WHITEPAPER

THE MADWYN PRINCIPLE

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

GPT by OpenAI and other language models deliver impressive results in processing information – but they don't always solve problems reliably. To make them suitable for industrial use, cognitive and algorithmic solutions must be combined. The new PROSTEP white paper on the MADWYN principle explains how a modular approach can significantly reduce the development effort associated with this integration.

Table of Contents

Abstract

Artificial intelligence today demonstrates impressive capabilities – from text generation and complex analysis to dynamic interaction with external services through tool calling and the Model Context Protocol (MCP). Yet in practice, it quickly becomes evident that language models do not automatically do what is actually needed. They excel at semantic analysis but are weak in deterministic, repeatable processes.

The **MADWYN** principle – **Make AI do what YOU need** – therefore focuses on the deliberate differentiation between algorithmic and cognitive solution elements for complex tasks and their flexible integration within hybrid AI workflows.

- **Algorithmic elements** ensure stability, traceability, and reproducibility.
- **Cognitive elements** provide flexibility, language understanding, and pattern recognition.

Using the example of automated chapter summarization, it becomes obvious that robust solutions emerge only through the deliberate division of labor between deterministic programming and the cognitive capabilities of AI. The significant development effort required to empirically approximate this boundary for each specific use case is systematically reduced by a modular, pattern-based approach: declarative building blocks (patterns) for algorithmic and cognitive steps, reusable and auditable, complemented by an engine that integrates external tools both statically (REST) and dynamically (MCP). This makes it possible to flexibly adjust the boundary between algorithmic and cognitive solution components for any given task.

In this way, AI becomes a reliable instrument – transparent, scalable, and secure. MADWYN stands for the consistent and purpose-driven use of AI as the foundation of systems suitable for the industry that not only work experimentally but deliver sustainable value in real-world environments.

Cognitive Foundation – Why AI Only Thinks Almost Like a Human

A key theoretical foundation for understanding modern AI systems is the model of human thinking developed by **Daniel Kahneman,** who received the 2002 **Nobel Prize in Economic Sciences** for it. Kahneman describes human cognition as the interplay of two complementary systems that differ both functionally and evolutionarily.

System 1 represents fast, automatic, and intuitive thinking. It operates without conscious control, recognizes patterns, reacts to stimuli, and makes decisions based on probabilistic assessments. This system is evolutionarily older and can already be observed in higher mammals. It evolved to enable rapid responses to threats or opportunities in uncertain environments — in other words, to survive.

System 2, on the other hand, stands for conscious, reflective, and analytical thinking. It enables language, planning, and logical reasoning. It is more energy-intensive but capable of intentional behavioral control, complex problem-solving, and self-monitoring of decisions.

Kahneman's model makes clear that human thought is not a single rational process but rather the result of an evolutionary layering. System 1 provides the intuitive, impulsive basis, while System 2 acts as the conscious, examining, and corrective instance. In this understanding, consciousness is not a separate, immaterial entity but an emergent property of neural activity. Our brain is our consciousness — a physical system that generates thought.

This perspective helps to better interpret the behavior of modern language models. A **Large Language Model (LLM)** operates in a way functionally similar to System 1: it recognizes patterns, draws statistical inferences, and generates plausible linguistic responses. It is fast, flexible, and adaptive – but lacks genuine understanding, self-reflection, or goal awareness. Artificial intelligence thus operates on the level of rapid, automatic recognition, not on that of conscious comprehension.

In a sense, AI behaves like a human without the reflective instance – a System 1 without System 2. This is precisely why its use requires deliberate structuring and control by humans. Only when cognitive processes are intentionally complemented by algorithmic control can systems be built that operate reliably, reproducibly, and in an explainable way.

This consideration leads directly to the question of **where the boundary between algorithmic and cognitive components** in AI-based applications should lie – a boundary that continues to shift as models become more capable, but whose **conscious design** remains essential.

Algorithmic vs. Cognitive

The discussion about the most efficient use of artificial intelligence is currently dominated by the fascination with the capabilities of modern language models. Indeed, these models can deliver impressive results – from writing natural text to answering complex questions and analyzing large document collections. In addition, technically advanced concepts such as external tool calling and the Model Context Protocol (MCP) have further expanded the application potential of AI systems: models can now interact with external services, query databases, trigger computations, or dynamically control workflows. As a result, the boundaries between pure language processing and operational action competence are becoming increasingly blurred.

However, practical experience consistently shows that AI models do not solve every problem reliably and efficiently – and they do not always do what is actually needed. Every AI-related task includes parts that are best solved **algorithmically** – following clear rules and producing reproducible results—and others that rely on **cognitive processing** such as pattern recognition or language understanding.

The distinction between these two domains is reminiscent of the interplay between the two systems of thought described by Daniel Kahneman. A language model functions much like the human **System 1** – it recognizes patterns, reacts intuitively, and produces rapid results, but without reflection or self-control. The algorithmic components of a hybrid system, by contrast, take on the role of **System 2:** they structure, verify, validate, and guide the processes to ensure stability and traceability. Only through the interaction of both levels does an overall behavior emerge that is both adaptive and reliable.

An important difference from human thinking, however, is that AI models possess no intrinsic mechanism for self-monitoring.They lack an internal System 2 that could question results or detect contradictions. This shortcoming can, however, be partially compensated **through deliberate prompt design.** More advanced prompting techniques — such as **prompt chaining, self-reflection prompts, or structured pattern sequences** – compel the model to review intermediate results, compare hypotheses, and articulate explicit decision steps.In this way, a form of **external reflection** can be created — something the model itself cannot perform, but which can be methodically imposed from the outside. In a sense, this creates a simulated form of System 2 – a controlling framework that constrains the model's spontaneous behavior through language, structure, and repetition.

This analogy illustrates why AI should not be regarded as an autonomous "thinking entity." Without the algorithmic control layer, it remains at the level of automatic reaction — a powerful but directionless System 1. The deliberate integration of deterministic, rule-based structures corresponds to the human capacity for reflection and correction – and is the prerequisite for building AI systems that are reproducible, auditable, and suitable for industrial use.

Algorithmic **Cognitive**

Until AI models appeared and were made available to a wide audience of developers, the cognitive part of a task required human interaction or was approximated by more or less complex heuristic processing. AI models now allow us to delegate certain classes of cognitive task elements to a machine and get remarkably appropriate results. The boundary between the two domains, however, is not static. With the growing capabilities of AI models, their expanding context windows, and the ability to dynamically integrate external tools, this boundary continues to shift. Tasks that only a few years ago were considered unsolvable by machines can today be handled with remarkable precision by an LLM, often in combination with external tool calls.

Nevertheless, the distinction remains crucial: anyone seeking to build robust, traceable, and industrial-grade AI-powered systems must deliberately decide which parts of a given task should be implemented algorithmically and which should rely on cognitive processing by an LLM. The **MADWYN principle – Make AI do what YOU need** – addresses this exact challenge. It is about not letting AI models "just do something," but instead deploying their strengths purposefully where they truly add value, while complementing them with algorithmic procedures and controlled tool calling where necessary. In doing so, AI evolves from an uncontrolled experiment into a reliable instrument.

Finding the Boundary

The question of where the boundary between algorithmic and cognitive elements should be drawn for a given task or use case cannot be determined theoretically in advance; it must be found empirically. The reason is that our understanding of why language models behave consistently in some situations and unpredictably in others remains incomplete. In many cases, only practical testing reveals how a model handles a given task – which prompts it processes reliably, and where it fails.

The process of identifying this boundary is therefore inherently iterative. **Prototyping and partial implementations** are essential to determine which tasks a model can handle reliably and which are better delegated to traditional software logic. However, this process is time-consuming: developers must invest considerable effort to find the right balance through trial and error, optimizing prompts, and refining workflows. Without a structured methodological framework, development tends to drift into unproductive experimentation instead of progressing systematically toward a robust, industrial-grade solution.

Example: Chapter Summarization of Documents

The challenge of identifying the proper boundary between algorithmic and cognitive elements of a task can be illustrated using the example of an AI workflow for generating chapter summaries. The objective is to create concise summaries of chapters from extensive PDF documents – a seemingly straightforward task that, in practice, turns out to be a complex blend of algorithmic and cognitive elements.

Cognitive Component

Analyzing the Table of Contents (TOC) of a document works best with a Large Language Model (LLM). The model can reliably identify chapter numbers, titles, and page references even in partially unstructured text. It can also handle linguistic variations that would quickly cause errors in conventional parsers or require extensive heuristics like pattern matching.

Summarizing an individual chapter is likewise a typical cognitive task for an LLM. However, this already pushes the boundaries of simple use cases: for an LLM-generated summary to remain meaningful, the amount of text to be analyzed must stay within a certain limit – typically around 15–30 pages, depending on token density.

Algorithmic Component

This limitation on input size imposes corresponding requirements on the algorithmic part. It is not sufficient to simply identify all chapters at the top or second level and summarize them. Instead, it is necessary to detect chapters — regardless of their depth in the hierarchy — that do not exceed the page limit, and to generate summaries for these first. For some use cases, this level of summarization is sufficient; if not, the resulting subchapter summaries can then be further condensed by an LLM.

The overall workflow (without the final condensation step up to the top level) looks like this:

1. **Cognitive Step: Extract a flat TOC**
   The LLM produces a linear list of all chapters and subchapters with number, title, and start page. For highly complex or deeply nested TOCs, language models tend to fail at hierarchical reconstruction. Therefore, this step is deliberately kept *flat*. The subsequent processing of this one-level chapter list is handled algorithmically.

2. **Algorithmic Step: Build a hierarchical TOC**
   From the flat list, a structured tree is created by evaluating chapter numbers and reconstructing parent–child relationships. Each chapter additionally receives its end page and a reference to its successor.

3. **Algorithmic Step: Create a worklist with a maximum of *n* pages per unit**
   To maintain stability in LLM processing, chapter sections must not exceed a fixed size limit (e.g., 20 pages). Oversized chapters are therefore recursively divided into smaller units.

   The algorithm checks the length of each chapter: if the limit is exceeded, it traverses the subchapters recursively. If even the lowest level remains too large, the chapter is split into blocks of up to n pages and added to the worklist. Such recursive decomposition would overwhelm any LLM and can only be achieved through deterministic code.

4. **Cognitive Step: Generate chapter summaries**
   The algorithm iterates over the worklist. For each chapter or subchapter, a structured prompt is executed. The result is a concise, well-formed summary of the chapter's content - a task that plays to the strengths of the LLM.

Insight

In practice, it became evident that language models struggled with hierarchical TOCs and recursive processing in very large or deeply nested documents. This observation led to the introduction of a flat TOC format, which leveraged the model's cognitive strengths while representing strict structural logic algorithmically.

This interplay reveals a fundamental property of hybrid AI systems: The algorithmic component not only performs deterministic tasks but also compensates for the model's lack of self-reflection. While the LLM recognizes patterns and extracts meaning, the algorithmic controller defines the macrostructure of the process and validates the model's outputs. In doing so, it establishes a kind of **second-order reflective layer** – something the model itself does not possess but which can be recreated through algorithmic control.

The effort required to empirically determine this dividing line was considerable: several iterations of prompt design, workflow adjustment, and algorithmic fine-tuning were necessary before the process reliably handled documents exceeding 600 pages with more than 10 pages of Table of Contents.

Reducing Development Effort with No-Code AI Patterns

The chapter summarization example illustrates the challenge of integrating both types of processing into a robust workflow. This effort arises primarily because developers must still work deeply within code and frameworks to make processes flexible and adaptable.

A framework specifically designed for modular AI workflows – based on so-called **patterns** – addresses this exact problem. It replaces individual ad-hoc programming with a **declarative building-block system** in which reusable patterns can be combined in various configurations. What was once a difficult-to-control experiment is becoming a reproducible and auditable process.

Types of Patterns

The approach distinguishes between several types of patterns that cover both algorithmic and cognitive tasks and can be freely combined or nested within one another:

- **Prompt:**
  A basic step, such as a single prompt to an LLM or a simple algorithmic operation that delivers either a human-readable response or a machine-readable intermediate result that is bound to be processed further.
  Example: "Extract TOC from document."
- **Coded Action:**
  An algorithmic step implemented deterministically in code, for instance, recursion, partitioning, or validation.
  Example: Recursive subdivision of large chapters into sections of a maximum of 20 pages.
- **Sequence:**
  A chain of multiple steps executed in linear order.
  Example: Extract TOC → Consolidate TOC → Iterate over chapters.
- **Iteration:**
  Repetition of a pattern over a list of inputs.
  Example: Generate a summary for each chapter in the worklist.
- **Condition (If-Then-Else):**
  Control of workflow execution based on intermediate results.
- **Loop:**
  Repetition of a sequence as long as a defined condition is met.
- **Composite Patterns (Voter, Concurrent):**
  Advanced control structures:
  Voter: Executes the same step multiple times and compares results.
  Concurrent: Executes different patterns in parallel with subsequent aggregation.

Each of these patterns encapsulates a clearly defined functionality. This makes workflows transparent, traceable, and modularly composable. Taken together, the patterns form a kind of **external reflection** layer: they impose a reliable structure on the workflow - one that the AI model alone could not establish - and enable consistency checks, repetitions, and decisions based on multiple results. In doing so, they complement the missing self-control of the LLM with an algorithmically structured form of self-monitoring.

Through pattern chains (prompt chaining) in particular, an iterative, step-by-step convergence toward stable results can be achieved – a process functionally similar to human reasoning about intermediate outcomes. This iterative reflection does not occur *inside* the model but is instead simulated through procedural structure, allowing cognitive exploration to be framed by deterministic logic.

Engine and Tool Calling

Patterns are executed by a program called the Pattern Execution Engine – or simply the Engine. The Engine reads the pattern definition and performs it. In the simplest case, this means sending a single prompt to an LLM and returning its result. For structured patterns, execution involves recursively invoking the contained sub-patterns according to their defined logic. For example, executing a Sequence means running the patterns that define its steps in order, where the output of one step may serve as the input to the next.

To integrate algorithmic elements into these workflows, the Engine also supports external tool calls. A tool in this context refers to a remotely callable function that performs an operation on an external server – for example, retrieving or generating data. Patterns access such functions through toolsets. A toolset is simply a collection of tools grouped into a single management unit, made available to the Engine when executing a pattern. There are two operational modes, which differ only in how the tool definitions are obtained:

- **Static Tool Definitions (REST):**
  The available tools (operations, parameters, schemas) are described in JSON and provided to the engine in advance. At runtime, the engine makes this toolset available to the LLM within the prompt. The LLM decides which tools to call; the engine executes the requested calls and feeds the results back into the next prompt.
- **Dynamic Tool Definitions (MCP – Model Context Protocol):**
  The engine discovers the available tools at runtime via the discovery functionality of the MCP tool server (including signatures and schemas) and exposes them to the LLM in the current context. Here too, the LLM decides which-one of the dynamically discovered tools to call. The engine orchestrates the sequence of requests and responses in a loop until the model produces a final answer instead of further tool calls.

Regardless of the mode, the execution cycle follows the same logic: when a pattern with available tools is executed, the engine provides the AI model with the current prompt and a list of accessible tools. The model requests tool calls, the engine executes them, and the results are incorporated into the subsequent prompt—until a final answer is generated.

Relation to the MADWYN Principle

External tool calls represent a simple yet highly effective means of integrating algorithmic functionality into an AI workflow.

- Statically defined REST tools offer maximum predictability (e.g., for strictly regulated, auditable steps) and allow integration with conventional REST services.
- MCP provides maximum flexibility when tool landscapes change or expand – without the need to redefine or modify toolsets.

In both modes, the cognitive decision remains with the LLM (which tool to use, in what order), while the engine handles the algorithmic orchestration (execution, error handling, retries, and context updates).

It is the responsibility of the prompt engineer to control how much freedom the LLM has in selecting tools. Through precise prompt design, the engineer can tightly guide the model (e.g. “use Tool A first, then Tool B”) or allow greater autonomy to enable adaptive behavior. Only by carefully managing this balance can reliable and repeatable results be ensured.

Industrial Maturity and Robustness

The transition from a functional prototype to a production-grade AI solution is a demanding process. While experimentation and failure are acceptable during prototyping, industrial deployment requires a different set of standards: reliability, traceability, and scalability are essential prerequisites.

Auditability and Traceability

A key objective of systems apt for industrial use is the ability to verify processes at any time. With the pattern-based approach, every step within a workflow is explicitly defined and can be documented, versioned, and tested. The decisions made by the LLM become traceable because both the prompts and the tools used – including their results – are logged. This creates an audit trail that is crucial for internal quality assurance as well as for regulatory compliance.

Scalability and Repeatability

Industrial systems must be capable of reliably handling large volumes of tasks. Purely LLM-based solutions quickly reach their limits because such models do not behave in a strictly deterministic manner. The introduction of algorithmic patterns – for instance, for document partitioning, list iteration, or result validation – provides the required stability. At the same time, the engine's capabilities, such as **parallel execution** of iterations or voter patterns, make it possible to compare the outcomes of multiple model runs and retain only the most consistent results.

Security and Control

In enterprise environments, the protection of data and access rights is of paramount importance. Since toolsets are defined declaratively and can be precisely restricted, it is possible to control exactly which external functions are accessible to an LLM.

From Concept to Operation

The pattern principle supports a structured development cycle of an AI solution:

- **Prototyping:** Rapid experimentation with new workflows through flexible pattern combinations.
- **Pilot Phase:** Small-scale validation with clearly documented workflows.
- **Production:** Scalable operation with well-defined control mechanisms, audit trails, and tool integrations.

In this way, an experimental workflow evolves into an **industrial-grade AI system** that leverages the cognitive strengths of modern AI models while maintaining the necessary robustness and security for real-world operation.

Conclusion

The core objective of **MADWYN – Make AI do what YOU need** is not to let artificial intelligence act unchecked, but to deploy it purposefully where its strengths lie – and to constrain or avoid it where it is structurally weak.

Solution apt for industry solutions require a carefully balanced combination of deterministic control and cognitive flexibility. The pattern-based approach provides the methodological foundation for this:

- It makes complex AI workflows declaratively definable, transparent, and reproducible.
- It enables the flexible combination of LLM-driven steps with deterministic logic.
- It integrates external tools via REST or MCP – orchestrated by the engine but cognitively leveraged by the LLM.
- It gives the prompt engineer the ability to guide the model so that results remain reliable and consistent.

In this way, AI evolves from an experimental technology into a reliable instrument that is transparent, controllable, auditable, and scalable. MADWYN stands for the consistent, purpose-driven use of artificial intelligence – and the pattern-based approach provides the architecture that makes it a practical reality.

PROSTEP GROUP

PDF version of the white paper:
www.prostep.com/whitepapers
or scan the QR Code

Do you have any comments or questions?

We look forward to your feedback at
infocenter@prostep.com

LEGAL NOTICE

Published by
PROSTEP AG

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

Edition 1, 2025

PROSTEP AG
Heinrich-Hertz-Strasse 3-7 · 64295 Darmstadt · Germany