

MICROSOFT FABRIC ALS DATENBACKBONE FÜR GENAI-ANWENDUNGEN

Von Daten zu Antworten

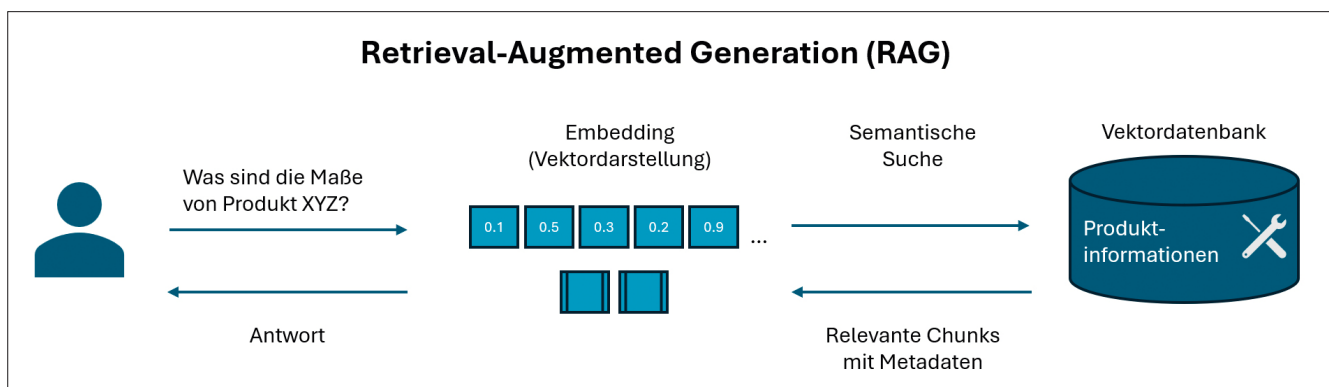
RAG, Vektorsuche und Agenten: Warum moderne GenAI-Projekte eine starke Datenplattform brauchen und wie Microsoft Fabric sie liefert.

Wer GenAI ernsthaft nutzen will, muss bei den Daten anfangen. Ohne strukturierte, kontextreiche und zugängliche Informationen bleibt selbst das leistungsfähigste Sprachmodell blind für das Wissen im Unternehmen. Dieser Artikel zeigt, wie Microsoft Fabric als zentrale Datenplattform die Verbindung zwischen moderner Datenstrategie und innovativen KI-Anwendungen schafft.

Die rasante Entwicklung generativer KI hat in den letzten Jahren neue Maßstäbe gesetzt. Sprachmodelle wie GPT-4

gezielt Inhalte aus einer externen Wissensbasis abrufen. Diese Inhalte werden dem Modell als zusätzlicher Kontext mitgegeben. So entsteht eine hybride Architektur, in der das Sprachmodell nicht mehr nur auf sein internes Wissen angewiesen ist, sondern gezielt auf unternehmensspezifische Informationen zugreifen kann.

Damit dieses Prinzip funktioniert, braucht es eine leistungsfähige Datenplattform, die sowohl strukturierte als auch unstrukturierte Daten effizient verwalten, transformieren und



Retrieval-Augmented Generation zur semantischen Suche (Bild 1)

oder Mistral erzeugen beeindruckend natürliche Texte, beantworten komplexe Fragen und unterstützen bei der Automatisierung von Geschäftsprozessen.

Doch trotz ihrer Leistungsfähigkeit bleiben diese Modelle in einem entscheidenden Punkt limitiert: Sie verfügen nicht über aktuelles, unternehmensspezifisches Wissen. Ihre Antworten basieren auf Trainingsdaten, die oft veraltet oder zu allgemein sind, um im Unternehmenskontext verlässlich zu funktionieren.

Damit generative KI in der Praxis echten Mehrwert liefert, braucht sie Zugriff auf relevante, strukturierte und kontextreiche Daten. Erst wenn ein Modell mit den richtigen Informationen angereichert wird, kann es präzise, nachvollziehbare und vertrauenswürdige Antworten geben. Die Qualität dieser Daten entscheidet über die Qualität der KI. Wer GenAI ernsthaft nutzen will, muss bei den Daten anfangen.

Ein vielversprechender Ansatz, um Sprachmodelle mit aktuellem Wissen zu versorgen, ist Retrieval-Augmented Generation. Dabei wird das Modell nicht allein gelassen, sondern durch eine Suchkomponente ergänzt, die bei jeder Anfrage

bereitstellen kann. Microsoft Fabric bietet genau das. Mit OneLake als zentralem Datenspeicher, Lakehouse und Eventhouse als Analyse-Engines sowie integrierten Tools für Datenpipelines, Transformationen und Machine Learning stellt Fabric eine durchgängige Umgebung bereit, um Daten für GenAI-Anwendungen nutzbar zu machen.

Dieser Artikel zeigt, wie sich mit Microsoft Fabric eine vollständige RAG-Pipeline aufbauen lässt. Von der Datenaufnahme über die Vektorisierung bis hin zur Integration in intelligente Agenten und Workflows wird Schritt für Schritt erläutert, wie Daten aufbereitet und in generative KI-Anwendungen eingebunden werden können.

Warum GenAI ohne Kontext nicht gut funktioniert

Generative KI, insbesondere in Form von sogenannten Large Language Models (LLMs), ist in der Lage, Texte zu verfassen, Fragen zu beantworten, Code zu generieren oder komplexe Inhalte zusammenzufassen. Diese Modelle basieren auf Milliarden von Parametern und wurden mit gewaltigen Mengen

an Textdaten trainiert. Ihre Stärke liegt in der Fähigkeit, Sprache zu verstehen und kontextbezogen zu reagieren.

Doch genau hier liegt auch eine zentrale Schwäche. Denn ein LLM kennt nur das, was es beim Training gesehen hat. Es hat keinen Zugriff auf aktuelle Informationen, keine Verbindung zu unternehmensspezifischen Datenquellen und keine Möglichkeit, neues Wissen selbstständig zu erschließen. In der Praxis führt das dazu, dass Antworten oft veraltet, ungenau oder schlicht erfunden sind. Dieses Phänomen ist als Halluzination bekannt und stellt ein zentrales Problem beim produktiven Einsatz von generativer KI dar.

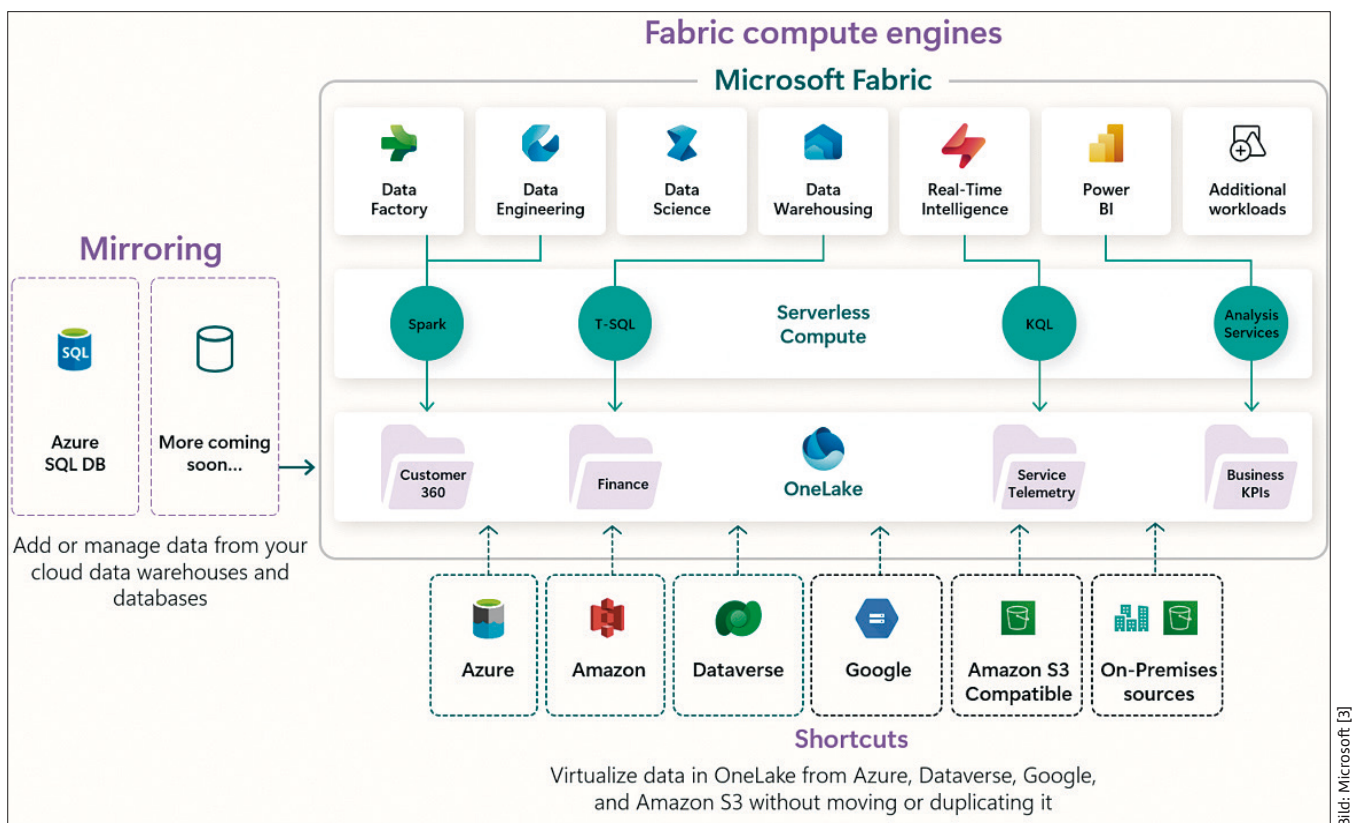
Die Idee liegt nahe, das Modell einfach mit neuen Daten nachzutrainieren. Doch in der Praxis ist dies selten sinnvoll. Das Nachtrainieren eines LLM ist technisch aufwendig, teuer und erfordert spezialisierte Infrastruktur. Zudem besteht das Risiko, dass das Modell dabei bestehendes Wissen verliert oder unerwünschte Verzerrungen übernimmt. Vor allem aber ist das Nachtrainieren nicht dynamisch. Neue Informationen lassen sich nicht kurzfristig einpflegen, was gerade in sich schnell ändernden Geschäftskontexten problematisch ist.

das Modell abdriftet oder irrelevante Inhalte verarbeitet. Der Prompt ist kein Speicherort für Wissen, sondern ein situativer Kontextgeber.

Um diese Einschränkungen zu überwinden, wurde das Konzept der Retrieval-Augmented Generation (RAG) entwickelt [1]. Dabei wird das Sprachmodell durch eine semantische Suchkomponente ergänzt (siehe Bild 1).

Wenn ein Nutzer eine Frage stellt, wird diese in einem ersten Schritt in eine Suchanfrage umgewandelt. Die Suche liefert dann relevante Inhalte aus einer externen Wissensquelle, etwa aus Dokumenten, Datenbanken oder Logs. Diese Inhalte werden dem Modell als zusätzlicher Kontext mitgegeben, bevor es eine Antwort generiert. Das Modell bleibt also generativ, aber es antwortet auf Basis von aktuellen, überprüfbaren Informationen.

Damit eine solche Architektur funktioniert, sind mehrere technische Schritte erforderlich. Sie reichen von der Zerlegung der Ausgangsdokumente über die semantische Aufbereitung bis hin zur Speicherung und Abfrage der Daten. Microsoft Fabric bietet für all diese Schritte eine integrierte



Microsoft Fabric Foundation mit OneLake als zentralem Datenspeicher (Bild 2)

Ein anderer Ansatz besteht darin, möglichst viele Daten direkt in den Prompt des Modells zu geben. Doch auch das ist keine tragfähige Lösung. Prompts unterliegen technischen Begrenzungen hinsichtlich ihrer Länge, und je mehr Kontext man einfügt, desto schwieriger wird es für das Modell, die relevanten Informationen korrekt zu gewichten. Zudem steigt mit der Menge an Kontext auch die Wahrscheinlichkeit, dass

Plattform, die es ermöglicht, den gesamten Prozess konsistent und skalierbar umzusetzen. Von der Datenaufnahme über die Transformation bis hin zur Speicherung und Analyse stehen alle notwendigen Werkzeuge zur Verfügung. Besonders relevant sind dabei OneLake als zentraler Datenspeicher und Lakehouse sowie Eventhouse als Analyse-Engines. In Kombination mit Azure OpenAI, Azure AI Foundry oder anderen ►

Services zum Betreiben von eigenen Modellen lassen sich so leistungsfähige RAG-Pipelines aufbauen, die generative KI mit unternehmensspezifischem Wissen versorgen.

Die Datenbasis für intelligente Systeme: Microsoft Fabric im Überblick

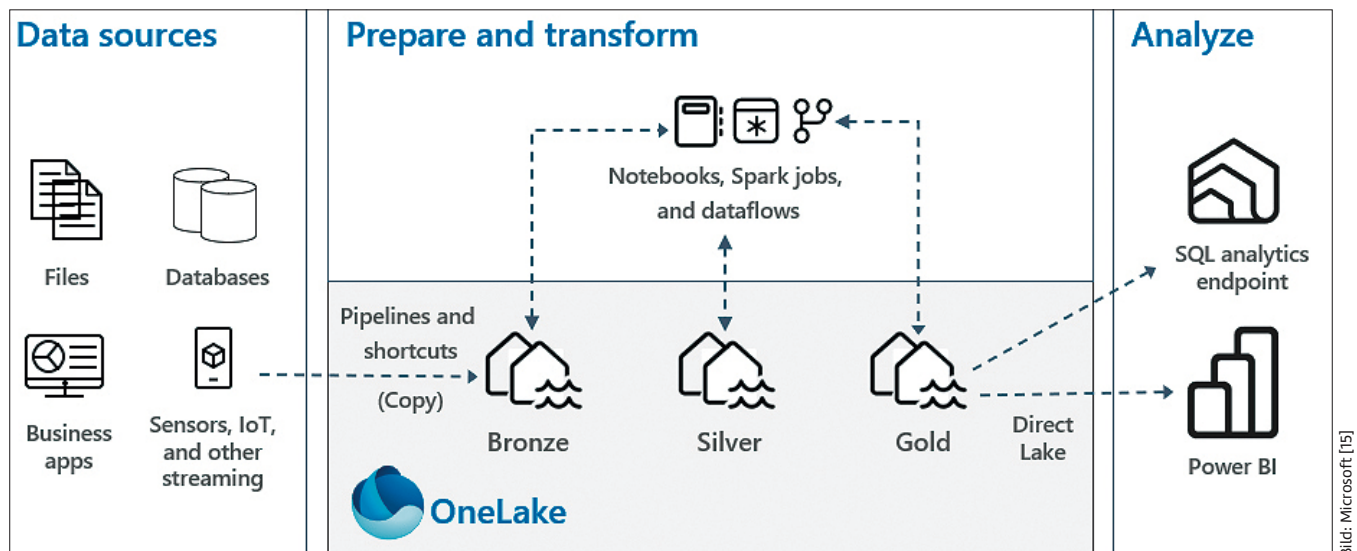
Microsoft Fabric [2] ist eine moderne Datenplattform, die darauf ausgelegt ist, Datenintegration, Datenspeicherung, Datenverarbeitung und Datenanalyse in einer einheitlichen Umgebung zusammenzuführen (siehe Bild 2 und [3]). Für Unternehmen, die generative KI produktiv einsetzen wollen, bietet Fabric eine zentrale Grundlage, um Daten effizient zu verwalten und für intelligente Systeme nutzbar zu machen. Im Kontext von Retrieval-Augmented Generation spielt Fabric eine Schlüsselrolle, da es alle notwendigen Bausteine für eine durchgängige Datenpipeline bereitstellt.

Im Zentrum von Microsoft Fabric steht OneLake [4], ein unternehmensweiter Data Lake, der als zentraler Speicher für

können sowohl über Apache Spark [7] als auch über automatisch generierte SQL-Endpunkte verarbeitet werden.

Das Eventhouse [8] ist auf zeitbasierte, semistrukturierte Daten wie Logs, Telemetrie oder Events spezialisiert. Es basiert auf dem Azure Data Explorer und eignet sich für Szenarien, in denen Daten kontinuierlich anfallen und schnell analysiert werden müssen. In der aktuellen Version unterstützt das Eventhouse auch Vektor-Datentypen. Dabei handelt es sich um Felder, die numerische Vektoren speichern, wie sie etwa bei Embeddings entstehen. Diese Vektoren können dann für eine semantische Suche genutzt werden, um die relevantesten Inhalte abzufragen [9].

Mit der aktuellen Public Preview von Azure Cosmos DB in Microsoft Fabric erweitert sich das Spektrum an KI-fähigen Speicherlösungen innerhalb der Plattform [10]. Cosmos DB in Fabric bringt Funktionen wie Vektorindizierung, Volltextsuche und hybride Suche direkt in die Fabric-Umgebung. Ein wesentlicher Vorteil liegt darin, dass sich Nutzer nicht um das



Microsoft Fabric und die Medallion-Lakehouse-Architektur (Bild 3)

alle Komponenten dient. OneLake speichert strukturierte, semistrukturierte und unstrukturierte Daten gemeinsam und ermöglicht so eine konsistente Datenhaltung über verschiedene Anwendungsbereiche hinweg. Die Daten stehen zentral und zugriffsoptimiert zur Verfügung, indem Daten entweder gespiegelt oder direkt über sogenannte Shortcuts von ihren Quellsystemen abgerufen werden.

Fabric nutzt auf OneLake aufbauend verschiedene logische Speicherformen, die jeweils für bestimmte Datenarten und Analysezwecke optimiert sind. Das Lakehouse [5] eignet sich für strukturierte und unstrukturierte Daten, die analytisch verarbeitet werden sollen. Es basiert auf dem offenen Delta-Lake-Format [6], das die Skalierbarkeit eines Data Lake mit der Abfragefähigkeit eines Data Warehouse kombiniert. Anders als klassische relationale Datenbanken ist ein Lakehouse nicht auf transaktionale Workloads ausgelegt, sondern auf große, analytische Datenmengen. Die Daten

Management der Datenbank kümmern müssen. Durch die tiefe Integration mit OneLake ist Cosmos DB in Fabric vollständig in das Fabric-Ökosystem eingebunden und kann dort genutzt werden.

Für die Datenaufnahme unterstützt Fabric verschiedenste Formate. Neben strukturierten Tabellen können auch unstrukturierte Dokumente wie zum Beispiel PDF, Word oder PowerPoint verarbeitet werden. Wenn diese Dateien nicht direkt als Text vorliegen, kann Azure AI Search [11] genutzt werden, um Inhalte zu extrahieren, zu analysieren oder automatisch zu beschreiben. Der OneLake File Indexer [12] in Azure AI Search ermöglicht es, solche Dateien in einen Index zu überführen und mit Metadaten anzureichern. Die so gewonnenen Inhalte lassen sich über Azure AI Search abrufen. Aber auch über eigene Data Pipelines und Spark Notebooks können Dateien durch eigene Implementierung in strukturierte Daten umgewandelt werden.

Ein wesentlicher Aspekt bei der Nutzung von Microsoft Fabric ist das sogenannte Capacity-Modell [13]. Statt einzelne Dienste separat zu lizenzieren, basiert Fabric auf einem einheitlichen Ressourcenmodell, das über sogenannte Capacities bereitgestellt wird. Eine Capacity ist eine reservierte Menge an Rechenleistung, Speicher und IO-Ressourcen, die einem bestimmten Tenant oder Arbeitsbereich zugewiesen wird. Sie bildet die Grundlage für alle Workloads in Fabric, darunter Lakehouse, Data Pipelines, Notebooks, Eventhouse oder Power BI.

Capacities werden in sogenannten Capacity Units (CUs) gemessen. Die kleinste Einheit ist F2 mit zwei Capacity Units, größere CUs bieten entsprechend mehr Leistung. Die Capacity Unit gibt aber nicht nur die Kapazität vor, sondern bestimmt auch, welche Features vorhanden sind. Bei F64 beginnen diverse Features wie zum Beispiel die Möglichkeit, mit einem integrierten Copiloten die Daten zu befragen und zu durchsuchen.

und Speicherorte parallel verwaltet werden müssen. Auch die Integration mit Power BI erfolgt meist über separate Schritte.

Azure Databricks hingegen ist eine offene, stark auf Data Science und Machine Learning ausgerichtete Plattform. Sie bietet maximale Flexibilität für komplexe Datenpipelines, Modelltraining und fortgeschrittene Analysen. Databricks ist besonders geeignet für Teams mit tiefem technischem Know-how, die individuelle Lösungen auf Basis von Apache Spark, MLflow oder Delta Lake entwickeln wollen. Die Plattform ist leistungsfähig, aber in der Bedienung komplexer und weniger auf Endanwender ausgerichtet.

Microsoft Fabric positioniert sich zwischen diesen beiden Welten. Es kombiniert die Benutzerfreundlichkeit und Governance eines BI-Systems mit der Flexibilität und Skalierbarkeit einer modernen Datenplattform. Durch OneLake als gemeinsamem Speicher, einheitliche Workspaces, rollenbasierte Sicherheit und native Unterstützung für Spark, SQL und Power BI entsteht eine Umgebung, in der Daten nicht nur

● Listing 1: Erstellen von Markdown Pages aus PDF als PySpark Notebook

```
# Create markdown based on text from PDF files
%pip install pdfminer.six
from pdfminer.high_level import extract_pages

input_folder =
    "/lakehouse/default/Files/bronze/product-pdfs"
output_folder =
    "/lakehouse/default/Files/silver/product-mds"
chunk_output_folder = output_folder + "/chunks/"

def extract_markdown_pages(pdf_path):
    for index, page_layout in enumerate(
        extract_pages(pdf_path), start=1):
        text = ""
        for element in page_layout:
            if hasattr(element, "get_text"):
                text += element.get_text()
        yield { "page_number": index, "page_content":
            text, "filename": pdf_path }

pdf_files = get_files(input_folder, "*.pdf")
for pdf_file in pdf_files:
    markdown_pages = extract_markdown_pages(
        pdf_file.filename)
    write_files(output_folder, markdown_pages)
```

Das Preismodell ist flexibel gestaltet. Capacities können entweder stundenweise oder monatlich gebucht werden, was eine gute Planbarkeit und Skalierbarkeit ermöglicht. Unternehmen können ihre Kapazitäten jederzeit anpassen, um auf Lastspitzen oder neue Projekte zu reagieren. Microsoft stellt dafür ein SKU Estimator [14] zur Verfügung, mit dem sich der Ressourcenbedarf für verschiedene Workloads wie Spark, SQL oder Power BI im Voraus abschätzen lässt.

Im Vergleich zu anderen Plattformen wie Azure Synapse Analytics oder Azure Databricks verfolgt Microsoft Fabric einen konsequent integrierten Plattformansatz.

Azure Synapse Analytics ist besonders stark im Bereich strukturierter Daten und Data Warehousing. Es bietet dedizierte SQL-Pools, Spark-Integration und eine Vielzahl an Konnektoren. Die Plattform eignet sich gut für klassische BI-Szenarien, bei denen strukturierte Daten in hoher Qualität für Reporting und Analyse aufbereitet werden. Allerdings ist die Benutzerführung fragmentierter, da verschiedene Engines

verarbeitet, sondern auch direkt für Analyse- und KI-Anwendungen nutzbar gemacht werden können. Besonders für Organisationen, die sowohl strukturierte als auch unstrukturierte Datenquellen in GenAI-Szenarien integrieren wollen, bietet Fabric eine konsistente und zukunftssichere Lösung.

Für GenAI-Projekte bedeutet das: Mit Fabric steht eine Plattform zur Verfügung, die alle relevanten Schritte von der Datenaufnahme über die Transformation bis zur Bereitstellung für Agenten und Workflows in einem konsistenten System abbildet. Das reduziert Komplexität, erhöht die Wiederverwendbarkeit und schafft eine solide Grundlage für skalierbare KI-Anwendungen.

Wie Daten in Fabric GenAI-fähig werden

Die Qualität generativer KI steht und fällt mit der Qualität der Daten, auf denen sie basiert. Damit ein Sprachmodell im Rahmen von Retrieval-Augmented Generation präzise, kontextbezogene Antworten liefern kann, müssen die zu- ►

grunde liegenden Informationen nicht nur vorhanden, sondern auch strukturiert, semantisch sinnvoll aufbereitet und technisch zugänglich sein. Microsoft Fabric bietet eine durchgängige Plattform, um diesen Prozess effizient und skalierbar umzusetzen.

Die Datenaufbereitung beginnt nicht direkt mit dem Zerlegen von Dokumenten in Vektoren, sondern idealerweise mit einer mehrstufigen Datenarchitektur, wie sie aus dem Data Warehousing bekannt ist: Bronze, Silber und Gold [15] (siehe Bild 3). In der Bronze-Stufe werden Rohdaten aus verschiedenen Quellen wie PDFs, Office-Dokumenten oder Webseiten gesammelt und in OneLake abgelegt. Um aus einem Dokument wie zum Beispiel einem PDF eine semantische Struktur zu bekommen, eignet sich oft die Konvertierung nach Markdown, welche deutlich mehr Strukturinformationen weitergeben kann (siehe Listing 1).

In der Silber-Stufe erfolgt eine erste semantische Strukturierung, das sogenannte Chunking [16]. Hier werden Inhalte beispielsweise nach Kapiteln, Abschnitten oder logischen Einheiten gruppiert (siehe Listing 2). Besonders wichtig ist da-

Formen und Strukturen gebracht, die für den jeweiligen Use Case effizient und sinnvoll sind. Bei einem GenAI-Fall stellt sich die Frage, ob die Texte direkt aus den Rohdaten übernommen oder erst einmal umgewandelt werden sollten. Oft ist es sinnvoll, die Inhalte sprachlich oder strukturell so anzupassen, dass sie besser zu typischen Nutzerfragen oder Suchanfragen passen. Beispielsweise können Tabellen in beschreibende Texte überführt oder redundante Informationen entfernt werden. Diese semantische Optimierung erhöht die Relevanz der späteren Antworten deutlich, da das Sprachmodell mit besser zugeschnittenem Kontext arbeitet. Die Transformation kann regelbasiert, modellgestützt oder manuell erfolgen und ist ein zentraler Hebel für die Qualität der gesamten RAG-Pipeline.

Im Anschluss daran werden die Chunks mit Metadaten angereichert. Dazu zählen Informationen wie zum Beispiel Titel, Quelle, Dokumententyp, Erstellungsdatum oder thematische Kategorien. Diese Metadaten sind essenziell für spätere Filterungen und Relevanzbewertungen. Sie ermöglichen es, Suchergebnisse gezielt einzugrenzen, etwa nach Abteilung,

● Listing 2: Erstellen von Chunks aus Markdown als PySpark Notebook

```
# Create chunks from markdown files
from langchain.text_splitter import
    MarkdownTextSplitter

def create_chunks(markdowns, filename, page_number=1):
    splitter = MarkdownTextSplitter(chunk_size = 100,
        chunk_overlap=0)
    documents = splitter.create_documents(markdowns)
    return [{
        "page_number": page_number,
        "page_content": doc.page_content,
        "filename": filename,
    } for doc in documents]

md_files = get_files(output_folder, "*.md")
for idx, md_file in enumerate(md_files):
    # create chunks overlapping with previous and next
    # page
    chunks = create_chunks([
        get_previous_page(md_files, md_file.filename),
        md_file.content,
        get_next_page(md_files, md_file.filename)
    ], md_file.filename, page_number=idx + 1)
    # save chunks with page as metadata
    write_files(chunk_output_folder, chunks)
```

bei, dass Chunks nicht willkürlich nach Seitenumbrüchen oder Zeichenlängen geschnitten werden, sondern entlang inhaltlicher Grenzen. Ein Abschnitt, der Text aus mehreren Kapiteln oder Themenbereichen vermischt, ist für semantische Suche meist wenig aussagekräftig. Ziel ist es, inhaltlich kohärente Einheiten zu schaffen, die auch über Seiten hinweg zusammengehören können. Chunks sollten eine ähnliche Länge haben, die sich an der maximalen Kontextlänge des verwendeten Sprachmodells orientiert. Aber auch leichte Überlappungen zwischen Chunks sind sinnvoll, um den richtigen Kontext zwischen Chunks zu erhalten. Die Zerlegung kann über Spark Notebooks oder Data Flows in Fabric erfolgen. Letztere bieten eine visuelle Oberfläche, mit der sich auch komplexe Textoperationen ohne Code realisieren lassen.

In der Gold-Stufe werden diese vorbereiteten Inhalte weiter optimiert. Hier werden die Daten in eine oder mehrere

Aktualität oder Dokumentenart. Auch für Governance und Auditing sind konsistente Metadatenstrukturen von zentraler Bedeutung. Dabei greifen Konzepte in Fabric wie Row Level Security oder auch Object Level Security, um genau zu definieren, welche Nutzer auf welche Daten Zugriff haben [17].

Für jeden Chunk wird nun ein semantischer Vektor berechnet, ein sogenanntes Embedding (siehe Listing 3). Dieser Vektor bildet die Bedeutung des Textes in einem mehrdimensionalen Raum ab und ist die Grundlage für spätere Ähnlichkeitsvergleiche.

In Fabric können Embeddings über verschiedene Wege erzeugt werden. Neben dem manuellen Einbinden von Azure OpenAI Service stehen auch AI Services in Fabric in der Public Preview zur Verfügung [18]. Dieses erlauben den Zugriff auf Azure AI Services wie Embedding, Text Analytics, Translator oder OpenAI direkt innerhalb von Fabric. Der große Vor-

teil: Die Nutzung dieser Modelle fällt in die gebuchte Fabric Capacity und verursacht keine zusätzlichen Kosten.

Die erzeugten Embeddings müssen anschließend gespeichert werden. Fabric bietet dafür mehrere Optionen. Im Lakehouse können Embeddings als Vektor-Spalten in Delta-Tabellen abgelegt werden. Diese Lösung ist besonders geeignet, wenn die Daten zusätzlich analytisch ausgewertet oder mit anderen strukturierten Informationen verknüpft werden sollen. Für performante semantische Suchen empfiehlt sich jedoch die Speicherung im Eventhouse oder in einer SQL-Datenbank mit Vektorunterstützung.

einheitlichen Sicherheitsmodell, zentralem Monitoring und konsistenter Governance. Drittens lassen sich Spark-Jobs, Pipelines, AI Services und Power BI direkt auf denselben Datenbestand anwenden, ohne Medienbrüche oder Synchronisationsprobleme. Für viele Unternehmen ist das ein entscheidender Vorteil, insbesondere wenn Datenschutz, Compliance und Skalierbarkeit eine Rolle spielen.

Eine spezialisierte Vektordatenbank ist sinnvoll, wenn maximale Performance, Skalierbarkeit und unabhängige Funktionen für Vektorsuchen benötigt werden, etwa bei großen Datenmengen oder besonders komplexen Suchanfragen, die

● Listing 3: Erstellen und Speichern der Embeddings als PySpark Notebook ins Eventhouse

```
# Create embeddings for each chunk
%pip install openai azure-kusto-data

from pyspark.sql.functions import udf, col
from pyspark.sql.types import ArrayType, FloatType
from openai import AzureOpenAI
import os

def get_embedding(text):
    client = AzureOpenAI(
        api_key=os.environ.get("AZURE_OPENAI_API_KEY"),
        azure_endpoint=os.environ.get(
            "AZURE_OPENAI_ENDPOINT"),
        azure_deployment=os.environ.get(
            "AZURE_OPENAI_DEPLOYMENT_ID"),
        api_version = "2024-02-01")

    response = client.embeddings.create(
        input=text,
        model= os.environ.get(
            "AZURE_OPENAI_EMBEDDING_MODEL",
            "text-embedding-ada-002"))

    return response.data[0].embedding

data = get_chunks(chunk_output_folder)
df_chunks = spark.createDataFrame(data, ["filename",
    "chunk_index", "text", "page_number"])

embedding_udf = udf(get_embedding,
    ArrayType(FloatType()))
df_embeddings = df_chunks.withColumn("embedding",
    embedding_udf(col("text")))

# Save embeddings to Eventhouse
df_embeddings
    .select("filename", "chunk_index", "text",
        "embedding", "page")
    .write
    .format(
        "com.microsoft.kusto.spark.synapse.datasources")
    ...
    .save()
```

Beide Varianten unterstützen Vektor-Felder nativ und ermöglichen Ähnlichkeitsabfragen [19] direkt über Kusto oder SQL zur semantischen Suche. Die Entscheidung für Lakehouse oder Eventhouse hängt vom Anwendungsfall ab: Das Lakehouse eignet sich für analytische Szenarien mit hohem Datenvolumen, das Eventhouse für schnelle, interaktive Suchen auf semistrukturierten Daten.

Ein häufiger Diskussionspunkt ist die Frage, ob man nicht besser eine spezialisierte Vektordatenbank wie Pinecone, Weaviate oder Qdrant verwenden sollte. Diese Systeme sind leistungsfähig und bieten zahlreiche Features für semantische Suche. Dennoch spricht vieles dafür, die Embeddings direkt in Fabric zu speichern. Erstens entfällt die Notwendigkeit, Daten zwischen Systemen zu bewegen, was sowohl die Komplexität als auch das Sicherheitsrisiko reduziert. Zweitens profitieren alle Komponenten von Fabric von einem

über die Standardfunktionen von Fabric hinausgehen. In solchen Fällen bieten spezialisierte Systeme oft optimierte Algorithmen, dedizierte Indizes und schnellere Suchzeiten.

Ein weiterer entscheidender Vorteil von Fabric ist aber die strategische Perspektive. Daten, die für GenAI aufbereitet werden, sind oft auch für andere Analysezwecke relevant, etwa für Reporting, Business Intelligence, Data Science oder klassische Machine-Learning-Modelle. Wenn die Embeddings und ihre Quellinformationen bereits in Fabric liegen, können sie ohne zusätzliche Infrastruktur für verschiedenste Anwendungsfälle genutzt werden. Fabric wird damit nicht nur zur Plattform für GenAI, sondern zur zentralen Datenbasis für das gesamte Unternehmen. Diese Wiederverwendbarkeit und Offenheit machen Fabric zu einer zukunftssicheren Alternative gegenüber spezialisierten, isolierten Vektordatenbanken. ►

Eine typische Daten-Pipeline in Fabric sieht dabei wie folgt aus: Eine Datei wird in OneLake geladen, zum Beispiel durch eine Synchronisation mit einem On-Premises-Datenspeicher, triggert automatisch eine Data Pipeline, welche über ein Spark Notebook oder einen Data Flow die Datei in Chunks zerlegt und mit Metadaten versieht. Anschließend werden Embeddings berechnet und im Eventhouse gespeichert. Von dort aus können sie direkt über Kusto semantisch durchsucht werden (siehe Bild 4). Dieser Prozess lässt sich vollständig innerhalb von Fabric orchestrieren. Alternativ können Daten auch im Lakehouse gesammelt und über Azure AI Search vektorisiert und indiziert werden, wenn eine hybride Sucharchitektur gewünscht ist.

Ein oft unterschätzter Aspekt ist die Aktualisierung der Embeddings. Wenn sich Inhalte ändern, müssen auch die zugehörigen Vektoren neu berechnet werden. Fabric bietet dafür

ideale Voraussetzungen. Über Data Pipelines lassen sich Änderungen erkennen, betroffene Chunks neu generieren und die Embeddings gezielt aktualisieren. So bleibt der Vektorindex konsistent und aktuell, ohne dass das gesamte System neu aufgebaut werden muss.

Für eine robuste Datenbasis empfiehlt es sich, einige Best Practices zu beachten: Die Chunk-Größe sollte konsistent gewählt werden, um eine gleichmäßige semantische Dichte zu gewährleisten. Zu große Chunks enthalten oft irrelevante Informationen, zu kleine verlieren den Kontext. Eine einheitliche Metadatenstruktur erleichtert nicht nur die Suche, sondern auch die Wartung und Erweiterung der Pipeline. Die Trennung von Text- und Vektorlogik sorgt dafür, dass Änderungen in einem Bereich nicht unbeabsichtigt andere Teile beeinflussen. Und schließlich sollten Pipelines modular aufgebaut sein, sodass einzelne Schritte wie Chunking, Embed-

● Listing 4: Einfacher Semantic Kernel Assistant mit RAG über Fabric Eventhouse

```
var openAIApiKey = Environment.GetEnvironmentVariable(
    "AZURE_OPENAI_API_KEY");
var openAIEndpoint =
    Environment.GetEnvironmentVariable(
        "AZURE_OPENAI_ENDPOINT");
var openAIDeployment =
    Environment.GetEnvironmentVariable(
        "AZURE_OPENAI_DEPLOYMENT");
var embeddingDeployment =
    Environment.GetEnvironmentVariable(
        "AZURE_OPENAI_EMBEDDING_DEPLOYMENT");

var kustoConnString =
    Environment.GetEnvironmentVariable(
        "KUSTO_CONNECTION_STRING");
var kustoDbName =
    Environment.GetEnvironmentVariable("KUSTO_DB_NAME");

var kernelBuilder = Kernel.CreateBuilder()
    .AddAzureOpenAIChatCompletion(
        deploymentName: openAIDeployment,
        apiKey: openAIApiKey,
        endpoint: openAIEndpoint
    )
    .AddAzureOpenAIEmbeddingGenerator(
        deploymentName: embeddingDeployment,
        apiKey: openAIApiKey,
        endpoint: openAIEndpoint);

var kernel = kernelBuilder.Build();
var embeddingGenerator =
    kernel.GetRequiredService<IEmbeddingGenerator>(
        string, Embedding<float>>>());
var chatCompletionService =
    kernel.GetRequiredService<IChatCompletionService>();

var memoryStore = new KustoMemoryStore(
    kustoConnString, kustoDbName);
var textMemory = new SemanticTextMemory(
    memoryStore, embeddingGenerator);

ChatHistory history = [];
history.AddSystemMessage("You are an expert in
    technical manuals. Only answer questions relevant
    to the technical manual context provided by the
    user.");

while (true)
{
    Console.WriteLine("Query: ");
    var prompt = Console.ReadLine();
    if (string.IsNullOrEmpty(prompt)) continue;

    // Retrieve top 5 most relevant manual sections
    var results = await textMemory.SearchAsync(
        "productInfoEmbeddings", prompt, limit: 5,
        minRelevanceScore: 0.79);
    var context = string.Join("\n--\n",
        results.Select(r => r.Metadata.Text));

    var userPrompt = prompt +
        (string.IsNullOrEmpty(context) ? "" :
        "\nContext:\n" + context);
    history.AddUserMessage(userPrompt);

    var response = await
        chatCompletionService.GetChatMessageContentAsync(
            history, kernel: kernel);
    Console.WriteLine($"Answer: {response.Content}\n");
}
```

ding oder Speicherung unabhängig voneinander angepasst oder wiederverwendet werden können. Diese Prinzipien erhöhen die Wartbarkeit, Skalierbarkeit und Transparenz der gesamten Architektur.

Wenn Daten sprechen lernen: Agenten, Workflows und KI-Logik mit Microsoft Fabric

Strukturierte, semantisch aufbereitete Daten sind die Grundlage jeder leistungsfähigen GenAI-Anwendung. Doch erst durch Agenten, Workflows und orchestrierte Logik entsteht daraus ein intelligentes System, das auf Nutzeranfragen reagieren, Entscheidungen treffen oder Prozesse anstoßen kann.

Die eigentliche Interaktion mit den Daten erfolgt in der Regel über spezialisierte Dienste, die sich an Fabric anbinden. Für Fachanwender stehen mit Microsoft Copilot Studio und den Fabric Data Agents intuitive Werkzeuge zur Verfügung,

gen stellen, etwa zu Verkaufszahlen, Produktvergleichen oder operativen Kennzahlen, und erhalten datengestützte Antworten, ohne SQL schreiben oder die Datenstruktur kennen zu müssen. Das setzt natürlich voraus, dass das Datenmodell sowie Tabellen und Spalten sinnvoll benannt wurden. Denn auch der schlaueste Agent kann nichts mit „Tabelle A“ und „Feld X1“ anfangen.

Data Agents senken die Einstiegshürde für datengetriebene Entscheidungen erheblich und machen Business Intelligence konversationsfähig. Sie ergänzen klassische Agentenlogik um eine direkte Verbindung zu analytischen Datenquellen innerhalb von Fabric und lassen sich nahtlos in bestehende Workflows integrieren.

Während Fabric Data Agents vor allem für direkte Abfragen in der Fabric-Umgebung gedacht sind, bietet Microsoft Copilot Studio [21] eine Low-Code-Umgebung, um eigene Chat-

The screenshot displays the Microsoft Fabric Eventhouse interface. On the left, a sidebar shows navigation options like Home, Workspaces, OneLake, Monitor, Real-Time, Workloads, GenAI, and SilverLakehouse. The main area is titled 'Vectors' and shows a KQL query being executed. The query is as follows:

```

1 -0.016585903, -0.0020781392, 0.014586184, -0.009273206, 0.009900568, -0.00030714634, -0.0041399407, 0.015480818, 0.0014793671, 0.00627036, -0.01832422,
2 -0.03419127, -0.005953411, -0.006681385, -0.030923756, -0.009090225, 0.0115147205, 0.0136451395, 0.00037780634, -0.02702888, 0.0034831704, 0.039471574,
3 -0.007907385, -0.0047607683, -0.016285291, -0.024793899, 0.04328803, -0.02266348, -0.020010257, -0.0040190425, -0.008593563]);
4 extend similarity = series_cosine_similarity(query, embedding)
5 project id, text, similarity*similarity*100
6 order by similarity desc
7 limit 10

```

Below the query, a table titled 'Table 1' shows the results of the query. The table has three columns: 'id', 'text', and 'similarity'. The results are as follows:

id	text	similarity
product_manual_page_15_chunk_471	/RS485 3 x USB3.0 1 x USB...	79.69329093800668
product_manual_page_15_chunk_181	evice according to the data ...	79.23698777091084
product_manual_page_15_chunk_483	ions Wiring diagrams Digita...	79.15047625665628
product_manual_page_15_chunk_452	thernet port COM (X30) * 1 ...	78.68131060096717
product_manual_page_15_chunk_348	es and function M.2 modul...	78.5279676020704
product_manual_page_15_chunk_550	Flash, 76 Front view, 10 G G...	78.06053455173362
product_manual_page_15_chunk_464	dule High current Per slot ...	78.0562441209742
product_manual_page_15_chunk_65	 8...	78.04031479937014
product_manual_page_15_chunk_472	escriptions 8.8.2 External Int...	78.0107310420925
product_manual_page_15_chunk_349	s 2242, 2260, 3042, 3052, 3...	77.55958494114398

Embedding und Ähnlichkeit zum User Prompt im Fabric Eventhouse (Bild 4)

um auf Daten in natürlicher Sprache zuzugreifen oder einfache Automatisierungen zu erstellen. Entwickler hingegen greifen auf Services und Frameworks wie Azure AI Foundry, Semantic Kernel oder das Model Context Protocol zurück, um komplexe Agenten mit modularer Logik, semantischer Suche und Tool-Integration zu realisieren. Allen Ansätzen gemeinsam ist, dass sie auf dieselbe Datenbasis in Fabric zugreifen, sei es über SQL-Endpunkte, Kusto-Abfragen, CosmosDB, GraphQL oder auch direkt auf Dateien über das Azure Blob Filesystem. Damit wird Fabric zur verbindenden Schicht zwischen Daten und Intelligenz.

Für Fachanwender ohne tiefes technisches Know-how bieten die Fabric Data Agents [20] einen vielversprechenden Einstieg. Diese Funktion befindet sich zum Zeitpunkt der Abfassung dieses Artikels in der Public Preview. Mit einem Data Agent lassen sich kleine Chatbots erstellen, die direkt auf Datenquellen wie Lakehouse oder Eventhouse zugreifen. Nutzerinnen und Nutzer können in natürlicher Sprache Fra-

bots und Automatisierungen zu erstellen. Diese ist zwar nicht Teil von Fabric, befindet sich aber in der Microsoft-365-Welt und kommt entweder über Microsoft-Copilot-Lizenzen mit oder kann via Pay-as-you-go genutzt werden. In Copilot Studio lassen sich Agents definieren, die auf natürliche Sprache reagieren, sowie Flows, die bestimmte Aktionen auslösen. Für den Zugriff auf Daten aus Microsoft Fabric stehen derzeit jedoch nur begrenzte Optionen zur Verfügung. Eine native Verbindung zu Lakehouses oder Eventhouse besteht aktuell nicht. Stattdessen gibt es zwei praktikable Wege, um Fabric-Daten dennoch in Copilot Studio nutzbar zu machen: Zum einen kann ein vorgelagerter Fabric Data Agent eingesetzt werden, der die Datenabfrage übernimmt und die Ergebnisse über einen Flow oder ein API an Copilot Studio zurückliefert. Zum anderen besteht die Möglichkeit, Fabric-Daten über Tools einzubinden, etwa über SQL-Endpunkte von Lakehouses oder als Knowledge Source über Azure AI Search, sofern die Daten aus Fabric indiziert und aufbereitet wurden. ►

Während Low-Code-Werkzeuge wie Copilot Studio und die Fabric Data Agents vor allem Fachanwendern den Zugang zu Unternehmensdaten erleichtern, benötigen komplexere GenAI-Szenarien oft deutlich mehr Flexibilität, Steuerbarkeit und technische Tiefe. Für Entwicklerteams, die maßgeschneiderte Agentenlogik, semantische Suche oder Modellorchestrierung umsetzen möchten, bietet das Microsoft-Ökosystem eine Reihe leistungsfähiger Tools und Frameworks, die sich direkt oder indirekt mit Fabric verbinden lassen. Den Anfang macht Azure AI Foundry [22], eine Plattform speziell für die Entwicklung und den Betrieb von GenAI-An-

Preview verfügbar ist. Zusätzlich gibt es in Azure AI Foundry dieselben Anbindungen wie in Copilot Studio, weil Letzteres auf Azure AI Foundry und seiner Funktionalität aufbaut.

Für Agenten und Workflows, die auf strukturierte Daten zugreifen sollen, kann zusätzlich noch das Model Context Protocol (MCP) eingesetzt werden [23]. MCP ermöglicht es, einem großen Sprachmodell strukturierte Informationen, Fähigkeiten und Werkzeuge bereitzustellen. Ein zentrales Merkmal von MCP ist die Definition sogenannter Tools, die von KI-Agenten genutzt werden können, um Aufgaben auszuführen. Diese Tools beschreiben beispielsweise, wie auf

● **Tabelle 1: Technologieübersicht für GenAI im Microsoft-Kosmos**

Technologie / Dienst	Funktion im GenAI-Kontext	Besonderheiten / Vorteile
Microsoft Fabric	Zentrale Plattform für Datenintegration, -speicherung, -verarbeitung und -analyse	Einheitliche Umgebung mit OneLake, Lakehouse, Eventhouse, Pipelines und Notebooks
OneLake	Tenant-weiter Datenspeicher für alle Fabric-Komponenten	Einheitliches Sicherheitsmodell, Data Sharing, Versionierung
Lakehouse	Speicherung und Analyse strukturierter und unstrukturierter Daten	Delta-Lake-Format, SQL- und Spark-Zugriff, ideal für ETL und Embedding-Vorbereitung
Eventhouse	Speicherung zeitbasierter, semistrukturierter Daten mit Vektorunterstützung	Native Vektorsuche mit KQL, ideal für semantische Suche in RAG-Workflows
Data Pipelines und Data Flows	Visuelle und codebasierte Datenintegration und -transformation	Automatisierung von Chunking, Metadatenanreicherung, Embedding-Erzeugung
Spark Notebook	Entwicklung und Ausführung von Datenvorverarbeitung, Chunking, Embedding-Erzeugung und semantischer Suche	Vollständig anpassbar, unterstützt Python und PySpark, ideal für komplexe Textverarbeitung, Tokenisierung und Vektorisierung. Besonders geeignet, wenn Data Flows oder Low-Code-Tools an ihre Grenzen stoßen
AI Services in Fabric	Zugriff auf Azure-AI-Modelle (OpenAI, Embedding, Translator, Text Analytics)	Nutzung innerhalb der Fabric Capacity, keine Zusatzkosten, direkt in Pipelines integrierbar
Azure AI Foundry	Plattform zur Entwicklung und Verwaltung von KI-Anwendungen	Modellorchestrierung, Agentenentwicklung, Teamkollaboration
Microsoft Copilot Studio	Low-Code-Umgebung für Chatbots und Workflows	Integration mit Microsoft 365, Verbindung von Agenten mit Daten und Aktionen
Semantic Kernel	Open-Source-Framework für Agentenentwicklung	Modularer Aufbau mit Skills, Prompts und Tool-Integration, ideal für wiederverwendbare Logik
CosmosDB in Microsoft Fabric	KI-fähige Datenbank mit Vektorindizierung, Volltext- und hybrider Suche	In Public Preview, tief in Fabric integriert, ideal für semantische Suche und KI-Anwendungen

wendungen. Sie richtet sich an Teams, die Sprachmodelle, Retrieval-Komponenten und Agenten modular kombinieren und orchestrieren wollen. Azure AI Foundry bietet eine Umgebung, in der sich Modelle von OpenAI, Anthropic und Co. integrieren, orchestrieren und überwachen lassen. Besonders für agentenbasierte Architekturen ist Azure AI Foundry interessant, da es die Verbindung zwischen Daten, Modellen und Geschäftslogik unterstützt. Entwickler können hier nicht nur Modelle feinjustieren oder einbinden, sondern auch Workflows definieren, die auf bestimmte Datenereignisse oder Nutzeranfragen reagieren. Fabric kann in Azure AI Foundry bereits als Knowledge Source eingebunden werden, auch wenn diese Funktion aktuell noch im Rahmen einer Public

Datenquellen zugegriffen wird oder wie Aktionen ausgelöst werden können. Ziel des Protokolls ist es, eine standardisierte Schnittstelle zu schaffen, über die Sprachmodelle mit externen Systemen interagieren können, etwa mit Datenbanken, APIs oder andere Anwendungen. Bislang existiert noch kein offizieller Fabric-MCP-Server, aber es gibt bereits Open-Source-Implementierungen, die sich in Eigenregie hosten lassen und Daten aus Fabric über MCP bereitstellen. So werden Fabric-Daten einfach durchsuchbar.

Für die Implementierung und Orchestrierung komplexer Agentenlogik bietet sich der Microsoft Semantic Kernel [24] an, ein Open-Source-Framework von Microsoft, das speziell für die Entwicklung von KI-Agenten mit modularer Logik,

Gedächtnis und Tool-Integration entwickelt wurde. Semantic Kernel ermöglicht es, sogenannte Skills zu definieren, die bestimmte Aufgaben übernehmen, etwa das Ausführen einer semantischen Suche im Eventhouse, das Extrahieren von Informationen oder das Auslösen eines Workflows. Diese Skills können mit Prompts, Funktionen oder externen APIs kombiniert werden. Ein Semantic Kernel Agent kann beispielsweise so konfiguriert werden, dass er bei einer Nutzeranfrage automatisch relevante Daten aus dem Eventhouse abrufen, diese mit einem Sprachmodell verarbeiten und anschließend eine strukturierte Antwort generiert oder eine Aktion auslöst (siehe [Listing 4](#)).

In unserem Beispiel beginnt die Entwicklung einer GenAI-Anwendung auf Basis von Microsoft Fabric mit einem Agenten, der mit dem Semantic Kernel entwickelt wurde. Dieser Agent wird als Chatbot in einer Webanwendung oder einem internen Portal bereitgestellt. Nutzerinnen und Nutzer können dort in natürlicher Sprache Fragen stellen, etwa zu Produkten, Prozessen oder internen Richtlinien.

Die Anfrage wird vom Agenten entgegengenommen und in eine semantische Suchanfrage umgewandelt. Diese wird an das Eventhouse übergeben, wo eine Vektorsuche nach den relevantesten Textabschnitten durchgeführt wird. Die gefundenen Inhalte werden als zusätzlicher Kontext in den Prompt des Sprachmodells eingebettet. Das Sprachmodell generiert auf dieser Grundlage eine Antwort, die nicht nur auf seinem Trainingswissen basiert, sondern auch auf aktuellen und unternehmensspezifischen Informationen. Die Antwort wird dem Nutzer direkt im Chat angezeigt.

Alle beteiligten Komponenten, von der Datenabfrage über die semantische Suche bis zur Antwortgenerierung, greifen auf dieselbe Datenbasis in Microsoft Fabric zu. Dadurch entsteht eine konsistente, sichere und nachvollziehbare Datenarchitektur, die sich flexibel erweitern und an unterschiedliche Anwendungsfälle anpassen lässt (vergleiche [Tabelle 1](#)).

Für den produktiven Einsatz solcher Agenten empfiehlt es sich, einige Prinzipien direkt in die Architektur zu integrieren. Die Trennung von Benutzeroberfläche, Modell und Datenzugriff sorgt für bessere Wartbarkeit und Flexibilität. Agentenlogik sollte modular aufgebaut sein, sodass sie in verschiedenen Kontexten wiederverwendet werden kann. Semantic Kernel unterstützt dies durch seine Plug-in-Architektur, bei der einzelne Skills unabhängig voneinander entwickelt und kombiniert werden können. Logging und Monitoring sind essenziell, um das Verhalten der Agenten zu analysieren und kontinuierlich zu verbessern. Feedbackschleifen, etwa durch Nutzerbewertungen oder automatische Auswertungen, helfen dabei, die Qualität der Antworten und die Relevanz der Datenbasis stetig zu optimieren. Auch die Wiederverwendbarkeit von Prompt-Vorlagen, Skill-Komponenten und Datenzugriffsmodulen trägt dazu bei, dass Agenten nicht nur schneller entwickelt, sondern auch langfristig gepflegt und erweitert werden können.

Fazit

Die Idee eines unternehmensweiten Assistenten, der Mitarbeiterinnen und Mitarbeitern auf natürliche Spracheingaben

hin präzise und kontextbezogene Antworten liefert, ist längst keine Zukunftsvision mehr. Mit Microsoft Fabric steht eine Plattform zur Verfügung, die alle notwendigen Bausteine vereint, um diese Vision Realität werden zu lassen. Dazu gehören die Datenspeicherung und -integration, die semantische Aufbereitung sowie die Integration in intelligente Agenten und Workflows.

Im Verlauf dieses Artikels wurde gezeigt, wie strukturierte, semistrukturierte und unstrukturierte Daten aus unterschiedlichen Quellen wie SQL-Tabellen, PDFs, HTML-Seiten oder Office-Dokumenten in OneLake zentralisiert und entlang einer Bronze-Silber-Gold-Architektur aufbereitet werden. Aus den Rohdaten entstehen durch semantisches Chunking, Metadatenanreicherung und Vektorisierung hochwertige Embeddings, die im Eventhouse gespeichert und für semantische Suchen genutzt werden können. Die Suche erfolgt über Agenten, die mit Semantic Kernel entwickelt wurden. Diese sind in der Lage, relevante Inhalte aus dem Eventhouse zu extrahieren, mit Sprachmodellen zu kombinieren und daraus fundierte Antworten zu generieren.

Trotz dieser Stärken ist Fabric keine Lösung, die ohne Weiteres sofort einsatzbereit ist. Die Plattform bietet viele Wege zum Ziel, aber nicht alle sind bereits vollständig ausgereift. Zwar lassen sich PDFs in Data Flows in Abschnitte zerlegen, doch die semantische Erkennung von inhaltlichen Einheiten ist dort noch begrenzt. In der Praxis greifen viele Teams daher auf Spark Notebooks mit Python zurück, um das Chunking präzise zu steuern, oder indizieren Dokumente, die in OneLake landen, direkt und einfacher mit Azure AI Search. Auch die Erzeugung von Embeddings erfolgt häufig über eigene Implementierungen mit Azure OpenAI oder Open-Source-Modellen, da die Integration in Fabric zwar möglich, aber noch nicht durchgängig verfügbar ist.

Ein weiteres Beispiel ist die Speicherung der Embeddings. Diese können im Lakehouse, im Eventhouse oder in Cosmos DB abgelegt werden. Die Anbindung an Frontends wie Semantic Kernel oder Copilot Studio erfordert jedoch häufig zusätzliche Logik. Es fehlen standardisierte Schnittstellen, um Vektorindizes direkt in Agenten zu nutzen oder semantische Suchen ohne Zwischenschritte zu orchestrieren. Viele dieser Komponenten befinden sich noch in der Public Preview. Das bedeutet, dass sich Funktionen, APIs und Stabilität noch verändern können und zudem meist kein Support für den Produktivbetrieb gegeben ist.

Auch das Thema Governance ist nicht vollständig gelöst. Wer Dokumente im Spark Notebook in Chunks zerlegt und Embeddings erzeugt, muss selbst sicherstellen, dass Berechtigungen aus den Quelldaten korrekt übernommen werden. Fabric bietet zwar Konzepte wie Row- und Object-Level-Security, doch diese müssen auch in den entsprechenden Zielformaten und -strukturen mit übernommen werden. Damit ein Nutzer nur die Embeddings sieht, auf deren Ursprungsdaten er Zugriff hat, muss die gesamte Pipeline im Nutzerkontext ausgeführt werden. Das ist technisch möglich, aber nicht trivial.

Ein weiterer zentraler Erfolgsfaktor liegt in der Qualität der Daten selbst und in der Art und Weise, wie sie aufbereitet werden. In diesem Artikel wurden viele Schritte im Code ►

vereinfacht dargestellt. In der Praxis entscheiden jedoch gerade die semantisch sinnvolle Zerlegung von Texten, die Auswahl geeigneter Chunk-Größen und die gezielte Anreicherung mit Metadaten über die Qualität der Antworten. Ein schlecht geschnittener Abschnitt oder fehlende Kontextinformationen können dazu führen, dass ein Sprachmodell irrelevante oder ungenaue Antworten liefert. Umgekehrt kann eine durchdachte Datenstruktur mit klaren Metadaten, konsistenten Formaten und inhaltlich kohärenten Abschnitten die Relevanz und Präzision der Ergebnisse erheblich steigern. Dieser Prozess ist selten beim ersten Versuch perfekt. Vielmehr bedarf es oft mehrerer Iterationen, Tests und Anpassungen, um die optimale Balance zwischen Kontexttiefe, Antwortqualität und Performance zu finden.

Trotz dieser Herausforderungen bietet Fabric einen entscheidenden strategischen Vorteil. Es schafft eine gemeinsame Dateninfrastruktur, die nicht nur für generative KI, sondern auch für klassische Business Intelligence, Data Science und Automatisierung sowohl von Fachanwendern als auch von Entwicklern genutzt werden kann. Daten müssen nicht zwischen Systemen bewegt werden, sondern stehen zentral zur Verfügung. Die Wiederverwendbarkeit von Daten, Pipelines und Sicherheitskonzepten reduziert Komplexität und erhöht die Konsistenz über alle Anwendungsfälle hinweg.

Fabric ist noch nicht perfekt, aber ein mächtiges Fundament. Wer bereit ist, in Architektur, Datenqualität und Governance zu investieren, erhält eine Plattform, die nicht nur einzelne Anwendungsfälle ermöglicht, sondern eine skalierbare und zukunftssichere Umgebung für datengetriebene Intelligenz schafft. Die Kombination aus technischer Tiefe, strategischer Offenheit und enger Integration in das Microsoft-Ökosystem macht Fabric zum zentralen Baustein einer unternehmensweiten Datenplattform, die nicht nur AI-, sondern jegliche Data-Analytics- und Reporting-Szenarien unterstützt. ■

- [1] Build advanced retrieval-augmented generation systems, <https://learn.microsoft.com/en-us/azure/developer/ai/advanced-retrieval-augmented-generation>
- [2] Microsoft Fabric, www.microsoft.com/de-de/microsoft-fabric
- [3] What is Microsoft Fabric?, <https://learn.microsoft.com/en-us/fabric/fundamentals/microsoft-fabric-overview>
- [4] OneLake, the OneDrive for data, <https://learn.microsoft.com/en-us/fabric/onelake/onelake-overview>
- [5] What is a lakehouse in Microsoft Fabric?, <https://learn.microsoft.com/en-us/fabric/data-engineering/lakehouse-overview>
- [6] Lakehouse and Delta Lake tables, <https://learn.microsoft.com/en-us/fabric/data-engineering/lakehouse-and-delta-tables>
- [7] What is Apache Spark compute in Microsoft Fabric?, <https://learn.microsoft.com/en-us/fabric/data-engineering/spark-compute>
- [8] Eventhouse overview, <https://learn.microsoft.com/en-us/fabric/real-time-intelligence/eventhouse>
- [9] Kusto: series_cosine_similarity(), <https://learn.microsoft.com/en-us/kusto/query/series-cosine-similarity-function>

- [10] Announcing Cosmos DB in Microsoft Fabric (Preview), <https://blog.fabric.microsoft.com/en-US/blog/22987/>
- [11] What's Azure AI Search?, <https://learn.microsoft.com/en-us/azure/search/search-what-is-azure-search>
- [12] Index data from OneLake files and shortcuts, <https://learn.microsoft.com/en-us/azure/search/search-how-to-index-onelake-files>
- [13] Plan your capacity size, <https://learn.microsoft.com/en-us/fabric/enterprise/plan-capacity>
- [14] Estimate your Fabric capacity needs, www.microsoft.com/en-us/microsoft-fabric/capacity-estimator
- [15] Understand medallion lakehouse architecture for Microsoft Fabric with OneLake, <https://learn.microsoft.com/en-us/fabric/onelake/onelake-medallion-lakehouse-architecture>
- [16] RAG chunking phase, <https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-chunking-phase>
- [17] Security in Microsoft Fabric, <https://learn.microsoft.com/en-us/fabric/security/security-overview>
- [18] AI services in Fabric (preview), <https://learn.microsoft.com/en-us/fabric/data-science/ai-services/ai-services-overview>
- [19] Azure Data Explorer for Vector Similarity Search, <https://techcommunity.microsoft.com/blog/azuredataexplorer/azure-data-explorer-for-vector-similarity-search/3819626>
- [20] Create a Fabric data agent (preview), <https://learn.microsoft.com/en-us/fabric/data-science/how-to-create-data-agent>
- [21] Copilot Studio overview, <https://learn.microsoft.com/en-us/microsoft-copilot-studio/fundamentals-what-is-copilot-studio>
- [22] What is Azure AI Foundry?, <https://learn.microsoft.com/en-us/azure/ai-foundry/what-is-azure-ai-foundry>
- [23] Build Agents using Model Context Protocol on Azure, <https://learn.microsoft.com/en-us/azure/developer/ai/intro-agents-mcp>
- [24] Introduction to Semantic Kernel, <https://learn.microsoft.com/en-us/semantic-kernel/overview/>



Florian Bader

ist Solution Architect bei Lunarix Digital Solutions und Microsoft MVP für Azure IoT. Mit langjähriger Erfahrung im Bereich DevOps, Cloud und Digitalisierung unterstützt er Unternehmen bei ihren digitalen Projekten.

florian.bader@lunarix.digital



Lars Roith

ist Solution Consultant bei Lunarix Digital Solutions und berät Unternehmen und Entwicklungsteams ganzheitlich bei der Umsetzung ihrer Softwareentwicklungsprojekte.

lars.roith@lunarix.digital