rechnungsnummer

Datum

- **invoice_date** - Rechnungsdatum
- **supply_date_or_period** - Leistungsdatum/-zeitraum

Beträge

- **net_amount** - Nettobetrag

- **vat_rate** - Steuersatz (7% oder 19%)
- **vat_amount** - Steuerbetrag
- **gross_amount** - Bruttobetrag

Text/Semantik

- **supply_description** - Leistungsbeschreibung (muss zum Projekt passen!)

Merkmal-Schema (maschinenlesbar)

Jedes Feature ist strukturiert definiert:

```
{
  "feature_id": "supplier_tax_or_vat_id",
  "name_de": "Steuernummer oder USt-ID",
  "name_en": "Tax ID or VAT ID",
  "legal_basis": "§ 14 Abs. 4 Nr. 2 UStG",
  "required_level": "REQUIRED",
  "extraction_type": "STRING",
  "validation": {
    "regex": "^DE\\d{9}$"
  },
  "generator_rules": {
    "can_be_missing": true,
    "typical_errors": ["missing", "wrong_format"]
  }
}
```

Sonderfall: Kleinbetragsrechnung

Bei Rechnungen ≤ 250 € brutto (§ 33 UStDV) gelten **reduzierte Anforderungen**:

Vollständige Rechnung	Kleinbetragsrechnung
11 Pflichtmerkmale	5 Pflichtmerkmale
Kundendaten erforderlich	Keine Kundendaten nötig
Netto + MwSt getrennt	Nur Bruttobetrag

Reduzierte Merkmale: - supplier_name - invoice_date - supply_description - gross_amount - vat_rate (implizit)

---

2. Die Prüfung (Pipeline)

Analyse-Pipeline Übersicht

Pipeline-Schritte im Detail

Schritt 1: Parse

- **Komponente:** pdfplumber / PyMuPDF
- **Aufgabe:** Text extrahieren, Felder per Regex finden
- **Output:** ParseResult mit extrahierten Werten

Schritt 2: Precheck (Regel-Engine)

- **Komponente:** Rule Engine
- **Aufgabe:** Pflichtfelder prüfen (ohne KI!)
- **Prüfungen:** Datumsformat, Betragsberechnung, Regex-Validierung

Schritt 3: RAG-Kontext

- **Komponente:** ChromaDB
- **Aufgabe:** Ähnliche Fälle aus Feedback finden
- **Output:** Top-K ähnliche Beispiele mit Korrekturen

Schritt 4: Legal Retrieval (optional)

- **Komponente:** Legal Retrieval Service
- **Aufgabe:** Relevante Rechtstexte finden

- **Output:** EU-Verordnungen, nationale Gesetze

Schritt 5: LLM-Analyse

- **Komponente:** Ollama / OpenAI / Anthropic
- **Aufgabe:** Semantische Prüfung (Projektpassung, Wirtschaftlichkeit)
- **Output:** Strukturiertes JSON-Ergebnis

Schritt 6: Finalize

- **Komponente:** Conflict Resolver
- **Aufgabe:** Regel + KI + User-Feedback zusammenführen
- **Output:** FinalResult mit Gesamtbewertung

Ergebnis-Struktur

```
{
  "features": [
    {
      "feature_id": "supplier_tax_or_vat_id",
      "status": "PRESENT",
      "value": "DE123456789",
      "source": "LLM",
      "confidence": 0.95
    }
  ],
  "semantic_check": {
    "project_fit": "RELEVANT",
    "economic_efficiency": "OK"
  },
  "overall_assessment": "APPROVED",
  "warnings": []
}
```

Status-Werte

- **PRESENT** - Merkmal vorhanden und korrekt
- **MISSING** - Merkmal fehlt
- **UNCLEAR** - Merkmal unklar/mehrdeutig

Gesamtbewertung

- **APPROVED** - Rechnung in Ordnung
- **NEEDS_REVIEW** - Manuelle Prüfung empfohlen
- **REJECTED** - Schwerwiegende Mängel

---

3. Systemarchitektur

Komponenten-Übersicht

Frontend (React)

- **Port:** 3000
- **Technologie:** React 18 + TypeScript + Vite + Tailwind
- **Aufgaben:** Upload, Ergebnisanzeige, Feedback

Backend (FastAPI)

- **Port:** 8000
- **Technologie:** Python 3.11+ / FastAPI
- **Aufgaben:** API, Business Logic, Task-Steuerung

PostgreSQL (Datenbank)

- **Port:** 5432
- **Aufgaben:** Dokumente, Ergebnisse, Regelwerke speichern

ChromaDB (Vector DB)

- **Port:** 8001
- **Aufgaben:** RAG-Beispiele, Legal Retrieval

Redis (Queue)

- **Port:** 6379
- **Aufgaben:** Celery Broker, Task Queue

Ollama (Local LLM)

- **Port:** 11434
- **Modelle:** llama3.2, mistral, etc.

Worker (Celery)

- **Aufgaben:** Hintergrundverarbeitung (Parse, Analyze)

Architekturprinzipien

Prinzip	Bedeutung
Container-First	Jede Komponente im eigenen Docker-Container
API-First	Kommunikation nur über REST-API
Stateless Backend	Kein Session-State, alles in DB/Cache
Transparency by Design	Alle KI-Ein-/Ausgaben werden gespeichert

---

Datenfluss

```
1. Frontend → POST /documents/upload → Backend → Storage (PDF)

2. Backend → task: parse_document → Worker → parse_results (DB)

3. Backend → task: precheck → Worker → precheck_results (DB)

4. Backend → task: prepare → Worker + ChromaDB → prepare_payloads (DB)

5. Backend → task: llm_run → Worker + Ollama → llm_runs (DB)

6. Backend → task: finalize → Worker → final_results (DB)
```

Datenpersistenz

Haupttabellen (PostgreSQL)

Tabelle	Inhalt
projects	Vorhabendefinitionen
documents	Hochgeladene PDFs
parse_results	Parser-Ausgaben
precheck_results	Regel-Engine-Ergebnisse
prepare_payloads	KI-Input (persistiert!)
llm_runs	KI-Responses
final_results	Zusammengeführte Ergebnisse
feedback	Menschliche Korrekturen
rag_examples	Gelernte Beispiele
rulesets	Regelwerk-Versionen

4. RAG-Lernmechanismus

Warum RAG statt Fine-Tuning?

Aspekt	Fine-Tuning	RAG (FlowAudit)
Trainingsaufwand	Hoch (GPU, Stunden)	Gering (CPU, Sekunden)
Datenbedarf	Hunderte Beispiele	Ab 1 Beispiel wirksam
Aktualisierung	Neues Modell nötig	Sofort nach Feedback
Nachvollziehbarkeit	Blackbox	Transparent
Didaktischer Wert	Gering	Hoch (zeigt Lernprozess)

RAG Learning Pipeline

Der Lernkreislauf

```
1. FEEDBACK        →  User korrigiert LLM-Ergebnis
       ↓
2. EMBEDDING       →  Text wird zu 384-dim. Vektor
       ↓
3. STORAGE         →  Vektor + Metadaten in ChromaDB
       ↓
4. RETRIEVAL       →  Ähnlichkeitssuche bei neuer Rechnung
       ↓
5. INJECTION       →  Beispiele in LLM-Prompt einfügen
       ↓
6. BETTER OUTPUT   →  LLM macht weniger Fehler
```

Lernprozess im Detail

Schritt 1: User gibt Feedback

LLM-Ergebnis:

```
supplier_tax_or_vat_id = PRESENT ("12345")
```

User korrigiert:

```
supplier_tax_or_vat_id = MISSING
Grund: "12345 ist Kundennummer, keine Steuernummer!"
```

Schritt 2: Embedding erstellen

```
embedding_text = """
INVOICE: Bauunternehmen XY, Rechnung 2024-001,
         Sanitärarbeiten, Betrag: 5.000€...
CORRECTION supplier_tax_or_vat_id: PRESENT -> MISSING
(Kundennummer statt Steuernummer)
"""

# Sentence Transformer erzeugt 384-dim. Vektor
vector = model.encode(embedding_text)
```

Schritt 3: In ChromaDB speichern

```
collection.add(
    ids=["rag_example_001"],
    embeddings=[vector],
    documents=[embedding_text],
    metadatas={
        "feature_id": "supplier_tax_or_vat_id",
        "ruleset_id": "DE_USTG",
        "created_at": "2025-12-26"
    }
)
```

Retrieval bei neuer Rechnung

Schritt 4: Ähnliche Beispiele finden

```
# Neue, ähnliche Rechnung kommt
query = "Handwerkerrechnung, Maurerarbeiten, ID: 54321..."

# Vektor-Suche in ChromaDB
similar = collection.query(
    query_embeddings=[model.encode(query)],
    n_results=3
)

# → Findet das korrigierte Beispiel mit 87% Ähnlichkeit!
```

Schritt 5: Few-Shot im Prompt

```
## Ähnliche Fälle aus früheren Prüfungen:

1. (Ähnlichkeit: 87%) Bei einer Handwerkerrechnung wurde
   eine Kundennummer fälschlicherweise als Steuernummer
   erkannt. Richtig: MISSING, weil deutsche Steuernummern
   dem Muster XX/XXX/XXXXX folgen.

2. (Ähnlichkeit: 72%) Der Leistungszeitraum "Q3 2025"
   wurde als UNCLEAR klassifiziert. Richtig: PRESENT,
   weil Q3 eindeutig Juli-September bedeutet.

Bitte berücksichtige diese Erfahrungen bei der Prüfung.
```

Lernkurve

Fehlerreduktion über Zeit

Ergebnis: Nach 18 Beispielen **56% weniger Fehler** bei supplier_tax_or_vat_id!

Qualitätssicherung

Konfidenz-Score berechnen

```
def calculate_confidence(feedback, corrections):
    score = 1.0

    # Viele Korrekturen = komplexer Fall
    if len(corrections) > 3:
        score -= 0.2

    # Begründungen erhöhen Konfidenz
    if all(c.note for c in corrections):
        score += 0.1

    return max(0.0, min(1.0, score))
```

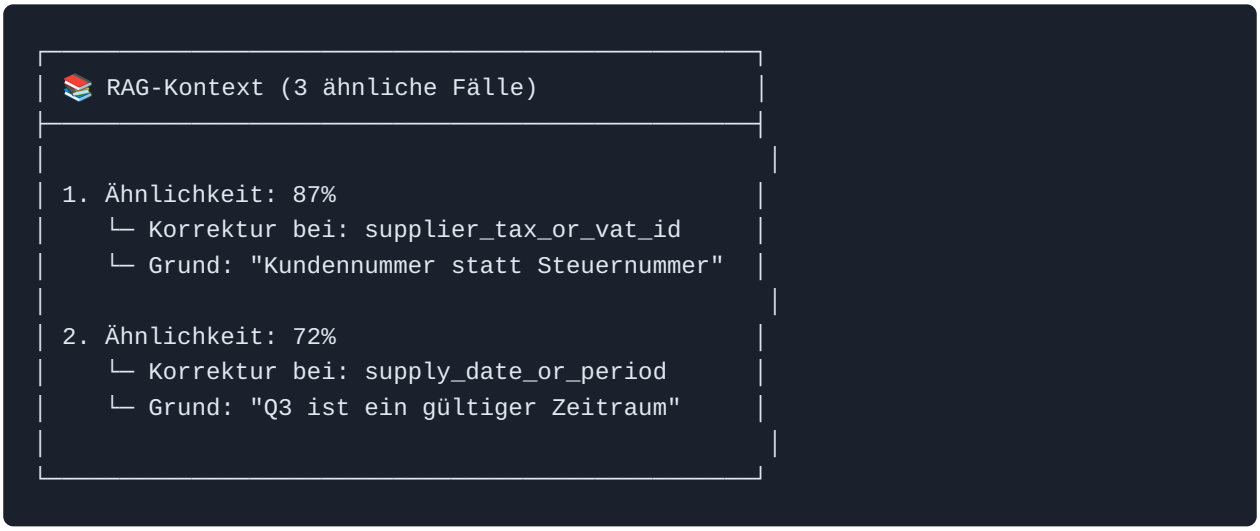
Wann wird ein RAG-Beispiel erstellt?

- ✅ CORRECT mit Korrekturen → JA
- ✅ PARTIAL mit Korrekturen → JA
- ❌ CORRECT ohne Korrekturen → NEIN

- ❌ WRONG ohne Details → NEIN

UI-Transparenz

RAG-Kontext wird angezeigt



Zusammenfassung

Kernbotschaft

> ***FlowAudit prüft nicht „mit KI"**, sondern **gegen Recht**, mit **strukturierter Datenverarbeitung**, unterstützt durch **KI dort, wo Regeln enden**.*

Die 4 Säulen

1. Merkmale

Explizite, versionierte Regelwerke (DE/EU/UK)

2. Pipeline

Parser → Regeln → RAG → LLM → Finalize

3. Architektur

Container-basiert, API-First, transparent

4. Lernen

RAG mit Few-Shot statt Blackbox-Fine-Tuning

Vorteile des Ansatzes

Vorteil	Beschreibung
Nachvollziehbar	Jede Entscheidung ist dokumentiert
Lernfähig	Verbessert sich durch Feedback
Regelkonform	Basiert auf echten Rechtsgrundlagen
Flexibel	Neue Regelwerke einfach hinzufügbar
Datenschutz	Lokale LLMs möglich (Ollama)

Technologie-Stack

Komponente	Technologie
Frontend	React 18 + TypeScript + Tailwind
Backend	Python 3.11 + FastAPI
Datenbank	PostgreSQL 16
Vector DB	ChromaDB
LLM	Ollama (lokal) oder Cloud-APIs
Queue	Redis + Celery
Container	Docker Compose

FlowAudit - KI-gestützte Rechnungsprüfung Version 1.0.0