WHITEPAPER

DAS MADWYN PRINZIP

Make AI do what YOU need
(and not what it thinks you need or wants to do)

GPT von OpenAI und und andere Sprachmodelle liefern beeindruckende Ergebnisse bei der Verarbeitung von Informationen, aber sie lösen nicht alle Probleme zuverlässig. Um sie industrietauglich zu machen, muss man kognitive und algorithmische Lösungsansätze kombinieren. Wie sich der damit verbundene Entwicklungsaufwand durch einen Baukasten reduzieren lässt, erläutert das neue PROSTEP-Whitepaper über das MADWYN-Prinzip.

Inhaltsverzeichnis

Einleitung

Künstliche Intelligenz zeigt heute beeindruckende Fähigkeiten – von der Textgenerierung über komplexe Analysen bis hin zur dynamischen Interaktion mit externen Diensten über Toolcalling und das Model Context Protocol (MCP). Doch in der Praxis wird schnell deutlich: Sprachmodelle tun nicht automatisch das, was gebraucht wird. Sie sind stark in der semantischen Analyse, aber schwach in deterministischen, wiederholbaren Abläufen.

Das **MADWYN**-Prinzip – **Make AI do what YOU need** – fokussiert deshalb auf die bewusste Differenzierung zwischen algorithmischen und kognitiven Lösungsanteilen für komplexe Aufgabenstellungen und ihre flexible Integration in hybride KI-Workflows.

- **Algorithmische Anteile** sichern Stabilität, Nachvollziehbarkeit und Reproduzierbarkeit.
- **Kognitive Anteile** steuern Flexibilität, Sprachverständnis und Mustererkennung bei.

Anhand des Beispiels automatisierter Kapitelzusammenfassungen zeigt sich, dass robuste Lösungen erst durch die gezielte Arbeitsteilung zwischen deterministischer Programmierung und kognitiver Leistung der KI entstehen. Durch einen modularen Baukasten-Ansatz wird der hohe Entwicklungsaufwand, der erforderlich ist, um diese Trennlinie für den jeweiligen Anwendungsfall empirisch zu approximieren, systematisch reduziert: Deklarative Bausteine (Patterns) für algorithmische und kognitive Schritte, wiederverwendbar und auditierbar, ergänzt durch eine Engine, die externe Tools sowohl statisch (REST) als auch dynamisch (MCP) einbindet. Damit lässt sich für eine konkrete Aufgabenstellung die Grenze zwischen algorithmischen und kognitiven Lösungsanteilen flexibel justieren.

So wird aus KI ein verlässliches Werkzeug: transparent, skalierbar und sicher. MADWYN steht für den konsequent zweckgerichteten Einsatz von KI – als Grundlage industrietauglicher Systeme, die nicht nur experimentell funktionieren, sondern nachhaltig Mehrwert schaffen.

Kognitives Fundament – Warum KI nur fast wie ein Mensch denkt

Ein wesentliches theoretisches Fundament für das Verständnis moderner KI-Systeme bildet das von **Daniel Kahneman** entwickelte Modell des menschlichen Denkens, für das er 2002 den **Nobelpreis für Wirtschaftswissenschaften** erhielt. Kahneman beschreibt die menschliche Kognition als Zusammenspiel zweier komplementärer Systeme, die sich funktional wie auch evolutionär unterscheiden.

System 1 steht für das schnelle, automatische und intuitive Denken. Es operiert ohne bewusste Kontrolle, erkennt Muster, reagiert auf Reize und trifft Entscheidungen aufgrund probabilistischer Einschätzungen. Dieses System ist evolutionär älter und bereits bei höheren Säugetieren nachweisbar. Es entstand, um in unsicheren Umgebungen schnelle Reaktionen auf Bedrohungen oder Chancen zu ermöglichen – mit anderen Worten: um zu überleben.

System 2 hingegen ist das bewusste, reflektierte und analytische Denken. Es ermöglicht Sprache, Planung und logisches Schlussfolgern. Es ist energieaufwendiger, aber dafür in der Lage, Verhalten gezielt zu steuern, komplexe Probleme zu lösen und Entscheidungen zu überprüfen.

Kahnemans Modell macht deutlich, dass das menschliche Denken kein einheitlicher, rationaler Prozess ist, sondern das Ergebnis einer evolutionären Schichtung. System 1 bildet die intuitive, impulsive Basis – System 2 die bewusste, prüfende und korrigierende Instanz. Bewusstsein ist in diesem Verständnis keine vom Körper getrennte Entität, sondern eine emergente Eigenschaft neuronaler Aktivität. Unser Gehirn ist unser Bewusstsein – ein physisches System, das Denken erzeugt.

Diese Perspektive hilft, das Verhalten moderner Sprachmodelle besser einzuordnen. **Ein Large Language Model (LLM)** arbeitet auf eine Weise, die funktional an System 1 erinnert: Es erkennt Muster, zieht statistische Schlüsse und erzeugt plausible sprachliche Antworten. Es ist schnell, flexibel und anpassungsfähig – aber ohne eigenes Verständnis, Selbstreflexion oder Zielbewusstsein. KI operiert damit auf der Ebene des schnellen, automatischen Erkennens, nicht auf der des bewussten Verstehens.

Das bedeutet: KI verhält sich in gewissem Sinne wie ein Mensch, dem die reflektierende Instanz fehlt – ein System 1 ohne System 2. Gerade deshalb bedarf ihr Einsatz einer bewussten Strukturierung und Kontrolle durch den Menschen. Nur wenn kognitive Prozesse gezielt durch algorithmische Steuerung ergänzt werden, entstehen Systeme, die zuverlässig, reproduzierbar und erklärbar arbeiten.

Diese Überlegung führt unmittelbar zur Frage, **wo die Grenze zwischen algorithmischen und kognitiven Anteilen** in KI-basierten Anwendungen verlaufen sollte – eine Grenze, die sich mit zunehmender Leistungsfähigkeit moderner Modelle fortwährend verschiebt, deren bewusste Gestaltung aber entscheidend bleibt.

Algorithmisch versus Kognitiv

Die Diskussion um den sinnvollen Einsatz von Künstlicher Intelligenz wird derzeit stark von der Faszination für die Leistungsfähigkeit moderner Sprachmodelle bestimmt. Tatsächlich können diese Modelle beeindruckende Ergebnisse liefern – vom Verfassen natürlichsprachlicher Texte über die Beantwortung komplexer Fragen bis hin zur Analyse umfangreicher Dokumentbestände. Hinzu kommt, dass moderne Ansätze wie Toolcalling und das Model Context Protocol (MCP) die Einsatzmöglichkeiten von KI-Systemen nochmals erheblich erweitern: Modelle können externe Dienste ansprechen, Datenbanken abfragen, Berechnungen anstoßen oder Workflows dynamisch steuern. Damit verschwimmen die Grenzen zwischen rein sprachlicher Verarbeitung und operativer Handlungskompetenz zunehmend.

Doch in der Praxis zeigt sich immer wieder, dass KI-Modelle nicht jedes Problem zuverlässig lösen, und sie tun auch nicht immer das, was wir eigentlich brauchen. Aufgaben im Bereich der KI umfassen Anteile, die sich am besten **algorithmisch** lösen lassen – sie folgen klaren Regeln und liefern reproduzierbare Ergebnisse – sowie Anteile, die **kognitives Verständnis** erfordern, etwa Sprachverstehen und Mustererkennung.

Die Unterscheidung zwischen diesen beiden Bereichen erinnert in gewisser Weise an das Zusammenspiel der beiden Denksysteme, das Daniel Kahneman beschrieben hat. Ein Sprachmodell agiert funktional ähnlich wie das **System 1** des Menschen – es erkennt Muster, reagiert intuitiv und liefert schnelle Ergebnisse, aber ohne Reflexion oder Selbstkontrolle. Die algorithmischen Komponenten eines hybriden Systems übernehmen dagegen die Rolle des **Systems 2:** Sie strukturieren, prüfen, validieren und steuern die Abläufe, um Stabilität und Nachvollziehbarkeit sicherzustellen. Erst im Zusammenspiel beider Ebenen entsteht ein Gesamtverhalten, das sowohl adaptiv als auch zuverlässig ist.

Ein wichtiger Unterschied zum menschlichen Denken besteht jedoch darin, dass KI-Modelle keine eigenständige Instanz der Selbstüberprüfung besitzen. Sie verfügen nicht über ein inneres „System 2", das Ergebnisse hinterfragt oder Widersprüche erkennt. Diesen Mangel kann man jedoch teilweise **durch gezieltes Prompt-Design kompensieren.** Komplexere Prompting-Techniken – etwa **Prompt-Chaining, Selbstreflexions-Prompts oder strukturierte Pattern-Abfolgen** – zwingen das Modell, Zwischenergebnisse zu prüfen, Hypothesen zu vergleichen und explizit Entscheidungsschritte zu formulieren. Dadurch lässt sich eine Form von **externer Reflexion** erzeugen, die das Modell selbst nicht leisten kann, aber methodisch von außen eingebracht wird. In gewisser Weise entsteht so eine simulierte Form von System 2 – ein kontrollierender Rahmen, der durch Sprache, Struktur und Wiederholung das spontane Verhalten des Modells einhegt.

Diese Analogie verdeutlicht, warum KI nicht als autonomes „Denkwesen“ betrachtet werden darf. Ohne die algorithmische Kontrollschicht bleibt sie auf der Ebene des automatischen Reagierens stehen – ein mächtiges, aber ungerichtetes System 1. Die bewusste Integration deterministischer, regelbasierter Strukturen entspricht der menschlichen Fähigkeit zur Reflexion und Korrektur und ist die Voraussetzung dafür, dass KI-Systeme reproduzierbare, auditierbare und industrietaugliche Ergebnisse liefern.

Algorithmisch **Kognitiv**

Die Trennlinie zwischen algorithmischen und kognitiven Anteilen ist dabei nicht statisch. Mit der zunehmenden Leistungsfähigkeit von KI-Modellen, ihren wachsenden Kontextfenstern und der Möglichkeit, externe Tools dynamisch einzubinden, verschiebt sie sich fortlaufend. Aufgaben, die vor wenigen Jahren noch als unlösbar für Maschinen galten, können heute erstaunlich präzise durch ein LLM bearbeitet oder in Kombination mit Tool-Aufrufen erledigt werden.

Dennoch bleibt die Grenze bedeutsam: Wer robuste, nachvollziehbare und industrietaugliche Systeme bauen will, muss bewusst entscheiden, welche Teile einer Aufgabenstellung algorithmisch und welche kognitiv umgesetzt werden. Das Konzept **MADWYN – Make AI do what YOU need** setzt genau hier an. Es geht darum, KI-Modelle nicht einfach „machen zu lassen“, sondern ihre Stärken gezielt dort einzusetzen, wo sie wirklich gebraucht werden – und sie gleichzeitig durch algorithmische Verfahren und kontrolliertes Toolcalling zu ergänzen, wo dies notwendig ist. Damit wird KI vom unkontrollierten Experiment zum verlässlichen Werkzeug.

Die Grenze bestimmen

Die Frage, wo die Trennlinie für einen bestimmten Anwendungsfall verläuft, lässt sich jedoch nicht im Vorhinein theoretisch bestimmen. Sie muss empirisch gefunden werden. Der Grund: Unser Wissen darüber, warum Sprachmodelle in manchen Situationen konsistent arbeiten und in anderen unvorhersehbar reagieren, ist bis heute unvollständig. In vielen Fällen zeigt sich erst in praktischen Tests, wie ein Modell mit einer bestimmten Aufgabenstellung umgeht, welche Prompts es stabil bearbeitet – und wo es versagt.

Das Herausarbeiten dieser Grenze ist deshalb ein iterativer Prozess. **Prototyping und Teilumsetzungen** sind unverzichtbar, um zu erkennen, welche Aufgaben ein Modell zuverlässig bewältigt und welche besser in klassische Softwarelogik ausgelagert werden. Doch dieser Prozess ist aufwendig: Entwicklerinnen und Entwickler investieren viel Zeit, um durch Versuch und Irrtum die richtige Balance zu finden, Prompts zu optimieren und Workflows anzupassen. Ohne methodische Unterstützung droht die Entwicklung in unproduktive Experimente abzugleiten, anstatt Schritt für Schritt in eine robuste, industrietaugliche Lösung zu münden.

Beispiel: Kapitelzusammenfassungen von Dokumenten

Wie anspruchsvoll es ist, die passende Abgrenzung zwischen algorithmischen und kognitiven Lösungsanteilen für eine konkrete Aufgabenstellung zu finden, lässt sich am Beispiel eines KI-Workflows zur Erzeugung von Kapitelzusammenfassungen illustrieren. Ziel ist es, aus umfangreichen PDF-Dokumenten eine Übersicht der Kapitel mit prägnanten Zusammenfassungen zu gewinnen – eine scheinbar einfache Aufgabe, die sich in der Praxis als komplexe Mischung aus algorithmischen und kognitiven Anteilen erweist.

Kognitiver Anteil

Die Analyse der Inhaltsverzeichnisse (TOCs) eines Dokuments gelingt am besten mit einem LLM. Modelle sind in der Lage, Kapitelnummern, Titel und Seitenangaben aus teilweise unstrukturiertem Text zuverlässig zu erkennen. Sie können zudem mit sprachlichen Variationen umgehen, die bei klassischen Parsern schnell zu Fehlern führen würden.

Auch die Zusammenfassung eines einzelnen Kapitels ist eine typische Aufgabe für ein LLM. Allerdings verlassen wir damit auch schon den Bereich einfacher Lösungen, denn damit eine durch ein LLM erzeugte Zusammenfassung auch aussagekräftig ist, darf die Menge des zu analysierenden Textes eine bestimmte Größe nicht überschreiten. Typischerweise sind das ca. 15-30 Seiten Text, je nach Token-Dichte.

Algorithmischer Anteil

Aus der Notwendigkeit, die Menge des jeweils zusammengefassten Textes zu begrenzen, ergeben sich Anforderungen an den Algorithmus. Es ist nicht ausreichend, einfach alle Kapitel auf der obersten Ebene oder der Ebene darunter zu finden und jeweils zusammenzufassen. Vielmehr ist es nötig, Kapitel (egal auf welcher Ebene) zu finden, die die Seitengrenze nicht überschreiten und zunächst für diese jeweils eine Zusammenfassung zu erzeugen. Es gibt Anwendungsfälle, für die das bereits ausreichend ist. Falls nicht, kann man dann die Zusammenfassungen der Teilkapitel durch ein LLM verdichten lassen.

Der Gesamtablauf (ohne abschließende Verdichtung bis auf die oberste Ebene) sieht also so aus:

1. **Kognitiver Schritt: Flachen TOC extrahieren:**
   Das LLM erzeugt eine lineare Liste aller Kapitel und Unterkapitel mit Nummer, Titel und Startseite. Bei sehr komplexen oder tief verschachtelten TOCs sind Sprachmodelle mit der Hierarchisierung überfordert. Deshalb wird dieser Schritt bewusst flach gehalten. Die weitere Verarbeitung der einstufigen Kapitelliste erfolgt dann algorithmisch.

2. **Algorithmischer Schritt: Hierarchischen TOC aufbauen:**
   Aus der flachen Liste wird ein strukturierter Baum erzeugt, indem die Kapitelnummern ausgewertet und Parent-Child-Beziehungen rekonstruiert werden. Jedes Kapitel erhält zusätzlich seine Endseite und einen Verweis auf den Nachfolger.

3. **Algorithmischer Schritt: Arbeitsliste mit maximal n Seiten bilden:**
   Um die Verarbeitung durch das LLM stabil zu halten, dürfen Kapitelabschnitte eine feste Seitengrenze (z. B. 20 Seiten) nicht überschreiten. Zu große Kapitel werden daher rekursiv in kleinere Einheiten zerlegt.

   Der Algorithmus prüft zunächst die Länge jedes Kapitels. Wenn die maximale Länge überschritten wird, werden die Unterkapitel rekursiv durchlaufen. Bleibt auch auf der untersten Ebene ein Kapitel zu groß, wird es in Blöcke von maximal n Seiten geschnitten und in die Arbeitsliste aufgenommen. Ein solches rekursives Vorgehen überfordert jedes LLM und kann daher nur mit deterministischem Code realisiert werden.

4. **Kognitiver Schritt: Kapitelzusammenfassungen erzeugen:**
   Über die Arbeitsliste wird iteriert. Für jedes darin enthaltene Kapitel oder Teilkapitel wird ein weiterer strukturierter Prompt ausgeführt. Ergebnis ist eine prägnante Zusammenfassung des Kapitelinhalts. Diese Aufgabe wiederum kann gut durch ein LLM ausgeführt werden.

Erkenntnis

In der Praxis zeigte sich: Bei sehr komplexen oder tief verschachtelten TOCs waren Sprachmodelle mit der Hierarchisierung und der rekursiven Verarbeitung überfordert. Das führte zur Einführung eines flachen TOC-Formats, das die kognitiven Fähigkeiten des Modells optimal nutzte, während die strenge Strukturlogik algorithmisch abgebildet wurde.

Dieses Zusammenspiel verdeutlicht eine grundlegende Eigenschaft hybrider KI-Systeme: Der algorithmische Anteil übernimmt nicht nur deterministische Aufgaben, sondern kompensiert zugleich die fehlende Reflexionsfähigkeit des Modells. Während das LLM Muster erkennt und Bedeutungen erschließt, definiert die algorithmische Steuerung die Makrostruktur des Prozesse und validiert die Ergebnisse des KI-Modells. Damit entsteht gewissermaßen eine **reflexive Ebene zweiter Ordnung,** die das Modell selbst nicht besitzt, die aber durch algorithmische Kontrolle nachgebildet werden kann.

Der Entwicklungsaufwand, diese Trennlinie empirisch herauszuarbeiten, ist nicht unerheblich – mehrere Iterationen von Prompting, Workflow-Anpassung und algorithmischem Feintuning waren erforderlich, bis der Prozess zuverlässig auch mit Dokumenten von über 600 Seiten und mehr als 10 Seiten Inhaltsverzeichnis ausgeführt wurde.

Entwicklungsaufwand reduzieren mit No-Code KI-Patterns

Das Beispiel der Kapitelzusammenfassungen zeigt, wie viel Feinarbeit notwendig ist, um in einer konkreten Aufgabenstellung eine angemessene Trennung zwischen algorithmischen und kognitiven Aufgaben zu finden und in einem robusten Workflow zu integrieren. Dieser Aufwand entsteht vor allem dadurch, dass Entwicklerinnen und Entwickler bisher tief in Code und Frameworks eingreifen müssen, um Abläufe flexibel zu gestalten und zu variieren.

Ein speziell auf modulare KI-Workflows ausgerichtetes Framework aus sogenannten **Patterns** adressiert genau dieses Problem. Er ersetzt individuelle Ad-hoc-Programmierung durch ein **deklaratives Baukastensystem,** in dem wiederverwendbare Bausteine (Patterns) miteinander kombiniert werden können. So wird aus einem schwer steuerbaren Experiment ein reproduzierbarer, auditierbarer Prozess.

Arten von Pattern

Der Ansatz unterscheidet verschiedene Typen von Pattern, die sowohl algorithmische als auch kognitive Aufgaben abdecken und beliebig kombiniert und ineinander verschachtelt werden können:

- **Prompt:**
  Ein einzelner Prompt an ein LLM, der entweder eine für einen Menschen bestimmte Textantwort oder ein maschinenlesbares Resultat liefert.
  Beispiel: „Extrahiere TOC aus Dokument".
- **Coded Action:**
  Ein algorithmischer Schritt, der deterministisch in Code umgesetzt ist, etwa für Rekursionen, Partitionierungen oder Validierungen.
  Beispiel: Rekursive Aufteilung von zu großen Kapiteln in maximal 20 Seiten.
- **Sequence:**
  Eine Verkettung mehrerer Schritte zu einem linearen Ablauf.
  Beispiel: TOC extrahieren → TOC konsolidieren → über Kapitel iterieren.
- **Iteration:**
  Wiederholung eines Patterns über eine Liste von Eingaben.
  Beispiel: Erzeuge für jedes Kapitel in der Arbeitsliste eine Zusammenfassung.
- **Bedingung (If-Then-Else):**
  Steuerung des Ablaufs abhängig von Zwischenergebnissen.
- **Loop:**
  Wiederholung eines Ablaufs, solange eine Bedingung erfüllt ist.
- **Composite Patterns (Voter, Concurrent):**
  Erweiterte Steuerungsmuster:
  Voter: Mehrfachausführung des gleichen Schrittes mit Ergebnisvergleich.
  Concurrent: Parallele Ausführung unterschiedlicher Patterns mit nachgelagerter Aggregation.

Jedes dieser Patterns kapselt eine klar abgegrenzte Funktionalität. Dadurch sind Workflows transparent, nachvollziehbar und lassen sich modular zusammensetzen. In ihrer Gesamtheit bilden diese Pattern eine Art **externe Reflexionsschicht:** Sie geben dem Ablauf eine verlässliche Struktur, die das KI-Modell alleine nicht herstellen kann und ermöglichen Konsistenzprüfungen, Wiederholungen und Entscheidungen auf Basis mehrerer Ergebnisse. Damit ergänzen sie die

fehlende Selbstkontrolle des LLMs durch eine algorithmisch strukturierte Form von Selbstüberwachung. Besonders durch Pattern-Ketten (Prompt-Chaining) lässt sich eine iterative, schrittweise Annäherung an stabile Ergebnisse realisieren – ein Prozess, der dem menschlichen Nachdenken über Zwischenergebnisse funktional ähnelt.

Engine und Toolcalling

Patterns werden von einem Programm ausgeführt, das als Pattern Execution Engine (kurz Engine) bezeichnet wird. Die Engine liest die Definition des Patterns und führt es aus. Im einfachsten Fall ist das ein einzelner Prompt gegen ein LLM, der ein Ergebnis erzeugt, das die Engine zurückgibt. Für strukturierte Patterns besteht die Ausführung darin, die enthaltenen Patterns nach der jeweiligen Logik rekursiv aufzurufen. So besteht die Ausführung einer Sequence z.B. darin, die Patterns, die die einzelnen Schritte definieren, nacheinander auszuführen, wobei das Ergebnis eines vorhergehenden Schrittes als Input des nächsten verwendet werden kann.

Um algorithmische Anteile in die Workflows zu integrieren, steht zusätzlich das Mittel externer Toolcalls zur Verfügung. Ein Tool bezeichnet in diesem Kontext eine remote aufrufbare Funktion, die auf einem entfernten Server eine Aktion ausführt, z.B. Daten ermittelt oder erzeugt. Einem Pattern werden diese Tools über Toolsets zur Verfügung gestellt. Ein Toolset bezeichnet dabei lediglich eine Menge verschiedener Tools, die zu einer Verwaltungseinheit gruppiert sind. Dabei gibt es zwei Betriebsarten, die sich ausschließlich in der Art der Ermittlung der Tool-Definitionen unterscheiden:

- **Statische Tool-Definitionen (REST):**
  Die verfügbaren Tools (Funktionen mit Parameterdefinition) werden vorab als JSON beschrieben und der Engine bekannt gemacht. Zur Laufzeit stellt die Engine dieses Toolset dem LLM im Prompt zur Verfügung. Das LLM entscheidet, welche Tools es aufruft; die Engine führt die angeforderten Calls aus und schleift die Ergebnisse in den nächsten Prompt ein.
- **Dynamische Tool-Definitionen (MCP – Model Context Protocol):**
  Die Engine entdeckt zur Laufzeit die verfügbaren Tools über die Discovery-Funktionalität des MCP-Toolservers (inkl. Signaturen/Schemata) und macht sie dem LLM im jeweiligen Kontext verfügbar. Auch hier entscheidet das LLM, welche der gefundenen Tools es aufruft; die Engine orchestriert die Requests/Responses in einer Schleife, bis das Modell statt weiterer Toolcalls eine finale Antwort liefert.

Unabhängig von der Betriebsart folgt die Ausführung demselben Muster: Wird ein Pattern ausgeführt, dem Tools zur Verfügung stehen, übergibt die Engine dem KI-Modell den aktuellen Prompt inklusive einer Liste der verfügbaren Tools. Das Modell fordert Toolaufrufe an, die Engine führt aus und ergänzt die Ergebnisse im nächsten Prompt – bis die finale Antwort entsteht.

Was das mit MADWYN zu tun hat

Externe Toolaufrufe stellen eine einfache und sehr effektive Möglichkeit dar, algorithmische Funktionalität in einen KI-Workflow zu integrieren.

- Statisch definierte REST-Tools geben maximale Planbarkeit (z. B. für streng geregelte, auditierbare Schritte) und erlauben die Integration herkömmlicher REST-Server.
- MCP ermöglicht höchste Flexibilität, wenn sich Tool-Landschaften ändern oder erweitert werden – ohne dass Toolsets neu definiert oder angepasst werden müssen.

In beiden Modi bleibt die kognitive Entscheidung beim LLM (welches Tool, in welcher Reihenfolge), während die Engine die algorithmische Orchestrierung übernimmt (Aufruf, Fehlerbehandlung, Retrys, Kontext-Update).

Es ist Aufgabe des Prompt-Engineers, den Freiheitsgrad des LLM bei der Auswahl der Tools zu steuern. Durch präzises Prompt-Design kann er das Modell entweder stark leiten (z. B. „nutze zuerst Tool A, dann Tool B“) oder mehr Entscheidungsspielraum geben, um adaptive Lösungen zu ermöglichen. Damit lässt sich sicherstellen, dass die gewünschten Ergebnisse zuverlässig und wiederholbar erreicht werden.

Industrietauglichkeit und Robustheit

Der Schritt von einem funktionsfähigen Prototyp hin zu einer produktiv eingesetzten KI-Lösung ist anspruchsvoll. Während im Prototyp noch experimentiert wird und Fehlversuche tolerierbar sind, gelten im industriellen Betrieb andere Maßstäbe: Zuverlässigkeit, Nachvollziehbarkeit und Skalierbarkeit sind zwingend erforderlich.

Auditierbarkeit und Nachvollziehbarkeit

Ein wesentliches Ziel industrietauglicher Systeme ist, dass Abläufe jederzeit überprüfbar sind. Mit dem Pattern-Ansatz wird jeder Schritt eines Workflows explizit beschrieben und kann dokumentiert, versioniert und getestet werden. Entscheidungen des LLM werden nachvollziehbar, weil sowohl die Prompts als auch die genutzten Tools und deren Ergebnisse protokolliert sind. Damit entsteht ein Audit-Trail, der sowohl für interne Qualitätssicherung als auch für regulatorische Anforderungen essenziell ist.

Skalierbarkeit und Wiederholbarkeit

Industrielle Systeme müssen große Mengen von Aufgaben zuverlässig bewältigen. Reine LLM-basierte Lösungen stoßen hier schnell an Grenzen, weil Modelle nicht konsistent deterministisch arbeiten. Durch die algorithmischen Patterns – etwa zur Partitionierung von Dokumenten, zur Iteration über Arbeitslisten oder zur Validierung von Ergebnissen – entsteht die notwendige Stabilität. Gleichzeitig erlaubt die Engine durch Features wie **parallele Ausführung** von Iterationen oder „Voter"-Patterns, Ergebnisse mehrerer Modellläufe zu vergleichen und nur die stabilsten auszuwählen.

Sicherheit und Kontrolle

In Unternehmensumgebungen spielt auch der Schutz von Daten und Zugriffsrechten eine zentrale Rolle. Da Toolsets deklarativ definiert und gezielt eingeschränkt werden können, lässt sich exakt steuern, welche externen Funktionen einem LLM zugänglich sind.

Von der Idee zum Betrieb

Das Pattern-Prinzip unterstützt den gesamten Lebenszyklus:

- **Prototyping:** Schnelles Experimentieren mit neuen Abläufen durch Kombination von Patterns.
- **Pilotierung:** Validierung im kleinen Maßstab mit klarer Dokumentation der Workflows.
- **Produktion:** Skalierbarer Betrieb mit klaren Kontrollmechanismen, Audit-Trails und Tool-Integrationen.

So wird aus einem experimentellen Workflow ein **industrietaugliches KI-System,** das die Stärken kognitiver Modelle gezielt nutzt, ohne die Anforderungen an Robustheit und Sicherheit zu vernachlässigen.

Fazit

Das zentrale Anliegen von **MADWYN – Make AI do what YOU need** ist es, Künstliche Intelligenz nicht unkontrolliert „machen zu lassen", sondern sie gezielt dort einzusetzen, wo ihre Stärken liegen – und sie dort zu begrenzen, wo sie strukturell schwach ist.

Die Erfahrungen zeigen klar:

- **Sprachmodelle sind stark** in der semantischen Analyse, im Sprachverständnis und in der flexiblen Mustererkennung.
- **Sprachmodelle sind schwach** in streng deterministischen, wiederholbaren Abläufen.

Industrietaugliche Lösungen entstehen erst durch das sorgfältig austarierte Zusammenspiel algorithmischer und kognitiver Verfahren. Der Pattern-Ansatz bietet hierfür das methodische Fundament:

- Er macht komplexe KI-Workflows deklarativ beschreibbar, nachvollziehbar und wiederholbar.
- Er ermöglicht die flexible Kombination von LLM-Schritten mit deterministischen Algorithmen.
- Er integriert externe Tools über REST oder MCP – gesteuert durch die Engine, aber kognitiv genutzt vom LLM.
- Er gibt dem Prompt-Engineer die Möglichkeit, das LLM so zu leiten, dass die Ergebnisse zuverlässig und konsistent bleiben.

Damit wird KI vom Experiment zum verlässlichen Werkzeug, das transparent, kontrollierbar, auditierbar und skalierbar ist. MADWYN bedeutet, KI konsequent zweckgerichtet einzusetzen – und mit dem Pattern-Ansatz steht dafür die geeignete Architektur bereit.

PDF Version des Whitepapers:
www.prostep.com/whitepaper
oder scannen Sie den QR Code

Sie haben Anmerkungen oder Fragen?

Wir freuen uns auf Ihr Feedback an
infocenter@prostep.com

IMPRESSUM

Herausgeber
PROSTEP AG

Ansprechpartner:
Dr. Norbert Lotter
norbert.lotter@prostep.com

Edition 1, 2025

PROSTEP AG
Heinrich-Hertz-Straße 3-7 · 64295 Darmstadt · Deutschland