Nexus Playground Fachkonzept & Release-Planung

Version 1.0 · 18. November 2026 · Status: Fachkonzept zur externen Review

Ein integrierter Arbeitsbereich zum Experimentieren, Vergleichen, Bewerten und Diagnostizieren von KI-Modellen und Nexus-Routing. Dieses Dokument konsolidiert die gemeinsam getroffenen Produktentscheidungen und ergänzt sie um schematische Mockups sowie eine inkrementelle Release-Planung.

Review-Hinweis: Die Abbildungen sind konzeptionelle Mockups, keine Screenshots der implementierten Anwendung. Alle dargestellten Modellantworten und Messwerte sind fiktiv.

1. Vision und Produktabgrenzung

Nexus Playground ist die integrierte Entwicklungs-, Test- und Analyseumgebung für lokale und angebundene KI-Modelle. Im Mittelpunkt stehen vier Fähigkeiten: Experimentieren, Vergleichen, Routing untersuchen und Ergebnisse nachvollziehbar sichern.

• Chat: schlanke, alltägliche Chat-Nutzung mit Historie, Modellwahl und optionalem TTS.

• Playground: technischer Workspace mit Sessions, Testläufen, Konfiguration, Diagnose, Vergleich und Bewertung.

• Gemeinsame Basis: vorhandene Nexus-API, Router, Scheduler, Engine-Lifecycle, Model Registry und Telemetrie. Keine zweite Inferenz- oder Routing-Infrastruktur.

2. Verbindliche Konzeptentscheidungen

Bereich	Beschluss
Produktstruktur	Chat und Playground getrennt
Testorganisation	Test-Sessions als primärer Container
Routing	Automatik mit unabhängigen manuellen Overrides
Testkontext	Isolierte Einzeltests und Multi-Turn-Dialoge
Vergleiche	Historische Runs und neue Multi-Modell-Vergleiche
Diagnose	Erweiterte technische Diagnose auf Abruf
Konfiguration	Globale Presets plus Session-Konfiguration
Promptverwaltung	Zentrale versionierte Prompt-Bibliothek
Ausführung	Normal über Scheduler oder kontrolliert
Persistenz	Vollständige immutable Run-Historie
Navigation	Session-Sidebar; Dialog/Testläufe/Vergleiche
Workspace	Einklappbare, größenveränderbare Panels
Testdialog	Hybrid mit Rerun, Branching und Vergleichen
Vergleichsansicht	Antwortspalten plus Metrikmatrix
Qualität	Manuelle Bewertung und LLM-Judge
Export	JSON/CSV/Markdown, Berichte, Testpakete
Automatisierung	Wiederverwendbare Testsuites auf Knopfdruck
Routing-Diagnose	Kandidatenanalyse und simulationsbasierte Prüfung
Umsetzung	Inkrementelle, nutzbare Releases

3. Logische Zielarchitektur

Der Playground ist ein Nexus-Client mit eigenen Application Services für Testorganisation, Auswertung und Austausch. Der produktive Router und Scheduler bleiben die einzige Instanz für reale Ausführung.

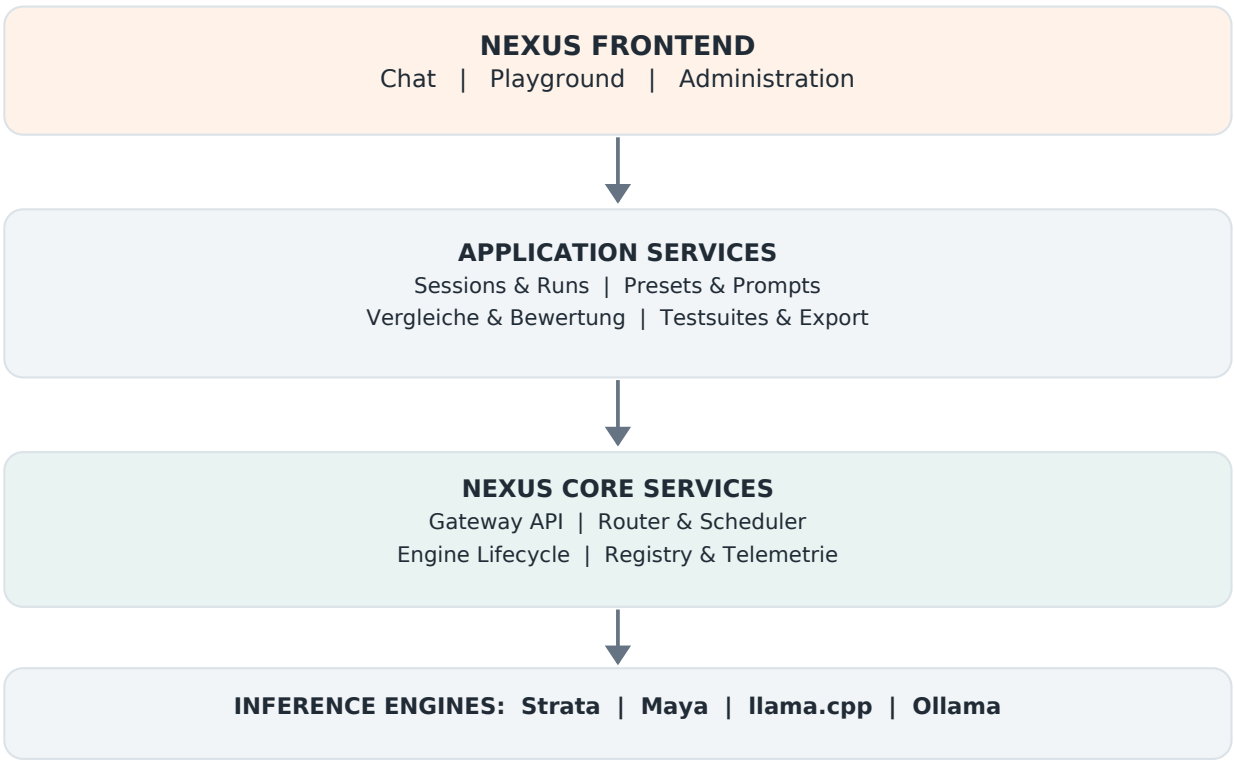


Abb. 1: Logische Schichten. Die Darstellung legt keine konkreten Prozesse, Deployments oder vorhandenen Implementierungen fest.

Technische Leitplanken

• OpenAI-kompatible Gateway-API bleibt bestehen; Playground-spezifische Funktionen erhalten bei Bedarf eigene Endpunkte.

• Durchgängige Korrelations-IDs verknüpfen Session, Testlauf, Gateway-Anfrage, Routing und Engine-Logs.

• Die Routing-Simulation verwendet dieselbe Entscheidungslogik wie der produktive Router, jedoch mit einem simulierten Zustand und ohne Inferenz.

• Kontrollierte Ausführung reserviert ausschließlich Nexus-verwaltete Ressourcen; externe Last wird dokumentiert, nicht weggezaubert.

4. Domänenmodell und Datenintegrität

TestSession organisiert Experimente und hält die aktuelle Arbeitskonfiguration. TestRun ist eine konkrete Ausführung mit unveränderlichem Ergebnis und Snapshot der tatsächlich übertragenen Eingaben. Comparison referenziert mehrere Runs. Evaluation enthält manuelle oder LLM-basierte Bewertungen. TestSuite ist eine wiederverwendbare Testdefinition; TestSuiteExecution eine konkrete Ausführung.

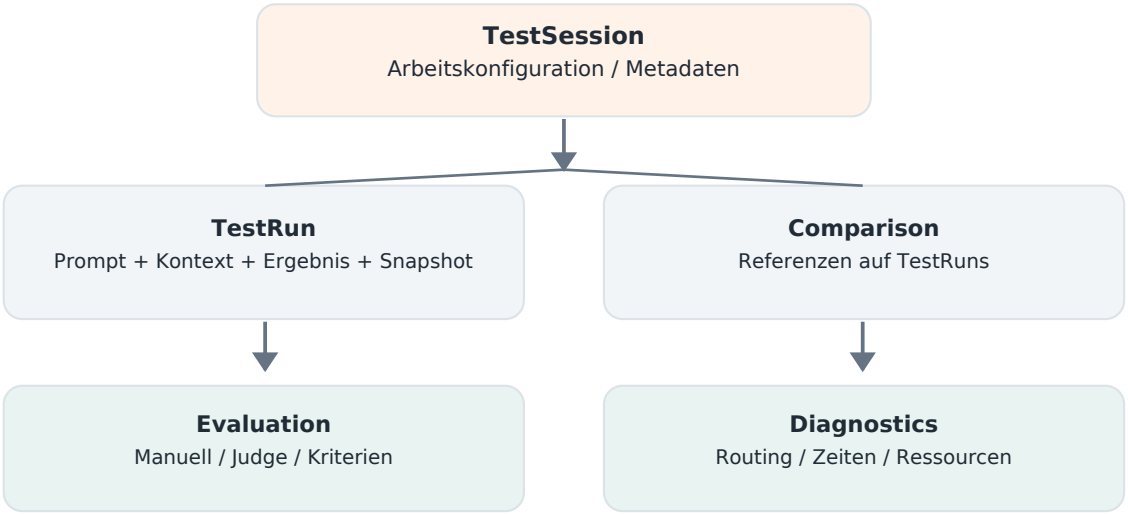


Abb. 2: Vereinfachte Objektbeziehungen. PromptVersion, Preset, TestSuite und RoutingSimulation sind eigenständige zusätzliche Objekte.

Run-Snapshot: Mindestumfang

• Prompt-Texte und exakt gesendeter Dialogkontext; referenzierte Prompt-Versionen.

• Angefragte und tatsächlich verwendete Engine, Modell, Quantisierung, Profil, Parameter und Overrides.

• Engine-/Backend-Version, Ausführungsmodus, Routing-Gründe, Queue- und Ladezeiten, Tokenzahlen, TTFT, Tokens/s, Laufzeit, Fehler und Ressourcen-Snapshots.

• Abgeschlossene Runs sind unveränderlich; Kommentare, Tags und Bewertungen werden getrennt gespeichert.

5. UX- und Workspace-Konzept

Die Navigation besteht aus einer einklappbaren Session-Sidebar und drei Tabs innerhalb der Session: Dialog, Testläufe, Vergleiche. Konfiguration und Diagnose liegen in separaten, größenveränderbaren Panels. Panel-Sichtbarkeit und -Breiten werden gespeichert, ohne Nutzeranpassungen beim Tabwechsel zu überschreiben.

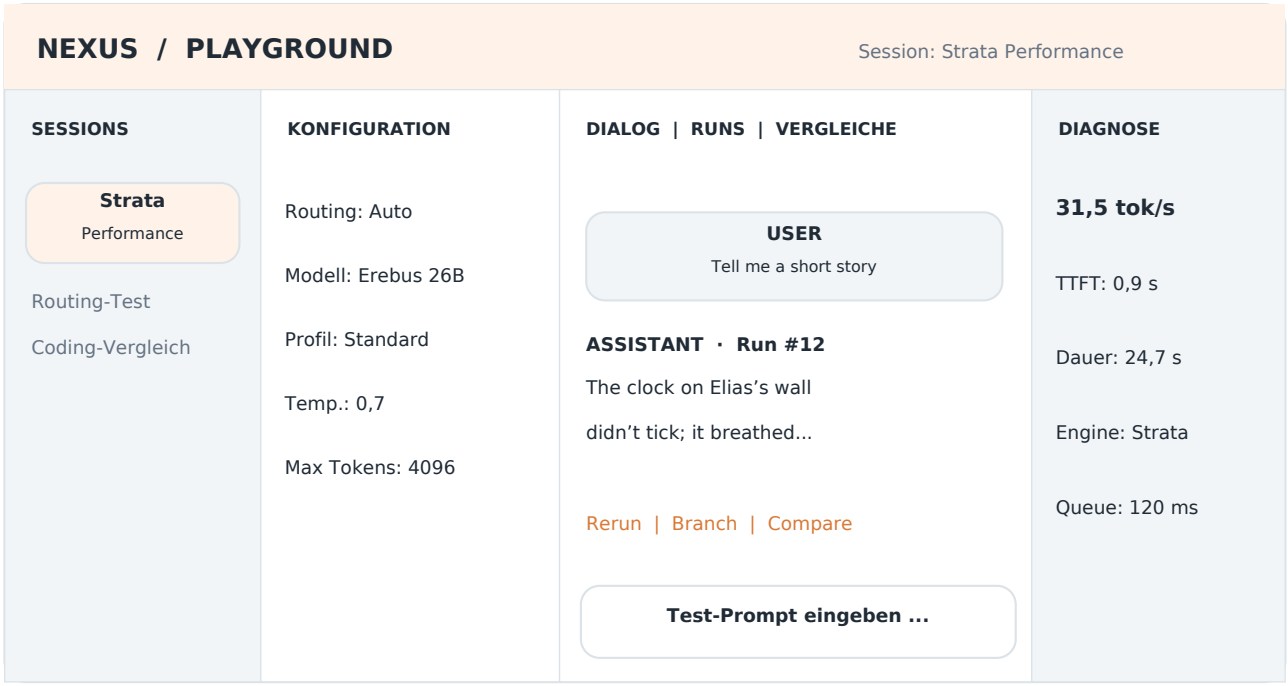


Abb. 3: Schematischer Desktop-Workspace. Die Größen sind illustrativ; die Panels sind im Produkt veränderbar.

Hybrider Testdialog

• Chatähnliche Darstellung mit Umschaltung zwischen isoliertem Prompt und Dialogkontext.

• Rerun eines früheren Prompts erzeugt einen neuen TestRun; ursprünglicher Run bleibt erhalten.

• Branching von einer früheren Nachricht erzeugt einen neuen Dialogzweig mit explizit festgehaltenem Kontext.

• Einzelne Runs können direkt zu Vergleichen hinzugefügt werden; Kopieren und TTS bleiben verfügbar.

6. Vergleichsansicht und Qualitätsbewertung

Vergleiche können bestehende historische Runs referenzieren oder neue Testläufe über mehrere Modelle/Settings auslösen. Antworten erscheinen nebeneinander, mit optional synchronem Scrollen. Eine separate Metrikmatrix zeigt TTFT, Tokens/s, Laufzeiten, Tokenzahlen, Engine, Modell und Ausführungsbedingungen. Unterschiede in Parametern und Ressourcen werden markiert.

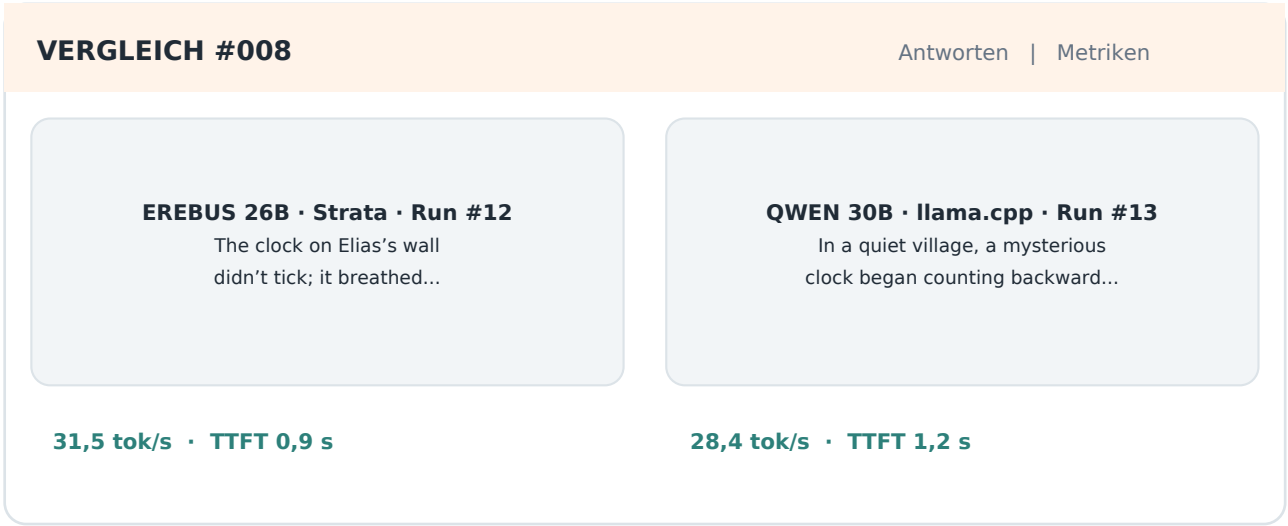


Abb. 4: Antwortvergleich. Modellnamen und Messwerte sind fiktiv.

Metrik	Erebus 26B	Qwen 30B
Tokens/s	31,5	28,4
TTFT	0,9 s	1,2 s
Gesamtdauer	24,7 s	29,1 s
Output-Tokens	779	826
Engine	Strata	llama.cpp

Abb. 5: Zugehörige Metrikmatrix (Beispieldaten).

Bewertungen

• Konfigurierbare Bewertungsschemata mit Kriterien wie Korrektheit, Vollständigkeit und Verständlichkeit; Coding- und Kreativaufgaben dürfen unterschiedliche Kriterien nutzen.

• Manuelle Bewertungen und LLM-Judge-Ergebnisse bleiben getrennt. Bewertungs-Prompt, Judge-Modell, Version, Kriterien und Begründung werden gespeichert.

• Blindbewertung ohne Modellnamen wird unterstützt, um Modell-Bias zu reduzieren. Kein automatischer Gesamtsieger allein aufgrund der Geschwindigkeit.

7. Routing-Diagnose und Simulation

Nexus zeigt nicht nur die gewählte Route, sondern auch geprüfte Kandidaten, Ausschlussgründe, Ressourcen- und Prioritätsbedingungen sowie angefragte und tatsächliche Konfigurationen. In der Simulation werden alternative Systemzustände geprüft, ohne eine Engine zu starten.



Strata / Erebus 26B	**Ausgewählt**
llama.cpp / Erebus 26B	**Keine Kapazität**
Maya / GLM	**Modellvorgabe**

Abb. 6: Fiktive Routing-Diagnose mit Kandidaten und Entscheidungsgründen.

• Simulationsszenarien: GPU ausgelastet oder nicht verfügbar; Engine ausgefallen; Modell nicht geladen; VIP-Priorität; manuelle Overrides.

• Simulation und reale Ausführung verwenden denselben Entscheidungsalgorithmus, aber unterschiedliche Zustandsquellen. Simulation ist keine Zusicherung späterer Kapazität.

8. Testsuites, Export und Reproduzierbarkeit

Testsuites bündeln wiederverwendbare Einzeltests, Dialogtests und Vergleiche. Sie können manuell gestartet werden, mit konfigurierbarer Wiederholungszahl, Ausführungsreihenfolge und Bewertung. Zeitpläne und Ereignistrigger sind ausdrücklich nicht Teil des beschlossenen initialen Umfangs.

ENGINE REGRESSION SUITE

3 Testfälle · 9 geplante Ausführungen · Modus: kontrolliert

Prompt-Verarbeitung	Einzeltest	**3×**
Kontextverhalten	Dialog	**3×**
Modellvergleich	Vergleich	**3×**

Abb. 7: Beispiel einer wiederverwendbaren Testsuite.

• Exportformate: JSON für strukturierte Daten, CSV für Tabellen, Markdown für Dokumentation sowie strukturierte Testberichte.

• Importierbares Nexus-Testpaket mit versioniertem Manifest, referenzierten Prompts, Konfigurationen, Modell-/Engine-Informationen und Ergebnissen.

• Importvalidierung meldet fehlende Modelle, inkompatible Engines und nicht unterstützte Schema-Versionen; kein stillschweigender Ersatz.

• Reproduzierbarkeit bedeutet nachvollziehbare Testbedingungen, nicht garantierte identische Modellantworten.

9. Inkrementelle Release-Roadmap

R1 – Playground Core

Umfang: Sessions, flexibler Workspace, hybrider Testdialog, Routing-Auto/Overrides, Run-Historie, grundlegende Diagnose.

Abnahme: Session erstellen, mehrere Tests ausführen, Konfiguration ändern, unveränderte Ergebnisse wieder öffnen.

R2 – Experimentation & Comparison

Umfang: Presets, Prompt-Versionierung, Rerun, Dialog-Branching, gespeicherte Vergleiche und kombinierte Vergleichsansicht.

Abnahme: Denselben Prompt mit zwei Konfigurationen testen, Ergebnisse vergleichen und später wiederholen.

R3 – Diagnostics & Quality

Umfang: Erweiterte Routing- und Ressourcen-Diagnose, konfigurierbare Bewertungsschemata, manuelle Bewertungen und LLM-Judge.

Abnahme: Qualität und Performance zweier Runs bewerten und Routing-Entscheidungen erklären.

R4 – Benchmarking & Automation

Umfang: Kontrollierte Ausführung, Testsuites, Wiederholungen, Aggregation, Berichte, Export und importierbare Testpakete.

Abnahme: Vor und nach einem Engine-Update dieselbe Suite ausführen und Ergebnisse vergleichen.

R5 – Routing Simulation

Umfang: Simulierte Systemzustände, Kandidatenanalyse, Ausschlussgründe, Prioritäts- und Override-Szenarien.

Abnahme: GPU-Ausfall simulieren und die Route untersuchen, ohne Inferenz auszulösen.

Reihenfolge: R1 → R2 → R3 → R4 → R5. Jedes Release soll nutzbar, testbar und dokumentiert sein. Die Planung ist funktionsorientiert, nicht kalendergetrieben.

10. Nichtfunktionale Anforderungen

Bereich	Anforderung
Logging	Strukturiert, aussagekräftig, durchgängig korrelierbar
Fehlerbehandlung	Explizite Status- und Fehlermeldungen
Qualitätssicherung	Unit-, Integrations- und API-Tests; CI-Prüfungen
Git	GitFlow-Branches; Conventional Commits
Persistenz	Immutable abgeschlossene Runs; versionierte Migrationen
Sicherheit	Berechtigungen, Importvalidierung, sichere Log- und Prompt-Verarbeitung
Ausführung	Streaming, asynchrone lange Tests, gemeinsamer Scheduler
Modularität	Wiederverwendung statt doppelter Engine-/Router-Verwaltung

11. Vorbedingungen und offene technische Prüfaufträge

Dieses Dokument definiert das fachliche Zielbild, aber nicht den Ist-Zustand der laufenden Nexus-Implementierung. Vor jeder Implementierung muss ein Coding-Agent den Bestand prüfen und die Release-Grenzen gegen die vorhandenen Komponenten abgleichen.

- Welche Chat-, Routing-, Scheduler- und Diagnosefunktionen existieren bereits?
- Welche APIs, Datenmodelle, Frontend-Komponenten und Telemetrie sind wiederverwendbar?
- Wie werden Requests, Runs, Engine-Zustände und Logs heute korreliert?
- Welche Datenbank- und Migrationsstrategie ist vorhanden?
- Welche API-Verträge und Daten-Snapshots sind für R1 zwingend notwendig?
- Welche Risiken entstehen für bestehende Chat-Clients und die OpenAI-kompatible API?

12. Empfohlener Review- und Umsetzungsprozess

1. Fachkonzept einfrieren: Diese Version dient als fachliche Baseline. Neue Ideen kommen zunächst in den Backlog.

2. Technische Bestandsaufnahme: Agent untersucht Nexus lesend und dokumentiert vorhandene Funktionen, Wiederverwendung, Lücken und Risiken.

3. Technisches Design: Module, Datenmodelle, Schnittstellen, Migrationen, Akzeptanztests und priorisierter R1-Backlog werden konkretisiert.

4. Umsetzung: Erst nach Freigabe Entwicklung auf Feature-Branch, automatisierte Tests, CI und Abnahme.

13. Fragen an Claude für die kritische Review

• Ist die Trennung zwischen Session, Run, Vergleich, Bewertung und Testsuite konsistent?

• Wo drohen doppelte Zuständigkeiten mit dem vorhandenen Nexus-Core?

• Sind immutable Snapshots vollständig genug, um spätere Vergleiche glaubwürdig zu machen?

• Ist die Release-Reihenfolge technisch sinnvoll oder müssen Grundlagen früher gelegt werden?

• Welche Anforderungen sind zu komplex für den ersten nutzbaren Release?

• Welche Fehlerfälle, Berechtigungen, Limits und Sicherheitsaspekte fehlen?

• Wie lässt sich die Routing-Simulation ohne zweite Routing-Implementierung realisieren?

• Welche messbaren Abnahmekriterien und automatisierten Tests fehlen?

Review-Auftrag: Bitte zunächst kritisch prüfen, Widersprüche und Risiken benennen, konkrete Verbesserungen vorschlagen und den bestehenden Nexus-Code analysieren. Keine Implementierung oder Codeänderung ohne gesonderte Freigabe.