Contents

Core Agentic patterns

Chain-of-Thought (CoT)

Chain-of-Thought is a reasoning pattern in which a model explicitly decomposes a problem into a sequence of intermediate reasoning steps before producing a final answer.

Historical perspective

The idea behind Chain-of-Thought did not emerge in isolation; it is rooted in long-standing research on human problem solving, symbolic reasoning, and step-by-step explanation in cognitive science and artificial intelligence. Early expert systems and logic-based AI relied on explicit inference chains, but these systems required hand-crafted rules and did not generalize well. In parallel, educational psychology emphasized the importance of "showing your work" as a way to improve reasoning accuracy and learning outcomes.

In modern machine learning, the immediate precursors to Chain-of-Thought appeared in work on prompting large language models around 2020–2021, when researchers observed that models often failed at multi-step reasoning despite strong performance on surface-level tasks. The key insight, formalized in 2022, was that large language models could be induced to perform significantly better on arithmetic, symbolic, and commonsense reasoning tasks if they were prompted to generate intermediate reasoning steps. This marked a shift from treating models as black-box answer generators toward viewing them as systems capable of producing structured reasoning traces when guided appropriately.

The pattern explained

At its core, Chain-of-Thought is about **externalizing intermediate reasoning**. Instead of asking a model to jump directly from a question to an answer, the prompt encourages or requires the model to articulate the steps that connect the two. These steps may include decomposing the problem, applying rules or heuristics, performing intermediate calculations, or evaluating partial conclusions.

In practice, the pattern works by conditioning the model to follow a reasoning

trajectory. This can be achieved in several ways: by including exemplars that show step-by-step reasoning, by explicitly instructing the model to "think step by step," or by structuring the task so that intermediate outputs are required before the final response. Regardless of the mechanism, the effect is the same: the model allocates part of its generation capacity to reasoning, rather than compressing all inference into a single opaque prediction.

From an agentic systems perspective, Chain-of-Thought serves as a **foundational reasoning primitive**. It enables decomposition of complex tasks into manageable subproblems, supports inspection and debugging of model behavior, and provides a substrate for more advanced patterns such as self-reflection and tree-based exploration. CoT is especially valuable in tasks that require arithmetic, logical consistency, planning, or constraint satisfaction, where single-step answers are brittle or unreliable.

However, Chain-of-Thought also introduces trade-offs. Generating explicit reasoning increases token usage and latency, and the reasoning trace itself may contain errors even when the final answer is correct—or vice versa. As a result, CoT is best understood not as a guarantee of correctness, but as a tool that increases the *probability* of correct reasoning and improves transparency. Later agentic patterns often build on CoT while selectively summarizing, pruning, or validating reasoning steps to manage these costs.

References

1. Wei, J., et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. NeurIPS, 2022. https://arxiv.org/abs/2201.11903
2. Kojima, T., et al. *Large Language Models are Zero-Shot Reasoners*. NeurIPS, 2022. https://arxiv.org/abs/2205.11916
3. Nye, M., et al. *Show Your Work: Scratchpads for Intermediate Computation with Language Models*. arXiv preprint, 2021. https://arxiv.org/abs/2112.00114
4. Cobbe, K., et al. *Training Verifiers to Solve Math Word Problems*. arXiv preprint, 2021. https://arxiv.org/abs/2110.14168

ReAct (Reason + Act)

ReAct is a prompting pattern where an LLM alternates *explicit reasoning steps* with *tool/environment actions*, using observations from those actions to steer the next reasoning step.

ReAct appears in the 2022 wave of "reasoning by prompting." The immediate precursor is **Chain-of-Thought (CoT)** prompting (early 2022), which showed that giving exemplars with intermediate reasoning steps can unlock multi-step problem solving in large models. (2) But CoT on its own is "closed-book": it improves decomposition and planning, yet still forces the model to *invent* facts and compute purely in-token—making it vulnerable to hallucination and compounding errors when external information is needed.

In parallel, multiple lines of research were converging on "LLMs as agents" that *act* via external interfaces. **WebGPT** (late 2021) trained a model to browse the web in a text environment, explicitly collecting citations during interaction. (arXiv) **MRKL systems** (mid 2022) articulated a modular neuro-symbolic architecture: keep the LLM as a language/coordination layer, and route specialized subproblems to tools/knowledge modules. (arXiv) Around the same time, grounding work like **Do As I Can, Not As I Say (SayCan)** explored selecting feasible actions via an affordance model while using an LLM for high-level planning. (arXiv)

Core ideas

ReAct (first posted Oct 2022; later published via ICLR venue) crystallized these threads into a simple, general prompt format: *interleave* reasoning traces with actions so that the model's "thoughts" can request information or execute steps, then immediately incorporate the resulting observations into the next reasoning step. (arXiv)

ReAct structures an agent's trajectory as a repeating loop:

1. **Thought (Reasoning trace):** the model writes a brief internal/explicit rationale describing what it knows, what it needs, and what it will do next.
2. **Action (Tool/environment step):** the model emits a structured action (e.g., `Search[...]`, `Lookup[...]`, `Click[...]`, `UseCalculator[...]`, `TakeStep[...]`) that is executed by the system.
3. **Observation:** the system returns the tool result (snippet, retrieved fact, environment state, error), which is appended to the context.
4. Repeat until a **Final** response is produced.

What distinguishes ReAct from simpler tool-augmented prompting is the *granularity* of this interaction. Reasoning and acting are interleaved at every step, rather than separated into large phases. This reduces hallucination by encouraging the model to seek external information when needed, improves robustness by enabling mid-course correction, and yields trajectories that are interpretable as step-by-step decision processes.

A typical ReAct prompt provides 1–2 exemplars of full trajectories in a consistent schema, then a new task. For example (conceptually):

- `Thought: ...`
- `Action: Search[...]`
- `Observation: ...`
- `Thought: ...`
- `Action: Lookup[...]`
- `Observation: ...`
- `Thought: ...`
- `Final: ...`

From a system-design perspective, ReAct also reinforces a clean separation of concerns. The language model is responsible for proposing reasoning and actions, while the surrounding runtime is responsible for enforcing action schemas, executing tools, handling failures, and maintaining state. This separation makes ReAct a natural foundation for more complex agentic systems and later patterns built on top of it.

References

- **ReAct: Synergizing Reasoning and Acting in Language Models** (Yao et al., 2022; ICLR venue) (arXiv)
- **Chain-of-Thought Prompting Elicits Reasoning in Large Language Models** (Wei et al., 2022) (arXiv)
- **WebGPT: Browser-assisted question-answering with human feedback** (Nakano et al., 2021) (arXiv)
- **MRKL Systems: A modular, neuro-symbolic architecture...** (Karpas et al., 2022) (arXiv)
- (Related "tool delegation" line) **Toolformer: Language Models Can Teach Themselves to Use Tools** (Schick et al., 2023) (arXiv)

Tools

MCP

- MCP motivation and scope: Why a standardized protocol is needed to connect agents with tools, data, and context.
  - https://www.anthropic.com/news/model-context-protocol
  - https://modelcontextprotocol.io/docs/learn/introduction
- MCP architectural model: Host, client, and server roles and how they map onto an agent execution loop.
  - https://modelcontextprotocol.io/docs/learn/architecture
- Protocol layers: Separation between the JSON-RPC data model and transport mechanisms.
  - https://modelcontextprotocol.io/specification/2025-06-18/basic
  - https://www.jsonrpc.org/specification
- Lifecycle and capability negotiation: Initialization, versioning, and feature discovery for interoperable agents.
  - https://modelcontextprotocol.io/specification/2025-06-18/basic/lifecycle
- Tools primitive: How agents invoke side-effectful actions via typed, schema-defined tools.

  - https://modelcontextprotocol.io/specification/2025-06-18/server/tools

- Resources primitive: Supplying structured, addressable context (files, data, memory) to agents.
  - https://modelcontextprotocol.io/specification/2025-06-18/server/resources

- Prompts primitive: Reusable prompt templates as shared cognitive scaffolding.
  - https://modelcontextprotocol.io/specification/2025-06-18/server/prompts

- Transport choices: stdio vs HTTP/Streamable HTTP and their implications for deployment and scaling.
  - https://modelcontextprotocol.io/specification/2025-06-18/basic/transports

- Security and authorization: Threat boundaries, safe server design, and OAuth 2.1 for remote access.
  - https://modelcontextprotocol.io/specification/2025-06-18/basic/security_best_practices
  - https://modelcontextprotocol.io/specification/2025-06-18/basic/authorization

- Practical agent integration: How planners and executors use MCP clients in real agent systems.
  - https://modelcontextprotocol.io/docs/learn/client-concepts