23. JANUAR 2026

KI PLATTFORM STRATEGIE

ONPREM AI STACK VS. MICROSOFT FOUNDRY

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

bwiiw
but what if IT works.

ZUSAMMENFASSUNG

Zwei strategische Zielarchitekturen für den produktiven Einsatz von Künstlicher Intelligenz auf Enterprise-Niveau: einen vollständig selbst betriebenen On-Prem / Private-Cloud-KI-Stack und eine plattformbasierte Architektur auf Basis von Azure AI Foundry. Ziel ist es, eine fundierte, langfristig tragfähige Entscheidung für den Betrieb von KI als strategische IT-Plattform zu ermöglichen. Es sind explizit zwei Technologiestacks benannt. Natürlich können auch einzelne Komponenten getauscht werden wie beispielsweise Rancher durch Openshift, vLLM durch Triton oder NeMo durch LlamaGuard. Alternativ kann auch ein souveränder Stack gewählt werden. Stackit AI Model Serving ist aktuell noch kein vollumfänglicher KI Stack. Er bedient zwar Inferenz, hat aber noch fehlende Komponenten wie Control Plane, RAG Control Plane, Data Ingest Pipelines, Policy Management etc.

Beide hier dargestellten Ansätze sind technisch leistungsfähig. Der entscheidende Unterschied liegt jedoch nicht primär in der Modellqualität oder Feature-Tiefe, sondern in der Frage, wie Verantwortung, Betrieb, Kosten und Governance organisatorisch verankert werden.

Ein On-Prem-Stack bedeutet maximale Kontrolle über Daten, Modelle, Inferenz und Laufzeit. Diese Kontrolle geht jedoch mit maximaler Verantwortung einher. Die Organisation übernimmt nicht nur den Betrieb der Hardware, sondern auch die dauerhafte Optimierung der Inferenz, das Lifecycle-Management von Modellen und Frameworks sowie die vollständige Sicherstellung von Stabilität, Auditfähigkeit und Skalierbarkeit. In der Praxis wird KI damit zu einem eigenen Plattformprodukt innerhalb der IT, das dedizierte Engineering- und Betriebsressourcen erfordert. On-Prem ist daher nur dann wirtschaftlich und organisatorisch sinnvoll, wenn ein stabiler, hoher Inferenzdurchsatz erwartet wird, entsprechende Plattformkompetenz dauerhaft verfügbar ist und Souveränitäts- oder regulatorische Anforderungen diesen Weg erzwingen.

Der Azure-AI-Foundry-Ansatz verlagert einen wesentlichen Teil der technischen Laufzeit- und Skalierungsverantwortung auf die Plattform. Inferenz, Basisskalierung und große Teile des Runtime-Betriebs werden standardisiert bereitgestellt. Dadurch reduziert sich die operative Komplexität erheblich. Die Rolle der IT verschiebt sich von Low-Level-Optimierung hin zu Governance, Use-Case-Steuerung, Kostenkontrolle und Qualitätsmanagement. Für viele Organisationen ist dies der realistischere Weg, da er

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

eine schnellere Wertschöpfung ermöglicht, das Betriebsrisiko reduziert und eine bessere betriebswirtschaftliche Steuerbarkeit von KI-Use-Cases erlaubt.

Ein zentrales Ergebnis des Dokuments ist, dass Inferenz der dominante Kosten- und Komplexitätstreiber ist. Im On-Prem-Betrieb ist Inferenz eine eigenständige Ingenieursdisziplin, die aktiv optimiert werden muss, um wirtschaftlich zu bleiben. In der plattformbasierten Architektur wird Inferenz zu einem standardisierten Service, dessen Kosten primär über Modellwahl, Limits und Budgets gesteuert werden. Dies erleichtert die Transparenz gegenüber Fachbereichen und Controlling erheblich.

Unabhängig vom gewählten Ansatz zeigt das Dokument, dass produktiver KI-Betrieb zwingend zentrale Plattformfunktionen erfordert: Model Routing, Sensitivity- und DLP-Kontrollen (Data Loss Prevention, Sicherstellen, dass personenbezogene, besonders schützenswerte oder vertrauliche Daten nicht in unzulässige KI-Kontexte gelangen), Guardrails, Agenten- und Tool-Steuerung, Observability sowie eine explizite Inference-FinOps-Disziplin. Diese Fähigkeiten sind keine optionalen Add-ons, sondern Voraussetzung für Skalierung, Nachvollziehbarkeit und verantwortbaren Betrieb.

Die Architekturentscheidung ist gleichzeitig eine Organisations- und Betriebsentscheidung. Sie legt fest, ob KI als eigenes, technisch betriebenes Plattformprodukt etabliert wird oder als gesteuerte, gemanagte Plattform mit klaren Governance- und Kostenmechaniken. Für die Mehrheit der Organisationen mit heterogenen Use Cases, schwankenden Lastprofilen und begrenzten Plattformressourcen (in Form von Mitarbeitenden und Infrastruktur) ist ein plattformbasierter Ansatz wie Azure AI Foundry der strategisch sinnvollere Einstieg und Skalierungspfad. Ein On-Prem-Stack sollte bewusst als Spezialfall betrachtet werden, der dort eingesetzt wird, wo Souveränität, regulatorische Anforderungen oder besondere Betriebsbedingungen dies zwingend erfordern.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

bwiiw
but what if IT works.

INHALTSVERZEICHNIS

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

ÜBERBLICK: PLATTFORMKOMPONENTEN UND ARCHITEKTUR-GEGENÜBERSTELLUNG

Das folgende Kapitel stellt alle zentralen Plattformkomponenten beider Architekturen gegenüber. Ziel ist eine konsolidierte Sicht für Architektur-, Betriebs- und Investitionsentscheidungen.

Architektur-baustein	On-Prem / Private Cloud	Azure AI Foundry
Physische Infrastruktur	Eigene GPU-Server, DC-Betrieb, Strom/Kühlung, Hardware-Lifecycle	Azure Compute (Managed), keine eigene Hardware
Orchestrierung	Kubernetes + Rancher, GPU Operator	Azure Managed Services, Plattformorchestrierung
Inferenz Runtime	vLLM (dediziert, selbst betrieben)	Managed Inference Endpoints
Modellkatalog	Eigenverantwortlich (lokal + extern)	Azure AI Foundry Model Catalog
UI / Zugriff	Open WebUI oder LibreChat	Foundry Control Plane + eigenes Frontend
AI Gateway	LiteLLM Proxy	Native Endpoint-/Routing-Mechaniken
Model Router	OPA + LiteLLM (Policy-as-Code)	Foundry Endpoint-Portfolio + Policies
API Credits	Zentral über LiteLLM	In Plattform integriert
RAG / Retrieval	Eigenbetrieb Vector DB + Storage	Azure AI Search + Azure Storage
MCP / Tooling	Dedizierte MCP-Server + Tool Gateway (PEP)	Foundry Agents + Tool Gateway (PEP)
Websearch	SearXNG + MCP Search Server	Managed Search-Tools + Policies
Guardrails	NeMo Guardrails + Guardrails AI	Plattform- + Policy-basierte Guardrails
Sensitivity / DLP	Presidio + Medical Layer + OPA	Managed PII + App-Layer DLP + Policies
Observability	OpenTelemetry + Prometheus + Grafana + Tracing	Azure Monitor + App Insights + Tracing
Inference-FinOps	Eigene Kostenmodelle + GPU-Sekunden	Native Cost Management + Token-Abrechnung
Budgetsteuerung	LiteLLM + OPA + Policies	Budgets/Quotas pro Endpoint
Auditfähigkeit	Eigenbau (Logs, Policies, Versionierung)	Plattformunterstützt + Policies
Release / Lifecycle	Eigenprozesse für Modell/Prompt/RAG	Plattform-gestützt (Evaluations, Canary)
Betriebs-verantwortung	Vollständig in der Organisation	Teilweise an Plattform ausgelagert

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

AUSGANGSLAGE

Organisationen stehen unter massivem Druck, KI produktiv einzusetzen. Gleichzeitig steigen die Anforderungen an Nachvollziehbarkeit, Risikosteuerung und verantwortlichen Betrieb von KI-Systemen. In der Praxis ergibt sich ein Spannungsfeld zwischen:

- Wunsch nach maximaler Daten- und Systemhoheit
- Realitäten moderner KI-Stacks (Modellzyklen, Inferenz-Frameworks, Toolchains)
- Erwartungshaltung der Fachbereiche an Geschwindigkeit und Qualität
- Ausreichende Personaldecke mit dem nötigen Knowhow

Die zentrale Frage lautet daher nicht ob KI eingesetzt wird, sondern wer dauerhaft die technische und regulatorische Systemverantwortung trägt.

ARCHITEKTURANSATZ A: ON-PREM FULL-STACK

Ein On-Premise- oder Private-Cloud-KI-Stack beschreibt den vollständigen Eigenbetrieb sämtlicher technischer, organisatorischer und prozessualer Komponenten, die notwendig sind, um moderne KI-Systeme produktiv, skalierbar und verantwortbar zu betreiben. Im Gegensatz zu klassischen Applikationsplattformen ist ein KI-Stack kein statisches System, sondern ein hochdynamisches Gefüge aus Modellen, Inferenzmechaniken, Datenflüssen und Steuerungslogiken.

Der Anspruch dieses Ansatzes ist maximale Kontrolle. Die Konsequenz ist maximale Verantwortung.

1. PHYSISCHE INFRASTRUKTUR UND RECHENZENTRUMSDESIGN

Die physische Infrastruktur bildet das Fundament des gesamten KI-Stacks. Moderne KI-Systeme sind ohne spezialisierte Hardware nicht wirtschaftlich betreibbar. CPUs spielen lediglich eine unterstützende Rolle für Steuerungs-, Vor- und Nachverarbeitungsschritte.

Im Zentrum stehen GPU-basierte Compute-Nodes. Die Wahl der GPU-Generation bestimmt nicht nur die Performance einzelner Inferenzanfragen, sondern die gesamte Kostenstruktur des Systems. Speicherbandbreite, VRAM-Größe, Interconnects und Energieverbrauch wirken sich direkt auf Durchsatz, Latenz und Kosten pro Token aus.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

Diese Entscheidung ist nicht kurzfristig reversibel. Inferenz-Frameworks, CUDA-Versionen, Treiber und Optimierungen sind eng an die Hardware gebunden. Ein GPU-Wechsel ist faktisch ein Plattformwechsel.

Zusätzlich erfordert der Betrieb leistungsfähiger GPUs angepasste Strom-, Kühl- und Rack-Konzepte. KI-Infrastruktur ist kein klassisches Rechenzentrumsthema, sondern ein Hochleistungsbetrieb mit entsprechenden Investitions- und Betriebsfolgen.

2. STORAGE- UND DATENINFRASTRUKTUR

KI-Systeme erzeugen und konsumieren große Mengen unterschiedlicher Datenarten. Modellartefakte, Checkpoints, Embeddings, Logs, Prompt-Historien und Agentenprotokolle haben jeweils unterschiedliche Zugriffs- und Latenzanforderungen.

In der Praxis entsteht eine mehrschichtige Storage-Architektur: Hochperformantes NVMe-Storage für aktive Modelle, skalierbarer Objekt-Storage für Artefakte und Vektordaten sowie separate Systeme für Logging und Monitoring.

Diese Heterogenität erhöht die Komplexität erheblich. Konsistenz, Backup, Wiederherstellung und Zugriffskontrollen müssen über alle Ebenen hinweg gewährleistet werden.

3. NETZWERKARCHITEKTUR

Netzwerke werden im KI-Kontext häufig unterschätzt. Verteilte Inferenz, Agentenarchitekturen und RAG-Pipelines erzeugen hochfrequente, latenzkritische Kommunikationsmuster zwischen Compute, Storage und Steuerungsebenen.

Unzureichende Netzwerkperformance führt unmittelbar zu erhöhten Antwortzeiten und sinkendem Durchsatz. Damit wird das Netzwerk zu einem direkten Kostentreiber. Dies kann auch im Labor bereits nachgestellt werden. Ein auf mehrere Nodes verteiltes Modell kann bei eingeschränkter Netzwerkperformance langsamer sein als ein Single-Node System, das aber begrenzt vertikal skaliert.

4. CONTAINER- UND ORCHESTRIERUNGSPLATTFORM

Um Modelle reproduzierbar und horizontal skalierbar bereitzustellen, ist eine Containerplattform zwingend erforderlich. Kubernetes-Distributionen haben sich als De-facto-Standard etabliert. In vielen Organisationen kommt dabei Rancher als zentrale Management- und Betriebsplattform für Kubernetes-Cluster zum Einsatz.

Der Betrieb von GPU-Workloads auf Kubernetes ist jedoch hochkomplex. GPU-Scheduling, Ressourcenzuteilung, Isolation konkurrierender Workloads und

Hochverfügbarkeit müssen explizit entworfen werden. Fehler in dieser Schicht wirken sich unmittelbar auf Kosten, Stabilität und Sicherheit aus.

Damit wird Kubernetes selbst zu einem kritischen Teil der KI-Architektur – und zu einem eigenständigen Risikofaktor.

In eigenen Tests sehen wir ca. 20-30% Verlust bei Containerisierung von Inferenzlayern (durch Netzwerkpfade, CPU Overhead, GPU Zugriff etc.).

5. INFERENZ-STACK UND PERFORMANCE-OPTIMIERUNG (VLLM-REFERENZSTACK)

Der Inferenz-Layer ist der zentrale wirtschaftliche Engpass des gesamten On-Prem-Stacks. In der Praxis ist es sinnvoll, sich bewusst auf einen primären Inferenz-Stack festzulegen, um Komplexität, Skill-Anforderungen und Betriebsschnittstellen zu reduzieren. Als Referenzarchitektur wird hier ein vLLM-basierter Inferenz-Stack beschrieben, weil vLLM für LLM-Inferenz in vielen Szenarien ein sehr gutes Verhältnis aus Performance, Funktionsumfang und operativer Integrationsfähigkeit bietet.

Der vLLM-Stack ist nicht „ein Container mit einem Modell", sondern eine eigenständige Laufzeitplattform, deren Parameter, Caching-Strategien und Skalierungslogik den Großteil der Kosten- und Qualitätscharakteristik bestimmen.

5.1 VLLM SERVING LAYER

vLLM wird typischerweise als dedizierter Service pro Modell (oder pro Modellfamilie) betrieben. Der Dienst stellt standardisierte Schnittstellen (z. B. OpenAI-kompatible APIs) bereit, kapselt die Modellgewichte und implementiert Inferenz-spezifische Optimierungen.

Entscheidend ist, dass der Betrieb nicht nur aus „Starten des Services" besteht, sondern aus der kontinuierlichen Pflege einer stabilen Serving-Konfiguration:

- Wahl und Begrenzung der Kontextlänge
- Konfiguration von Parallelisierung und Batching
- Steuerung von Timeouts, Retries und Backpressure
- Definition von Request- und Token-Limits pro Mandant, Anwendung oder User

Diese Parameter sind nicht kosmetisch. Sie entscheiden darüber, ob ein Modell im Alltag stabil antwortet oder ob Lastspitzen zu Queue-Build-up, erhöhten Latenzen oder Out-of-Memory-Situationen führen.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

5.2 PERFORMANCE-MECHANIKEN

Ein vLLM-Stack wird wirtschaftlich erst durch konsequente Nutzung der Inferenz-Mechaniken:

- Continuous Batching: Inferenzanfragen werden so zusammengeführt, dass GPU-Auslastung steigt und Leerlauf sinkt. Ohne Batching steigen Kosten pro Antwort drastisch.
- KV-Cache-Strategie: Der Key-Value-Cache reduziert Rechenarbeit bei langen Konversationen, ist aber VRAM-intensiv. Cache-Größe und -Policy sind direkte Kostenhebel.
- Prefill vs. Decode Optimierung: Prompt-Verarbeitung (Prefill) und Token-Generierung (Decode) skalieren unterschiedlich. Im Betrieb ist das Nutzungsprofil entscheidend: viele kurze Prompts verhalten sich anders als wenige lange.
- Quantisierung: Reduziert Speicherbedarf und kann Throughput erhöhen, verändert aber teils Qualität und muss pro Modell validiert werden.

Die wichtigste Konsequenz für Architekturentscheidungen ist: Modellgröße und Inferenzkosten korrelieren nicht linear. Kontextlänge, Prompt-Länge, Batching-Effizienz und Cache-Verhalten beeinflussen die realen Kosten oft stärker als die reine Parameterzahl.

5.3 SKALIERUNG UND BETRIEBSMODI

On-Prem-Skalierung ist grundsätzlich begrenzt und muss explizit entworfen werden. Ein praxistauglicher vLLM-Betrieb unterscheidet typischerweise:

- Horizontale Skalierung: Mehrere vLLM-Instanzen hinter einem Load Balancer. Dies erhöht Verfügbarkeit und Durchsatz, erfordert aber saubere Session-/State-Strategien und konsistentes Routing.
- Vertikale Skalierung: Größere GPU-Nodes oder Multi-GPU-Setups. Dies ist teuer und erhöht Pfadabhängigkeiten.
- Isolation nach Kritikalität: Trennung von „kritischen“ Produktionsanwendungen und „explorativen“ Workloads, um Stabilität zu schützen.

Damit wird der Inferenz-Layer zu einem eigenen „Service-Portfolio“ innerhalb der IT: Es entstehen Betriebsstufen, SLAs, Prioritäten und Kapazitätsmodelle.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

5.4 OBSERVABILITY, MONITORING UND FINOPS FÜR INFERENZ

Ohne tiefes Monitoring ist ein vLLM-Inferenzbetrieb nicht steuerbar. Klassisches Infrastrukturmonitoring (CPU/RAM) reicht nicht aus. Benötigt wird KI-spezifische Observability auf drei Ebenen: Service und Requestebene (Requestrate, Fehlerrate, Timeouts, Queue-Längen, etc.) Token und Kostenebene (Tokens per Request, pro Minute/Anwendung/Mandant) und Hardware- und Runtime Ebene.

Diese Metriken sind nicht nur zur Fehleranalyse nötig, sondern zur Governance und Kostensteuerung: Ein On-Prem-Stack benötigt ein internes Äquivalent zu Cloud-FinOps, weil sonst Kosten nicht mehr erklärbar sind.

Zusätzlich ist Tracing entscheidend, sobald RAG und Agenten ins Spiel kommen. Inferenz ist dann nur ein Teil einer Kette aus Retrieval, Tool-Aufrufen und Post-Processing. Ohne Ende-zu-Ende-Tracing lassen sich Latenzursachen und Qualitätsprobleme nicht sauber zuordnen.

5.5 LOGGING, AUDITFÄHIGKEIT UND BETRIEBSNACHWEISE

Ein produktiver Inferenzbetrieb benötigt kontrollierte Logs, die technische Analyse ermöglichen, ohne unkontrollierte Datenschatten zu erzeugen. Dazu gehören:

- strukturierte Request-Logs (ohne oder mit kontrollierter Payload)
- Modell- und Konfigurationsversionen als Pflichtmetadaten
- Prompt- und Policy-Versionen als Pflichtmetadaten
- Ereignislogs für Rate Limits, Blockierungen und Guardrail-Entscheidungen

Ohne diese Disziplin ist ein späterer Nachweis, warum ein System zu einem bestimmten Zeitpunkt ein bestimmtes Verhalten gezeigt hat, faktisch nicht möglich.

6. MODELL-, PROMPT- UND LIFECYCLE-MANAGEMENT

Im On-Prem-Ansatz trägt die Organisation die vollständige Verantwortung für den gesamten Modell-Lifecycle. Dies umfasst Auswahl, Versionierung, Validierung, Freigabe, Betrieb und Rücknahme von Modellen.

Prompts sind dabei kein triviales Konfigurationselement. Sie definieren Verhalten, Tonalität, Entscheidungslogik und Risikoprofile eines Systems. Änderungen an Prompts können funktional relevanter sein als Änderungen am Modell selbst.

Ohne strukturierte Prozesse für Prompt-Versionierung, Tests und Rollbacks ist ein kontrollierter Betrieb nicht möglich.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

7. MCP, AGENTEN UND STEUERUNGSLOGIK

Model Context Protocols (MCP) und Agentenarchitekturen existieren im On-Prem-Umfeld nicht als fertige Standardprodukte. Für einen produktiven Betrieb ist daher eine dedizierte MCP-Server-Umgebung zu spezifizieren, die Tool-Zugriffe standardisiert, kontrolliert und auditierbar macht.

Festlegung: MCP als Standard-Tool-Bus

- Pro Tool/Datenquelle wird ein eigener MCP-Server betrieben (z. B. „Search", „Knowledge", „Tickets", „DB Read", „DB Write").
- Alle MCP-Server liegen in einer separaten „Tool-Zone" und werden über einen zentralen Policy Enforcement Point abgesichert.

Festlegung: Tool Gateway / Policy Enforcement Point (PEP)

- Zwischen Agent und MCP-Servern steht ein Gateway, das:
  - Whitelists pro Agent-Rolle erzwingt
  - Parameter per JSON-Schema validiert
  - Rate Limits/Quotas pro Agent, Anwendung und User anwendet
  - jeden Tool Call vollständig protokolliert (Request-Metadaten, Ergebnisstatus)
  - optional Human-in-the-Loop für kritische Aktionen erzwingt

Damit wird Agentensteuerung nicht als „Code in einem Use Case" implementiert, sondern als wiederverwendbarer Plattformstandard.

8. UI, ZUGRIFFSSCHICHT UND MANDANTENFÄHIGKEIT (FESTGELEGT)

Ein On-Prem-Stack benötigt eine standardisierte UI, um Nutzung zu bündeln, Rollen sauber zu trennen und zentrale Governance umzusetzen.

Open WebUI oder LibreChat beispielsweise kann als Standard-Frontend dienen

- Open WebUI / LibreChat dient als primäre Nutzeroberfläche für Chat, RAG und (optional) Workspaces.
- Die UI spricht OpenAI-kompatible APIs.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Authentifizierung wird über das Unternehmens-Identity-System angebunden (SSO-Pattern), die feingranulare Zugriffskontrolle erfolgt über nachgelagerte Policies.

Diese Festlegung ist bewusst: Sie reduziert Shadow-Tools, bündelt Telemetrie und vereinfacht die Einführung in Fachbereiche.

9. API-CREDITS, MULTI-PROVIDER-ROUTING UND KOSTENKONTROLLE (FESTGELEGT)

Ein spezifizierter On-Prem-Stack muss externe Modellanbieter (API Credits) kontrolliert einbinden können – ohne dass jede Anwendung eigene Keys verwaltet.

LiteLLM Proxy als AI Gateway

- LiteLLM übernimmt als zentrales Gateway:
  - Routing zwischen lokalen vLLM-Endpunkten und externen Providern (z. B. Azure OpenAI, OpenAI, Anthropic)
  - zentrale Key-Verwaltung und Rotation
  - Kosten- und Nutzungs-Telemetrie pro Team/Use Case
  - Load Balancing und Fallback-Regeln

Betriebslogik

- Standardpfad ist „local-first“ (vLLM). Externe Credits werden gezielt genutzt für:
  - besonders hochwertige Aufgaben
  - temporäre Peaks
  - spezielle Modellfähigkeiten
- Entscheidung erfolgt über Policies (Use Case/Endpoint) statt über Ad-hoc-Entwicklerentscheidungen.

10. MODEL ROUTER, POLICY DECISION UND PROVIDER-AUSWAHL

Der Model Router ist eine eigenständige Plattformkomponente, die entscheidet, ob eine Anfrage an ein lokales vLLM-Modell oder an ein externes API-basiertes Modell (z. B. GPT-5.2) geroutet wird. Ziel ist eine zentrale, nachvollziehbare und auditierbare Steuerung von Qualität, Kosten, Sensitivität und Verfügbarkeit.

Der Router ist kein Client-Feature, sondern ein Plattformservice. Applikationen dürfen nicht direkt entscheiden, welches Modell sie nutzen.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

Open Policy Agent (OPA) fungiert als zentrale Entscheidungsinstanz. Routing-Entscheidungen werden als Policy-as-Code implementiert und versioniert.

OPA erhält strukturierte Metadaten je Anfrage:

- data_sensitivity (public | internal | confidential | highly_confidential)
- usecase_class (chat | summarization | extraction | coding | decision_support)
- quality_tier (standard | high)
- context_length_bucket (<8k | 8–32k | >32k)
- budget_class (low | standard | premium)
- need_websearch (true/false)
- need_tools (true/false)
- tenant / business_unit / user_role
- runtime_signals (local_capacity_ok, api_quota_remaining, provider_health)

OPA liefert eine deterministische Entscheidung:

- route = local-vllm
- route = api-gpt52
- route = fallback-policy

Beispielhafte, verbindliche Referenzregeln:

- Wenn data_sensitivity = highly_confidential → zwingend local-vllm
- Wenn quality_tier = high UND data_sensitivity erlaubt extern → api-gpt52
- Wenn context_length_bucket > lokales Limit → api-gpt52
- Wenn budget_class = low → local-vllm
- Wenn api_quota_exhausted = true → local-vllm
- Wenn local_capacity_ok = false → api-gpt52 (sofern erlaubt)

Diese Regeln sind nicht im Code, sondern ausschließlich in OPA-Policies definiert.

LiteLLM setzt die von OPA getroffene Routing-Entscheidung technisch um:

- Auswahl des konkreten Providers
- Übersetzung auf OpenAI-kompatible APIs

- Load Balancing zwischen mehreren vLLM-Instanzen
- Fallback auf definierte Provider bei Fehlern

LiteLLM stellt zusätzlich sicher:

- zentrale API-Key-Verwaltung
- Quotas und Budgets pro Team/Use Case
- konsistente Telemetrie unabhängig vom Provider

Auditfähigkeit und Nachvollziehbarkeit

Für jede Anfrage werden folgende Metadaten protokolliert:

- gewählter Provider (local-vllm / api-gpt52)
- auslösende Policy-Regel
- relevante Routing-Attribute (Sensitivity, Quality, Budget)
- Fallback-Ereignisse

Damit ist jederzeit nachvollziehbar, warum ein bestimmtes Modell genutzt wurde.

Fallback- und Degradationsstrategien

Der Router implementiert explizite Degradationspfade:

- API-Provider nicht verfügbar → Fallback auf local-vllm (wenn erlaubt)
- lokale Kapazität erschöpft → API-Provider (wenn erlaubt)
- Budget erschöpft → erzwungener local-vllm-Pfad oder Blockierung

Diese Mechaniken sind Teil der Betriebsarchitektur und werden nicht ad hoc im Anwendungscode umgesetzt.

11. WEBSEARCH UND EXTERNE WISSENSANBINDUNG (FESTGELEGT)

Für produktive Assistenten ist Websearch ein eigener Capability-Baustein, der kontrolliert und auditierbar angeboten werden muss.

SearXNG als interne Search-Engine

- SearXNG wird intern betrieben und aggregiert Suchresultate aus mehreren Suchdiensten.
- Zugriff erfolgt ausschließlich über einen dedizierten MCP-Search-Server.

Policy-Festlegungen

- Domain Allow-/Deny-Lists (z. B. nur definierte Quellen für bestimmte Use Cases)
- Rate Limits pro User/Agent
- Ergebnis-Persistenz und Retention-Regeln
- Vollständiges Audit der Suchabfragen (ohne unnötige Inhaltskopien) und externe Wissensanbindung (festgelegt)

Für produktive Assistenten ist Websearch ein eigener Capability-Baustein, der kontrolliert und auditierbar angeboten werden muss.

Festlegung: SearXNG als interne Search-Engine

- SearXNG wird intern betrieben und aggregiert Suchresultate aus mehreren Suchdiensten.
- Zugriff erfolgt ausschließlich über einen dedizierten MCP-Search-Server.

Policy-Festlegungen

- Domain Allow-/Deny-Lists (z. B. nur definierte Quellen für bestimmte Use Cases)
- Rate Limits pro User/Agent
- Ergebnis-Persistenz und Retention-Regeln
- Vollständiges Audit der Suchabfragen (ohne unnötige Inhaltskopien)

11. GUARDRAILS, SAFETY UND POLICY-STEUERUNG (FESTGELEGT)

Guardrails müssen als zentrale Plattformfunktion umgesetzt werden – nicht als verstreute Prompt-Regeln.

NeMo Guardrails als Policy-Layer zwischen App und LLM

- NeMo Guardrails wird als programmierbarer Guardrail-Layer vor die Modellaufrufe geschaltet.
- Anwendungsfälle:
  - Prompt-Injection-Erkennung und -Abwehr
  - Topic/Intent-Control
  - Output-Moderation und Redaction-Patterns
  - Eskalationslogiken (z. B. human approval)

Festlegung: Guardrails AI für strukturierte Outputs und Validierung

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Guardrails AI wird eingesetzt, wenn Outputs verlässlich strukturiert sein müssen (z. B. JSON für Workflows) und wenn Validierungen mit Re-Ask-Mechanik benötigt werden.
- Ergänzend können spezifische Validatoren (z. B. PII) eingebunden werden.

Damit entstehen zwei klar getrennte Guardrail-Dimensionen:

- Policy/Safety-Steuerung (NeMo)
- Output-Verlässlichkeit/Validierung (Guardrails AI)

12. SENSITIVITY & DLP GUARD (VOLL SPEZIFIZIERT)

Der Sensitivity & DLP (Data Loss Prevention) Guard ist eine eigenständige Plattformkomponente, deren Zweck es ist, die Verarbeitung personenbezogener, sensitiver und beispielsweise medizinischer Informationen technisch durchzusetzen. Er ist kein optionales Feature, sondern ein verpflichtender Kontrollpunkt im gesamten Anfrage- und Antwortfluss.

Zweck und Architekturrolle

Der DLP Guard erfüllt vier zentrale Funktionen:

- Klassifikation der Sensitivität von Eingaben und Ausgaben
- Erkennung personenbezogener Daten (PII)
- Erkennung medizinischer Inhalte im Kontext personenbezogener Daten (PHI-ähnliche Muster)
- Erzwingung von Allow-, Redact- oder Block-Entscheidungen auf Basis zentraler Policies

Der DLP Guard ist sowohl vor der Inferenz (Input) als auch nach der Inferenz (Output) platziert. Zusätzlich ist er vor Logging-, Tracing- und Persistenzkomponenten geschaltet, um sicherzustellen, dass keine sensitiven Inhalte unbeabsichtigt in Betriebsartefakten landen.

Platzierung im Referenz-Flow:

Open WebUI / Applikation → DLP Guard (Input) → Model Router → vLLM / API → DLP Guard (Output) → UI

Parallel dazu:

DLP Guard → Logging/Tracing (nur Metadaten, keine Rohinhalte)

Damit wird der DLP Guard zu einem verpflichtenden Gatekeeper für alle Inhalte.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

PII (PERSONALLY IDENTIFIABLE INFORMATION)-ERKENNUNG MIT PRESIDIO

Microsoft Presidio (Open-Source Framework zur Erkennung und Anonymisierung von personenbezogenen Daten in Texten und Bildern) wird als primäre PII-Detection- und Anonymization-Engine eingesetzt.

- Presidio Analyzer für Standard-PII (Name, E-Mail, Telefonnummer, Adresse, IBAN, etc.)
- Custom Recognizers für:
  - organisationsspezifische Identifikatoren
  - Kundennummern
  - interne Personalnummern
  - projektspezifische Kennungen
- Sprachmodelle und NLP-Pipelines für Deutsch

Confidence Management

- Mindest-Confidence-Schwellen je Entity-Typ
- Unterschiedliche Schwellen für Input vs. Output
- False-Positive-Management durch Whitelists und Pattern-Ausnahmen

Presidio liefert strukturierte Entity-Listen inklusive Typ, Position und Confidence-Score.

MEDICAL CONTEXT DETECTION

Zusätzlich zur PII-Erkennung wird ein Medical-Context-Layer betrieben.

Nicht jede medizinische Information ist per se verboten. Kritisch wird sie, wenn sie im Zusammenhang mit identifizierbaren Personen steht.

Spezifikation Medical Layer

- Regelbasierte Erkennung medizinischer Begriffe (z. B. Diagnosen, Medikamente, Prozeduren)
- Ontologie-Listen (z. B. ICD, ATC, SNOMED-Synonyme)
- Kontext-Dichte-Scoring (Anteil medizinischer Begriffe im Text)

Der Medical Layer liefert strukturierte Signale:

- has_medical_terms (true/false)
- medical_term_density

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- medical_categories

POLICY DECISION MIT OPA

Open Policy Agent (OPA) fungiert als zentrale Entscheidungsinstanz für Sensitivity-Policies.

OPA erhält strukturierte Features:

- has_pii (true/false)
- pii_types
- pii_confidence
- has_medical_terms
- medical_term_density
- direction (input/output)
- usecase_class
- tenant / business_unit / user_role

OPA trifft eine deterministische Entscheidung:

- action = allow
- action = redact
- action = block

HARTE REFERENZREGELN

Verbindliche Plattformregeln:

- Wenn has_pii = true → mindestens redact
- Wenn has_pii = true UND has_medical_terms = true → block
- Wenn medical_term_density > definierter Schwelle UND identifizierende Merkmale vorhanden → block
- Wenn direction = output UND block → standardisierte Fehlermeldung, keine Rohdaten

Diese Regeln sind Policy-as-Code und dürfen nicht in Applikationslogik implementiert werden.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

REDACTION UND MASKIERUNG

Wenn action = redact:

- Presidio Anonymizer maskiert erkannte Entitäten
- Maskierungsregeln sind je Entity-Typ konfigurierbar (z. B. vollständige Entfernung vs. Platzhalter)
- Redaction wird als Teil der Antwort markiert (Transparenz gegenüber dem Nutzer, optional)

LOGGING, AUDIT UND DATENSCHUTZ

Der DLP Guard erzwingt eine strikte Trennung zwischen Inhaltsverarbeitung und Betriebsartefakten.

Pflichtvorgaben

- Keine Rohtexte in zentralen Logs
- Logging nur von:
  - Entity-Typen
  - Anzahl erkannter Entitäten
  - getroffene Policy-Entscheidung
  - Rule-ID der auslösenden Policy

Der Betrieb des DLP Guards berücksichtigt explizit:

- False Positives durch zu aggressive Erkennung
- Fachlich legitime Sonderfälle

Referenzprozesse

- Review-Workflow für False Positives
- Anpassung von Recognizern und Confidence-Schwellen
- Policy-Overrides nur über genehmigte Ausnahmeprozesse

Damit wird der DLP Guard zu einer kontinuierlich gepflegten Plattformkomponente und nicht zu einer einmalig konfigurierten Regelmaschine.

Spätestens an dieser Stelle wird deutlich: Der On-Prem-KI-Stack ist kein Infrastrukturprojekt, sondern die Entwicklung und der Betrieb eines hochkomplexen Softwaresystems.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

13. OBSERVABILITY & INFERENCE-FINOPS

Observability und Inference-FinOps sind eigenständige Plattformfähigkeiten. Ziel ist nicht nur technische Überwachung, sondern die kontinuierliche betriebswirtschaftliche Steuerung von Latenz, Qualität und Kosten pro Use Case.

13.1 TELEMETRIE-ARCHITEKTUR

Referenz-Toolchain (On-Prem)

- OpenTelemetry Collector als standardisierte Telemetrie-Einsammelstelle
- Prometheus für Metriken
- Grafana für Dashboards
- Loki für strukturierte Logs
- Tempo/Jaeger für Distributed Tracing

Alle Plattformkomponenten (Open WebUI, LiteLLM, vLLM, MCP-Server, DLP Guard, Tool Gateway) instrumentieren OpenTelemetry nativ. Correlation IDs werden am UI erzeugt und entlang der gesamten Kette propagiert.

13.2 METRIKMODELL (VERBINDLICH)

Service-Ebene

- Requests per Second (RPS)
- Error Rate
- Timeout Rate
- p50/p95/p99 Latenz (getrennt für Prefill und Decode)
- Queue Length / Backpressure-Indikatoren

Inferenz-Ebene

- Tokens/s pro vLLM-Service
- Tokens pro Request (Prompt vs. Completion)
- KV-Cache Utilization
- Prefill/Decode Ratio

GPU-Ebene

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- GPU Utilization
- VRAM Usage
- Memory Fragmentation Proxy
- Power Draw und Thermal Throttling Events

RAG- und Agent-Ebene

- Retrieval Hit Rate
- Re-Ranking Anteil
- Quellenverteilung
- Tool Calls pro Anfrage
- Tool Error Rate

13.3 STANDARD-DASHBOARDS

Verbindliche Referenz-Dashboards:

- Platform Health Dashboard (gesamt)
- Inference Efficiency Dashboard (Tokens/s, GPU-Sekunden)
- Cost per Use Case Dashboard
- Long-Context Usage Dashboard
- Agent Tooling Dashboard

Diese Dashboards sind Teil der Plattformabnahme und kein optionales Add-on.

13.4 END-TO-END TRACING

Tracing umfasst mindestens: UI → LiteLLM → OPA → vLLM/API → MCP → RAG → LLM → DLP Guard → UI

Damit lassen sich Latenzursachen eindeutig zuordnen (z. B. Retrieval vs. Inferenz vs. Tool Calls).

13.5 KOSTENMODELL UND ALLOKATION

Referenz-Kostenmodell

- GPU-Sekunden pro 1k Tokens
- Kosten pro Anfrage
- Kosten pro Use Case

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Kosten pro Business Unit / Mandant

Kosten werden über LiteLLM, vLLM-Telemetrie und OPA-Routing-Attribute korreliert.

Budgetsteuerung und Quotas

Verbindliche Mechaniken:

- Budgets pro Use Case
- Quotas pro Endpoint
- Rate Limits pro Nutzer/Team
- Eskalationsschwellen bei Budgetüberschreitung

Budgetverletzungen können folgende automatische Aktionen auslösen:

- Downgrade auf kleineres Modell
- Erzwingen local-vLLM
- Blockieren nicht-kritischer Anfragen

Inference-FinOps als Betriebsdisziplin

Inference-FinOps ist eine eigene Betriebsrolle.

Aufgaben:

- Optimierung des Modellportfolios
- Reduktion unnötig großer Kontextlängen
- Erhöhung der Batching-Effizienz
- Identifikation teurer Use Cases

Ziel ist die kontinuierliche Senkung der Kosten pro fachlichem Outcome.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

ARCHITEKTURANSATZ B: MICROSOFT AI FOUNDRY STACK

Der Azure-AI-Foundry-Stack ist kein einzelner Dienst, sondern eine Kombination aus Control-Plane, Managed Inference und Governance sowie den angrenzenden Plattformservices für Identität, Netzwerk, Logging und Datenanbindung. Der zentrale Architekturgedanke ist, dass Inferenz, Modellbereitstellung und agentische Orchestrierung als Plattformfähigkeiten verstanden werden – mit klaren, überprüfbaren , gemanagten Kontrollen.

1. NETZWERK- UND SICHERHEITSGRUNDLAGE

Alle KI-Endpunkte und Datenzugriffe erfolgen privat und kontrolliert, ohne öffentliche Exposition, und mit eindeutigen Verantwortlichkeiten.

Konkrete Bausteine

- Azure Virtual Network (VNet) als Netzwerkdomäne für KI-Workloads
- Private Endpoints / Private Link für Dienstzugriffe (Storage, AI Search, Key Vault etc.)
- ExpressRoute (oder Site-to-Site VPN) für Anbindung aus dem Unternehmensnetz
- Network Security Groups (NSG) und optional Azure Firewall für Egress-/Ingress-Kontrolle
- Zentrale DNS-Strategie (Private DNS Zones) für Private Link Auflösung

KI-Workloads sind daten- und logintensiv. Ohne Private Link entstehen unklare Datenpfade und potenzielle Schatten-Exposition. Eine private Netzarchitektur ist die Voraussetzung für belastbare Zugriffskontrolle und konsistente Nachvollziehbarkeit.

2. IDENTITÄT, ZUGRIFF UND SECRETS

- Microsoft Entra ID als Identity Provider
- Managed Identities für Dienst-zu-Dienst-Zugriffe (ohne statische Secrets)
- Azure Key Vault für Secrets, Zertifikate und Schlüsselmaterial (**Alternativ beispielsweise HashiCorp Vault onPrem, bei Key Vaults immer über onPrem nachdenken!**)
- RBAC auf Subscription/Resource Group/Resource Ebene sowie feingranular in den Diensten

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

KI-Systeme sind nicht nur Anwendungen, sondern Plattformen, die viele interne Tools und Datenquellen orchestrieren. Dadurch steigt die Relevanz von Identität als primärem Kontrollmechanismus. Managed Identities reduzieren operatives Risiko und sind ein zentraler Baustein für auditierbare Zugriffsketten.

3. CONTROL PLANE: AZURE AI FOUNDRY

Rolle der Control Plane Azure AI Foundry fungiert als Steuerungsebene für:

- Auswahl und Verwaltung von Modellen (Model Catalog)
- Bereitstellung und Betrieb von Inferenz-Endpunkten
- Agenten- und Tool-Orchestrierung inklusive Policies
- Evaluations- und Lifecycle-Funktionen

Konkrete Architekturentscheidung

- Foundry als zentraler Einstiegspunkt für KI-Produktteams
- Trennung von: Plattformbetrieb (IT) und Use-Case-Entwicklung (Fach-/ Produktteams)

Der Hauptgewinn der Control Plane ist nicht „mehr Features“, sondern die Standardisierung von Betriebs- und Governance-Prozessen. Ohne Control Plane zerfällt der KI-Betrieb in heterogene Einzelservices, die nicht mehr sauber kontrollierbar sind.

4. MODELLE UND INFERENZ ALS MANAGED SERVICE

Konkrete Bausteine

- Azure AI Foundry Model Catalog für Modellwahl
- Managed Inference Endpoints als standardisierte Laufzeit
- Optional: Azure OpenAI Service (wenn bestimmte Modelle/Capabilities benötigt werden)

Betriebslogik

- Pro produktivem Use Case ein eigener Inferenz-Endpunkt oder ein klar segmentiertes Endpoint-Portfolio (z. B. „Chat“, „Summarization“, „Extraction“)
- Einheitliche Limits und Policies pro Endpoint (Request Limits, Token Limits, Concurrency)
- Trennung von produktiven und explorativen Endpunkten

Inferenz ist der dominante Kostentreiber. Managed Inference reduziert die On-Prem-typische Optimierungs- und Patchlast und verschiebt die Komplexität auf eine standardisierte, betrieblich unterstützte Laufzeit.

5. RAG-STACK

Modelle bleiben generisch; organisationsspezifisches Wissen wird über Retrieval bereitgestellt.

Konkrete Bausteine

- Azure AI Search als Retrieval- und Index-Layer
- Azure Storage (Blob) als Dokument-Landing-Zone
- Optional: Azure Cosmos DB / Azure SQL für strukturierte Wissensobjekte
- Ingestion/ETL: Azure Data Factory oder Fabric Data Pipelines (je nach Data-Plattformstrategie)

RAG-Pipeline (spezifiziert)

- Dokumentingest (Blob) inkl. Metadaten (Owner, Klassifizierung, Gültigkeit, Quelle)
- Chunking-Strategie (regelbasiert pro Dokumenttyp) und Embedding-Erzeugung
- Indexierung in AI Search inkl. Filterfeldern (Mandant, Bereich, Klassifizierung)
- Retrieval-Policy: Top-k, Hybrid (BM25+Vector), Re-Ranking wenn nötig
- Antwortkonstruktion: Prompt-Template + Quellenreferenzen (Citations) + Policy-Checks

RAG trennt Wissensaktualität von Modellzyklen. Das reduziert die Notwendigkeit von Fine-Tuning und erhöht gleichzeitig Nachvollziehbarkeit, weil Quellen explizit referenzierbar sind.

6. MCP UND AGENTENSTEUERUNG

Tool-Nutzung und Kontextzugriffe werden zentral geregelt, nicht pro Use Case „frei" implementiert.

Konkrete Bausteine

- Foundry-Agenten/Agent Framework als Orchestrierungsschicht
- MCP-Server-Konzept als standardisierte Tool-Schnittstelle
- Tool-Gateway (Policy Enforcement Point) zwischen Agent und Backend-Systemen

Spezifikation Tool-Gateway

- Zentrale Whitelist erlaubter Tools pro Agent-Rolle
- Parametervalidierung und Schema-Checks pro Tool
- Rate Limits und Quotas pro Agent/Anwendung
- Vollständiges Tool-Audit: Wer hat wann welches Tool mit welchen Parametern aufgerufen
- Optional: Human-in-the-Loop für kritische Aktionen (Freigabe-Workflow)

Agenten werden gefährlich, wenn Tools unkontrolliert verfügbar sind. MCP ist die technische Grundlage, um Toolnutzung standardisiert und auditierbar zu machen.

7. OBSERVABILITY, MONITORING UND KOSTENSTEUERUNG

Konkrete Bausteine

- Azure Monitor / Log Analytics als zentrale Log-Plattform
- Application Insights für Request-Telemetrie
- Azure OpenTelemetry / Tracing über Correlation IDs (End-to-End)
- Cost Management + Budgets für Kostenkontrolle

Spezifizierte Telemetrieanforderungen

- Request Metriken: RPS, Fehlerquote, Timeoutquote, p50/p95/p99 Latenz
- Token Metriken: Prompt Tokens, Completion Tokens, Tokens pro Use Case
- Retrieval Metriken: Hit Rate, Re-Ranking Anteil, Quellenverteilung
- Agent Metriken: Tool Calls pro Anfrage, Tool Error Rate, Human Approvals
- Kosten: Kosten pro 1k Tokens pro Endpoint, Kosten pro Use Case, Trend über Zeit

Ohne Token- und Retrievalmetriken ist ein KI-Betrieb nicht steuerbar. Monitoring ist nicht „nice to have“, sondern die Voraussetzung für belastbare Betriebs- und Kostenentscheidungen.

8. RELEASE- UND LIFECYCLE-PROZESSE

Spezifikation

- Modell-/Endpoint-Änderungen laufen über Change-Prozess mit:
  - Evaluations (Qualität, Halluzinationsrate, Safety)

- Canary/Shadow-Deployment für neue Modellversionen
- Rollback-Plan mit definierter Metrikschwelle

- Prompt-Versionierung als Pflicht: jede produktive Prompt-Änderung ist versioniert, testbar, rollbackfähig

Der größte Fehler in KI-Betrieb ist, Prompts als „Text" zu behandeln. In produktiven Systemen sind Prompts Code.

9. FOUNDRY-UI, APPLIKATIONSZUGRIFF UND MANDANTENFÄHIGKEIT

Azure AI Foundry stellt eine zentrale Oberfläche und Entwicklungsumgebung bereit. Für den produktiven Einsatz ist jedoch zu spezifizieren, wie Endnutzer zugreifen und wie Mandanten-/Bereichstrennung umgesetzt wird.

Referenzentscheidung

- Foundry dient als Control Plane für Modell-, Agenten- und Endpoint-Management.
- Endnutzerzugriff erfolgt über eine standardisierte, organisationseigene UI (oder über ein kontrolliertes Frontend), das über Entra ID authentifiziert.

Mandantenfähigkeit

- Trennung wird über Entra ID (Rollen), RBAC (Role Based Access Control) auf Resource-Ebene und Filter-/Access-Policies auf Datenebene umgesetzt.
- Für Use Cases mit unterschiedlicher Schutzklasse werden getrennte Endpoints und getrennte Retrieval-Indizes (spezielle Such-/Indexstrukturen) betrieben.

10. MODEL ROUTER, PROVIDER-PORTFOLIO UND QUALITÄTS-/KOSTENSTEUERUNG (VOLL SPEZIFIZIERT)

Azure AI Foundry unterstützt ein Modellportfolio, das bewusst in Leistungs- und Kostenklassen strukturiert werden muss.

Referenz-Endpoint-Portfolio

- Endpoint-Klasse A: „Standard" (kostenoptimiert, Routine)
- Endpoint-Klasse B: „High Quality" (komplexe Fälle)
- Endpoint-Klasse C: „Long Context" (große Kontexte, selektiv)

Routingprinzip

- Die Applikation wählt nicht ad hoc ein Modell.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Routing erfolgt über definierte Policies (Use Case, Sensitivität, Budget, Kontextbedarf).

Spezifikation Routing-Attribute

- usecase_class
- data_sensitivity
- quality_tier
- context_length_bucket
- budget_class
- need_tools / need_websearch

Degradation

- Bei Budgetüberschreitung: Downgrade auf Standard-Endpoint oder Blockierung nicht-kritischer Requests.
- Bei Provider-/Endpoint-Degradation: Fallback auf alternative Endpoint-Klasse.

11. SENSITIVITY/DLP IN FOUNDRY

Für Foundry gilt dieselbe Grundanforderung wie On-Prem: Sensitivität muss vor und nach der Inferenz technisch durchgesetzt werden.

Referenzarchitektur

- DLP Guard als vorgelagerte und nachgelagerte Stufe in der Applikationskette (Input/Output).
- Klassifikation erfolgt policy-getrieben.

Implementationsmuster

- DLP-Komponente als separater Service (App Layer) mit Entra ID abgesichert.
- Nutzung von Managed Services für PII-Erkennung, sofern gewünscht, ansonsten Eigenbetrieb analog On-Prem.

Harte Regeln (identisch zur On-Prem-Policy)

- PII → mindestens Redaction
- PII + medizinischer Kontext → Block
- Keine Rohdaten in zentralen Logs; nur Metadaten.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

12. GUARDRAILS, PROMPT-SAFETY UND AGENT-RISIKOSTEUERUNG IN FOUNDRY

In Foundry ist Guardrail-Steuerung eine Kombination aus Plattformmechaniken und Applikations-/Policy-Schicht.

Referenzanforderungen

- Prompt-Injection-Abwehr (insbesondere bei RAG und Tooling)
- Topic/Intent-Control pro Use Case
- Output-Redaction-Patterns
- Human-in-the-Loop für kritische Tool-Aktionen

Umsetzung

- Guardrails werden als zentrale Policies pro Endpoint/Agent etabliert.
- Tool-Gateway als Policy Enforcement Point bleibt auch in Foundry der Standard, um Toolzugriffe auditierbar zu machen.

13. WEBSEARCH UND EXTERNE WISSENSZUGRIFFE IN FOUNDRY

Websearch ist eine eigene Capability, die kontrolliert angeboten werden muss.

Referenzprinzip

- Websearch nur über dedizierte Tools (MCP/Search-Tool), nie als „freier" Internetzugriff.
- Domain Allow-/Deny-Listen und Rate Limits sind verpflichtend.
- Auditfähigkeit über Tool-Logs.

14. OBSERVABILITY & COST GOVERNANCE IN FOUNDRY (REFERENZ)

Foundry-Workloads müssen auf denselben Steuerungsmetriken betrieben werden wie On-Prem:

- Requestmetriken (RPS, Fehler, Latenz p95)
- Tokenmetriken (Prompt/Completion, Tokens pro Use Case)
- Retrievalmetriken (Hit Rate, Quellenverteilung)
- Agentmetriken (Tool Calls, Human Approvals)

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Kosten (Kosten pro Use Case, Trends, Budgets)

Budgets und Limits werden pro Endpoint/Use Case gesetzt und sind Bestandteil der Betriebsabnahme.

15. RUNBOOKS, INCIDENT- UND PROBLEM-MANAGEMENT

Auch im Plattformansatz bleiben Betriebsprozesse notwendig. Referenz-Runbooks:

- Latenz-/Timeout-Anstieg
- Kostenanstieg einzelner Use Cases
- Retrieval-Qualitätsabfall (z. B. falsche Quellen, geringe Hit Rate)
- Tool-Missbrauch oder Tool-Fehlerwellen
- DLP-Block-Spikes (Hinweis auf Prompt-Injection oder Datenleckversuche)

Damit wird Foundry nicht „einfach nur betrieben“, sondern als kontrollierte Plattform gesteuert.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

VERGLEICH DER SYSTEME

Der Vergleich muss die entscheidende Realität abbilden: Zwei Stacks können funktional ähnlich wirken, unterscheiden sich aber massiv darin, wer die dauerhafte Optimierungs-, Patch- und Nachweispflicht trägt und wie sich Kosten in der Praxis verhalten. Entscheidend ist nicht nur die Architektur auf dem Papier, sondern die Betriebsökonomie über mehrere Jahre.

1. VERANTWORTUNGSVERTEILUNG

Im On-Prem-Ansatz verbleibt die vollständige Verantwortung für Hardware, Laufzeit, Inferenzoptimierung, Telemetrie, Incident Management und Change Control in der Organisation. Das ist grundsätzlich möglich, erfordert aber ein dauerhaftes Plattformteam und eine klare Produktisierung des Betriebs.

Konkret bedeutet dies:

- Die Organisation verantwortet die komplette GPU-Hardware-Lifecycle-Planung (Beschaffung, Ersatz, Abschreibung, Wartung).
- Die Organisation trägt die Verantwortung für Treiberstände, CUDA-Versionen, Inferenz-Framework-Versionen und deren Kompatibilität mit den eingesetzten Modellen.
- Kapazitätsplanung (Peak vs. Durchschnittslast) ist eine interne Aufgabe.
- Incident Management bei Inferenzproblemen (Latenz, OOM, Queue-Build-Up, GPU-Ausfälle) liegt vollständig in der eigenen Betriebsorganisation.
- Auditfähigkeit und Nachvollziehbarkeit müssen technisch selbst aufgebaut und dauerhaft gepflegt werden.

Im Foundry-Ansatz wird ein signifikanter Teil dieser Verantwortung auf die Plattform verschoben. Die Organisation fokussiert sich stärker auf Governance, Datenanbindung, Use-Case-Engineering und Kostensteuerung.

Das heißt in der Praxis:

- Hardware-Lifecycle, Inferenz-Runtime-Stabilität und Skalierungsmechaniken liegen primär beim Plattformanbieter.
- Patching, Sicherheitsupdates und viele Betriebsrisiken werden abstrahiert.
- Die Organisation bleibt verantwortlich für:
    - Modell- und Endpoint-Auswahl

- Budget- und Quotensteuerung
- Prompt-, RAG- und Tool-Governance
- fachliche Qualitätssicherung

Der fundamentale Unterschied ist:
On-Prem = Technische und betriebliche Gesamtverantwortung.
Foundry = Governance- und Steuerungsverantwortung bei ausgelagerter technischer Laufzeit.

2. INFERENZ ALS PRIMÄRER DIFFERENZFAKTOR

Inferenz ist der wirtschaftliche und technische Kern beider Architekturen. Der Unterschied liegt nicht in der Fähigkeit, Inferenz durchzuführen, sondern in der Art, wie Inferenz als Disziplin betrieben wird.

2.1 ON-PREM: INFERENZ ALS INGENIEURSDISZIPLIN

On-Prem zwingt zu einer ingenieurgetriebenen Optimierung der Inferenz. Der vLLM-Stack muss so konfiguriert werden, dass er Lastspitzen, Kontextlängen und unterschiedliche Use-Case-Profile beherrscht. Diese Optimierung ist dauerhaft, weil Modell- und Nutzungsprofile nicht stabil sind.

Charakteristika:

- Inferenzparameter (Batching, KV-Cache, Parallelisierung) sind Teil der Betriebsarchitektur.
- Latenz, Durchsatz und Kosten sind direkte Funktionen der internen Konfiguration.
- Jede Änderung an:
  - Promptlängen
  - RAG-Chunking
  - Agentenketten
  - Tool-Call-Anzahl
    verändert das reale Lastprofil und damit die Inferenzökonomie.

Damit wird Inferenz zu einer eigenen Engineering-Disziplin, vergleichbar mit dem Betrieb eines Hochlast-Transaktionssystems.

Organisatorische Konsequenz:

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Es braucht dedizierte Expertise für:
  - GPU-Auslastungsoptimierung
  - Lastprofile
  - Batching-Strategien
  - Kontextlängenmanagement
- Ohne diese Kompetenz eskalieren Kosten und Latenzen schleichend.

2.2 FOUNDRY: INFERENZ ALS PLATTFORMSERVICE

In Foundry wird Inferenz zu einem standardisierten Service. Die „Inferenzkompetenz" verlagert sich vom Low-Level-Optimieren hin zur bewussten Auswahl von Modellen, Limits, Endpoint-Portfolios und Budgets.

Charakteristika:

- Performance-Optimierung auf Runtime-Ebene ist weitgehend abstrahiert.
- Organisation entscheidet primär:
  - Welches Modell
  - Welche Kontextgrenzen
  - Welche Concurrency-Limits
  - Welche Budgetgrenzen
- Feinsteuerung von Batching, GPU-Fragmentierung etc. ist nicht Teil des operativen Alltags.

Organisatorische Konsequenz:

- Weniger Low-Level-Expertise nötig.
- Stärkerer Fokus auf:
  - Use-Case-Klassifizierung
  - Qualitätsstufen
  - Budget- und Kostensteuerung

On-Prem = maximale technische Hebel und Autarkie.
Foundry = reduzierte Hebel, dafür geringere Komplexität und geringeres Betriebsrisiko.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

3. KOSTEN- UND MODELLGRÖßENKORRELATION

Modellgröße korreliert mit Kosten, aber nicht linear, weil die realen Treiber im Inferenzbetrieb liegen.

Entscheidend sind nicht nur Parameterzahlen, sondern:

- Kontextlängen
- Promptgrößen
- Batching-Effizienz
- KV-Cache-Verhalten
- Agentenketten (mehrere Modellaufrufe pro fachlicher Anfrage)
- Retrieval- und Tool-Overhead

Damit entstehen Kostenprofile, die sich deutlich von einfachen „Preis pro Token"-Annahmen unterscheiden.

3.1 ON-PREM KOSTENLOGIK

On-Prem-Kosten sind primär Fixkosten und werden über Auslastung und Zeit amortisiert.

Kostenmodell (vereinfachte, praxistaugliche Näherung)

GPU-Stundenkosten =
(CAPEX + Wartung + Energie + Kühlung + Rechenzentrum + Betriebsaufwand)
÷ produktive GPU-Stunden

Effektive Inferenzkosten pro 1k Tokens =
GPU-Stundenkosten ÷ (Tokens pro Stunde bei realer Auslastung)

Warum das in der Praxis schwierig ist

1. Auslastung ist nicht konstant
   Peaks erfordern Überdimensionierung. Durchschnittslast liegt häufig deutlich unter Peak-Kapazität. Das senkt die reale Auslastung und erhöht die effektiven Kosten pro Token.
2. Kontextlängen verzerren Kosten
   Lange Kontexte erhöhen Prefill-Kosten und KV-Cache-Bedarf. Zwei Use Cases mit identischem Modell können sich um ein Vielfaches in den realen Kosten unterscheiden.

3. Agentenketten multiplizieren Inferenzkosten
   Eine fachliche Anfrage kann mehrere Inferenzdurchläufe erzeugen (Planung, Tool-Auswahl, Auswertung, Final Answer). Die Kosten pro fachlichem Outcome sind damit höher als „ein Prompt = eine Antwort".
4. Batching-Effizienz ist volatil
   Batching hängt stark vom Anfrageprofil ab. Viele kurze, synchrone Anfragen lassen sich besser batchen als wenige lange, interaktive Konversationen.

Konsequenz:

On-Prem wird wirtschaftlich nur dann attraktiv, wenn:

- Es einen stabilen, hohen, planbaren Inferenzdurchsatz gibt
- Die Organisation aktiv Inference-FinOps betreibt
- Kontextlängen und Agentenketten bewusst begrenzt und optimiert werden

Ohne diese Disziplin tendiert On-Prem dazu, teurer zu sein als erwartet, trotz fehlender Tokenpreise.

3.2 AZURE AI FOUNDRY KOSTENLOGIK

In Foundry sind Kosten nutzungsabhängig und korrelieren direkter mit Modellwahl und Verbrauch.

Kostenmodell (steuerbar)

Kosten pro Anfrage =
f(Prompt Tokens, Completion Tokens, gewähltes Modell, Kontextlänge)

Kosten pro Use Case =
Summe(Tokenkosten)

- Retrievalkosten
- Toolkosten
- ggf. Zusatzkosten für Long Context / Premium-Modelle

Warum das in der Praxis einfacher steuerbar ist

1. Direkte Korrelation zwischen Nutzung und Kosten
   Jede zusätzliche Tokenmenge erzeugt sichtbar Kosten. Es gibt keine „versteckten Fixkosten", die über Auslastung amortisiert werden müssen.

2. Budget- und Quotenmechaniken
   Budgets können pro Endpoint, Use Case oder Team gesetzt werden. Kostenexplosionen werden schneller sichtbar und sind technisch begrenzbar.
3. Modellportfolio als Kostensteuerungsinstrument
   Durch Staffelung (Standard, High Quality, Long Context) kann der Großteil der Anfragen auf günstigeren Modellen laufen, während teure Modelle gezielt eingesetzt werden.
4. Bessere Transparenz für Fachbereiche
   Kosten lassen sich leichter fachlich erklären („dieser Use Case kostet X pro Monat") als GPU-Auslastungsmodelle.

Konsequenz:

Foundry erleichtert die betriebswirtschaftliche Steuerung erheblich, insbesondere in Organisationen mit vielen Use Cases und stark schwankenden Lastprofilen.

4. SKALIERUNGSÖKONOMIE

4.1 On-Prem Skalierung

Skalierung bedeutet:

- zusätzliche GPUs
- zusätzliche Rack-Kapazität
- zusätzliche Strom- und Kühlleistung
- zusätzliche Betriebsaufwände

Skalierung ist diskret, kapitalintensiv und zeitverzögert. Fehlplanung führt entweder zu:

- Überkapazität (hohe Fixkosten)
- Unterkapazität (Latenzprobleme, Business-Impact)

4.2 FOUNDRY SKALIERUNG

Skalierung ist elastisch und primär ein Kosten- statt ein Kapazitätsproblem.

Das reduziert Planungsrisiken, erhöht aber die Abhängigkeit von Plattformpreisen und Verfügbarkeiten.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

5. CHANGE-, PATCH- UND LIFECYCLE-AUFWAND

On-Prem:

- Eigene Verantwortung für:
  - Inferenz-Framework-Upgrades
  - CUDA/Treiber-Kompatibilität
  - Sicherheits-Patches
- Jede Änderung ist ein potenzieller Betriebs- und Stabilitätsfaktor

Foundry:

- Plattform übernimmt Großteil des technischen Patchings
- Organisation fokussiert sich auf:
  - Modellwechsel
  - Endpoint-Änderungen
  - Prompt- und Policy-Versionen

6. AUDIT- UND NACHWEISFÄHIGKEIT

On-Prem:

Auditfähigkeit ist ein Eigenbau:

- Logging-Design
- Policy-Versionierung
- Konfigurationshistorien
- manuelle Nachweisprozesse

Foundry:

Auditfähigkeit wird strukturell unterstützt:

- Plattformlogs
- standardisierte Telemetrie
- konsistente Endpoint-Historien

Aber: Auch hier muss die Organisation Governance und Policies aktiv durchsetzen. Die Plattform ersetzt keine Governance.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

MODELLGRÖßEN, FINE-TUNING UND DRIFT-RISIKEN

Die Wahl der Modellgröße und der Umgang mit Fine-Tuning sind zentrale Architekturentscheidungen, die sich in On-Prem- und Cloud-basierten Plattformen grundlegend unterscheiden. Beide Ansätze haben unterschiedliche Implikationen für Qualität, Betriebsaufwand, Wartbarkeit und langfristige Stabilität der KI-Systeme.

1. MODELLGRÖßEN ALS STRUKTURELLE RAHMENBEDINGUNG

On-Prem / Private Cloud

In lokal betriebenen Umgebungen ist die maximal sinnvoll betreibbare Modellgröße in der Praxis begrenzt. Typische produktive On-Prem-Setups bewegen sich im Bereich kleiner bis mittlerer Modelle (z. B. 7B–120B Parameter), da größere Modelle erhebliche Anforderungen an GPU-Topologie, Interconnects, VRAM und parallele Skalierung stellen.

Sehr große Modelle erfordern:

- Multi-GPU-Sharding
- hohe Interconnect-Bandbreiten (z. B. NVLink/InfiniBand)
- komplexe Orchestrierung
- geringere Auslastungseffizienz

Damit steigt der operative Aufwand und die Kosten pro Anfrage überproportional. On-Prem-Stacks sind daher strukturell auf kleinere Modellklassen optimiert.

Azure AI Foundry

Azure AI Foundry ermöglicht den Zugriff auf sehr große, hochoptimierte Modelle („Frontier-Modelle“) mit deutlich höherer Parameterzahl, größerem Kontextfenster und komplexeren Reasoning-Fähigkeiten. Diese Modelle werden auf speziell optimierten Hyperscaler-Infrastrukturen betrieben, die lokal nur mit erheblichem Aufwand und hohen Kosten nachzubilden wären.

Damit entsteht ein struktureller Unterschied:

- On-Prem: Fokus auf kleinere bis mittlere Modelle
- Foundry: Zugriff auf sehr große, leistungsfähige Modelle

2. FINE-TUNING ALS KOMPENSATIONSMECHANISMUS KLEINERER MODELLE

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

On-Prem

Da kleinere Modelle naturgemäß geringere allgemeine Modellkapazität besitzen, entsteht häufig der Impuls, diese durch Fine-Tuning stärker an den konkreten Anwendungsfall anzupassen.

Fine-Tuning wird On-Prem oft genutzt, um:

- domänenspezifisches Wissen zu integrieren
- Antwortstil und Struktur zu stabilisieren
- fehlende Modellkapazität zu kompensieren

Architektonisch ist Fine-Tuning damit in On-Prem-Stacks häufiger ein Mittel, um Qualitätslücken kleinerer Modelle zu schließen.

Azure AI Foundry

In Foundry-Umgebungen ist Fine-Tuning in vielen Fällen weniger notwendig, da größere Basismodelle bereits eine deutlich höhere allgemeine Leistungsfähigkeit, breiteres Weltwissen und bessere Reasoning-Fähigkeiten besitzen.

Statt Fine-Tuning werden dort häufiger eingesetzt:

- Prompt Engineering
- Retrieval-Augmented Generation (RAG)
- Tool-basierte Erweiterungen

Das reduziert den Bedarf, Modellgewichte selbst zu verändern, und verschiebt die Anpassung in steuerbare, versionierbare Konfigurations- und Datenebenen.

3. DRIFT-RISIKEN DURCH FINE-TUNING

Fine-Tuning ist nicht nur eine Qualitätsmaßnahme, sondern führt auch zu zusätzlichen Betriebs- und Governance-Risiken. Feinjustierte (fine-tuned) Modelle driften stärker, weil sie auf spezifischen, weniger umfangreichen und zeitlich begrenzten Trainingsdaten basieren und dadurch stärker von Veränderungen in Fachlogik, Datenbasis oder Nutzungsmustern beeinflusst werden als allgemeine Basismodelle. Zudem verstärken sich Abweichungen, da Änderungen an Prompts, RAG oder Tool-Logik auf feinjustierte Modelle oft unvorhersehbare Wechselwirkungen haben.

DRIFT-RISIKEN IN ON-PREM-STACKS

Durch Fine-Tuning entstehen potenziell mehrere Formen von Drift:

Model Drift

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Abweichung des feinjustierten Modells vom ursprünglichen Basismodell
- Verändertes Antwortverhalten über die Zeit

Data Drift

- Trainingsdaten spiegeln nicht mehr die aktuelle Realität wider
- Veraltete Fachlogik oder Dokumente fließen in das Modell ein

Behavioral Drift

- Änderungen an Prompts, RAG oder Tools wirken anders auf ein feinjustiertes Modell als auf ein Basismodell
- Schwerer vorhersagbare Wechselwirkungen

Je häufiger Fine-Tuning eingesetzt wird, desto stärker wird das Modell zu einem individuellen, schwer reproduzierbaren Artefakt. Das erhöht:

- Validierungsaufwand
- Regressionsrisiken
- Audit- und Nachweiskomplexität

Drift-Risiken in Foundry-Stacks

In Foundry-Umgebungen wird Drift häufiger auf den Ebenen oberhalb des Modells adressiert:

- RAG-Indizes
- Prompts
- Tool-Logik
- Policies

Das Modell selbst bleibt häufiger unverändert. Drift manifestiert sich damit eher in:

- Wissensbasis
- Prompt-Logik
- Tool-Workflows

Diese Ebenen sind:

- leichter versionierbar
- einfacher testbar
- besser rückrollbar

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

Damit ist das Drift-Risiko besser kontrollierbar, auch wenn es weiterhin existiert.

4. BETRIEBS- UND GOVERNANCE-IMPLIKATIONEN

On-Prem

Häufiges Fine-Tuning führt zu:

- wachsender Anzahl individueller Modellversionen
- komplexem Lifecycle-Management
- erhöhtem Validierungs- und Testaufwand
- steigenden Anforderungen an Dokumentation und Nachvollziehbarkeit

Das Modell wird selbst zu einem zentralen Governance-Objekt.

Azure AI Foundry

In Foundry-Stacks liegt der Governance-Schwerpunkt stärker auf:

- Endpoint-Portfolio
- Prompt- und Policy-Versionierung
- RAG-Datenpflege
- Tool-Governance

Das Modell bleibt häufiger ein standardisiertes Plattformartefakt, was:

- Betriebsaufwand reduziert
- Reproduzierbarkeit erhöht
- Auditfähigkeit erleichtert

5. ARCHITEKTUR-FAZIT

On-Prem-Stacks sind strukturell auf kleinere Modelle ausgerichtet. Um fachliche Qualität sicherzustellen, wird Fine-Tuning dort häufiger eingesetzt, was jedoch Drift-Risiken, zusätzlichen Betriebsaufwand und erhöhte Governance-Komplexität mit sich bringt.

Azure AI Foundry ermöglicht den Einsatz deutlich größerer Basismodelle, wodurch Fine-Tuning in vielen Fällen weniger notwendig ist. Anpassungen verlagern sich auf besser kontrollierbare Ebenen wie RAG, Prompts und Tool-Logik. Dadurch reduziert sich die Drift-Gefahr auf Modellebene und die langfristige Wartbarkeit wird verbessert.

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

bwiiw
but what if IT works.

ENTSCHEIDUNG

ZIEL DER ENTSCHEIDUNG

Die Entscheidung zwischen einem On-Prem-KI-Stack und einer Azure-AI-Foundry-Plattform ist keine rein technische Architekturentscheidung. Sie ist eine strategische Festlegung auf ein Betriebs-, Verantwortungs- und Kostenmodell für KI als Unternehmensplattform.

Die zentrale Frage lautet:

Will die Organisation KI als eigenes technisches Produkt betreiben – oder als gemanagte Plattform steuern?

ENTSCHEIDUNGSDIMENSIONEN

1. VERANTWORTUNGSMODELL

On-Prem

- Volle technische und betriebliche Verantwortung
- Hardware, Inferenz, Laufzeit, Patching, Stabilität liegen intern
- Hohe organisatorische Reife und dauerhaftes Plattformteam erforderlich

Azure AI Foundry

- Technische Laufzeit und Skalierung werden ausgelagert
- Organisation fokussiert auf Governance, Use Cases und Kostensteuerung
- Reduziertes operatives Risiko

Ist die Organisation bereit und fähig, KI als eigenes Plattformprodukt zu betreiben?

2. TIME-TO-VALUE

On-Prem

- Längere Aufbauphase (Hardware, Plattform, Betriebsprozesse)
- Verzögerte Skalierbarkeit
- Höhere Anfangskomplexität

Azure AI Foundry

- Schnell produktiv

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

- Elastische Skalierung
- Geringere Einstiegshürde

Ist schnelle Wertschöpfung wichtiger als maximale technische Kontrolle?

3. KOSTENLOGIK

On-Prem

- Hoher Fixkostenanteil
- Wirtschaftlich nur bei:
  - hohem, stabilem Durchsatz
  - aktiver Inferenzoptimierung
- Kosten schwerer fachlich erklärbar
- Azure AI Foundry
- Variable Kosten pro Nutzung
- Direkte Korrelation zwischen Use Case und Kosten
- Einfachere Budgetierung und Kostenallokation

Gibt es einen stabilen, planbaren Inferenzbedarf – oder stark schwankende Nutzung?

4. INFERENZKOMPETENZ

On-Prem

- Inferenz ist Ingenieursdisziplin
- Dauerhafte Optimierung erforderlich
- Eigene Expertise für GPU, Batching, Kontextmanagement nötig

Azure AI Foundry

- Inferenz als Plattformservice
- Fokus auf Modellwahl und Limits statt Runtime-Optimierung

Will die Organisation dauerhaft Inferenz-Engineering als Kernkompetenz aufbauen?

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

5. SKALIERUNGSRISIKO

On-Prem

- Diskrete Skalierung
- Risiko von Über- oder Unterdimensionierung
- Kapitalbindung

Azure AI Foundry

- Elastische Skalierung
- Skalierung = Kostenentscheidung
- Geringeres Kapazitätsrisiko

Sind Lastprofile stabil oder volatil? Akzeptiere ich die Kapitalbindung?

6. GOVERNANCE UND AUDITFÄHIGKEIT

On-Prem

- Auditfähigkeit muss aktiv gebaut werden
- Hoher Implementierungs- und Pflegeaufwand

Azure AI Foundry

- Plattformunterstützung für Logging, Telemetrie und Nachvollziehbarkeit
- Governance weiterhin erforderlich, aber technisch einfacher durchsetzbar

Soll Governance primär technisch selbst gebaut oder durch Plattformmechaniken unterstützt werden?

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

ENTSCHEIDUNGSMATRIX (EXECUTIVE SUMMARY)

Situation	Strategisch sinnvoller Ansatz
Hoher Souveränitätszwang, tatsächlich nur wenn Regulatorisch gefordert.	On-Prem oder Sovereign Cloud Angebote
Stabil hoher Inferenzdurchsatz, Kostenprobleme in einer Cloud-Kalkulation	On-Prem
Aufbau interner KI-Plattformkompetenz, dringliches Training eigener Modelle für hochspezifische Umfänge	On-Prem
Schnelle Wertschöpfung erforderlich	Azure AI Foundry
Stark schwankende Nutzung	Azure AI Foundry
Viele parallele Use Cases	Azure AI Foundry
Begrenzte interne Plattformressourcen	Azure AI Foundry
Fokus auf Governance statt Runtime	Azure AI Foundry
Maximale, neueste Modellqualität	Azure AI Foundry
Keine Kapazitäten für Finetuning (vor allem Expertise und Zeit) von kleinen Modellen	Azure AI Foundry

STRATEGISCHE KERNAUSSAGE FÜR DAS MANAGEMENT

On-Prem ist eine Investition in eine eigene KI-Plattformorganisation, die auch dementsprechend sauber gestaffed sein muss.
Er ist nur dann sinnvoll, wenn die Organisation bereit ist, KI als **dauerhaftes** Kern-Engineering-Thema zu betreiben.

Azure AI Foundry oder vergleichbare Plattformen sind eine Investition in Geschwindigkeit, Risikoreduktion und Steuerbarkeit.
Er ist für die Mehrheit der Organisationen der wirtschaftlich und organisatorisch realistischere Weg.

Für Organisationen mit:

- mehreren Fachbereichen
- heterogenen Use Cases
- dynamischen Lastprofilen
- begrenzten Plattformteams

florian.mertl@bwiiw.com
www.bwiiw.com
+491728918803

ist ein plattformbasierter Ansatz (Microsoft AI Foundry. oder ähnliche) in der Regel der strategisch sinnvollere Einstieg und Skalierungspfad.

Ein On-Prem-Stack sollte bewusst als Spezialfall betrachtet werden, nicht als Standard – und nur dort eingesetzt werden, wo Souveränität oder regulatorische Anforderungen dies zwingend erfordern und die nötige Personaldecke mit entsprechendem Knowhow vorhanden ist.

DISCLAIMER

Dieses Dokument stellt keine Rechtsberatung dar. Es dient ausschließlich als fachliche, strategische und operative Orientierungshilfe zur Gestaltung und Bewertung von KI-Plattformarchitekturen, Betriebsmodellen und Governance-Strukturen aus Sicht von IT- und Unternehmensorganisationen.

Die Inhalte basieren auf dem zum Zeitpunkt der Erstellung verfügbaren technischen, organisatorischen und regulatorischen Rahmen sowie auf allgemein zugänglichen Informationen, Marktstandards und branchenüblichen Best Practices. Trotz sorgfältiger Erstellung wird keine Gewähr für die Vollständigkeit, Richtigkeit oder Aktualität der dargestellten Inhalte übernommen.

Insbesondere ersetzt dieses Dokument nicht die individuelle rechtliche, regulatorische oder organisatorische Prüfung durch qualifizierte Fach- und Rechtsberatung. Anforderungen, Auslegungen und Verpflichtungen können sich ändern, unterschiedlich interpretiert werden oder je nach Branche, Organisationsstruktur, Einsatzszenario und regionalem Kontext variieren.

Jegliche Haftung für Entscheidungen oder Maßnahmen, die auf Grundlage dieses Dokuments getroffen werden, ist ausgeschlossen. Die Verantwortung für die fachlich, organisatorisch, wirtschaftlich und rechtlich sachgerechte Auslegung sowie für die konkrete Umsetzung verbleibt bei den jeweiligen Organisationen und deren Leitungsorganen.

KONTAKT

Florian Mertl
Gründer und Geschäftsführer der bwiiw GmbH
www.bwiiw.com
florian.mertl@bwiiw.com
+49 1728918803