LLM-Anwendungen testen und absichern

Qualität, Robustheit und Sicherheit generativer KI-Systeme systematisch prüfen

01 INTRO

Klassische Softwaretests gehen meist davon aus, dass sich für eine definierte Eingabe ein erwartbares Ergebnis bestimmen lässt. Bei Anwendungen mit Sprachmodellen gilt diese Annahme nur eingeschränkt. Antworten können variieren, fachlich plausibel und trotzdem falsch sein, von kleinen Änderungen des Kontexts abhängen oder sich nach einem Modellupdate verändern. Klassische Testverfahren bleiben deshalb notwendig, reichen allein aber nicht aus.

Der Kurs vermittelt einen systematischen Quality-Engineering-Ansatz für LLM-Anwendungen. Die Teilnehmer entwickeln Evaluationsdatensätze, Qualitätsmetriken und automatisierte Tests, untersuchen Halluzinationen und Robustheit und testen RAG-, Tool- und agentische Systeme. Security wird dabei nicht als separates Randthema behandelt: Prompt Injection, untrusted Content, Datenabfluss, Berechtigungen und missbräuchliche Tool-Aufrufe werden als konkrete Testgegenstände in die Teststrategie integriert.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Erfahrung in Softwaretest, Qualitätssicherung oder Softwareentwicklung und Grundverständnis LLM-basierter Anwendungen

02 TRAININGSPROGRAMM

Inhalte

Was ist bei LLM-Systemen anders?

- probabilistisches Verhalten
- mehrere valide Antworten
- sprachliche Variation
- scheinbar plausible Fehler
- Abhängigkeit vom Kontext
- Modelländerungen
- klassische Testorakel reichen nicht immer aus
- Qualität muss operationalisiert werden

Das Testobjekt bestimmen

- Modell
- Prompt
- System Instructions
- Kontext
- Anwendungslogik
- Retrieval
- Tools
- Agenten
- Gesamtsystem statt nur Modellantwort

Qualitätsmerkmale von LLM-Anwendungen

- fachliche Richtigkeit
- Relevanz
- Vollständigkeit
- Konsistenz
- Faithfulness
- Robustheit
- Sicherheit
- Nachvollziehbarkeit

Funktionale und nichtfunktionale Qualität

- fachliche Funktion
- Antwortqualität
- Latenz
- Kosten
- Verfügbarkeit
- Datenschutz
- Security
- Benutzererfahrung
- unterschiedliche Qualitätsziele explizit machen

Anforderungen testbar machen

- eine gute Antwort ist kein Kriterium
- erwartete Eigenschaften
- unerlaubte Eigenschaften
- Muss-Kriterien
- Qualitätsbereiche
- Toleranzen
- Akzeptanzkriterien
- messbare Anforderungen entwickeln

Das Oracle-Problem

- eindeutiges erwartetes Ergebnis
- mehrere korrekte Antworten
- teilweise korrekte Antworten
- semantische Gleichwertigkeit
- subjektive Bewertung
- Referenzantworten
- Eigenschaften statt exakter Strings
- mehrere Oracles kombinieren

Deterministische Prüfungen

- Schema
- Datentyp
- Pflichtfelder
- erlaubte Werte
- Format
- Länge
- enthaltene oder verbotene Elemente
- deterministische Regeln zuerst nutzen

Semantische Prüfungen

- Bedeutung statt exakter Formulierung
- Ähnlichkeit
- relevante Inhalte
- fachliche Aussagen
- Widersprüche
- Vollständigkeit
- Grenzen semantischer Metriken
- nicht jede sprachliche Ähnlichkeit bedeutet Korrektheit

Testdaten für LLM-Systeme

- repräsentative Eingaben
- typische Benutzerfragen
- schwierige Fälle
- Grenzfälle
- fehlerhafte Eingaben
- mehrdeutige Eingaben
- unerwartete Sprache
- realistische Verteilung

Evaluationsdatensätze

- Golden Dataset
- Testfälle
- Eingabe
- Kontext
- erwartete Eigenschaften
- Referenzinformationen
- Qualitätskriterien
- Versionierung

Ground Truth

- fachlich geprüfte Referenzen
- bekannte Fakten
- Referenzdokumente
- strukturierte Daten
- menschliche Annotation
- Unsicherheit der Ground Truth
- Änderungen
- Herkunft nachvollziehbar halten

Testdatensatz entwickeln

- reale Anwendungsfälle sammeln
- Produktionsfälle anonymisieren
- synthetische Fälle ergänzen
- Fehlerklassen abdecken
- seltene kritische Fälle
- Positiv- und Negativbeispiele
- Datensatz balancieren
- Coverage sichtbar machen

Testfall-Design für LLMs

- Äquivalenzklassen
- Grenzwerte
- Entscheidungstabellen
- Zustandsmodelle
- kombinatorische Verfahren
- risikobasiertes Testen
- exploratives Testen
- klassische Testentwurfsverfahren weiterverwenden

Metamorphes Testen

- erwartete Beziehungen statt exakter Antworten
- irrelevante Informationen ergänzen
- Reihenfolge verändern
- Formulierung variieren
- Sprache wechseln
- semantisch äquivalente Eingaben
- erwartete Stabilität
- unerwartete Verhaltensänderungen erkennen

Perturbation Tests

- Tippfehler
- Umgangssprache
- unterschiedliche Formulierungen
- zusätzliche Informationen
- fehlende Informationen
- Reihenfolge
- Format
- Robustheit gegenüber kleinen Änderungen

Halluzinationen testen

- erfundene Fakten
- erfundene Quellen
- nicht vorhandene Funktionen
- falsche Zusammenhänge
- fehlende Informationen
- Modell zum Nichtwissen zwingen
- unbeantwortbare Fragen
- Halluzinationsrate operationalisieren

Abstention testen

- wann darf das System antworten?
- wann sollte es nicht antworten?
- fehlende Informationen
- Unsicherheit
- Rückfragen
- Eskalation
- kontrolliertes Nichtwissen
- falsche Sicherheit vermeiden

Konsistenz

- gleiche Frage mehrfach
- semantisch gleiche Fragen
- unterschiedliche Reihenfolge
- Gesprächskontext
- widersprüchliche Aussagen
- Selbstkonsistenz
- fachliche Konsistenz
- notwendige gegenüber unnötiger Varianz

Reproduzierbarkeit

- Modellversion
- Parameter
- Promptversion
- Kontext
- Retrieval-Zustand
- Tool-Version
- Testumgebung
- vollständige Testkonfiguration protokollieren

Automatisierte Evals

- Evaluationspipeline
- Datensatz
- System ausführen
- Ergebnisse erfassen
- Kriterien anwenden
- Metriken berechnen
- Vergleich mit Baseline
- automatischer Report

Regelbasierte Evaluation

- exakte Werte
- reguläre Ausdrücke
- Schemas
- fachliche Regeln
- bekannte Fakten
- erlaubte Kategorien
- verbotene Inhalte
- deterministische Bewertung bevorzugen, wo möglich

Modellbasierte Evaluation

- LLM-as-a-Judge
- Bewertungs-Prompt
- Kriterien
- Referenzantwort
- Scores
- Klassifikation
- Paarvergleich
- Begründungen nicht mit Wahrheit verwechseln

Grenzen von LLM-as-a-Judge

- Judge ist selbst probabilistisch
- Bias
- Positionseffekte
- Präferenz für bestimmte Formulierungen
- Modellabhängigkeit
- Promptabhängigkeit
- Selbstbewertung
- menschliche Kalibrierung notwendig

Human Evaluation

- fachliche Experten
- Bewertungsrubriken
- konsistente Kriterien
- mehrere Bewerter
- Uneinigkeit
- Stichproben
- Kosten
- Human Evaluation gezielt einsetzen

Metriken kombinieren

- deterministische Checks
- Retrieval-Metriken
- semantische Metriken
- Modellbewertung
- Human Review
- Performance
- Kosten
- kein einzelner Quality Score beschreibt das Gesamtsystem

Baselines

- bestehendes System
- bisheriges Modell
- bisheriger Prompt
- einfache Lösung
- menschliche Leistung, sofern sinnvoll messbar
- Kosten
- Latenz
- Änderungen gegen Baseline bewerten

Regressionstests

- Promptänderungen
- Modellwechsel
- Providerwechsel
- neue Modellversion
- Kontextänderungen
- neue Tools
- Retrieval-Änderungen
- Qualitätsverlust automatisch erkennen

Nichtdeterministische Regressionen

- einzelne Abweichung gegenüber systematischem Problem
- Wiederholungen
- Stichprobengröße
- Verteilungen
- Schwankungsbereiche
- statistische Betrachtung
- Schwellenwerte
- Flaky Evals vermeiden

Prompt-Versionen testen

- Prompt als Softwareartefakt
- Versionierung
- A/B-Vergleich
- gleiche Testdaten
- Qualitätsmetriken
- Kosten
- Regression
- Änderung nachvollziehbar machen

Modelle vergleichen

- gleiche Aufgaben
- gleiche Qualitätskriterien
- unterschiedliche Modelle
- Qualität
- Latenz
- Kosten
- Robustheit
- Auswahl nach Anwendungsfall

RAG-Systeme testen

- Ingestion
- Chunking
- Retrieval
- Ranking
- Kontext
- Generierung
- Quellen
- Pipeline-Stufen getrennt prüfen

Retrieval testen

- relevante Dokumente
- Precision
- Recall
- Hit Rate
- Ranking
- Ground Truth
- unbeantwortbare Queries
- Retrieval unabhängig von der Antwort bewerten

Groundedness und Faithfulness

- Antwort durch Kontext gedeckt?
- Aussagen Quellen zuordnen
- zusätzliche Behauptungen
- widersprüchliche Quellen
- falsche Interpretation
- Quellenbezug
- erfundene Belege
- Grounding systematisch evaluieren

RAG-End-to-End-Tests

- Frage
- Retrieval
- Kontext
- Antwort
- Quelle
- fachliche Richtigkeit
- fehlendes Wissen
- Regression über gesamte Pipeline

Tool Use testen

- richtiges Tool auswählen
- Tool nicht unnötig verwenden
- Parameter
- Validierung
- Tool-Fehler
- Ergebnisverarbeitung
- Seiteneffekte
- unerlaubte Tool-Nutzung

Function Calling testen

- erwarteter Funktionsaufruf
- kein Aufruf erwartet
- Parameter korrekt
- fehlende Parameter
- ungültige Werte
- mehrere mögliche Funktionen
- Fehlermeldungen
- Modelloutput und tatsächliche Ausführung trennen

MCP-Integrationen testen

- angebotene Tools
- Tool Discovery
- Schemas
- Berechtigungen
- Serverfehler
- manipulierte Tool-Ergebnisse
- unerwartete Daten
- MCP-Komponente deterministisch und End-to-End testen

Agentische Systeme testen

- Zielerreichung
- Planung
- Tool-Auswahl
- Aktionsfolge
- Zwischenzustände
- Abbruch
- Eskalation
- unerwartete Wege zum Ergebnis

Agenten nicht nur am Endergebnis messen

- erlaubte Aktionen
- verbotene Aktionen
- unnötige Aktionen
- Ressourcenverbrauch
- Zwischenschritte
- externe Änderungen
- Policy-Verstöße
- Erfolg auf falschem Weg erkennen

Langlaufende Agenten testen

- mehrere Arbeitsschritte
- Zustand
- Kontextverlust
- wiederholte Aktionen
- Schleifen
- Fehlerfortsetzung
- Unterbrechung
- Recovery

Multi-Agent-Systeme testen

- Delegation
- Rollen
- Übergaben
- widersprüchliche Ergebnisse
- Kommunikationsfehler
- gemeinsame Zustände
- Fehlerfortpflanzung
- Gesamtverhalten

Security als Testgegenstand

- Angriffsoberfläche bestimmen
- Benutzerinput
- externe Dokumente
- Webseiten
- Tools
- APIs
- Memory
- andere Agenten

Direkte Prompt Injection

- System Instructions umgehen
- Rollenwechsel
- manipulierte Anweisungen
- vertrauliche Informationen anfordern
- Policy-Umgehung
- Angriffsvarianten
- Erfolgskriterien
- Regressionstests

Indirekte Prompt Injection

- Dokumente
- Webseiten
- E-Mails
- Suchergebnisse
- RAG-Dokumente
- Tool-Ausgaben
- versteckte Instruktionen
- untrusted Content systematisch testen

Datenexfiltration

- System Prompt
- vertraulicher Kontext
- andere Benutzerdaten
- Dokumente
- Secrets
- Tool-Ergebnisse
- externe Kommunikationskanäle
- mögliche Exfiltrationspfade testen

Berechtigungen testen

- erlaubte Aktionen
- verbotene Aktionen
- Benutzerrollen
- Mandanten
- Read und Write
- externe Systeme
- Least Privilege
- Modellentscheidung darf keine Autorisierung ersetzen

Tool-Missbrauch testen

- gefährliche Parameter
- unerwartete Kombinationen
- Path Traversal
- Injection in nachgelagerte Systeme
- unzulässige Schreiboperationen
- Zugriff außerhalb des vorgesehenen Bereichs
- Tool-Komposition
- serverseitige Kontrollen verifizieren

Memory testen

- falsche Informationen speichern
- sensible Informationen
- Benutzertrennung
- veraltete Informationen
- Manipulation
- Löschung
- ungewollte Persistenz
- Memory Poisoning

Adversarial Testing

- absichtlich schwierige Eingaben
- widersprüchliche Instruktionen
- ungewöhnliche Formate
- Encoding
- mehrsprachige Angriffe
- Kontextmanipulation
- lange Eingaben
- systematische Angriffsklassen

Red Teaming

- Angriffshypothesen
- Threat Model
- Angriffsziele
- Testkampagnen
- manuelle Exploration
- automatisierte Varianten
- Ergebnisse klassifizieren
- Regressionstests aus Findings ableiten

Threat Modeling für LLM-Anwendungen

- Assets
- Akteure
- Trust Boundaries
- Datenflüsse
- Eingabekanäle
- Tools und externe Systeme
- mögliche Auswirkungen
- Tests aus Bedrohungen ableiten

Abuse Cases

- unbeabsichtigte Nutzung
- absichtlicher Missbrauch
- fachlich unerlaubte Aktionen
- Datenzugriffe
- Ressourcenmissbrauch
- Automatisierung schädlicher Abläufe
- erwartete Gegenmaßnahmen
- konkrete Tests entwickeln

Datenschutz testen

- personenbezogene Daten
- Logging
- Modellprovider
- Persistenz
- Conversations
- Testdaten
- Datenminimierung
- Löschprozesse

Performance testen

- Antwortzeit
- Time-to-first-token
- lange Kontexte
- parallele Benutzer
- Tool-Aufrufe
- Retrieval
- Agent Loops
- Performance-Budgets

Kosten als Qualitätsmerkmal

- Tokens
- Modellaufrufe
- Retrieval
- Reranking
- Agentenschritte
- Retries
- Kosten pro Aufgabe
- Kostenregressionen erkennen

Last- und Stabilitätstests

- parallele Requests
- Rate Limits
- Providerfehler
- Timeouts
- Queueing
- Backpressure
- Fallback
- kontrolliertes Verhalten unter Last

Resilience Testing

- Modell nicht erreichbar
- Retrieval ausgefallen
- MCP-Server ausgefallen
- Tool-Timeout
- partielle Fehler
- ungültige Modellantworten
- Retry
- Degraded Mode

Observability für QA

- Promptversion
- Modellversion
- Eingabe
- Kontext
- Retrieval
- Tool Calls
- Antwort
- Qualitäts- und Betriebsmetriken

Produktionsdaten für Qualität nutzen

- reale Fehlerfälle
- Benutzerfeedback
- abgebrochene Interaktionen
- Eskalationen
- schlechte Antworten
- Datenschutz
- neue Regressionstests
- Evaluation kontinuierlich erweitern

Quality Gates

- Mindestqualität
- kritische Testfälle
- Security Tests
- Regression
- Kosten
- Performance
- Freigabekriterien
- Änderungen bei Nichterfüllung blockieren

Evals in CI/CD

- automatisierte Evaluationssuite
- Pull Request
- Promptänderung
- Modellkonfiguration
- Retrievaländerung
- Vergleich gegen Baseline
- Report
- Releaseentscheidung auf nachvollziehbare Daten stützen

Testpyramide für LLM-Anwendungen

- deterministische Unit Tests
- Component Tests
- Contract Tests
- LLM Evals
- Integrationstests
- End-to-End-Tests
- Security Tests
- Produktionsmonitoring als zusätzliche Rückkopplung

Teststrategie entwickeln

- Risiken
- Qualitätsmerkmale
- Testobjekte
- Teststufen
- Oracles
- Datensätze
- Metriken
- Quality Gates

Praktische Übungen

- Qualitätskriterien für eine LLM-Anwendung definieren
- Evaluationsdatensatz entwickeln
- deterministische Checks implementieren
- semantische Bewertung ergänzen
- LLM-as-a-Judge aufbauen und kritisch prüfen
- Halluzinationen und Abstention testen
- Promptvarianten als Regression vergleichen
- RAG-Retrieval separat evaluieren
- RAG-Antworten auf Groundedness prüfen
- Tool Calls automatisiert testen
- Prompt-Injection-Testfälle entwickeln
- indirekte Prompt Injection untersuchen
- Berechtigungs- und Tool-Missbrauch testen
- Agentenlauf anhand von Aktionen statt nur Ergebnis bewerten
- Quality Gate für eine LLM-Anwendung entwickeln
- vollständige Teststrategie für einen Praxisfall erstellen

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	3 Tage
Durchführung	Präsenz oder Live Online, vorzugsweise als Inhouse-Schulung
Schwerpunkt	Inhouse-Schulungen für Unternehmen und Teams

Sprache	Deutsch, englische Fachbegriffe und Dokumentation
Unterlagen	Kursunterlagen, Beispiele und Übungsaufgaben
Technik	Eigener Rechner mit Entwicklungsumgebung und Zugang zu einem aktuellen Sprachmodell

> **Inhouse-Schulungen**
> Geeignete eigene Use Cases, Qualitätsanforderungen, RAG-Systeme oder agentische Anwendungen können als Grundlage für Evaluationsdatensätze, Security-Tests und die gemeinsame Teststrategie verwendet werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an Test Engineers, Test Automation Engineers, Quality Engineers, Softwareentwickler, AI Engineers, Softwarearchitekten und Security-affine technische Consultants, die für die Qualität und Absicherung von LLM-basierten Anwendungen verantwortlich sind.

Er eignet sich insbesondere für Teams, die LLM-Anwendungen, RAG-Systeme oder AI Agents nicht nur funktional ausprobieren, sondern systematisch testen, evaluieren und für einen produktiven Einsatz absichern wollen.

> **Voraussetzungen**
> Praktische Erfahrung in Softwaretest, Qualitätssicherung oder Softwareentwicklung wird vorausgesetzt. Grundlegendes Verständnis von generativer KI und LLM-Anwendungen sollte vorhanden sein; Programmierkenntnisse sind für die praktischen Automatisierungsübungen hilfreich. Spezielle Kenntnisse in RAG, MCP oder Agentenarchitekturen werden nicht vorausgesetzt und aus Sicht des Testens so weit erläutert, wie es für die Übungen erforderlich ist.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- Qualitätsmerkmale für LLM-Anwendungen operationalisieren
- testbare Anforderungen und Akzeptanzkriterien formulieren
- geeignete Testorakel für probabilistische Systeme auswählen
- Evaluationsdatensätze und Ground Truth entwickeln
- klassische Testentwurfsverfahren auf LLM-Systeme übertragen
- metamorphe und Robustheitstests entwickeln
- Halluzinationen und Abstention systematisch testen
- automatisierte Evaluationspipelines aufbauen
- deterministische, semantische und modellbasierte Bewertungen kombinieren

- LLM-as-a-Judge kritisch und kontrolliert einsetzen
- Regressionen bei Prompt- und Modelländerungen erkennen
- RAG-Systeme auf Retrieval- und Antwortqualität testen
- Tool-, MCP- und Agentenverhalten prüfen
- Prompt-Injection- und Datenexfiltrationsszenarien entwickeln
- Berechtigungen und Tool-Grenzen verifizieren
- Threat Modeling und Red Teaming in die Teststrategie integrieren
- Performance-, Kosten- und Resilience-Anforderungen testen
- Produktionsbeobachtungen in neue Tests überführen
- Evals und Quality Gates in CI/CD integrieren
- eine vollständige Teststrategie für eine LLM-Anwendung entwickeln

06 FAQ

Häufige Fragen

Was unterscheidet diesen Kurs von KI in Qualitätssicherung und Testautomatisierung?

Jener Kurs betrachtet das gesamte Quality Engineering unter dem Einfluss generativer KI. Dazu gehört sowohl der Einsatz von KI für Testanalyse, Testdesign und Testautomatisierung als auch ein Überblick über das Testen AI-basierter Systeme.

Dieser Kurs spezialisiert sich vollständig auf LLM-basierte Systeme als Testobjekt. Evaluation, RAG-Qualität, Tool Use, Agenten, Prompt Injection, Red Teaming, Regression und Quality Gates werden deshalb wesentlich tiefer behandelt.

Was unterscheidet ihn von der LLM-Anwendungsentwicklung mit Python?

Der Entwicklungskurs behandelt den Aufbau einer LLM-Anwendung einschließlich grundlegender Tests und Evaluation. Dieser Kurs setzt auf dieser Systemklasse auf und konzentriert sich vollständig auf deren Verifikation, Qualitätssicherung, Robustheit und Security.

Ist der Kurs nur für Tester?

Nein. Auch Entwickler, AI Engineers und Architekten profitieren davon, insbesondere wenn sie für die Qualität eigener LLM-Systeme verantwortlich sind. Kenntnisse klassischer Testmethoden sind hilfreich, die relevanten Verfahren werden aber auf LLM-Systeme übertragen und eingeordnet.

Werden klassische Softwaretests durch Evals ersetzt?

Nein. Deterministische Softwarekomponenten sollten weiterhin mit klassischen Unit-, Integrations- und Systemtests geprüft werden. Evals ergänzen diese Verfahren dort, wo probabilistisches Modellverhalten bewertet werden muss.

Was ist der Unterschied zwischen Test und Eval?

Die Begriffe überschneiden sich in der Praxis. Im Kurs steht Eval insbesondere für wiederholbare Bewertungen probabilistischer AI-Ausgaben anhand definierter Datensätze und Kriterien. Diese Evals sind Bestandteil einer umfassenderen Teststrategie.

Wird LLM-as-a-Judge behandelt?

Ja, einschließlich seiner Schwächen. Ein zweites Sprachmodell ist kein objektives Testorakel. Deshalb wird modellbasierte Bewertung mit deterministischen Prüfungen, Referenzdaten und menschlicher Evaluation kombiniert.

Werden RAG-Systeme praktisch getestet?

Ja. Retrieval und Generierung werden zunächst getrennt und anschließend End-to-End bewertet. Dadurch lässt sich feststellen, ob eine schlechte Antwort durch fehlendes Retrieval, falschen Kontext oder die eigentliche Generierung verursacht wurde.

Werden AI Agents getestet?

Ja. Dabei wird nicht nur geprüft, ob ein Agent am Ende das gewünschte Ergebnis erreicht. Auch Aktionsfolge, Tool-Nutzung, Berechtigungen, Abbruchbedingungen und unerlaubte Nebenwirkungen werden als Testgegenstände betrachtet.

Wird Prompt Injection praktisch behandelt?

Ja. Direkte und indirekte Prompt-Injection-Szenarien gehören zum praktischen Teil. Der Schwerpunkt liegt darauf, Sicherheitsannahmen tatsächlich zu testen und nicht nur Schutzanweisungen in einen System Prompt zu schreiben.

Ist das eine Penetration-Testing-Schulung?

Nein. Security Testing und Red Teaming von LLM-Anwendungen sind wichtige Bestandteile, der Kurs ist aber eine Quality-Engineering-Schulung für das Gesamtsystem. Klassisches Infrastruktur- oder Web-Penetration-Testing ist nicht Gegenstand des Kurses.

Kann ich Evals in eine CI/CD-Pipeline integrieren?

Ja. Automatisierte Regressionsevaluation und Quality Gates sind ein eigener Bestandteil des Kurses. Dabei wird auch berücksichtigt, dass probabilistische Ergebnisse andere Freigabekriterien benötigen können als klassische Unit Tests.

Warum dauert der Kurs drei Tage?

LLM-Qualität lässt sich nicht sinnvoll auf einige Prompt-Tests reduzieren. Testorakel, Evaluationsdatensätze, Halluzinationen, RAG, Tool Use, Agenten, Security, Regression, Performance und CI/CD bilden gemeinsam ein eigenständiges Quality-Engineering-Gebiet. Drei Tage erlauben, diese Bereiche praktisch zu verbinden, ohne den Kurs zu einer bloßen Übersicht zu reduzieren.

Kann der Kurs an unsere eigene LLM-Anwendung angepasst werden?

Ja. Bei Inhouse-Schulungen können geeignete eigene Use Cases, Qualitätsanforderungen, RAG-Systeme oder agentische Anwendungen als Grundlage für Evaluationsdatensätze, Security-Tests und die gemeinsame Teststrategie verwendet werden.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/llm-anwendungen-testen/