Foundations

Conversations and Reasoning

Chat Mechanics, Memory, and Reasoning Patterns

Michael J Bommarito II · Daniel Martin Katz · Jillian Bommarito

December 21, 2025

Working Draft Chapter

Version 0.1

This chapter is from the textbook *Artificial Intelligence for Law and Finance*, currently under development. It covers the transition from single-shot completions to multi-turn conversations, context management strategies, and the reasoning patterns that enable complex problem-solving.

The most current copy of the project is available at:
`https://github.com/mjbommar/ai-law-finance-book/`

Contents

How to Read This Chapter

This chapter extends the single-turn prompting concepts from Chapter 1 (The LLM Primer) into the domain of multi-turn conversations and structured reasoning. Whether you are a legal professional building client-facing AI assistants, a financial analyst designing research workflows, or an architect implementing enterprise AI systems, this chapter provides the conceptual and practical tools you need to maintain coherent dialogues and extract reliable reasoning from large language models.

Reading Paths

Path 1: Practitioner Quick Start (20–30 minutes)

If you are already experimenting with LLMs and want immediately applicable techniques:

- **Section 2**: Learn the system/user/assistant role architecture and how to maintain context in long conversations
- **Section 3**: Understand when to use chain-of-thought prompting and how to structure reasoning for accuracy
- **Section 4**: Apply the decision framework to select strategies based on your task's risk and time constraints

This path gives you practical patterns you can deploy immediately while building the conceptual foundation for more advanced techniques.

Path 2: Full Technical Understanding (60–90 minutes)

If you need to understand the underlying mechanics for system design or architecture decisions:

- Read all sections in order, paying particular attention to the memory strategies in Section 2.2 and the "Lost in the Middle" phenomenon
- Study the complete reasoning taxonomy in Section 3, including Tree of Thoughts and Graph of Thoughts for complex planning tasks
- Examine the few-shot example selection strategies and bootstrapping techniques for building your own prompt libraries

This path prepares you to design sophisticated conversational systems with appropriate reasoning topologies.

Path 3: Governance and Risk Focus (30–45 minutes)

If your primary concern is risk management, compliance, or audit:

- **Section 2.4**: Understand guardrails, Constitutional AI, and the role of system prompts in enforcing policy
- **Section 3.1.2**: Learn why reasoning traces should remain private and how to separate audit logs from user-facing outputs
- **Section 4**: Apply the risk-based decision framework for high-stakes applications

This path emphasizes the governance dimensions that Part III will expand upon in detail.

Prerequisites

This chapter assumes familiarity with the concepts introduced in Chapter 1 (The LLM Primer):

- How tokens and tokenization work
- The autoregressive generation process
- Basic prompt structure and the role of context
- Common failure modes (hallucination, sensitivity to phrasing)

If these concepts are unfamiliar, we recommend reviewing the LLM Primer before proceeding.

What This Chapter Covers

Key Objectives

By the end of this chapter, you will be able to:

1. **Design multi-turn conversations** with clear role separation, effective memory management, and robust context handling
2. **Select appropriate reasoning strategies** (chain-of-thought, self-consistency, ReAct, Tree of Thoughts) based on task complexity, accuracy requirements, and cost constraints
3. **Implement few-shot learning** with properly selected examples, understanding similarity-based retrieval and bootstrapping techniques
4. **Apply safety guardrails** at the system prompt level while understanding their limitations and the need for defense-in-depth
5. **Balance accuracy, latency, and cost** using the strategy selection framework for profes-

sional applications

Visual Cues (Boxes)

Throughout the chapter, colored boxes signal intent:

- **Key takeaways (`keybox`):** objectives, frameworks, and decision rules
- **Notes (`highlightbox`):** quick-start paths, intuition, and plain-English context
- **Optional technical detail (`technicalbox`):** deeper mechanics you can skip on a first read
- **Warnings (`cautionbox`):** risk and governance pitfalls
- **Code/examples (`listingbox`):** reproducible snippets and technical artifacts (when included)

What This Chapter Does Not Cover

To maintain focus, we defer several related topics to subsequent chapters:

- **Structured outputs and schemas**: Covered in Chapter 3 (Structured Outputs and Tool Use)
- **Tool use and function calling**: Introduced conceptually here (ReAct), but implementation details appear in Chapter 3
- **Retrieval-Augmented Generation (RAG)**: The full RAG architecture is covered in Chapter 4 (Retrieval and Knowledge); we focus here on when retrieval is needed, not how to implement it
- **Fine-tuning and training**: We focus on prompt-based techniques that require no model modification
- **Agent architectures**: Autonomous agent design with planning and execution loops appears in Part II (Agents)
- **Governance frameworks**: Comprehensive regulatory and compliance considerations appear in Part III (Governance)

Bridge from the LLM Primer

In the previous chapter, we examined how LLMs process and generate text in single-turn interactions. We saw that these models are fundamentally stateless: each inference call is independent, with the model retaining no memory of previous interactions. We explored how prompt design influences output quality and how various failure modes can compromise reliability.

This chapter extends those foundations in two critical directions:

From Single-Turn to Multi-Turn.. Real-world applications rarely consist of isolated queries. Legal research assistants must maintain context across dozens of exchanges. Financial analysis tools need to remember constraints established early in a session. Customer service applications must track the history of an issue. We will examine how to create the *illusion* of memory in a stateless system, managing conversation state through careful prompt construction and external memory mechanisms.

From Pattern Matching to Reasoning.. Single-turn prompts often rely on the model's ability to pattern-match from its training data. But complex professional tasks—legal analysis, financial modeling, medical diagnosis—require genuine multi-step reasoning. We will explore techniques that force models to externalize their reasoning process, dramatically improving accuracy on tasks that require logical inference, arithmetic, or planning.

Together, these extensions transform the LLM from a sophisticated autocomplete engine into a capable conversational partner and reasoning system. The techniques we introduce here form the foundation for the structured outputs, tool use, and agent architectures covered in subsequent chapters.

1 Introduction and Scope

The architectural evolution of Large Language Models from static, single-turn text completion engines to dynamic, stateful conversational agents represents a fundamental paradigm shift in artificial intelligence. At its core, a pre-trained Transformer model is a *stateless function*: it accepts an input array (often called a *tensor*) of token IDs and outputs a probability distribution for the subsequent token, retaining no memory of the transaction once the inference cycle concludes (Vaswani et al. 2017). To engineer the illusion of a continuous, coherent dialogue—and, more importantly, to instill the capacity for multi-step reasoning—developers must construct a sophisticated orchestration layer that manages context, creates artificial state, and enforces logical structure upon the stochastic generation process.

This chapter provides a comprehensive analysis of the mechanisms required to transform a raw language model into a reliable conversational partner and a rigorous reasoning engine. We extend the view from single-turn prompt engineering into the complex domain of multi-turn workflows, where the challenges of context window management, information retrieval, and state retention become paramount.

1.1 Why Conversations and Reasoning Matter for Professionals

For legal and financial professionals, the limitations of single-turn prompts become apparent almost immediately in practice. Consider these common scenarios:

Legal Research.. An attorney researching case law needs to refine queries iteratively based on results, maintain awareness of jurisdiction constraints established early in the session, and synthesize findings across multiple exchanges. A single-turn model cannot remember that the research is limited to Ninth Circuit precedents or that the client's situation involves specific statutory provisions discussed three turns earlier.

Financial Analysis.. A portfolio manager analyzing market conditions needs to establish baseline assumptions, explore alternative scenarios, and refine recommendations based on client risk tolerance—all while maintaining consistency with constraints defined at the start of the session. The model must "remember" that the client has a 10-year investment horizon and moderate risk tolerance when generating subsequent analyses.

Due Diligence.. A compliance officer reviewing vendor documentation needs to track issues identified across multiple documents, maintain a running list of concerns, and ensure that follow-up questions reference findings from earlier in the review. The conversation must maintain coherent state across potentially dozens of exchanges.

These scenarios share a common requirement: the AI system must maintain *conversational state* that persists across turns, even though the underlying model has no inherent memory capability.

1.2 The Dual Challenge: State and Reasoning

This chapter addresses two interconnected challenges that emerge when deploying LLMs in professional contexts:

Challenge 1: Conversational State

How do we create the illusion of persistent memory in a fundamentally stateless system? This involves managing the conversation history, handling context window limitations, and ensuring that critical information established early in a dialogue remains accessible and influential throughout the interaction.

Challenge 2: Structured Reasoning

How do we move beyond pattern-matching to enable genuine multi-step reasoning? Standard "zero-shot" prompting often fails on complex tasks because the model attempts to map input directly to output in a single forward pass, relying on surface-level statistical correlations rather than causal logic (Wei et al. 2022). This leads to frequent hallucinations on math, logic, and planning tasks.

These challenges are deeply interrelated. Effective reasoning often requires maintaining state across reasoning steps, while conversational coherence depends on the system's ability to reason about

what information is relevant to the current exchange. The techniques we introduce—from role-based prompt engineering to chain-of-thought reasoning to tool-augmented generation—address both challenges in an integrated framework.

1.3 The Emergence of Reasoning Capabilities

Recent research has demonstrated that forcing a model to externalize its latent logic significantly enhances performance on symbolic and arithmetic tasks. The emergence of intermediate reasoning topologies—specifically Chain-of-Thought (CoT), Self-Consistency, and ReAct—has fundamentally changed what we can expect from LLM-based systems (Wei et al. 2022; Wang et al. 2023a; Yao et al. 2023b).

However, these capabilities introduce new variables into the system architecture. Reasoning traces consume tokens from the limited context window. Multi-path verification (self-consistency) multiplies inference costs. Tool-augmented reasoning (ReAct) introduces network latency and external dependencies. Most critically, the "Lost in the Middle" phenomenon (Liu et al. 2024) means that retrieval accuracy degrades in long contexts, requiring careful attention to how we construct prompts and manage conversation history.

1.4 Chapter Roadmap

We will systematically examine the components of a modern conversational architecture:

Section 2: Conversational Models and State.. We begin with the definition of roles (System, User, Assistant) and the management of short-term versus long-term memory. We examine how the system prompt functions as an "immutable constitution" for agent behavior, and how context window limitations necessitate active memory management strategies including sliding windows, recursive summarization, and vector-enhanced retrieval.

Section 3: Reasoning Patterns.. We proceed to a rigorous evaluation of reasoning strategies, analyzing the computational trade-offs between linear reasoning (Chain-of-Thought), ensemble verification (Self-Consistency), tool-augmented reasoning (ReAct), and graph-based exploration (Tree and Graph of Thoughts). We examine when to keep reasoning traces private versus visible, and how few-shot examples can bootstrap reasoning capabilities.

Section 4: Strategy Selection.. We synthesize these elements into a decision framework for strategy selection, guiding architectural decisions based on task complexity, risk tolerance, and the immutable constraints of cost and compute.

Section 5: Synthesis.. We integrate the concepts and prepare the transition to structured outputs, tool use, and multimodal inputs covered in subsequent chapters.

1.5 A Note on Terminology

Throughout this chapter, we use several terms with specific technical meanings:

- **Context window**: The maximum number of tokens the model can consider in a single inference call. This includes both the input (prompt, history, retrieved content) and the output (generated response).
- **State** or **memory**: Information that persists across conversation turns. Since LLMs are stateless, "memory" is an illusion created by re-injecting previous context into each new prompt.
- **Reasoning trace** or **scratchpad**: The intermediate steps a model generates when solving a problem. These may be visible to users (for explainability) or kept private (for safety and simplicity).
- **Orchestration layer**: The software infrastructure that manages conversation state, constructs prompts, invokes the model, and processes outputs. This layer transforms a stateless model into a stateful conversational system.
- **Guardrails**: Safety mechanisms that constrain model behavior, typically implemented through system prompts, output classifiers, or Constitutional AI principles.

With these foundations established, we turn to the mechanics of conversational state management.

2 Conversational Models and State

The distinction between a text-completion engine and a conversational agent lies entirely in the management of state. Since the underlying model weights are frozen and stateless, "memory" is purely a function of the input context. To maintain a conversation, the orchestration layer must recursively re-inject the history of the interaction into the model's context window with each new turn. This process, often termed **context rehydration**, is the fundamental mechanic of conversational AI.

However, treating the context window as an infinite append-only log is computationally unsustainable and algorithmically flawed. Context windows have fixed limits—ranging from 4,096 tokens in older models to 128,000+ in newer iterations—and the attention mechanism itself has limitations that affect how well the model can access different parts of a long context.

In this section, we examine the architectural components of conversational state: the role-based prompt structure that defines who is speaking and what constraints apply, the memory strategies that maintain coherence across turns, and the safety mechanisms that bound model behavior.

2.1 Role-Based Prompt Engineering

Modern conversational architectures enforce a strict separation of concerns through **role-based prompting**. This is not merely a formatting convention but a control mechanism that delineates the model's operational constraints from user inputs. The architecture recognizes three primary roles: the **System**, the **User**, and the **Assistant**.

2.1.1 The System Role: The Immutable Constitution

The **system prompt** (often designated with special control tokens in the model's vocabulary) serves as the immutable "constitution" of the agent. It establishes the behavioral baseline, tone, output format, and, crucially, the safety boundaries of the interaction (Zheng et al. 2023).

Unlike user prompts, which are dynamic, untrusted, and variable, the system prompt is intended to be static and privileged. In architectures like Llama 2, the input is structured to wrap user messages in specific tags while separating the system instruction, ensuring the attention heads distinguish between instructions to be followed and text to be processed (Touvron et al. 2023). This structural separation is vital for defense against "jailbreaking" or "prompt injection" attacks, where a user attempts to override the model's safety protocols by mimicking authoritative instructions.

Signpost to Chapter 1.. Prompt injection is not only a conversational concern but also a core LLM failure mode with security implications. Chapter 1 provides a detailed taxonomy of prompt injection and defense-in-depth mitigations.

Effective System Prompt Design

An effective system prompt should include:

1. **Identity and role**: Define who the assistant is and its primary purpose
2. **Behavioral constraints**: Specify what the assistant should and should not do
3. **Output format**: Establish expected structure for responses (when applicable)
4. **Knowledge boundaries**: Define what the assistant knows and when to acknowledge uncertainty
5. **Escalation criteria**: Specify when to refuse, ask for clarification, or defer to human judgment

Example: Legal Research Assistant.. Consider a system prompt for a legal research assistant:

```
You are a legal research assistant specializing in U.S. federal
law. You help attorneys find relevant case law, statutes, and
regulations. You always cite specific sources with proper legal
```

```
citations. You acknowledge when a question falls outside your
knowledge or when case law may have evolved since your training.
You never provide legal advice or make predictions about case
outcomes. When asked about jurisdiction-specific rules, you note
that local practice may vary and recommend verifying with current
local sources.
```

This prompt establishes identity (legal research assistant), scope (U.S. federal law), expected behavior (cite sources, acknowledge limitations), prohibitions (no legal advice or predictions), and appropriate hedging (verify with local sources).

Instruction Drift and Mitigation.. However, the efficacy of the system prompt is not absolute. A phenomenon known as **instruction drift** or **context dilution** can occur as the conversation lengthens. As the distance between the initial system prompt and the current generation increases, the model's adherence to the initial constraints can degrade, primarily due to the "recency bias" inherent in the attention mechanism (Liu et al. 2024).

To combat this, advanced prompting strategies involve:

- **Repeating critical constraints**: Restate key rules near the end of the prompt, immediately before the generation trigger
- **Periodic reinforcement**: Insert reminder statements in long conversations (e.g., "Remember, you should not provide legal advice")
- **Sentinel phrases**: Use specific phrases that trigger compliance checks (though this requires custom orchestration logic)

2.1.2 The User and Assistant Roles: Managing the Dialogue Loop

The **user role** represents external input—the queries, commands, and information provided by the human interacting with the system. The **assistant role** captures the model's generated outputs. To maintain state, the application must append each user query and assistant response to a growing list of messages. This list is serialized into a single prompt string (or a structured token sequence) at every turn.

The Dialogue Loop

A typical multi-turn conversation follows this pattern:

1. Receive user input
2. Construct prompt: `[System] + [History] + [User Input]`

3. Call model, generate response
4. Append `[User Input, Assistant Response]` to history
5. Return response to user
6. Repeat from step 1

The integrity of this dialogue loop is maintained through strict formatting. For instance, Llama 2 uses a specific syntax where the system prompt and user prompt are enclosed in special tags, but the model's response is left outside, creating a clear demarcation in the token stream. This formatting guides the model's stop-token prediction; without it, the model might hallucinate the user's next response rather than stopping after its own turn.

Practical Implications.. When constructing prompts:

- **Be explicit**: Don't assume the model remembers what "it" refers to from five turns back unless you've managed context accordingly
- **Reference context**: If a question relates to something said much earlier, remind the model: "Based on our discussion about X earlier, ..."
- **Use retrieval**: For long conversations, retrieve and inject the exact relevant snippet rather than relying on raw history

2.2 Memory Strategies: Maintaining Context in Dialogues

One of the biggest challenges in multi-turn conversations is how the model "remembers" earlier parts of the dialogue. Given the finite nature of the context window, managing conversation history is fundamentally a *resource allocation problem.* As the conversation exceeds the window size, the system must make active decisions about what to retain and what to discard.

2.2.1 The "Lost in the Middle" Phenomenon

A critical insight in context management is that LLMs do not access all parts of their context window with equal acuity. Extensive empirical analysis reveals a distinct U-shaped performance curve regarding information retrieval (Liu et al. 2024). Models excel at retrieving information located at the very beginning (primacy effect) and the very end (recency effect) of the input context. Information buried in the middle of a long sequence is significantly more likely to be ignored, hallucinated, or incorrectly associated.

The U-Shaped Retrieval Curve

When retrieving facts from context:

- **Beginning of context**: High retrieval accuracy (primacy effect)
- **Middle of context**: Significantly degraded accuracy
- **End of context**: High retrieval accuracy (recency effect)

This phenomenon is likely an emergent property of training data structure, where introductions and conclusions contain concentrated semantic signal.

Figure 1 illustrates this U-shaped retrieval curve, showing how information placement affects model attention.

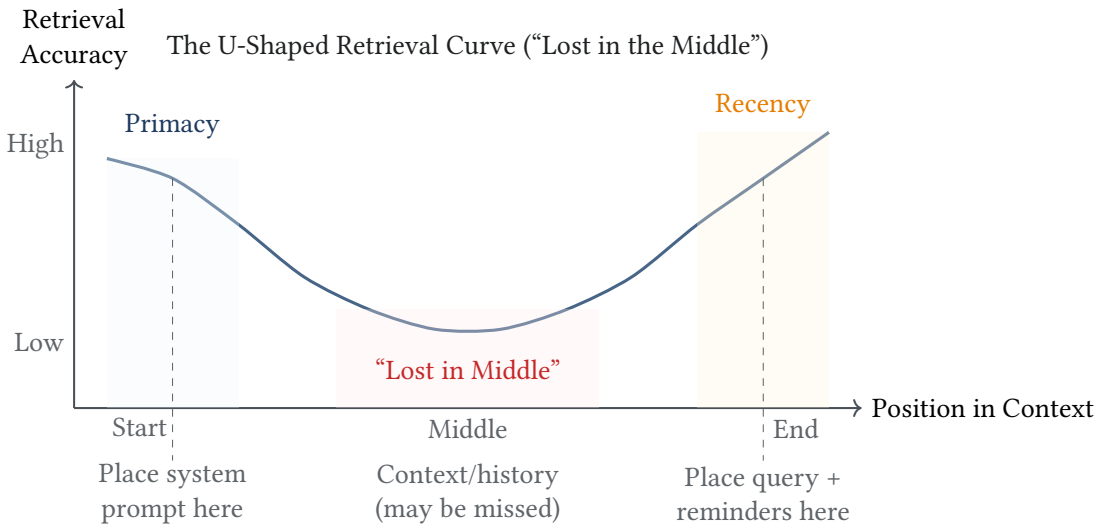

Figure 1: The "Lost in the Middle" phenomenon: LLMs exhibit a U-shaped retrieval curve, with high accuracy at the beginning (primacy) and end (recency) of context, but degraded performance in the middle. This has profound implications for prompt construction.

The implications for conversation design are profound. Simply filling the context window with relevant documents or history does not guarantee retrieval. To mitigate this:

- **Prompt re-ranking**: Relocate the most critical information to the beginning or immediate end of the prompt
- **Chunking with overlap**: Break long content into chunks with semantic boundaries, ensuring critical information appears at chunk boundaries
- **Explicit signaling**: Use formatting (headers, bullet points) to make critical information more salient

2.2.2 Sliding Windows

The simplest memory management strategy retains only the most recent N tokens or turns. This **sliding window** approach:

- **Advantages**: Computationally simple, ensures high recency, predictable token usage
- **Disadvantages**: Induces "catastrophic forgetting" of earlier context; critical constraints or facts established early are lost once they slide out of the window

When to Use.. Sliding windows are appropriate when:

- The conversation is inherently short-term (e.g., simple Q&A)
- Recent context is far more relevant than historical context
- You have other mechanisms (like system prompts) to maintain critical constraints

2.2.3 Recursive Summarization

A more sophisticated approach employs the LLM itself to periodically compress older turns into a concise narrative summary (Wang et al. 2024). For example, after 10 turns, the model might summarize the dialogue as "User asked about Python lists; Assistant explained syntax and provided examples." This summary is then prepended to the active context, replacing the raw tokens.

- **Advantages**: Preserves the semantic "gist" of the conversation, frees up token space for new content
- **Disadvantages**: Lossy compression; specific details (e.g., a phone number, a specific code snippet, exact client instructions) may be smoothed over

Implementation Considerations..

- **Summarization frequency**: Every N turns, or when the history exceeds a token threshold
- **Preservation of critical facts**: Maintain a separate "pinned facts" section for information that must never be summarized away
- **Hierarchical summarization**: For very long conversations, summarize summaries recursively

Legal Application: Client Interview Memory

In a legal intake scenario, you might configure memory as follows:

- **Pinned facts**: Client name, case type, jurisdiction, key dates (never summarized)

- **Active window**: Last 5 turns (full detail)
- **Summarized history**: Earlier turns compressed to key facts and decisions

This ensures the attorney always sees critical identifying information while maintaining context efficiency.

2.2.4 Vector-Enhanced Memory (Long-Term Retrieval)

For indefinite memory, systems employ external **vector stores**. Conversation turns are embedded into vectors and stored in a database (using indices like HNSW or FAISS). When a new user query arrives, the system retrieves semantically relevant past interactions—regardless of temporal distance—and injects them into the current context (Zhang et al. 2024).

Signpost to Chapter 1.. Chapter 1 provides the primary treatment of embeddings, similarity metrics, and hybrid retrieval. This section focuses on how retrieval is used to implement conversational memory.

- **Advantages**: Approximates human episodic memory; can retrieve relevant context from arbitrarily long histories; enables cross-session memory
- **Disadvantages**: Introduces latency; requires embedding model and vector database infrastructure; can surface conflicting memories if user preferences changed

Hybrid Approaches.. In practice, the most robust systems combine strategies:

1. **System prompt**: Immutable constraints and identity
2. **Pinned facts**: Critical session-specific information that must persist
3. **Retrieved context**: Semantically relevant prior exchanges or documents
4. **Summarized history**: Compressed earlier conversation
5. **Active window**: Recent turns in full detail
6. **Current query**: The user's latest input

This layered approach maximizes the utility of the limited context window while maintaining coherence.

Bridge to Agentic Systems

These memory patterns become architectural decisions in agentic systems. Chapter 6 introduces the six-property framework for understanding agents, and Chapter 7 examines Memory as one of ten core design questions—addressing how agents maintain state across iterations and sessions.

2.3 Context Management and Instruction Placement

Given the "Lost in the Middle" phenomenon, where you place information in the prompt matters as much as what information you include.

2.3.1 Token Budgeting

Before constructing any prompt, establish a token budget:

1. **Determine total available tokens**: Model context limit (e.g., 128K)
2. **Reserve for output**: Allocate space for the expected response (e.g., 2K–4K tokens for detailed answers)
3. **Allocate to components**:
   - System prompt: 500–1,000 tokens
   - Few-shot examples (if used): 1,000–3,000 tokens
   - Retrieved context (RAG): Variable, often 2,000–10,000 tokens
   - Conversation history: Remainder
   - Current query: Usually small
   - Final instructions/reminders: 100–200 tokens

2.3.2 Optimal Instruction Placement

Because LLMs weight recent tokens more heavily, the *end* of the prompt (just before generation begins) is prime real estate. Place critical instructions there:

The "Sandwich" Pattern

Structure your prompt as:

1. **Beginning**: System prompt with core identity and constraints
2. **Middle**: Context, history, retrieved documents

3. **End**: Current query + *reinforcement of critical constraints*

The final reinforcement ensures that key rules (output format, prohibited actions, required disclaimers) are fresh in the model's "attention" when it begins generating.

Figure 2 illustrates the recommended ordering for prompt construction, showing how each layer builds on the previous.

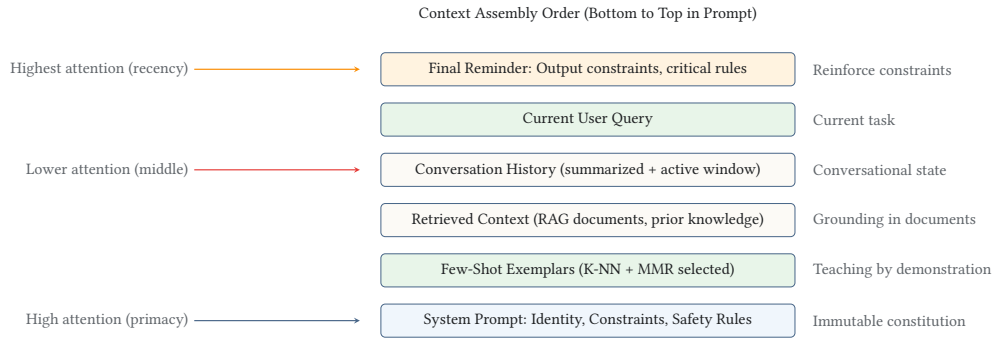

Figure 2: Context assembly order for multi-turn conversations. Place the system prompt and critical constraints at the beginning (primacy zone), context and history in the middle, and the current query with reminders at the end (recency zone) for maximum retention.

Example: Financial Compliance.. Consider a financial advisory system where legal disclaimers are mandatory:

```
[System: You are a financial information assistant...]

[History of conversation...]

[User: What stocks should I buy?]

[Reminder: You must not provide personalized investment advice. If
asked, explain that you provide educational information only and
recommend consulting a licensed financial advisor. Always include
a disclaimer.]
```

The final reminder, placed immediately before generation, increases compliance probability.

2.4 Safety, Guardrails, and Constitutional AI

As conversational agents become more capable and autonomous, the risk of harmful behaviors increases. Systems may generate toxic content, facilitate harmful activities, or reveal sensitive information. Governance controls must be baked into the conversational loop, distinct from the model's raw capabilities.

2.4.1 Guardrails: Classification-Based Defense

Guardrails are typically implemented as separate, lightweight classification models that scan both inputs and outputs (Dong, Zhao, et al. 2024; Inan et al. 2023). These guardrail models effectively "wrap" the main LLM:

1. **Input check**: Before the user input reaches the LLM, the guardrail checks for malicious intent, prohibited requests, or policy violations
2. **Generation**: If input passes, the main LLM generates a response
3. **Output check**: Before returning to the user, the guardrail checks the response for policy violations
4. **Intervention**: If a violation is detected at either stage, the guardrail intercepts the message and replaces it with a standard refusal or safe response

Defense in Depth

Robust systems implement multiple layers of protection:

- **System prompt**: First line of defense, establishing behavioral boundaries
- **Input guardrails**: Catch malicious inputs before they reach the model
- **Output guardrails**: Catch harmful generations before they reach the user
- **Audit logging**: Record all interactions for review and incident response
- **Human escalation**: Mechanism to route sensitive queries to human reviewers

From Guardrails to Governance

Guardrails evolve into governance surfaces in agentic systems. Chapter 7's Governance question addresses how to design agents that can be audited, overridden, and controlled—extending these safety concepts to autonomous operation.

2.4.2 Constitutional AI

A more advanced and theoretically robust approach is **Constitutional AI** (CAI), pioneered by Anthropic (Bai et al. 2022). Instead of relying solely on Reinforcement Learning from Human Feedback (RLHF)—which is hard to scale, expensive, and can encode implicit human biases—CAI uses a "constitution" of explicit principles (e.g., "Please choose the response that is most helpful, honest, and harmless").

The CAI process involves a self-supervised mechanism:

1. **Generation**: The model generates responses to prompts, including potentially harmful ones
2. **Critique**: The model critiques its own response based on the constitution (e.g., "Did this response encourage violence?")
3. **Revision**: The model revises the response to be compliant
4. **Training**: The model is fine-tuned on these revised, safe traces

This method creates a conversational agent that is "aligned" via explicit rules rather than implicit human preferences, making the model's refusal behavior more explainable, consistent, and robust against adversarial attacks.

2.4.3 Practical Safety Prompts

For practitioners without access to fine-tuning, safety must be implemented through prompt design:

- **Explicit prohibitions**: "Do not provide medical diagnoses, legal advice, or recommendations to harm."
- **Uncertainty acknowledgment**: "When uncertain, acknowledge limitations and recommend consulting an expert."
- **Refusal patterns**: "If asked to do X, respond with Y."
- **Clarification triggers**: "If the request is ambiguous or potentially harmful, ask for clarification rather than assuming intent."

Safety and Reasoning.. Note that revealing the model's reasoning (if you use techniques like chain-of-thought, discussed in Section 3) can sometimes conflict with safety. Often we keep detailed reasoning hidden precisely so the model can deliberate freely—even consider and reject a potentially harmful action—without ever exposing a harmful thought to the user. For example, an LLM might internally reason "The user is asking how to do something dangerous—I should refuse," but you wouldn't want it to reply with that reasoning visible. You just want it to refuse. Thus, part of safety is deciding which parts of the model's process to keep private.

2.5 Implementation Patterns for Professional Applications

Before synthesizing the complete architecture, it is worth examining specific implementation patterns that arise in legal and financial contexts. These domains present unique challenges that generic chatbot architectures do not address.

2.5.1 Legal Conversation Patterns

Legal applications require careful attention to several domain-specific constraints:

Privilege Protection.. Attorney-client privilege considerations must be embedded in the system architecture. The conversational system must never expose privileged information to unauthorized parties, even in error messages or debugging logs. This requires:

- Separate storage for privileged and non-privileged conversation content
- Access controls that verify privilege status before retrieving historical context
- Audit trails that distinguish between privileged and non-privileged interactions
- Memory strategies that respect privilege boundaries when summarizing conversations

Jurisdictional Awareness.. Legal analysis varies significantly by jurisdiction. A conversational legal assistant must maintain awareness of:

- The governing jurisdiction for the matter under discussion
- When the user shifts jurisdictional context (e.g., "What about under California law?")
- When an answer depends on jurisdictional variation and should note alternatives
- Conflict of laws considerations when multiple jurisdictions are relevant

These requirements influence memory architecture. The system should pin the primary jurisdiction in persistent memory, flag jurisdictional shifts as significant context changes, and retrieve jurisdiction-specific precedents when constructing prompts.

Temporal Sensitivity.. Legal rules change. A conversation about employment law may produce different answers depending on whether it occurred before or after a significant regulatory change. Conversational legal systems must:

- Track the effective dates of legal rules referenced in responses
- Warn users when advice may be affected by recent or pending legal changes
- Maintain metadata about when information was last verified
- Consider whether to use retrieval to verify current legal status

2.5.2 Financial Conversation Patterns

Financial applications present their own distinct requirements:

Regulatory Boundaries.. Financial services are heavily regulated, and conversational AI must navigate complex compliance requirements:

- Distinguishing between general information and personalized advice (which may trigger regis-

tration requirements)

- Ensuring disclosures are presented when required
- Maintaining records as required by securities, banking, or insurance regulations
- Avoiding forward-looking statements that could constitute market manipulation

Numerical Precision.. Financial conversations frequently involve numerical data where precision matters:

- Currency amounts should be tracked with appropriate precision
- Percentages, ratios, and rates must be consistently formatted
- The system should clarify ambiguous numerical references ("Did you mean 3% annually or monthly?")
- Calculations should be verified, ideally through tool use rather than model inference

Client Risk Profiling.. Financial conversations often need to consider the client's risk profile, which should be:

- Established early in the conversation or retrieved from persistent storage
- Updated when the client indicates changed circumstances
- Consulted before providing any investment-related information
- Protected as sensitive personal information

2.5.3 Multi-Session Continuity

Many professional applications span multiple conversation sessions. A client may return days or weeks later expecting the assistant to remember prior discussions. This requires:

Session Bridging.. When a user returns, the system must efficiently reconstruct relevant context:

- Retrieve high-level summaries of prior sessions
- Identify topics that may be relevant to the new session (based on initial user message)
- Load pinned facts and persistent preferences
- Present a natural transition ("Welcome back. When we last spoke, we were discussing...")

Context Staleness.. Prior conversation context may become stale:

- Facts discussed months ago may no longer be accurate

- The user's situation may have changed
- Regulatory requirements may have evolved
- Market conditions discussed previously may be outdated

Systems should track context freshness and prompt users to confirm critical facts when resuming long-dormant conversations.

Archival and Retrieval.. Professional contexts often require conversation archival:

- Legal hold requirements may prevent deletion of relevant conversations
- Compliance auditors may need to search historical conversations
- Users may need to reference prior discussions for their own records
- The system itself may need to retrieve historical context for current queries

These requirements influence storage architecture, retention policies, and search capabilities.

2.6 Putting It Together: A Conversational Architecture

To synthesize this section, consider the complete architecture of a production conversational system:

Conversational System Components

1. **Orchestration Layer**: Manages state, constructs prompts, handles tool calls
2. **Memory Store**: Persists conversation history, embeddings, pinned facts
3. **Retrieval System**: Searches memory and external knowledge for relevant context
4. **Input Guardrails**: Classifies user input for policy compliance
5. **LLM**: Generates responses given the constructed prompt
6. **Output Guardrails**: Classifies model output for policy compliance
7. **Audit System**: Logs all interactions for compliance and debugging

Each component plays a role in maintaining the illusion of a coherent, stateful, and safe conversational agent built on a fundamentally stateless foundation. The choices you make at each layer—what to remember, where to place instructions, how to structure guardrails—determine the reliability and safety of your system.

With the mechanics of conversational state established, we now turn to the second major challenge: how to elicit structured reasoning from these systems.

3 Reasoning Patterns and When to Use Them

Not every query to an AI is simple. Some questions require step-by-step reasoning, intermediate calculations, or access to external tools and knowledge sources. In this section, we cover different prompting strategies that help the model break down complex tasks or improve its accuracy on challenging problems.

We start with relatively lightweight techniques and progress to more elaborate ones, discussing when each is appropriate. The guiding principle is: *use the simplest strategy that gets the job done reliably*. Overly complex prompting can waste time or even confuse the model if the task didn't need it. On the other hand, a task that requires reasoning or multiple steps will benefit greatly from these patterns.

Figure 3 provides an overview of the reasoning pattern hierarchy, from simple direct prompting to complex graph-based exploration.

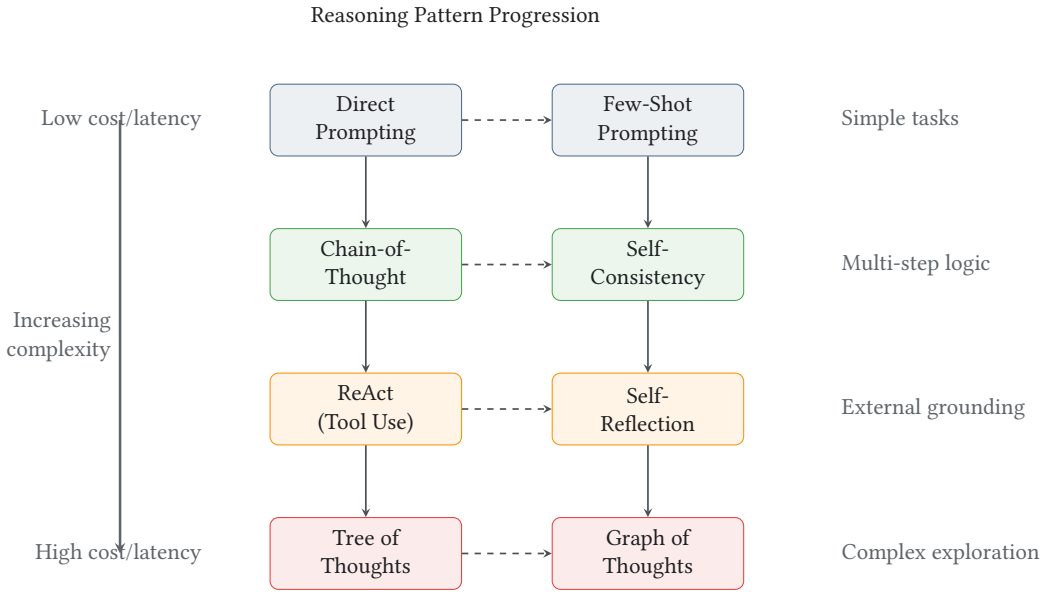

Figure 3: The hierarchy of reasoning patterns, from simple (top) to complex (bottom). Each level adds capability at the cost of increased latency and token usage. Select the simplest pattern that meets your accuracy requirements.

3.1 Chain-of-Thought Prompting

One powerful idea that has emerged is to prompt the model to produce a **chain of thought**—a series of intermediate reasoning steps—before giving the final answer (Wei et al. 2022). Think of this like showing your work in a math problem. Instead of asking directly "What is the result?", you ask the model to reason out loud and then conclude.

3.1.1 Mechanism and Theoretical Basis

The theoretical underpinning of Chain-of-Thought (CoT) is that it decouples the reasoning process from answer generation. In a standard prompt, the model must compute the logic and the output token simultaneously. In CoT, the intermediate tokens serve as a "scratchpad," allowing the model to dump its working memory into the context window. This effectively expands the computational depth of the model, as the attention mechanism can now attend to the intermediate steps it just generated to inform the next step.

Chain-of-Thought Prompting

Chain-of-Thought (CoT) prompting encourages the model to generate intermediate reasoning steps before arriving at a final answer. This decomposes complex problems into simpler sub-problems, improving accuracy on tasks requiring multi-step reasoning.

Research by Wei et al. (2022) showed that providing a few examples of this step-by-step approach in the prompt can significantly improve model performance on complex tasks (Wei et al. 2022). Simply adding examples where the model was guided to articulate intermediate steps—rather than jumping directly to the answer—made a substantial difference in solving arithmetic word problems, logic puzzles, and commonsense questions.

The improvement was dramatic: a large model (540 billion parameters) with chain-of-thought prompting solved math word problems at state-of-the-art levels, whereas it struggled if only given the question with no reasoning steps. For tasks like arithmetic (GSM8K) or symbolic reasoning, standard prompting yields a relatively flat scaling curve—adding more parameters doesn't significantly help. With CoT, performance scales log-linearly with model size, unlocking "emergent" capabilities that smaller models simply cannot perform.

Example: Legal Analysis.. Consider a question about contract liability:

> **Direct prompting**: "Is the seller liable for the defective goods under this contract?"
> **Model response**: "Yes." (potentially incorrect, no reasoning visible)
>
> **Chain-of-thought prompting**: "Analyze the contract liability step by step. First, identify the relevant warranty provisions. Second, determine whether the defect was discoverable at delivery. Third, assess whether proper notice was given. Finally, conclude on liability."
> **Model response**: "Step 1: The contract contains an express warranty in Section 4.2... Step 2: The defect in the heating element was latent and not discoverable... Step 3: Notice was provided within 10 days as required... Conclusion: The seller is likely liable under the express warranty."

The chain-of-thought version forces the model to consider each element systematically, reducing the

chance of overlooking a critical factor.

3.1.2 Private Scratchpads vs. User-Facing Rationales

A critical design decision in CoT implementation is the *visibility* of the reasoning trace:

- **User-facing rationales**: The reasoning is displayed to the user. This builds trust (explainability) and allows the user to verify the logic. However, it can be verbose and distracting for simple queries.
- **Private scratchpads**: The reasoning is generated in a hidden block (parsed out by the application before display) or within a strictly internal "thought" loop. The user sees only the final answer.

When to Keep Reasoning Private

Keep reasoning traces private when:

- The reasoning might explore harmful or sensitive considerations before reaching a safe conclusion
- The user interface should remain clean and simple
- The reasoning is for system debugging, not user consumption
- You want to maintain auditability without cluttering the user experience

Make reasoning visible when:

- Explainability is a requirement (regulatory, trust-building)
- Users need to verify the logic (legal, medical, financial analysis)
- The reasoning process itself is educational

Keeping reasoning traces private offers distinct advantages for safety and simplicity. It allows the model to "think" about sensitive topics or explore dead ends without exposing the user to confusing or potentially unsafe intermediate thoughts. Furthermore, it allows for a cleaner user interface where only the concise, final answer is presented, while the detailed log is retained for auditability.

3.1.3 The Cost of Reasoning: Token Overhead

The primary trade-off of CoT is latency and cost. Generating a rationale requires significantly more tokens than a direct answer. Analysis suggests that for complex math datasets, CoT can increase token usage by approximately 19–40% compared to direct answering (Xu et al. 2025a).

This "inference compute" is often a necessary investment; without it, accuracy on complex tasks drops dramatically. However, new techniques attempt to mitigate this overhead:

- **Chain of Draft (CoD)**: Encourages the model to generate concise, shorthand reasoning (e.g., "5*5=25, +5=30") rather than verbose natural language explanations. Research indicates CoD can achieve similar accuracy to full CoT while using up to 40% fewer tokens (Xu et al. 2025a).
- **TokenSkip**: Controllable compression of reasoning traces, pruning redundant tokens while preserving the logical structure (Xu et al. 2025b).

Practical Guidance.. Chain-of-thought is most useful in tasks like:

- Multi-step mathematics and calculations
- Logical reasoning and puzzle solving
- Complex analysis requiring consideration of multiple factors
- Any scenario where the model might otherwise "jump to a conclusion"

One caution: CoT tends to help more with larger models. Smaller LLMs (under a few billion parameters) often don't benefit and can even get confused by the additional verbosity. Always verify if it actually improves results for your particular case.

3.2 Self-Consistency: Taking a Majority Vote

Chain-of-thought can be taken a step further. What if the model's reasoning process could be run multiple times to see if it arrives at the same answer consistently? This is the idea behind **self-consistency** decoding (Wang et al. 2023a).

3.2.1 Operational Mechanics

Instead of prompting the model once, you prompt it multiple times (say 5 or 10 times) with some randomness injected so it might come up with different reasoning each time. You then examine all the answers it gave. Each chain-of-thought will lead to an answer, and you select the answer that appears most frequently among the outputs via **majority voting**.

Self-Consistency Decoding

Self-consistency generates k independent reasoning chains (samples) for the same query, typically with non-zero temperature ($T \approx 0.7$) to ensure diversity. The final answer is selected via majority voting—marginalizing out the reasoning paths to find the most consistent final answer.

The intuition is that a complex question might have multiple plausible solution paths, but they should converge to the same correct answer. Incorrect reasoning, on the other hand, might produce a variety of wrong answers. By sampling several reasoning paths, we increase the chance of hitting the correct

line of reasoning at least once. Then by taking a majority vote (if one answer appears in 6 out of 10 samples, for example), we eliminate outliers.

3.2.2 Efficacy vs. Compute

Self-consistency yields dramatic improvements. On the GSM8K benchmark, applying self-consistency can boost accuracy by over 17 percentage points compared to a single CoT path (Wang et al. 2023a). For math word problems and commonsense questions, accuracy jumped significantly—in some cases by 10–20 percentage points.

However, the cost is linear with the number of samples (k). Running 10 samples increases inference costs and latency by 10×. This creates a sharp accuracy-cost trade-off.

Self-Consistency Budgeting

- **Sweet spot**: Research suggests 5–10 samples often provide most of the benefit
- **Diminishing returns**: Going from 1 to 5 samples hugely improves reliability; going to 30 samples may not be much better than 10
- **Use selectively**: Reserve for high-stakes queries where the cost is justified

For a non-technical user, think of it this way: if you're unsure about an answer, you might "think it through" multiple times and see if you keep getting the same result. If 4 out of 5 times you arrive at answer A and once at answer B, you'd suspect A is more likely correct. The model can do similarly with self-consistency prompting.

Recent Optimizations.. Techniques like "Slim-SC" attempt to prune redundant chains early. If the first 3 chains produce identical answers, the system stops sampling. Alternatively, model confidence scores can determine if more samples are needed. These optimizations can reduce computational overhead by 25–45% while retaining accuracy benefits.

3.3 Tool-Augmented Reasoning: The ReAct Framework

Sometimes an LLM by itself isn't enough to solve a problem correctly—it might need to look up information or perform a precise calculation. **Tool-augmented reasoning** lets the model use external tools (search engines, calculators, databases) as part of its response. A leading approach integrating this is the **ReAct** framework (short for Reasoning and Acting) (Yao et al. 2023b).

3.3.1 The ReAct Loop

In ReAct, the model interleaves reasoning steps with actions. The pattern structures the interaction as a dynamic loop:

1. **Thought**: The model reasons about what it needs to know
2. **Action**: The model emits a call to a tool (e.g., `Search[query]`)
3. **Observation**: The system executes the tool and pastes the result back into the context
4. **Thought**: The model processes the observation
5. **Answer**: The model generates the final response (or loops back to step 2 if more information is needed)

Figure 4 illustrates this cycle visually.

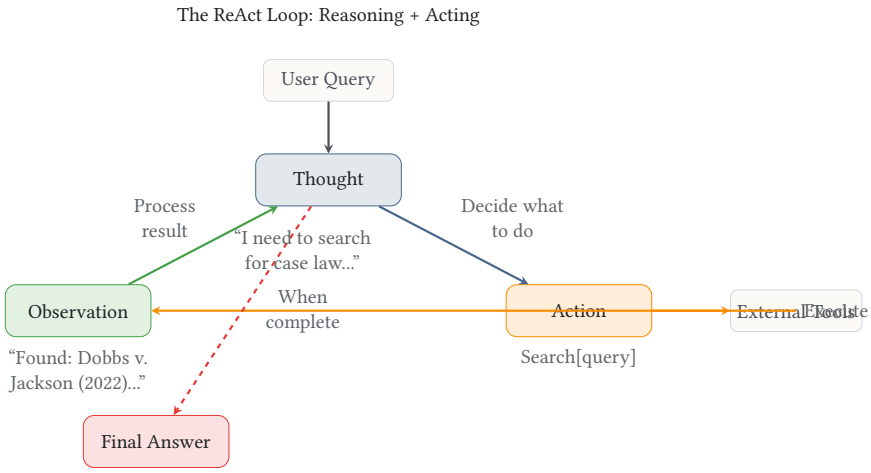

Figure 4: The ReAct loop: the model alternates between reasoning (Thought), taking action via external tools (Action), and processing results (Observation) until it can produce a final answer. This grounds reasoning in real-world data.

ReAct Example: Legal Research

User: "Has the Supreme Court ever overturned a precedent set by Roe v. Wade?"
Internal ReAct trace (not shown to user):

1. *Thought*: "The user is asking about overturning a precedent of Roe v. Wade. I recall Dobbs v. Jackson (2022) might be related. Let me verify."
2. *Action*: `Search[Dobbs Jackson 2022 Roe precedent]`
3. *Observation*: "Dobbs v. Jackson Women's Health Organization (2022) overturned Roe v. Wade."
4. *Thought*: "Confirmed. The precedent was overturned in 2022 by Dobbs."

Final Answer: "Yes. In 2022, the Supreme Court's decision in Dobbs v. Jackson Women's

Health Organization expressly overturned the precedent established by Roe v. Wade."

The user only sees the final answer. But by reasoning and using a tool in tandem, the model ensured accuracy and could provide a reference to the case.

3.3.2 Comparison with Chain-of-Thought

Where CoT is a *closed loop* (internal reasoning only), ReAct is an *open loop* (grounded in external reality). This significantly reduces hallucinations in knowledge-intensive tasks because the model is encouraged to fetch real data for uncertain parts rather than guessing (Yao et al. 2023b).

However, ReAct is inherently slower than CoT due to:

- Network latency of tool calls
- Verbose "thought-action-observation" traces that consume tokens
- Potential for multiple tool calls in a single query

Benchmarks show ReAct outperforms CoT on fact-checking tasks (HotpotQA) but may struggle with pure logic puzzles where no external information is needed, due to the overhead of the tool-use format.

Use Cases for Legal and Financial Professionals..

- **Legal research assistant**: The LLM can query legal databases (Westlaw, LexisNexis) for case citations
- **Financial assistant**: The LLM can pull stock prices, economic data, or run calculations via APIs
- **Due diligence**: The LLM can search document repositories for specific provisions or clauses
- **Regulatory compliance**: The LLM can query regulatory databases for current rules

In all these cases, the model's answer quality is improved by factual grounding rather than relying on potentially outdated training data.

3.3.3 Auditability Considerations

For professional applications, always separate the model's final answer from the reasoning trace. The reasoning + tool usage log is for you (or an expert user) to inspect if needed, but a normal end-user interacting with the AI should just get the polished result. This keeps the experience clean and avoids confusing the user with the model's internal deliberations.

From a governance perspective, the complete ReAct trace provides an audit trail: which tools were

called, what information was retrieved, and how the model synthesized its response. This is invaluable for compliance review and debugging.

ReAct as Proto-Agentic Pattern

The ReAct pattern—reasoning interleaved with action—represents a proto-agentic architecture. Chapters 6–7 formalize when such systems qualify as genuine agents and how to architect them. Chapter 6 provides a rigorous framework distinguishing agents from tool-using systems, while Chapter 7's Planning and Action questions address how agents decompose tasks and execute tool calls.

3.4 Advanced Topologies: Tree of Thoughts and Graph of Thoughts

Linear chains (CoT) and loops (ReAct) are sometimes insufficient for problems requiring exploration, backtracking, or combining multiple distinct ideas.

3.4.1 Tree of Thoughts (ToT)

Tree of Thoughts generalizes CoT by framing the reasoning process as a search over a tree (Yao et al. 2023a). At each step, the model generates multiple possible "next thoughts" (branches). It then self-evaluates these branches (using a heuristic or a prompt like "Is this a promising direction?") and proceeds only with the best ones, backtracking if a dead end is reached.

Tree of Thoughts

Tree of Thoughts (ToT) implements deliberate problem solving by:

1. Generating multiple candidate next steps at each node
2. Evaluating candidates for promise (self-evaluation)
3. Pruning unpromising branches
4. Backtracking when dead ends are reached

This is analogous to implementing Breadth-First Search (BFS) or Depth-First Search (DFS) via prompting.

ToT is particularly effective for planning tasks. In the "Game of 24" puzzle (form the number 24 using four numbers and basic operations), GPT-4 with standard CoT achieved only a 4% success rate, while ToT achieved 74% (Yao et al. 2023a).

When to Use ToT..

- Creative writing requiring exploration of alternatives

- Planning tasks with multiple valid approaches
- Puzzles and games requiring search
- Design problems with trade-offs

3.4.2 Graph of Thoughts (GoT)

Graph of Thoughts further extends this by modeling reasoning as a Directed Acyclic Graph (DAG) (Besta et al. 2024). It introduces operations that are impossible in trees:

- **Aggregation**: Combining multiple distinct thoughts into a stronger solution
- **Refinement**: Looping back to improve a thought based on later insights

Graph of Thoughts

Graph of Thoughts (GoT) models reasoning as a DAG, enabling:

- **Branching**: Exploring multiple paths simultaneously
- **Aggregation**: Combining insights from different branches
- **Refinement**: Iteratively improving partial solutions

This resembles MapReduce or dynamic programming patterns in computation.

For example, in a sorting task, GoT can break the list into sub-lists, sort them individually (Map), and then merge them (Reduce/Aggregate). While GoT offers the highest ceiling for complex creative or analytical tasks, the orchestration overhead is massive, making it suitable only for offline, high-value tasks where latency is secondary to quality.

Taxonomy of Reasoning Topologies.. Besta et al. (2025) provide a comprehensive taxonomy categorizing reasoning patterns from linear chains through branching trees to fully connected graphs, analyzing the computational trade-offs at each level (Besta et al. 2025).

These exploration patterns inform agent planning. Chapter 7 addresses how agents break complex jobs into steps, manage dependencies, and backtrack when needed—operationalizing ToT/GoT as planning strategies.

3.5 Self-Reflection and Critique Loops

Wouldn't it be useful if the AI could check its own work? There is a class of techniques where the model is prompted to critique, verify, or improve its initial answer. This approach mimics human metacognition—thinking about thinking.

3.5.1 The Critique-Improve Pattern

One simple approach: after the model provides an answer, you (automatically) ask it, "Are you sure about that? Is there any mistake in your solution?" Surprisingly, the model might catch its own error and correct it. This doesn't always work, but it often can for calculation mistakes or logical inconsistencies, especially if the model is high capacity.

A more systematic version is the **critique-and-revision loop**:

1. **Generate**: Model produces an initial draft
2. **Critique**: Model is prompted to evaluate the draft against criteria (e.g., "Critique this for errors, bias, or clarity")
3. **Revise**: Model generates a revised version based on its critique

Self-Reflection Example: Investment Recommendation

Initial response: "You should invest in technology stocks for growth."
Critique prompt: "Critique the above advice for any financial risks or missing considerations."
Model critique: "The advice doesn't consider the user's age, risk tolerance, existing portfolio allocation, or current market conditions. It also fails to include required disclaimers."
Revision prompt: "Given the critique, provide a more comprehensive response."
Revised response: "Technology stocks can offer growth potential, but the right allocation depends on your age, investment timeline, and risk tolerance. Consider consulting a licensed financial advisor who can assess your complete financial picture. This is general educational information, not personalized investment advice."

Research by Shinn et al. (2023) demonstrated that an iterative "reflective" strategy enabled agents to significantly improve performance on coding tasks and decision-making games by learning from their own mistakes (Shinn et al. 2023). Li & Zhao (2025) showed that self-reflection enhances LLM responses in academic contexts (Li and Zhao 2025).

In agentic contexts, self-consistency informs termination conditions—how agents know when they've reached a reliable answer. Chapter 7's Termination question addresses explicit stopping conditions, goal satisfaction verification, and resource budgets.

3.5.2 Practical Considerations

- **Limit iterations**: One round of self-reflection and revision yields most of the benefit. Beyond that, you risk the model oscillating or over-correcting. Two iterations is usually the maximum.
- **Different modes**: The critique prompt forces the model to focus on evaluation rather than generation, engaging a different "mode" of thinking (more analytical). This is why it can catch

errors it made during generation.

- **Keep critiques private**: The critique may contain sensitive exploration. Show only the final revised answer to users unless explainability requires otherwise.
- **High-stakes applications**: Self-reflection is particularly valuable when the cost of an error is high. In legal or medical contexts, a second opinion (even from the same model acting as a "critic") is valuable.

3.6 Few-Shot Examples: Teaching by Demonstration

Few-shot prompting is one of the simplest yet most effective ways to shape an LLM's behavior. It means giving the model a few examples of what you want within the prompt itself before asking it to perform. Think of it as on-the-fly training.

3.6.1 How Few-Shot Learning Works

This leverages the fact that large models have learned to continue patterns. The original GPT-3 paper demonstrated that providing a few examples of a task in the prompt allows the model to generalize to similar tasks without any parameter updates (Brown et al. 2020). In other words, GPT-3 showed that language models are *few-shot learners*. This property means we can guide them with just text examples instead of explicitly programming new rules.

Few-Shot Prompting

Few-shot prompting provides 1–5 examples of the desired input-output pattern within the prompt. The model generalizes from these examples to handle new inputs in the same format/style.

Example Structure..

```
Example 1: [Input A] → [Output A]
Example 2: [Input B] → [Output B]
Example 3: [Input C] → [Output C]
Now your turn: [New Input] → ?
```

The model will usually imitate the style and structure of the examples.

3.6.2 Designing Effective Examples

Types of Examples..

- **Positive exemplars**: Show what good outputs look like. If designing a contract analysis bot, provide a mock contract clause and an ideal analysis.

- **Negative exemplars**: Sometimes it helps to show what *not* to do. Use sparingly; heavy use can confuse the model. One or two can clarify boundaries.
- **Boundary cases**: If there are commonly tricky cases in your domain, include an example of how to handle one. For a financial assistant, a boundary case might be a user asking for stock predictions (which you want the assistant to refuse).

Example Selection Principles

1. **Relevance**: Select examples similar to the expected queries
2. **Diversity**: Cover different scenarios, not just variations of one
3. **Clarity**: Each example should have unambiguous input-output mapping
4. **Recency**: Use up-to-date examples reflecting current practices
5. **Locality**: Prefer examples from the same domain, jurisdiction, or context

3.6.3 Similarity-Based Selection

Selecting examples that are relevant to the user's query is crucial. Research shows that retrieving in-context examples that closely match the query's topic or wording can significantly improve performance, more so than random selection (Zebaze Dongmo et al. 2024).

The **K-Nearest Neighbors (K-NN)** approach involves:

1. Embedding the user's current query into a vector space
2. Retrieving the k most semantically similar examples from a labeled set
3. Including those examples in the prompt

For instance, if the user asks a question about mergers and acquisitions law, it's better to include few-shot examples about M&A legal analyses rather than personal injury examples, even if the logical task is similar.

3.6.4 Maximal Marginal Relevance for Diversity

A potential pitfall of pure K-NN is redundancy. If the user asks about "apples," K-NN might retrieve five examples about "apples." This lacks diversity and fails to show the model how to handle variations.

Maximal Marginal Relevance (MMR) addresses this by optimizing for both relevance and diversity:

$$\text{MMR} = \lambda \cdot \text{Sim}(d, q) - (1 - \lambda) \cdot \max_{d_j \in S} \text{Sim}(d, d_j)$$

where d is the candidate example, q is the query, S is the set of already selected examples, and λ balances relevance vs. diversity (typically $\lambda = 0.5$).

By ensuring few-shot examples cover a diverse range of reasoning patterns while remaining relevant, MMR prevents the model from overfitting to a narrow pattern.

3.6.5 Bootstrapping Exemplars

When few-shot examples are scarce (the "cold start" problem), you can use the model itself to generate candidates:

1. Use a strong model to generate candidate chains-of-thought for a set of questions
2. Filter these candidates: keep only those leading to correct final answers
3. Use these correct, synthetic chains as few-shot examples

This technique, sometimes called "Self-Instruct," allows rapid creation of high-quality exemplar libraries without manual human annotation (Wang et al. 2023b).

Bootstrapping Best Practices

- **Human review**: Always review model-generated examples before deployment; they may contain subtle errors or biases
- **Track provenance**: Note which examples are real vs. AI-generated
- **Version control**: Maintain history of your exemplar library as it evolves
- **Domain expertise**: For legal/medical/financial domains, have an expert vet synthetic examples

3.7 Reasoning in Professional Domains

The reasoning patterns described above take on specific characteristics when applied to legal and financial contexts. These domains require not only accurate reasoning but also adherence to domain-specific norms of argumentation and evidence.

3.7.1 Legal Reasoning Patterns

Legal analysis presents distinctive requirements that influence how we structure reasoning traces:

Issue-Rule-Application-Conclusion (IRAC).. Legal analysis traditionally follows a structured format: identify the legal issue, state the governing rule, apply the rule to the facts, and draw a conclusion. Chain-of-thought prompting can be structured to mirror this pattern:

- **Issue identification**: What legal question does this fact pattern present?
- **Rule statement**: What statutes, regulations, or case holdings govern this issue?
- **Application**: How do the specific facts map onto the legal rule's elements?
- **Conclusion**: What outcome follows from this analysis?

Few-shot examples that demonstrate this IRAC structure prime the model to organize its reasoning in a legally recognizable format.

Analogical Reasoning.. Legal reasoning frequently involves arguing by analogy to precedent cases. LLMs can perform analogical reasoning, but benefit from explicit prompting:

- "Compare the facts of this case to [precedent]. What are the key similarities and differences?"
- "Identify the distinguishing facts that might lead to a different outcome."
- "Rank the cited cases by relevance to the current facts."

Self-consistency is particularly valuable here, as different reasoning chains may identify different analogies, and majority voting can surface the most robust comparisons.

Counter-Argument Generation.. Legal advocacy requires anticipating opposing arguments. LLMs can be prompted to generate counter-arguments:

- "What would opposing counsel argue in response to this analysis?"
- "Identify the weakest points in this legal position."
- "Steelman the opposing position—what's the strongest case against our conclusion?"

This form of self-reflection helps identify vulnerabilities in legal analysis before they are exploited by adversaries.

Authority Verification.. Legal reasoning depends on valid authority. LLMs are prone to hallucinating case citations, statute numbers, and legal rules. ReAct patterns are essential for legal applications:

- Never cite a case without retrieval verification
- Verify that cited statutes remain current (not repealed or amended)
- Confirm that regulatory provisions are still effective
- Check for subsequent history that may have overruled a precedent

3.7.2 Financial Reasoning Patterns

Financial analysis requires its own adaptations of reasoning patterns:

Quantitative Reasoning Chains.. Financial analysis often involves sequential calculations. Chain-of-thought prompting should explicitly trace the computational steps:

- State each input value and its source
- Show intermediate calculations explicitly
- Carry units through the computation (dollars, percentages, shares)
- Verify the final result with a reasonableness check

For high-stakes calculations, consider using tool calls to perform arithmetic rather than relying on the model's internal computation, which is notoriously unreliable for complex math.

Scenario Analysis.. Financial decision-making often requires analyzing multiple scenarios. Tree of Thoughts is naturally suited to this:

- Base case: Expected market conditions
- Upside case: Favorable outcomes
- Downside case: Adverse developments
- Stress case: Extreme but plausible scenarios

Each branch can be explored independently, with conclusions synthesized across scenarios to inform risk-aware decisions.

Regulatory Compliance Reasoning.. Financial analysis must consider regulatory constraints. The reasoning trace should explicitly:

- Identify applicable regulatory frameworks
- Check whether proposed actions trigger regulatory requirements
- Verify compliance with disclosure obligations
- Flag potential conflicts of interest

Market Data Integration.. Financial reasoning requires current market data. ReAct patterns should incorporate:

- Real-time price lookups when valuations are needed

- Economic indicator retrieval for macroeconomic analysis
- Earnings data retrieval for company-specific analysis
- Rate lookups for fixed-income and derivatives calculations

Unlike legal research, where documents are relatively static, financial data is continuously changing. Systems must be designed to refresh data appropriately and timestamp conclusions.

3.7.3 Professional Domain Challenges

Both legal and financial domains share common challenges that influence reasoning pattern selection:

Explainability Requirements.. Both domains may require the ability to explain how conclusions were reached. This favors reasoning patterns that produce interpretable traces:

- Chain-of-thought produces linear, readable explanations
- Tree of Thoughts shows alternative paths considered
- ReAct shows evidence consulted
- Self-consistency shows agreement across multiple analyses

Patterns that aggregate or compress reasoning (like majority voting without trace preservation) may sacrifice explainability.

Audit Trail Requirements.. Regulated industries often require detailed records of decision processes. Reasoning traces should be:

- Timestamped with when the analysis was performed
- Associated with the model version used
- Linked to the specific data/documents consulted
- Preserved for the required retention period

Human-in-the-Loop Integration.. High-stakes professional applications typically require human review. Reasoning patterns should facilitate this:

- Produce structured outputs that highlight key decision points
- Flag areas of uncertainty or low confidence
- Present alternative conclusions where appropriate
- Provide sufficient detail for expert review without overwhelming verbosity

3.8 Selecting the Right Reasoning Pattern

With this toolkit of reasoning patterns, how do you decide which to use? The next section provides a decision framework, but here are initial heuristics:

Reasoning Pattern Selection Heuristics

- **If the task is straightforward**: Use direct prompting (zero-shot or simple few-shot). Don't complicate unnecessarily.
- **If the task requires multi-step logic**: Use Chain-of-Thought. The model needs to "show its work."
- **If correctness is paramount**: Use Self-Consistency. Multiple paths reduce the chance of a flawed single chain.
- **If up-to-date information is needed**: Use ReAct with appropriate tools. The model's training data may be outdated.
- **If the problem requires exploration**: Use Tree of Thoughts. Planning and creative tasks benefit from trying multiple approaches.
- **If quality trumps latency**: Use Graph of Thoughts or iterative self-reflection. Reserve for high-value, offline analysis.

The following section elaborates on these trade-offs and provides a more structured decision framework.

4 Choosing Strategies Under Constraints

The selection of a conversational model design and reasoning pattern is not a binary choice but a multi-dimensional optimization problem. Every application operates under constraints—time, cost, risk tolerance, accuracy requirements—and the optimal strategy depends on how you weight these factors. In this section, we synthesize the patterns discussed earlier into a practical decision framework.

4.1 The Decision Matrix

Different tasks call for different approaches. The following matrix provides guidance based on task type and requirements:

Task Type	Recommended Strategy	Context/Memory	Rationale
Chatbots / Creative Writing	Zero-shot or few-shot	Sliding window / Summarization	Latency is critical. Reasoning errors are low-risk.
Math Tutoring / Coding	Chain-of-Thought	K-NN few-shot selection	Precision is key. Users tolerate slight delay for correctness.
Medical / Financial Analysis	Self-Consistency	Vector retrieval (high recall)	**High risk**. Error cost outweighs compute cost.
Research Assistants	ReAct	Long-term vector memory	Model must interact with tools for real-time data.
Complex Planning / Discovery	Tree of Thoughts	Full context replay	Exploration is needed. High latency acceptable.

4.2 The Four Dimensions of Strategy Selection

When selecting a strategy, consider four key dimensions:

4.2.1 Accuracy Requirements

- **Low-stakes**: Casual conversation, creative writing, brainstorming. Hallucinations are tolerable; speed matters more.
- **Medium-stakes**: Educational content, summarization, drafting. Errors are undesirable but not catastrophic; can be caught in review.
- **High-stakes**: Legal advice, medical diagnosis, financial recommendations. Errors can cause real harm; maximum reliability required.

High-Stakes Decision Rule

For high-stakes applications:

1. Always use structured reasoning (CoT or better)
2. Consider self-consistency for critical determinations
3. Ground in external sources (ReAct + retrieval)

4. Include human review in the workflow
5. Log everything for audit

4.2.2 Latency Tolerance

- **Real-time** (< 2 seconds): Chatbots, autocomplete, live assistance. Use direct prompting; minimize reasoning overhead.
- **Interactive** (2–30 seconds): Research queries, analysis, document review. CoT and single-pass retrieval acceptable.
- **Batch/Offline** (> 30 seconds): Deep analysis, report generation, complex planning. Full self-consistency, ToT/GoT viable.

4.2.3 Cost Sensitivity

- **Cost-sensitive**: High-volume consumer applications. Minimize tokens; use smaller models where possible.
- **Moderate budget**: Professional tools with per-query value. Invest in accuracy for high-value queries.
- **Budget unconstrained**: Mission-critical applications. Maximize accuracy regardless of cost.

The cost of self-consistency scales linearly with samples (k samples = $k\times$ base cost). Token-efficient techniques like Chain of Draft can reduce CoT overhead by 40%.

4.2.4 Explainability Requirements

- **Black-box acceptable**: Internal tools, automation. Hide reasoning for simplicity.
- **Explainability preferred**: Professional tools. Offer reasoning on request.
- **Explainability mandatory**: Regulated domains (finance, healthcare, legal). Must show reasoning for compliance.

4.3 Context Assembly: The Final Step

Many conversational failures are, at their root, *context failures*. Before the model even begins to reason, you must construct the prompt correctly. Here is a systematic approach:

4.3.1 The Context Assembly Recipe

Assemble your prompt in this order:

1. **System Prompt**: Identity, constraints, safety rules
2. **Few-Shot Exemplars**: Selected via similarity (K-NN) and diversity (MMR)
3. **Retrieved Context**: Documents, data, or prior knowledge (RAG)
4. **Conversation History**: Summarized older turns + recent active window
5. **Current User Query**: The actual question or request
6. **Final Reminder**: Output constraints, format requirements, critical rules

Why Order Matters

Due to the "Lost in the Middle" phenomenon:

- Information at the **start** (system prompt) is retained well
- Information in the **middle** (retrieved context, history) is partially degraded
- Information at the **end** (query + reminder) receives highest attention

Place your most critical constraints at both the start AND end.

4.3.2 Token Budgeting in Practice

Before each call, verify your token allocation:

1. **Check total capacity**: Know your model's context limit (8K, 32K, 128K, etc.)
2. **Reserve for output**: Allocate 1K–4K tokens for the response, depending on expected length
3. **Allocate components**: Divide remaining budget across system prompt, examples, context, and history
4. **Truncate if necessary**: If over budget, compress history first (summarize), then reduce retrieved context (re-rank for relevance), then reduce examples
5. **Never truncate**: System prompt or final reminder (these are non-negotiable)

4.4 Application-Specific Guidance

4.4.1 Legal Applications

Legal AI Strategy Recommendations

- **Legal research**: ReAct with access to case databases; cite all sources
- **Document review**: Few-shot with domain-specific examples; CoT for analysis
- **Contract analysis**: CoT with explicit checklist prompting; self-consistency for critical clauses
- **Compliance questions**: ReAct for regulatory lookups; never rely solely on training data for current regulations
- **Always**: Include disclaimers; acknowledge limitations; recommend human review

4.4.2 Financial Applications

Financial AI Strategy Recommendations

- **Market data queries**: ReAct with API access; never use training data for prices
- **Risk analysis**: Self-consistency for quantitative assessments
- **Portfolio suggestions**: CoT with explicit constraint checking (risk tolerance, time horizon)
- **Regulatory queries**: ReAct for up-to-date rule lookups (SEC, FINRA, etc.)
- **Always**: Include suitability disclaimers; distinguish information from advice

4.4.3 Customer-Facing Applications

For general-purpose assistants where latency and user experience matter:

- Use direct prompting or simple few-shot for routine queries
- Escalate to CoT only for detected complexity (e.g., multi-part questions)
- Implement guardrails at both input and output stages
- Keep responses concise; offer to elaborate if user requests
- Use sliding window + summary for conversation history

4.5 Governance Implications

Your choice of reasoning strategy has governance implications:

Auditability.. Strategies that produce reasoning traces (CoT, ReAct) provide audit trails. Even if you hide traces from users, log them for compliance review.

Explainability.. Regulated industries may require explanations. CoT provides this naturally; private scratchpads can be disclosed on request.

Reproducibility.. Self-consistency with majority voting is more reproducible than single-sample generation (which varies with temperature). For regulated applications, consider deterministic settings (temperature = 0) or explicit seed values.

Version Control.. Track your system prompts, few-shot examples, and strategy configurations. When issues arise, you need to reproduce the exact conditions that produced a problematic output.

What Part III Covers

This chapter provides foundational strategies. Part III (Governing AI Agents) expands on:

- Comprehensive governance frameworks for AI deployment
- Regulatory requirements across jurisdictions
- Risk assessment methodologies
- Incident response and escalation protocols
- Organizational structures for AI oversight

4.6 Decision Flowchart

When facing a new task, use this simplified decision process:

1. **Assess stakes**: Is this high-risk (legal, medical, financial) or low-risk?
2. **Check data needs**: Does the task require current information beyond training data?
3. **Evaluate complexity**: Is multi-step reasoning required?
4. **Consider constraints**: What are the latency and cost limits?
5. **Select pattern**:
   - Low-risk, simple, real-time → Direct prompting
   - Low-risk, complex, real-time → CoT
   - Any risk, needs current data → ReAct
   - High-risk, complex → Self-consistency + CoT

- High-value, complex, offline → ToT/GoT

6. **Design memory**: Choose appropriate context management for conversation length
7. **Implement guardrails**: Add safety layers appropriate to risk level
8. **Test and iterate**: Validate on representative examples before deployment

The right strategy is the *simplest one that meets your accuracy and safety requirements.* Over-engineering wastes resources and can introduce unnecessary complexity. Start simple, measure results, and add sophistication only where needed.

4.7 Case Studies in Strategy Selection

To illustrate how these principles apply in practice, consider the following scenarios drawn from legal and financial contexts.

4.7.1 Case Study 1: Legal Research Assistant

Scenario: A law firm wants to deploy an AI assistant that helps associates research case law and statutes relevant to client matters.

Analysis:

- **Stakes**: High. Incorrect legal citations could embarrass the firm or, worse, lead to malpractice.
- **Data needs**: Critical. Legal research requires access to current case law and statutes.
- **Complexity**: High. Legal analysis requires multi-step reasoning, analogy, and application of rules to facts.
- **Constraints**: Moderate latency tolerance (research tasks are not real-time); cost is acceptable for high-value work.

Recommended Strategy:

- **ReAct with retrieval**: Absolutely essential. The model must search legal databases rather than rely on training data.
- **Chain-of-thought with IRAC structure**: Produce reasoning traces that follow the Issue-Rule-Application-Conclusion format familiar to legal professionals.
- **Self-consistency**: For important conclusions, run multiple reasoning paths and verify agreement.
- **Private scratchpad**: Keep exploratory reasoning hidden; show only polished analysis to users.
- **Citation verification guardrail**: Before presenting any case citation, verify it exists and check its subsequent history.

Memory Configuration: Long conversations tracking matter details. Pin client facts, governing jurisdiction, and key deadlines. Use recursive summarization for extended research sessions.

4.7.2 Case Study 2: Financial News Summarization

Scenario: An investment firm wants to automatically summarize daily financial news for portfolio managers.

Analysis:

- **Stakes**: Moderate. Summaries inform but don't directly drive trades; errors are caught in review.
- **Data needs**: Critical. News summarization inherently requires current content.
- **Complexity**: Moderate. Summarization is well within LLM capabilities.
- **Constraints**: Daily batch processing; cost-sensitive due to high volume; moderate latency tolerance.

Recommended Strategy:

- **ReAct with news retrieval**: Fetch articles from trusted sources.
- **Simple chain-of-thought**: Identify key points, note market implications, synthesize.
- **Single-pass generation**: Self-consistency is overkill for summarization.
- **Visible reasoning**: Summary structure naturally shows reasoning.
- **Topic-based guardrails**: Flag articles about portfolio companies for human attention.

Memory Configuration: Each news item is independent; minimal cross-article context needed. Consider tagging summaries with topics for later retrieval.

4.7.3 Case Study 3: Client Risk Assessment Chatbot

Scenario: A wealth management firm wants a chatbot that gathers client information and produces preliminary risk assessments.

Analysis:

- **Stakes**: High. Regulatory requirements around suitability assessments; fiduciary duties.
- **Data needs**: Low. Assessment based on client-provided information, not external data.
- **Complexity**: Moderate. Structured questionnaire with conditional logic.
- **Constraints**: Real-time (conversation with client); must feel natural and responsive.

Recommended Strategy:

- **Direct prompting with few-shot examples**: Model appropriate conversational patterns for gathering information.
- **Simple chain-of-thought for classification**: Show reasoning for risk category assignment.
- **No retrieval**: Assessment based on conversation content only.
- **Strict guardrails**: Never provide investment advice; only gather information and classify.
- **Mandatory human review**: Assessment is preliminary; always routed to advisor for confirmation.

Memory Configuration: Track all client responses in current session. Pin stated risk tolerance and investment horizon. Clear session-specific context between clients but maintain templates.

4.7.4 Case Study 4: Contract Review Workflow

Scenario: A corporate legal department wants AI assistance reviewing vendor contracts for risk terms.

Analysis:

- **Stakes**: High. Missing a problematic clause could expose the company to significant liability.
- **Data needs**: Moderate. Contract review is primarily about the document itself, but comparison to company playbook may require retrieval.
- **Complexity**: High. Contracts require careful parsing, cross-reference of definitions, and risk assessment.
- **Constraints**: Batch processing acceptable; thoroughness valued over speed.

Recommended Strategy:

- **Tree of Thoughts**: Explore the contract systematically by section, considering multiple interpretations of ambiguous language.
- **ReAct for playbook comparison**: Retrieve company's standard positions and risk thresholds for each clause type.
- **Self-consistency for risk ratings**: Multiple evaluations of high-risk clauses to verify assessment.
- **Structured output**: Produce standardized risk reports with clause-by-clause analysis.
- **Escalation triggers**: Automatically flag contracts meeting certain criteria for senior review.

Memory Configuration: Contract context must persist throughout review. Pin key definitions and parties. Maintain session with all identified issues for final summary.

4.7.5 Lessons from Case Studies

These examples illustrate several recurring patterns:

Strategy Selection Principles

1. **Match stakes to verification**: Higher stakes justify more expensive verification strategies (self-consistency, human review).
2. **Use retrieval when data matters**: Any task involving facts beyond training data requires tool-augmented reasoning.
3. **Structure reasoning for the audience**: Legal tasks benefit from IRAC; financial tasks from structured scenarios.
4. **Consider the full workflow**: AI is rarely the final word in professional contexts; design for handoff.
5. **Start simple**: Begin with the simplest strategy that might work; add complexity only when measurement shows it's needed.

5 Synthesis: From Stateless Models to Cognitive Systems

We have moved from viewing the LLM as a simple text predictor to seeing it as the kernel of a *cognitive operating system*—one that requires explicit memory management, structured reasoning topologies, and rigorous safety constitutions to function effectively in professional settings.

5.1 The Three Pillars of Conversational AI

This chapter has established three foundational pillars for building reliable conversational systems:

Pillar 1: State Management

Creating memory in a memoryless system.
Since LLMs are fundamentally stateless, maintaining conversational coherence requires an orchestration layer that:

- Constructs prompts with appropriate role separation (system, user, assistant)
- Manages context windows through sliding windows, summarization, or vector retrieval

- Places critical instructions strategically to combat attention degradation
- Implements safety guardrails at input and output stages

The illusion of memory is created by careful context construction, not by the model itself.

Pillar 2: Structured Reasoning

Moving beyond pattern matching to genuine inference.
Complex professional tasks require reasoning capabilities that raw LLMs lack. We achieve this through:

- **Chain-of-Thought**: Externalizing intermediate reasoning steps
- **Self-Consistency**: Sampling multiple paths and voting on answers
- **Tool Augmentation**: Grounding in external data sources (ReAct)
- **Exploration**: Tree and Graph of Thoughts for complex planning
- **Self-Reflection**: Critique and revision loops for quality improvement

Each technique trades off accuracy, latency, and cost differently.

Pillar 3: Strategic Selection

Matching techniques to requirements.
No single approach works for all tasks. Effective deployment requires:

- Assessing the stakes (risk tolerance for errors)
- Evaluating latency and cost constraints
- Determining explainability requirements
- Selecting the simplest strategy that meets requirements
- Implementing appropriate governance controls

The goal is not maximum sophistication but appropriate sophistication.

5.2 Key Takeaways

Chapter Summary

1. **Conversations require state management**: The model doesn't remember anything; your application must manage context explicitly through prompt construction.
2. **Position matters**: Due to the "Lost in the Middle" phenomenon, place critical information at the start and end of prompts, not buried in the middle.
3. **Reasoning improves accuracy**: Chain-of-thought and related techniques dramatically improve performance on complex tasks, but at the cost of additional tokens and latency.
4. **Tools ground in reality**: ReAct-style approaches reduce hallucinations by connecting the model to real-time data sources, essential for legal and financial applications.
5. **Multiple paths increase reliability**: Self-consistency (sampling multiple reasoning chains and voting) provides robust answers for high-stakes decisions.
6. **Safety is layered**: Effective guardrails combine system prompts, input/output classifiers, and human escalation paths.
7. **Simpler is often better**: Use the minimum necessary complexity; over-engineering wastes resources and can introduce new failure modes.

5.3 The Integration of State and Reasoning

These pillars are not independent. Effective reasoning often requires sophisticated state management:

- **ReAct** requires maintaining state across thought-action-observation cycles
- **Self-consistency** requires aggregating results across multiple independent runs
- **Tree/Graph of Thoughts** requires tracking branching states and backtracking
- **Self-reflection** requires maintaining drafts and critiques across iterations

Similarly, state management decisions affect reasoning capabilities:

- **Context window limits** constrain how much reasoning trace can be maintained
- **Memory strategies** determine what prior reasoning is available for reference
- **Few-shot example selection** primes the reasoning patterns the model will employ

Understanding this interplay is essential for designing robust conversational systems.

5.4 What We Have Not Covered

This chapter focused on the fundamental mechanics of conversations and reasoning. Several important topics are deferred to subsequent chapters:

Covered in Later Chapters

- **Structured Outputs** (Chapter 3): Forcing models to produce JSON, XML, or other schema-conformant outputs
- **Tool Use and Function Calling** (Chapter 3): Implementation details for integrating external APIs and tools
- **Retrieval-Augmented Generation** (Chapter 4): The full architecture for document retrieval and grounding
- **Agent Architectures** (Part II): Autonomous systems with planning, execution, and learning loops
- **Governance Frameworks** (Part III): Comprehensive regulatory and organizational considerations

What Comes Next: From Conversations to Agents

This chapter established the foundations for conversational AI and structured reasoning. These techniques become building blocks for agentic systems:

- **Chapter 6 (What is an Agent?)**: Provides a rigorous conceptual framework—the six-property rubric that distinguishes genuine agents from sophisticated tools
- **Chapter 7 (How to Design an Agent?)**: Translates these concepts into ten architectural questions that guide agent design, from Triggers to Governance

The memory strategies, reasoning patterns, and safety mechanisms introduced here scale to autonomous systems that iterate, adapt, and take action in the world.

5.5 Bridge to Structured Outputs and Tools

The techniques in this chapter prepare you for the next level of LLM capability: producing structured, machine-readable outputs and interacting with external systems.

From Conversations to Actions.. We introduced ReAct as a reasoning pattern, but we only sketched how tool calls actually work. The next chapter explains the mechanics of function calling, API integration, and the protocols that enable models to take actions in the world.

From Free Text to Schemas.. Reasoning produces answers, but professional applications often require those answers in specific formats. The next chapter covers techniques for constraining model outputs to conform to JSON schemas, legal citation formats, financial report structures, and other domain-specific templates.

From Memory to Knowledge.. We touched on vector-enhanced memory, but the full Retrieval-Augmented Generation (RAG) architecture involves sophisticated chunking, embedding, indexing, and retrieval strategies. These details follow in Chapter 4 (Retrieval and Knowledge).

With the foundations of state management and reasoning established, you are ready to explore how LLMs can produce structured outputs, use tools, and integrate with the broader ecosystem of professional applications.

6 Further Learning

This section provides an annotated guide to the foundational literature on conversational AI and reasoning in large language models. We organize sources by topic, with brief descriptions of what each contributes and why it matters for practitioners.

6.1 Foundational Papers on Reasoning

Chain-of-Thought Prompting.. Wei et al. (2022) introduced the chain-of-thought prompting technique, demonstrating that simply asking models to "think step by step" dramatically improves performance on arithmetic and symbolic reasoning tasks. This paper established that reasoning capabilities can be elicited through prompting alone, without fine-tuning. Essential reading for understanding why multi-step reasoning works in LLMs.

Self-Consistency.. Wang et al. (2023a) extended chain-of-thought by sampling multiple reasoning paths and selecting the most consistent answer via majority voting. Their experiments showed 10–20 percentage point improvements on challenging benchmarks. This paper is crucial for understanding how to improve reliability in high-stakes applications through redundancy.

ReAct Framework.. Yao et al. (2023b) introduced the paradigm of interleaving reasoning with tool use, showing that grounding model reasoning in external observations significantly reduces hallucinations. This paper bridges the gap between pure reasoning and agentic behavior, making it foundational for building research assistants and fact-checking systems.

Tree of Thoughts.. Yao et al. (2023a) generalized chain-of-thought into a tree structure with exploration and backtracking. Their results on planning tasks (Game of 24, creative writing) demonstrated that deliberate search over reasoning paths can dramatically outperform linear chains. Important for complex planning and creative applications.

Graph of Thoughts.. Besta et al. (2024) extended reasoning to directed acyclic graphs, enabling aggregation and refinement operations. Their taxonomy paper (Besta et al. 2025) provides a comprehensive framework for understanding the landscape of reasoning topologies. Advanced reading for those designing sophisticated reasoning systems.

6.2 Context and Memory Management

Lost in the Middle.. Liu et al. (2024) provided the definitive empirical analysis of how LLMs access information across long contexts, demonstrating the U-shaped retrieval curve. This paper fundamentally changed how practitioners think about prompt construction and is essential reading for anyone building systems with long context windows.

PagedAttention.. Kwon et al. (2023) introduced the memory management technique that enables efficient serving of large context windows. While technical, this paper explains the infrastructure innovations that make long conversations practical at scale. Important for those building production systems.

Memory Mechanisms Survey.. Zhang et al. (2024) provides a comprehensive survey of memory architectures for LLM agents, covering episodic, semantic, and working memory. This is the best current overview of how to implement various memory strategies discussed in this chapter.

6.3 Safety and Alignment

Constitutional AI.. Bai et al. (2022) introduced the paradigm of training models using explicit principles rather than human preferences alone. This paper explains how modern safety-focused models (like Claude) achieve consistent, explainable refusal behavior. Essential for understanding the alignment approaches behind commercial models.

Llama Guard.. Inan et al. (2023) describes a practical guardrail architecture for filtering harmful inputs and outputs. This paper provides a template for implementing the input/output classification approach to safety discussed in this chapter.

Guardrails Survey.. Dong, Zhao, et al. (2024) offers a comprehensive survey of guardrail architectures, covering prompt-based, classification-based, and hybrid approaches. Valuable for understanding the full landscape of safety options.

6.4 Few-Shot Learning and In-Context Learning

GPT-3 and Few-Shot Learning.. Brown et al. (2020) established that large language models are powerful few-shot learners, capable of generalizing from just a few examples provided in context. This foundational paper explains why in-context learning works and its limitations.

Self-Instruct.. Wang et al. (2023b) demonstrated how models can generate their own training data for instruction following. This paper is valuable for understanding bootstrapping techniques when human-labeled examples are scarce.

Reflexion.. Shinn et al. (2023) introduced verbal reinforcement learning through self-reflection, showing that agents can improve by critiquing their own performance. Important for understanding self-improvement loops in autonomous systems.

6.5 Legal and Financial AI

GPT-4 Passes the Bar Exam.. Katz et al. (2024) demonstrated that GPT-4 can pass the Uniform Bar Examination at a level competitive with human test-takers. This paper provides concrete evidence of LLM capabilities in legal reasoning and establishes benchmarks for legal AI performance.

ChatGPT Goes to Law School.. Choi et al. (2023) offered an early assessment of LLM capabilities in legal education contexts, identifying both promising capabilities and significant limitations. Valuable for understanding the state of legal AI as of 2023.

6.6 Implementation Resources

Prompt Engineering Guides.. OpenAI, Anthropic, and Google all publish prompt engineering guides that complement the academic literature:

- **OpenAI Cookbook**: Practical examples for API usage and prompting patterns
- **Anthropic Documentation**: Guidelines for Constitutional AI principles and Claude-specific techniques
- **Google AI**: Resources on PaLM/Gemini prompting and Vertex AI integration

Open-Source Frameworks.. Several frameworks implement the patterns discussed in this chapter:

- **LangChain**: Comprehensive framework for chaining LLM calls, implementing memory, and integrating tools
- **LlamaIndex**: Focused on document indexing, retrieval, and RAG patterns
- **DSPy**: Programmatic approach to prompt optimization and chain-of-thought compilation
- **Instructor**: Type-safe structured output extraction from LLMs

6.7 Staying Current

The field of LLM reasoning and conversation design evolves rapidly. Key venues for tracking developments:

- **arXiv cs.CL and cs.AI**: Pre-prints appear here first, often months before peer review
- **ACL, EMNLP, NAACL**: Premier NLP conferences with rigorous peer review
- **NeurIPS, ICML, ICLR**: Machine learning conferences with significant LLM content
- **Research Blogs**: Anthropic, OpenAI, Google DeepMind, and Meta AI publish technical updates

For legal and financial AI specifically:

- **Journal of Legal Education**: Academic perspectives on AI in legal practice
- **Legal Tech News**: Industry coverage of AI adoption in law
- **RegTech/FinTech publications**: Coverage of AI in financial compliance
- **OECD and FSB reports**: Policy perspectives on AI in finance

6.8 Recommended Reading Sequence

For readers who want to develop deep expertise in conversational AI and reasoning, we recommend the following structured approach:

Foundation Level (Week 1–2).. Begin with the foundational papers that established the field:

1. Start with Brown et al. (2020) to understand in-context learning capabilities
2. Read Wei et al. (2022) to grasp why step-by-step reasoning helps
3. Study Liu et al. (2024) to understand context limitations

This foundation equips you to understand why basic prompting techniques work and where they fail.

Intermediate Level (Week 3–4).. Progress to reliability and tool integration:

1. Wang et al. (2023a) for ensemble verification techniques
2. Yao et al. (2023b) for grounding reasoning in external observations
3. Bai et al. (2022) for understanding safety alignment

This level prepares you to build robust systems that verify their outputs and interact with the world.

Advanced Level (Week 5–6).. Explore sophisticated reasoning architectures:

1. Yao et al. (2023a) for deliberate search over reasoning paths
2. Besta et al. (2024) for graph-based reasoning composition

3. Shinn et al. (2023) for self-improving agent loops

This advanced material enables design of complex systems for planning, exploration, and autonomous improvement.

Domain Specialization (Ongoing).. For legal or financial applications, complement the technical literature with:

1. Domain-specific evaluation papers like Katz et al. (2024)
2. Regulatory guidance on AI in your specific sector
3. Professional ethics guidelines for AI use in your field

6.9 Common Misconceptions

Before concluding this chapter, it is worth addressing several common misconceptions that the literature does not always clearly dispel:

Misconception: More reasoning steps always helps.. Chain-of-thought improves performance on tasks that require multi-step logic, but not on all tasks. For simple factual questions or pattern matching, adding reasoning can actually hurt performance by introducing opportunities for the model to "overthink" and reach incorrect conclusions. Always evaluate whether reasoning is helping for your specific task.

Misconception: Longer context windows solve memory problems.. While extended context windows (100K+ tokens) enable longer conversations, they do not solve the fundamental "lost in the middle" problem. Information in the middle of very long contexts remains harder to access than information at the boundaries. Context length expansion should complement, not replace, active memory management strategies.

Misconception: Fine-tuning is always better than prompting.. Fine-tuning can improve performance on specific tasks, but it requires data, expertise, and compute. For many applications, few-shot prompting with well-selected examples approaches fine-tuned performance at a fraction of the cost. Prompting is also more flexible—you can adapt behavior without retraining the model.

Misconception: Safety guardrails eliminate risk.. Guardrails reduce risk but do not eliminate it. Adversarial users can sometimes bypass guardrails through creative prompting. Safety must be implemented in depth—prompt-level controls, input/output filtering, and downstream validation all play roles. Never rely on a single layer of protection for high-stakes applications.

Misconception: Reasoning traces reveal ground truth.. When a model produces a chain-of-thought, it appears to show "how it's thinking." But these traces may be post-hoc rationalizations

rather than accurate reflections of the underlying computation. Models can produce confident-looking reasoning for incorrect conclusions. Always verify final answers independently of how convincing the reasoning appears.

6.10 Exercises for Practitioners

To solidify understanding of the concepts in this chapter, we recommend the following exercises:

Exercise 1: Compare Reasoning Strategies.. Select a multi-step problem from your domain (e.g., a legal issue or financial calculation). Solve it using:

1. Direct prompting (zero-shot)
2. Chain-of-thought prompting
3. Self-consistency with 5 samples

Compare the quality of answers and the cost/latency trade-offs. Document which approach works best for your specific problem type.

Exercise 2: Build a Few-Shot Library.. Create a library of 10–20 high-quality examples for a task you frequently perform. Include:

1. Diverse problem types within your domain
2. Complete reasoning traces in your preferred format (e.g., IRAC for legal)
3. Edge cases that might confuse the model

Test how few-shot examples from your library affect performance compared to zero-shot prompting.

Exercise 3: Context Window Management.. Design a prompt structure for a multi-turn conversation in your domain. Specify:

1. What goes in the system prompt (and what doesn't)
2. How you will summarize older conversation turns
3. What facts you will pin in persistent context
4. How you will prioritize context when approaching the window limit

Test your design with a 20+ turn conversation and evaluate whether the model maintains coherence.

Exercise 4: Implement a ReAct Loop.. Build a simple ReAct-style system for a research task:

1. Define 2–3 tools (e.g., search, lookup, calculate)

2. Implement the thought-action-observation loop
3. Test with queries that require tool use to answer correctly

Evaluate how tool integration affects accuracy compared to relying on model knowledge alone.

Exercise 5: Safety Prompt Design.. Draft a comprehensive system prompt for a professional application. Include:

1. Role and scope definitions
2. Explicit prohibitions for your domain
3. Uncertainty acknowledgment instructions
4. Refusal patterns for common problematic requests

Test with adversarial prompts to evaluate robustness. Iterate on weak points you discover.

Conclusion

The transition from single-turn prompts to robust conversational agents requires a holistic engineering approach that treats the LLM not as a magic box, but as a component in a carefully designed stateful system. Throughout this chapter, we have examined the fundamental mechanics that make this possible.

What We Established

We demonstrated that maintaining conversation coherence requires active state management strategies—including role-based prompting, recursive summarization, and vector-enhanced retrieval—to combat context window limitations and attention biases like the "Lost in the Middle" effect. The system prompt serves as an immutable constitution, but its efficacy degrades with conversational length unless reinforced through strategic instruction placement.

We showed that reliability in complex tasks is a function of the reasoning topology employed. While Chain-of-Thought provides a necessary baseline for logical decomposition, advanced patterns like Self-Consistency offer scalable mechanisms to trade inference compute for accuracy. ReAct bridges internal reasoning with external grounding, essential for fact-intensive domains like law and finance. Tree and Graph of Thoughts extend exploration capabilities for problems requiring search and aggregation.

We introduced few-shot learning as both a teaching mechanism and a retrieval problem, with techniques like K-NN selection and Maximal Marginal Relevance balancing relevance and diversity. Bootstrapping methods enable the rapid creation of exemplar libraries, though human review remains

essential for high-stakes domains.

We established a decision framework for strategy selection based on accuracy requirements, latency constraints, cost sensitivity, and explainability needs. The right answer is always the simplest approach that meets requirements—over-engineering introduces complexity without commensurate benefit.

The Road Ahead

Ultimately, the successful deployment of conversational AI in professional settings rests on three pillars:

1. **The "Constitution"**: The system prompts and guardrails that define the agent's behavioral boundaries, implemented through defense-in-depth with multiple layers of protection.
2. **Strategic Pattern Selection**: Matching reasoning topologies to the specific utility function of the application—accuracy, latency, cost, and explainability trade-offs calibrated to the domain.
3. **Continuous Governance**: Audit trails, version control, and human oversight mechanisms that ensure accountability as these systems operate in high-stakes environments.

As models continue to scale and capabilities expand, the differentiation will shift from the raw capability of the underlying model to the sophistication of the cognitive architecture surrounding it. The techniques in this chapter—state management, structured reasoning, and strategic selection—form the foundation of that architecture.

What Comes Next

The next chapter extends these foundations into the realm of *structured outputs* and *tool use*. We will examine:

- How to constrain model outputs to conform to JSON schemas, legal citation formats, and other structured templates
- The mechanics of function calling and API integration that enable ReAct-style tool use
- The full Retrieval-Augmented Generation (RAG) architecture for grounding models in document knowledge
- Multimodal inputs that extend conversations beyond text

With the conversational and reasoning foundations now in place, you are prepared to build systems that not only converse and reason but also produce machine-readable outputs and take actions in the world. The journey from pattern-matching text predictor to capable AI assistant is well underway.

References

Bai, Yuntao, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. (2022). "Constitutional AI: Harmlessness from AI Feedback". In: *arXiv preprint arXiv:2212.08073*. Introduces constitutional AI for alignment via explicit principles. URL: https://arxiv.org/abs/2212.08073 (visited on 12/20/2025).

Besta, Maciej, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Nyczyk, Piotr Iff, et al. (2024). "Graph of Thoughts: Solving Elaborate Problems with Large Language Models". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.16. Introduces graph-based reasoning with aggregation and refinement operations, pp. 17682–17690. URL: https://arxiv.org/abs/2308.09687 (visited on 12/20/2025).

Besta, Maciej, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Nils Blach, Piotr Iff, Joanna Gajda, Hubert Nyczyk, Ales Kubicek, Lukas Gianinazzi, et al. (2025). "Demystifying Chains, Trees, and Graphs of Thoughts". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 47.12. Comprehensive taxonomy of reasoning topologies in LLMs, pp. 10967–10989. URL: https://arxiv.org/abs/2401.14295 (visited on 12/20/2025).

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. GPT-3 paper demonstrating in-context learning capabilities, pp. 1877–1901. URL: https://arxiv.org/abs/2005.14165 (visited on 12/20/2025).

Choi, Jonathan H., Kristin E. Hickman, Amy Monahan, and Daniel Schwarcz (2023). "ChatGPT Goes to Law School". In: *Journal of Legal Education* 71.3. Early assessment of LLM capabilities in legal education, pp. 387–400.

Dong, Yi, Ronghui Zhao, et al. (2024). "Building Guardrails for Large Language Models". In: *arXiv preprint arXiv:2402.01822*. Survey of guardrail architectures for LLM safety. URL: https://arxiv.org/abs/2402.01822 (visited on 12/20/2025).

Inan, Hakan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. (2023). "Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations". In: *arXiv preprint arXiv:2312.06674*. Classification model for guardrailing LLM inputs and outputs. URL: https://arxiv.org/abs/2312.06674 (visited on 12/20/2025).

Katz, Daniel Martin, Michael J. Bommarito, Shang Gao, and Pablo Arredondo (2024). "GPT-4 Passes the Bar Exam". In: *Philosophical Transactions of the Royal Society A*. Vol. 382. 2270. Demonstrates LLM performance on legal reasoning tasks. URL: https://royalsocietypublishing.org/doi/10.1098/rsta.2023.0254 (visited on 12/20/2025).

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica (2023). "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the 29th Symposium on Operating Systems Principles*. Introduces PagedAttention for efficient KV-cache management in inference, pp. 611–626. URL: https://arxiv.org/abs/2309.06180 (visited on 12/20/2025).

Li, Bo and Chen Zhao (2025). "Self-reflection enhances large language models towards substantial academic response". In: *npj Artificial Intelligence* 1.42. Empirical study of self-reflection improving LLM response quality. URL: https://www.nature.com/articles/s44387-025-00045-3 (visited on 12/20/2025).

Liu, Nelson F., Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang (2024). "Lost in the Middle: How Language Models Use Long Contexts". In: *Transactions of the Association for Computational Linguistics* 12. Demonstrates U-shaped retrieval accuracy in long contexts, pp. 157–173. URL: https://aclanthology.org/2024.tacl-1.9/ (visited on 12/20/2025).

Shinn, Noah, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao (2023). "Reflexion: Language Agents with Verbal Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 36. Introduces self-reflection for iterative improvement in LLM agents. URL: https://arxiv.org/abs/2303.11366 (visited on 12/20/2025).

Touvron, Hugo, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. (2023). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. Technical report on Llama 2 architecture including chat formatting. URL: https://arxiv.org/abs/2307.09288 (visited on 12/20/2025).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Original Transformer architecture paper. URL: https://arxiv.org/abs/1706.03762 (visited on 12/20/2025).

Wang, Qingyue, Yun Fu, Yang Cao, et al. (2024). "Recursively Summarizing Enables Long-Term Dialogue Memory in Large Language Models". In: *Neurocomputing*. Demonstrates recursive summarization for maintaining long conversation context. URL: https://arxiv.org/abs/2308.15022 (visited on 12/20/2025).

Wang, Xuezhi, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou (2023a). "Self-Consistency Improves Chain of Thought Reasoning in Language Models". In: *International Conference on Learning Representations*. Introduces self-consistency decoding with majority voting to improve reasoning accuracy. URL: https://arxiv.org/abs/2203.11171 (visited on 12/20/2025).

Wang, Yizhong, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi (2023b). "Self-Instruct: Aligning Language Models with Self-Generated Instructions". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguis-*

tics. Method for bootstrapping instruction-following data from the model itself, pp. 13484–13508. URL: https://arxiv.org/abs/2212.10560 (visited on 12/20/2025).

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou (2022). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 35. Foundational paper introducing chain-of-thought prompting for improved reasoning in LLMs, pp. 24824–24837. URL: https://arxiv.org/abs/2201.11903 (visited on 12/20/2025).

Xu, Yilun et al. (2025a). "Chain of Draft: Thinking Faster by Writing Less". In: *arXiv preprint arXiv:2502.18600*. Concise reasoning traces for faster inference. URL: https://arxiv.org/abs/2502.18600 (visited on 12/20/2025).

Xu, Yuxuan et al. (2025b). "TokenSkip: Controllable Chain-of-Thought Compression in LLMs". In: *Proceedings of EMNLP 2025*. Method for reducing chain-of-thought token overhead. URL: https://aclanthology.org/2025.emnlp-main.165/ (visited on 12/20/2025).

Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas Griffiths, Yuan Cao, and Karthik Narasimhan (2023a). "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 36. Extends chain-of-thought to tree-based exploration with backtracking. URL: https://arxiv.org/abs/2305.10601 (visited on 12/20/2025).

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao (2023b). "ReAct: Synergizing Reasoning and Acting in Language Models". In: *International Conference on Learning Representations*. Introduces the ReAct framework combining reasoning traces with tool use. URL: https://arxiv.org/abs/2210.03629 (visited on 12/20/2025).

Zebaze Dongmo, Arsene Romain, Benoit Sagot, and Rachel Bawden (2024). "In-Context Example Selection via Similarity Search Improves Low-Resource Machine Translation". In: *arXiv preprint arXiv:2408.00397*. Demonstrates semantic search for few-shot example selection. URL: https://arxiv.org/abs/2408.00397 (visited on 12/20/2025).

Zhang, Zeyu et al. (2024). "From Human Memory to AI Memory: A Survey on Memory Mechanisms in the Era of LLMs". In: *arXiv preprint arXiv:2504.15965*. Comprehensive survey of memory mechanisms for LLM agents. URL: https://arxiv.org/abs/2504.15965 (visited on 12/20/2025).

Zheng, Ming, Jiahui Pei, Lajanugen Logeswaran, Moontae Lee, and David Jurgens (2023). "When "A Helpful Assistant" Is Not Really Helpful: Personas in System Prompts Do Not Improve Performances of Large Language Models". In: *arXiv preprint arXiv:2311.10054*. Empirical study of system prompt persona effects on LLM performance. URL: https://arxiv.org/abs/2311.10054 (visited on 12/20/2025).