FROM AD-HOC PROMPTS TO DESIGN PATTERNS: A CRITERIA-DRIVEN REVIEW FOR LLMS

Лановий О.Ф., Скрипченко М. Д.

Харківський національний університет
радіоелектроніки
м. Харків, Україна

E-mail: oleksiy.lanovyy@nure.ua, myky-
ta.skrypchenko@nure.ua

Abstract

This review pursues two aims: (A1) define explicit criteria for assessing prompt patterns — factual accuracy, structure adherence/controllability, cost/latency, robustness to perturbations, reproducibility, and safety/policy compliance — and (A2) derive practical recommendations on which pattern families work best under specific conditions. We consolidate published "prompt pattern" approaches into seven families (Input Semantics, Output Customization, Error Identification, Prompt Improvement, Cognitive Verifier, Refusal Breaker, Interaction), summarize reported benefits and limitations of LLMs, and map patterns to evaluation practices (LLM-as-a-Judge, Agent-as-a-Judge, voting) and agentic workflows (RAG/StructRAG, memory, tools). Beyond synthesis, the contribution is a criteria-driven framework and gap analysis that inform design guidelines for reliable, efficient prompting pipelines and outline directions for model/agent development (e.g., structure-aware retrieval and bounded verification). Main conclusions: (i) for safety- or fact-critical tasks, combine Output Customization (schema/templating) with Error Identification and retrieval; (ii) for integration into pipelines, prefer Output Customization; (iii) for ambiguous domains, start with Input Semantics; (iv) for scarce budget/latency, avoid heavy interaction patterns; (v) use Cognitive Verifier selectively for math/logic.

1. Introduction

Large language models (LLMs) have increased demand for reproducible prompting practices that go beyond ad-hoc instructions. Practitioners describe prompt patterns — reusable prompting solutions — often by analogy to software design patterns. However, terminology and evaluation remain inconsistent. This review addresses those gaps by introducing explicit criteria and context-dependent recommendations.

Research questions. RQ1. What definitions best distinguish prompts, patterns, and their roles in shaping LLM behavior? RQ2. Which pattern families recur most often, and how do they relate (Input Semantics, Output Customization, Error Identification, Prompt Improvement, Cognitive Verifier, Refusal Breaker, Interaction)? RQ3. What benefits and failure modes are most frequently reported? RQ4. How are patterns assessed — benchmarks, LLM-as-a-Judge, Agent-as-a-Judge, multi-model voting? RQ5. How do patterns interface with agent architectures, memory mechanisms, and retrieval (e.g., RAG/StructRAG)?

1.1. Criteria for Analysis.

To make comparisons concrete, we evaluate pattern families along six criteria:

1. Factual accuracy (does the pattern reduce hallucinations / factual errors?).

2. Structure adherence / controllability (does output match a specified schema/format, enabling automation?).

3. Cost & latency (tokens, steps, and wall-clock time).

4. Robustness (sensitivity to prompt paraphrase, numeric changes, and distractors). 5.

Reproducibility (variance across runs/models).

6. Safety & policy compliance (propensity to refuse appropriately / avoid unsafe content).

1.2. Definitions and Core Concepts

Prompt patterns are reusable prompting solutions — by analogy to software design patterns — adapted to steering LLMs' behavior and outputs.

1.3. Key terms

Prompt. An instruction to an LLM that sets rules/constraints, automates steps, and targets specific qualities (and quantities) in the output.

Pattern. A reusable solution to a recurring problem encountered when interacting with LLMs across tasks and domains.

Prompt pattern. A lightweight "programming" device that shapes both the model's outputs and the interaction flow.

We adopt the following working definitions: a prompt is an instruction that sets rules/constraints and target qualities; a pattern is a reusable solution to a recurring prompting problem; a prompt pattern is a lightweight programming device shaping outputs and interaction flow. These definitions ground all subsequent comparisons and recommendations.

1.4. Catalog Structure and Classification

We group recurrent prompting solutions into seven families and compare each against the six criteria to enable criteria-driven (not ad-hoc) analysis and application [1]. The families are:

1. Input Semantics (Meta-Language Creation, controlled vocabulary/glossary bootstrapping). Typical strengths/risks. Improves accuracy and robustness by disambiguating terms; low cost; limited structure control unless combined with templates.

2. Output Customization (Templates, JSON/schema-constrained output, Persona, Visualization Generator, Recipe). Typical strengths/risks. Highest structure adherence and reproducibility; generally low latency; may mask errors if schema is satisfied but content is wrong — pair with verification.

3. Error Identification (Fact-Check List, Reflection, cite-to-source prompts). Typical strengths/risks. Raises accuracy and safety; adds cost/latency; effectiveness depends on retrieval/evidence availability.

4. Prompt Improvement (Question Refinement, Alternative Approaches, self-editing). Typical strengths/risks. Increases robustness to underspecified tasks; moderate extra cost; gains taper if initial prompts are already strong.

5. Cognitive Verifier (constrained CoT, self-consistency, unit checks). Typical strengths/risks. Boosts accuracy on math/logic/program synthesis; increases latency; must cap reasoning depth to control cost.

6. Refusal Breaker (with explicit guardrails). Typical strengths/risks. Reduces false refusals; safety risk if misused — apply only to correct misclassification, not to bypass policies.

7. Interaction (plan-critique-revise loops, multi-turn debate). Typical strengths/risks. Best for open-ended exploration; highest cost/latency; variable reproducibility; use when discovery outweighs budget constraints.

The literature consistently yields seven families — Input Semantics, Output Customization, Error Identification, Prompt Improvement, Cognitive Verifier, Refusal Breaker (guardrailed), Interaction — each exhibiting distinct trade-offs on the six criteria. This catalog provides the reference set we evaluate and use downstream.

1.5. Benefits and Use Cases of Patterns

Using a catalog of prompt patterns offers significant benefits for people working with LLMs: 1. Higher quality and control of outputs. Patterns help you deliberately shape dialogues with LLMs, enabling high-quality generations and reliable execution of complex, automated tasks. 2. Clearer communication. A well-crafted prompt reduces back-and-forth and leads to more accurate answers sooner.

3. Versatility. Although many patterns are discussed in the context of software development, they transfer well to arbitrary domains — from infinite story generation for entertainment to educational games and topic exploration.

4. Structured thinking. The catalog encourages a systematic approach to prompting problems and their solutions.

5. Support for automation. Catalog patterns are applicable to common LLM interaction problems and can drive automated software tasks.

1.6. Examples of pattern-specific advantages:

1. Meta Language Creation. Useful when ideas can be expressed more concisely, unambiguously, or clearly in a constructed meta-language. The pattern works by explaining to the LLM the meaning of symbols, words, or operators [2].

2. Template Pattern. Constrains LLM outputs to a formatting structure that matches user needs — particularly helpful when the model does not "know" the target format.

3. Software engineering use. Patterns can improve code quality, assist with refactoring, requirements elicitation, and software design.

4. Chain-of-Thought (CoT) prompting. While CoT is a general prompting technique, it can be embedded within patterns. It encourages the model to expose its reasoning so you can, for example, derive task lists (generated knowledge) from the reasoning chain [2].

2. Fundamental Limitations of LLMs

2.1. Hallucinations and Knowledge Inaccuracy

LLM limitations are central to agent design and prompt-engineering methods.

Hallucination is when a model generates output that is nonsensical or factually wrong yet appears coherent and plausible[3].

Software engineering context. Coding hallucinations include incomplete, non-functional, or incorrect source code that does not meet requirements [4].

Hallucinations raise the risk of ignorance: even well-educated non-experts may struggle to distinguish reliable facts from invented ones when clear error markers are absent.

2.2. Fragility of Reasoning

1. Some sources question whether current LLMs perform true formal reasoning. Instead, their "reasoning" can resemble probabilistic pattern-matching, retrieving nearest patterns from training data rather than understanding concepts [5].

2. Mathematical reasoning. On grade-school math, performance can vary widely across paraphrases of the same problem — highlighting fragile reasoning ability [5].

3. Sensitivity to irrelevant information. Inserting seemingly relevant but actually no-op statements into a reasoning chain can cause dramatic performance drops (reported up to ~65% for some models), as models may treat statements as operations without grasping their mathematical meaning.

4. Input perturbations. LLMs are relatively robust to name changes, but highly sensitive to changes in numeric values, which can degrade accuracy and increase variance [5].

2.3. Context-window and Memory Limits

Typical LLMs have limited context length, constraining their ability to maintain full conversation history — especially in multi-agent systems. Very long inputs can slow responses and harm quality [2].
1. This is acute in long-term conversational tasks, where models must maintain coherence across many sessions — difficult for traditional memory systems [6].

2. Remedies. Approaches like Retrieval-Augmented Generation (RAG) dynamically fetch relevant context; Agentic Memory (A-MEM) organizes memory in flexible, linked structures (akin to

Zettelkasten). Even as context windows grow, RAG remains attractive due to its effectiveness and much lower cost (often cited as up to ~99% cheaper than stuffing huge contexts).

Reported benefits include higher output control, clearer communication, versatility, structured thinking, and better automation. Recurring failure modes are schema-compliant yet incorrect content without verification, fragile reasoning (esp. math), sensitivity to numeric/distractor perturbations, retrieval dependence, and variance in multi-turn settings.

3. Evaluation Methodologies for LLMs and Agents

Assessing LLMs and agents requires specialized metrics and benchmarks, since traditional methods are often insufficient.

3.1. Evolution of LLM/Agent Evaluation

1. LLM-as-a-Judge. Using an LLM (e.g., GPT-4) to evaluate outputs — including dialogue quality and functional code correctness. Studies report high alignment with human preferences [2]. 2. Agent-as-a-Judge. A further step: an agentic system (not just an LLM) evaluates other agents. Early studies suggest this can outperform LLM-as-a-Judge and approach human raters, offering fair, information-rich assessments without the cost of human annotation [2].

3. Voting inference (multi-LLM). Combining outputs via majority vote across multiple LLMs can improve performance and accuracy in certain settings [2].

3.2. Open Challenges in Evaluation

1. Data contamination. Work on GSM-Symbolic suggests high scores on original GSM8K may reflect optimistic bias if test items leaked into training, undermining metric reliability [5].

2. Lack of standardization (agents). For LLM-agent research (e.g., requirements engineering), dataset construction often depends on practical projects and specific cases — leading to limited standardization and a shortage of large, public datasets [5].

3. Agent vs. LLM trade-offs. LLMs can excel at single-shot tasks (e.g., high-quality test generation with few-shot examples). LLM agents, via multi-agent collaboration, tend to be better at task decomposition and iterative optimization — but may be more expensive than plain LLMs on simple tasks [2][5].

Patterns are commonly assessed with domain benchmarks, LLM/Agent-as-a-Judge, and multi-model voting. For comparability, we recommend token/time-normalized metrics, fixed seeds, ablations, and contamination checks.

4. Applications and Evolution: LLM Agents

The evolution of LLMs is tightly linked to the development of autonomous agents, which aim to mitigate core model limitations.

4.1. Agents and Their Core Components

LLM-based agents leverage model capabilities for autonomous reasoning and decision-making. A typical agent architecture includes:

1. Customized profiles (Personas). Assign a specific role or personality to the model (Persona pattern) [7].

2. Long-term memory. Mechanisms beyond the context window, including RAG or specialized systems like A-MEM that build interlinked knowledge graphs [6].

3. Reasoning and planning. Algorithms that structure complex tasks (e.g., Chain-of-Thought, Tree-of-Thought) [7].

4. Action modules. Interfaces to external tools and APIs, enabling agents to carry out real-world tasks [7].

4.2. Mitigating Limitations via RAG and Structure

1. The RAG challenge. Many RAG pipelines struggle on knowledge-intensive reasoning when relevant information is scattered across documents, making precise identification and global reasoning difficult [8].

2. StructRAG as a remedy. The StructRAG framework draws on cognitive theories: humans transform raw information into structured knowledge (tables, graphs, catalogs, algorithms) to shorten reasoning paths. StructRAG selects an optimal structure per task, reconstructs documents into that format, and then uses the structured knowledge to answer — outperforming baseline RAG and long-context

models on complex scenarios. [8]

4.3. Examples of Use — and Remaining Limits

LLMs are applied successfully across domains:

1. Software engineering. Code generation, debugging, testing, requirements analysis, and documentation [9]. However, models like ChatGPT tend to perform worse on harder tasks such as code review and vulnerability discovery — areas requiring further advances.

2. Medicine, law, education. LLMs can pass elements of medical licensing exams or assist with legal analysis and learning tasks [1].

Despite progress in prompt engineering and tooling, core challenges remain. Many sources argue that hallucinations are intrinsic and cannot be fully eliminated.

4.4. Recommendations. What to use when.

To translate the six criteria into actionable guidance, we map common task conditions to recommended pattern families and briefly explain the trade-offs. The table below summarizes which families to prefer under typical constraints and why, with emphasis on accuracy, structure adherence, cost/latency, robustness, reproducibility, and safety/policy compliance. These recommendations are indicative rather than exhaustive and assume standard good practices (e.g., retrieval when facts matter).

In agentic systems, prompt patterns act as modular design primitives: align vocabularies for memory, enforce structured tool/API use, ground with retrieval (incl. StructRAG), and perform bounded verification before actuation. The ensuing “what-to-use-when” table operationalizes this integration in Table 1.

Table 1.

Condition / goal	**Recommended family (examples)**	**The Reason**
Strict format for downstream systems (APIs, ETL, grading)	Output Customization (Template, JSON schema)	Maximizes structure adherence and reproducibility at low cost.
High factual stakes (policy, specs, medicine)	Error Identification (fact-check, reflection) + RAG/StructRAG; optional Cognitive Verifier	Improves accuracy and safety; verifier helps logic/math; expect added latency.
Ambiguous domain language	Input Semantics (meta-language, glossary)	Raises accuracy/robustness by disambiguation; low cost.
Tight budget/latency Open-ended exploration / tutoring Math/logic/program synthesis	Output Customization (single-shot templates) over Interaction Interaction + light Prompt Improvement Cognitive Verifier (bounded CoT, self-consistency)	Reduces turns/tokens; keep reasoning concise. Better requirement elicitation; accept higher cost. Lifts accuracy without unbounded tokens when capped.
Benign false refusals	Refusal Breaker with guardrails	Fixes misclassification; preserve policy compliance.

Conclusion

This review replaces ad-hoc discussion of prompting with a criteria-driven account and provides direct answers to the research questions posed in the Introduction. RQ1 (definitions): we clarify prompt, pattern, and prompt pattern and use these terms to anchor evaluation and design. RQ2 (what recurs): seven recurrent families are consolidated—Input Semantics, Output Customization, Error Identification, Prompt Improvement, Cognitive Verifier, Refusal Breaker, Interaction—each contrasted along six criteria (factual accuracy, structure adherence/controllability, cost/latency, robustness, reproducibility, safety).

RQ3 (benefits/failure modes): structure improves controllability and automation; verifier-style reasoning helps logic/math when bounded; interaction aids exploration but increases variance and cost; persistent failure modes include schema-compliant yet incorrect outputs without verification, sensitivity to numeric/distractor perturbations, retrieval dependence, and multi-turn variability. RQ4 (how to assess): current practice converges on domain benchmarks, LLM/Agent-as-Judge, and voting; for comparability we recommend token-/time-normalized metrics, fixed seeds, ablations, and contamination checks. RQ5 (how patterns integrate in agents): patterns act as modular design primitives—align vocabularies for memory, enforce structured tool/API use, ground with retrieval (incl. structure-aware methods such as StructRAG), and apply bounded verification before actuation.

From these answers a consistent picture emerges. Structure-first methods (templates, schema-constrained output) yield the most reliable gains in controllability and downstream automation; factual reliability rises when structure is paired with retrieval and explicit verification. Input-semantics techniques usefully precede other interventions by stabilizing vocabulary and task framing. Cognitive-verifier prompts should be used selectively for math/logic with budget caps. Interaction patterns are valuable for requirement elicitation, yet they degrade efficiency and complicate reproducibility. Refusal-mitigation should be applied narrowly to correct benign misclassifications while preserving policy compliance.

What-to-use-when (actionable mapping). Strict formats → Output Customization; high factual stakes → Error Identification + retrieval (incl. StructRAG) + verification; ambiguous domains → Input Semantics; tight budget/latency → single-shot Output Customization with lightweight checks; exploratory/problem-finding work → Interaction; math/logic → bounded Cognitive Verifier. Viewed within agentic systems, these choices provide a compact, operational basis for selecting and evaluating prompting strategies in practice and for strengthening empirical rigor. Limitations & future work: we outline paths toward structure-aware retrieval, stronger verification circuits, and uniform reporting; a reference implementation of pattern bundles with token/time-normalized metrics is a natural next step.

Література

1. White J., Fu Q., Hays S., Sandborn M., Olea C., Gilbert H., Elnashar A., Spencer-Smith J., Schmidt D.C. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. Department of Computer Science, Vanderbilt University, Tennessee, Nashville, USA, 2023. DOI: 10.5555/3721041.3721046

2. Jin H., Huang L., Cai H., Yan J., Li B., Chen H. From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future. arXiv preprint (2024). DOI: 10.1145/3770084

3. Bianchini F. Evaluating Intelligence and Knowledge in Large Language Models. Topoi. Springer, 2024. Vol. 44, no. 1. P. 163–173. DOI: 10.1007/s11245-024-10072-5

4. Qian C., Liu W., Liu H., Chen N., Dang Y., Li J., Yang C., Chen W., Su Y., Cong X., Xu J., Li D., Liu Z., Sun M. ChatDev: Communicative Agents for Software Development. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024, Long Papers), Bangkok, Thailand, 2024. P. 15174–15186. DOI: 10.18653/v1/2024.acl-long.810

5. Mirzadeh I., Alizadeh K., Shahrokhi H., Tuzel O., Bengio S., Farajtabar M. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. arXiv preprint (2024). DOI:10.48550/arXiv.2509.25725

6. Xu W., Liang Z., Mei K., Gao H., Tan J., Zhang Y. A-MEM: Agentic Memory for LLM Agents. arXiv preprint (2025) (CoRR abs/2502.12110). DOI:10.48550/arXiv.2502.12110

7. Wang X., Chen Y., Yuan L., Zhang Y., Li Y., Peng H., Ji H. Executable Code Actions Elicit Better LLM Agents. arXiv preprint (2024). DOI: 10.48550/arXiv.2402.01030

8. Li Z., Chen X., Yu H., Lin H., Lu Y., Tang Q., Huang F., Han X., Sun L., Li Y. StructRAG: Boosting Knowledge Intensive Reasoning of LLMs via Inference-time Hybrid Information Structurization. arXiv preprint (2024). DOI: 10.48550/arXiv.2410.08815

9. Sami A.M., Rasheed Z., Waseem M., Zhang Z., Tomas H., Abrahamsson P. A Tool for Test Case Scenarios Generation Using Large Language Models. arXiv preprint (2024). DOI: https://doi.org/10.48550/arXiv.2410.08815