Aufgabenblatt 11

Colmant

12. Juli 2025

Aufgabe 1: AI infrastructure

Kernservices für Agenten

Google Cloud stellt mit **Vertex AI** eine durchdachte Agenten-Plattform bereit. Im Zentrum steht das **Agent Development Kit (ADK)**, ein quelloffenes Python-Framework, das Entwicklern erlaubt, in wenigen Zeilen codebasierte Agenten mit eigenem Reasoning, Tool-Zugriff und mehrstufigen Workflows zu definieren. Über das standardisierte **Agent2Agent-Protokoll (A2A)** lassen sich mehrere Agenten zu komplexen Netzwerken verbinden, in denen sie Informationen austauschen und Aufgaben arbeitsteilig erledigen. Für Produktion und Skalierung bietet die verwaltete **Agent Engine** automatische Lastverteilung, Health-Checks und ein eingebautes Monitoring. Kurz- und Langzeit-Gedächtnis werden über konfigurierbare Speicher-Backends gewährleistet.

Microsoft antwortet mit **Azure AI Foundry** (früher AI Studio) und dem **Foundry Agent Service**. Hier steht neben einer grafischen Oberfläche vor allem das **Semantic Kernel** sowie das Forschungsframework **AutoGen** im Fokus. Entwickler bauen Agenten, indem sie Prompts, Funktionen und spezialisierte Teilagenten definieren, die parallel laufen und über A2A oder das **MCP-Protokoll** interoperieren. Azure liefert zudem eine integrierte Observability-Suite mit Logging, Performance-Metriken und Fairness-Checks.

Modellanpassung und Integration

Bei Google bildet die **Gemini-Modellreihe** das Herzstück: Sie kombiniert multimodale Fähigkeiten mit starkem Reasoning. Vertex AI erlaubt serverseitiges Feintuning und unterstützt MLOps-Pipelines für kontinuierliches Retraining. Über den **Model Garden** können auch Open-Source-LLMs wie Mistral, Llama und BLOOM eingebunden werden.

Azure setzt primär auf die **GPT-4-Familie** sowie hunderte Partner- und Community-Modelle aus dem **Azure OpenAI Service**. GPT-4.1 bietet bis zu einer Million Token Kontext und ermöglicht lange, zusammenhängende Simulationen. Feintuning, Distillation und Integration externer Modelle (z. B. Hugging Face) sind vollständig unterstützt.

Daten, Vektorsuche und Gedächtnis

Google bietet mit **Vertex AI Vector Search** eine vollverwaltete Vektordatenbank mit nativer Integration in Vertex AI Search. Die Agent Engine kann direkt auf diese Indizes zugreifen und Kontext aus Dokumenten, Datenbanken oder Streaming-Quellen beziehen.

Azure nutzt **Azure AI Search** mit Vektor-Unterstützung: Skillsets transformieren Inhalte in Einbettungen, die in hochverfügbaren Indizes gespeichert werden. Foundry-Agenten greifen direkt über SDKs darauf zu. Für Echtzeitszenarien stehen Event-Hubs und Synapse-Integration zur Verfügung.

Eigene Einschätzung

Beide Plattformen sind im Juli 2025 sehr ausgereift. Google Cloud punktet mit Offenheit, Modularität und schneller Einstiegshürde. Entwickler, die Open-Source-Modelle nutzen oder flexibel experimentieren möchten, profitieren hier besonders.

Microsoft Azure überzeugt durch große Modellbibliotheken, tiefe Enterprise-Integration und leistungsfähige Werkzeuge für Skalierung, Sicherheit und Analyse. Für unternehmensweite Simulationssysteme mit hohen Compliance-Anforderungen ist Azure aktuell

die stärkere Wahl. Für kreative, modulare Agentenentwicklung mit maximaler Offenheit bietet Google Cloud jedoch Vorteile.

Aufgabe 2: Justice explained by AI

In dieser Aufgabe sollten wir einen AI-Agenten simulieren durch dei Fähigkeit von modernen LLM-AIs die Kontext und Kohärenz über mehrere Fragen hinweg beibehalten. Ich habe mir den Aufgaben Teil A vorgenommen, in dem ich eine Conservation zwischen einem AI-Experten und einem Alien simuliere durch gezieltes Prompting. Der AI-Experte soll dem Alien erklären, was Gerechtigkeit ist. Ich habe mich dazu entschieden die AI einen Philosophen spielen zu lassen. Dann habe ich die Argumente so gut wie ich konnte mit meinem Philosophie-Wissen versucht zu zerlegen. Das Ergebnis ist eine interessante Diskussion, die zeigt, wie komplex und vielschichtig das Thema Gerechtigkeit ist und wie wichtig es ist in Definitionen genau und präzise zu sein. Das Gespräch kann unter follgenden Link nach gelesen werden: AI conversation.

Evaluation

Mir ist gleich zu beginn aufgefallen, dass die AI eine sehr ungewöhnliche Definition von Gerechtigkeit gegeben hat. In ihrem versuch Gerechtigkeit einem ALien ohne gesellschaftlichen Vorwissen zu erklären war die Definition sehr abstrakt und eher unpräzise und lies viel Raum für Missinterpretationen. Fast bis zum Ende hat die AI aber Kohärenz bewahrt und hat den Character des Philosophen nicht gebrochen. Was ich aber schade fand ist dass, die AI nicht wirklich einen Philosophen gespielt, sondern eher alle Philosophischen Ideen wieder gegeben hat. Das führte dazu dass der Character keine wirklichen eigenen Ideen bzw. Feste standpunkte hatte. Das führte dazu dass, die AI die Einwände des Aliens so entkräftet hat indem es einfach eine reihe neuer Argumente gebracht hat, aus dennen sich dann eins ausgesucht werden konnte. Als das Alien dann am Ende zu erkennen gab dass, es die Argumente auf eine eher unmoralische Weise interpretiert brach die AI dann aber doch etwas aus dem Character und es wirkt so als würde eher ein teil des Gedanken Process in der Ausgabe sichtbar sein. In der aller letzten Antwort stimmte die AI dann auch noch dem Alien zu dass, die AI eigentlich gar nicht Gerechtigkeit definieren darf aufgrund der oben genannten Argumente. Für eine Philosophische Diskussion ist die AI unbrauchbar sie ist sowohl zu ungenau und gleichzeitig gibt sie zu viele Argumente auf einmal so dass es sehr schwer ist von einem Argument zum nächsten zu kommen. Das Kann natürlich auch an meinem prompting liegen.

Aufgabe 3:

1 Zielsetzung

Generierung eines plausiblen, zeitgestempelten Krisenverlaufs, Empfehlung von Ressourcenzuweisungen und Identifizierung potenzieller Engpässe bei Evakuierungsrouten basierend auf einer textuellen Schilderung einer urbanen Gefährdung.

2 Benötigte Werkzeuge (APIs)

Werkzeugkategorie	Zweck
Live Web Search	Aktuelle Informationen zu Wetter, Infrastruktur und Nachrichten abrufen
GIS/Text-Map-Parser	Umwandlung des textbasierten Stadtplans in Knoten-Kanten-Graphen
Code-Interpreter	Ausführen leichter Simulationen (z.B. Feuer- und Verkehrsflussmodell)
Rechner (Calculator)	Schnelle Berechnungen für Ankunftszeiten und Ressourcenbedarf
Datei I/O	Speichern von Zwischenergebnissen und Journaleinträgen
Benachrichtigungs-API	Automatischer Versand von Berichten an Stadtplaner

3 Denk- und Handlungsrahmen (ReAct-Stil)

1. **Eingabe verstehen**
   Reason: Extraktion von Entitäten (Gefährdungstyp, Ort, Schweregrad).
   Act: NLP-Parser aufrufen.

2. **Karteneinlesung**
   Reason: Bestimmung des relevanten Teilgebiets.
   Act: GIS/Text-Map-Parser konvertieren.

3. **Gefährdungsmodellierung**
   Reason: Auswahl des Ausbreitungsmodells (z.B. Feuerausbreitung hangaufwärts schneller).
   Act: Simulation via Code-Interpreter in Zeitschritten (z.B. 10 Minuten).

4. **Timeline-Generierung**
   Reason: Zuordnung der Ausbreitung zu Zeitstempeln in der Stadtgeografie.
   Act: Chronologische Ereignisliste erstellen.

5. **Ressourcenzuweisung**
   Reason: Entscheidung über Einsatz von Einheiten (Feuerwehr, Rettung, Polizei).
   Act: Greedy-Algorithmus im Code-Interpreter zur Zuweisung.

6. **Engpass-Analyse Evakuierungsrouten**
   Reason: Identifikation von Straßen mit Überlastungsgefahr.
   Act: Verkehrsfluss-Simulation (Warteschlangenmodell).

7. **Zusammenfassung und Bericht**
   Reason: Konsolidierung der Ergebnisse für Entscheidungsträger.
   Act: Logbuch und Maßnahmenvorschläge via Datei I/O speichern.

8. **Iteration / Feedback**
   Reason: Integration von Planer-Anpassungen (z.B. Straßensperrungen).
   Act: Rückkopplungsschleife, erneute Simulation.

4 Speicherarchitektur

- **Kurzzeit-Scratchpad**: Zwischenergebnisse der Simulation (Feuerfront, Ressourcenzustände)

- **Ereignisjournal (Datei)**: Endgültige Timeline-Einträge und Zuweisungen
- **Status-Cache (Key–Value)**: Geparster Kartengraph, aktuelle Ressourcendaten
- **Override-Log**: Nutzer-Trigger und manuelle Anpassungen

5 Potenzielle Fehlerquellen

- *Fehlerhafte Karteninterpretation*: Fehlende oder erfundene Straßendaten
- *Simulationsverzerrung*: Ungenaue numerische Modelle
- *Ressourcen-Staubild*: Überbuchung derselben Einheit für mehrere Einsätze
- *Endlosschleifen*: Unklare Abbruchkriterien bei Iterationen
- *Sprachliche Missinterpretation*: Unpräzise Ortsangaben führen zu falschen Zonen

Github(branch:main)

Wie immer sind alle meine benutzten Dateien auf meinem Github zu finden.