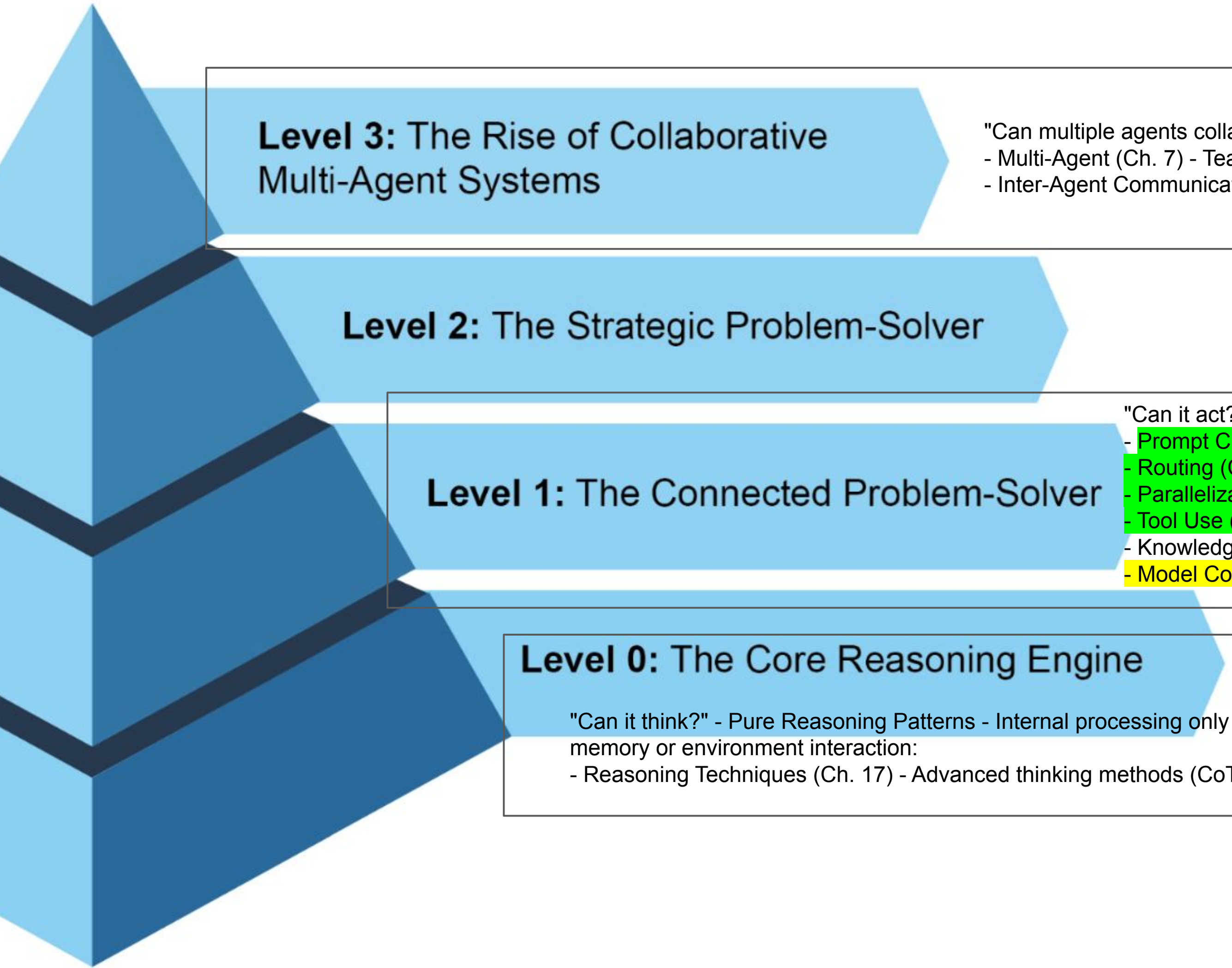
Chapters 8-12

Memory Management, Learning and Adaptation, MCP, Goal Setting and Monitoring, Exception Handling and Recovery

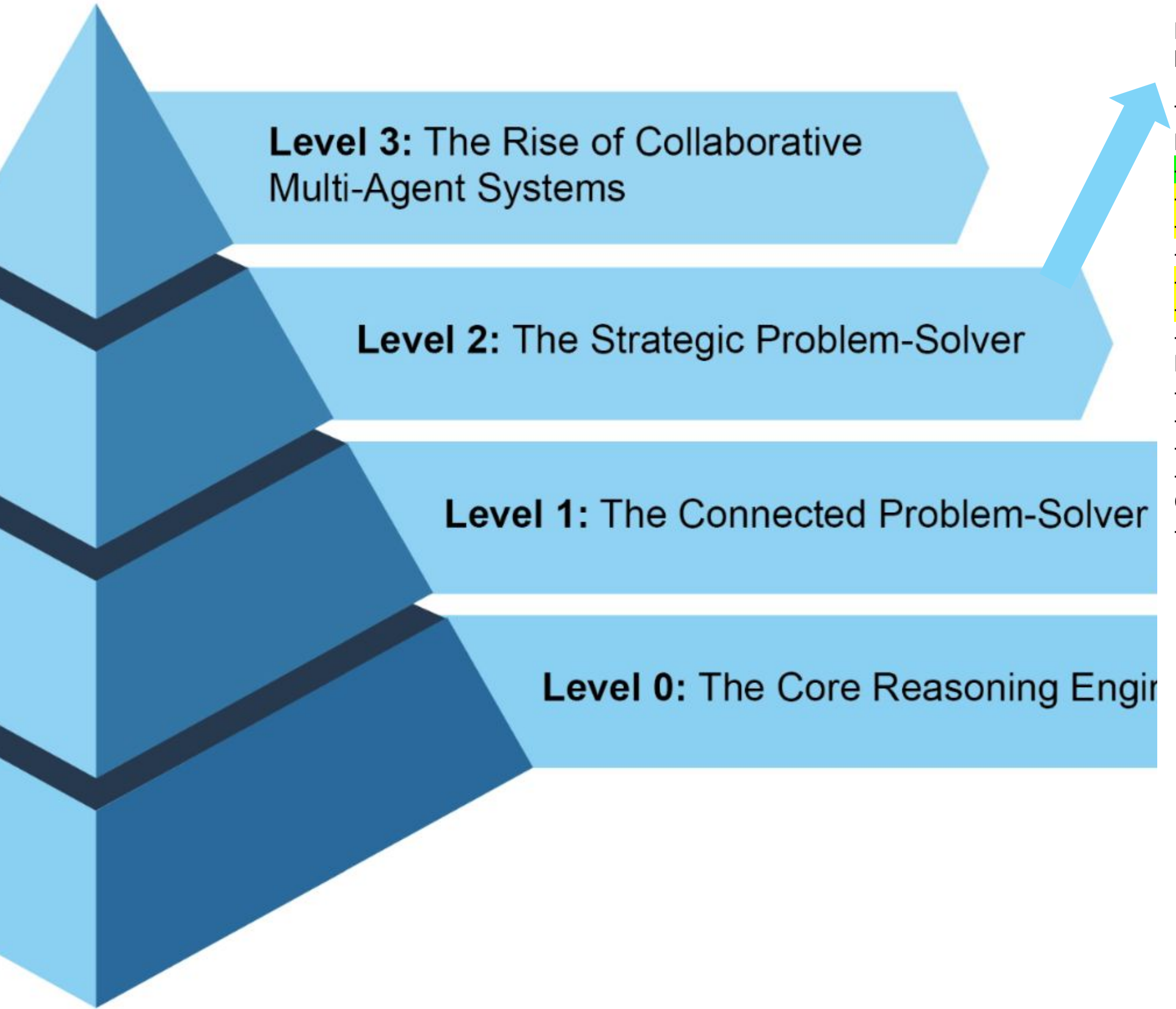


Level 3: The Rise of Collaborative Multi-Agent Systems
Level 2: The Strategic Problem-Solver
Level 1: The Connected Problem-Solver
Level 0: The Core Reasoning Engine

"Can multiple agents collaborate as an organization?"
- Multi-Agent (Ch. 7) - Team orchestration & specialization
- Inter-Agent Communication (A2A) (Ch. 15)

"Can it act?" - Connection & Interaction Patterns:
- Prompt Chaining (Ch. 1) - sequential task execution
- Routing (Ch. 2) - Conditional path selection
- Parallelization (Ch. 3) - Concurrent execution
- Tool Use (Ch. 5) - External API/service integration
- Knowledge Retrieval (RAG) (Ch. 14)
- Model Context Protocol (MCP) (Ch. 10)

"Can it think?" - Pure Reasoning Patterns - Internal processing only without tools, memory or environment interaction:
- Reasoning Techniques (Ch. 17) - Advanced thinking methods (CoT, ToT, etc)


Level 3: The Rise of Collaborative Multi-Agent Systems
Level 2: The Strategic Problem-Solver
Level 1: The Connected Problem-Solver
Level 0: The Core Reasoning Engi

Level 2: The Strategic Problem-Solver - "Can it strategize, learn, and operate reliably?"

This section is subdivided for human readability into:

Meta Cognition Patterns:
- Reflection (Ch. 4) - Self-evaluation capabilities
- Planning (Ch. 6) - Multi-step strategy formulation
- Goal Setting & Monitoring (Ch. 11) - Objective tracking
- Memory Management (Ch. 8)
- Learning & Adaptation (Ch. 9) - self-improvement loops
- Exception Handling & Recovery (Ch. 12) - Error resilience
- Prioritization (Ch. 20) - Action selection & ranking

Production readiness (safety and reliability):
- Resource-Aware Optimization (Ch. 16)
- Human-in-the-Loop (Ch. 13) - human oversight & intervention
- Guardrails/Safety Patterns (Ch. 18) - Behavioral constraints
- Evaluation & Monitoring (Ch. 19) - Performance tracking

Collaboration and exploration:
- Exploration & Discovery (Ch. 21) - novel solution finding

Chapter 10 - MCP

What is a difference between MCP and tool calling?

What are complexities and pitfalls related to using MCP?

How MCP approach is influencing software development?

Model Context Protocol is dynamically discovered tools, data and prompts

Client-server model: LLM apps (clients) discover/use capabilities from MCP servers with dynamic discovery: Clients query servers to learn available capabilities at runtime: Resources (data), Tools (functions), Prompts (templates)

MCP vs. Tool Function Calling:

- Function calling: Direct 1:1 LLM-to-tool, tightly coupled
- MCP: Universal adapter enabling any LLM to leverage server's tools

Transport & Security:

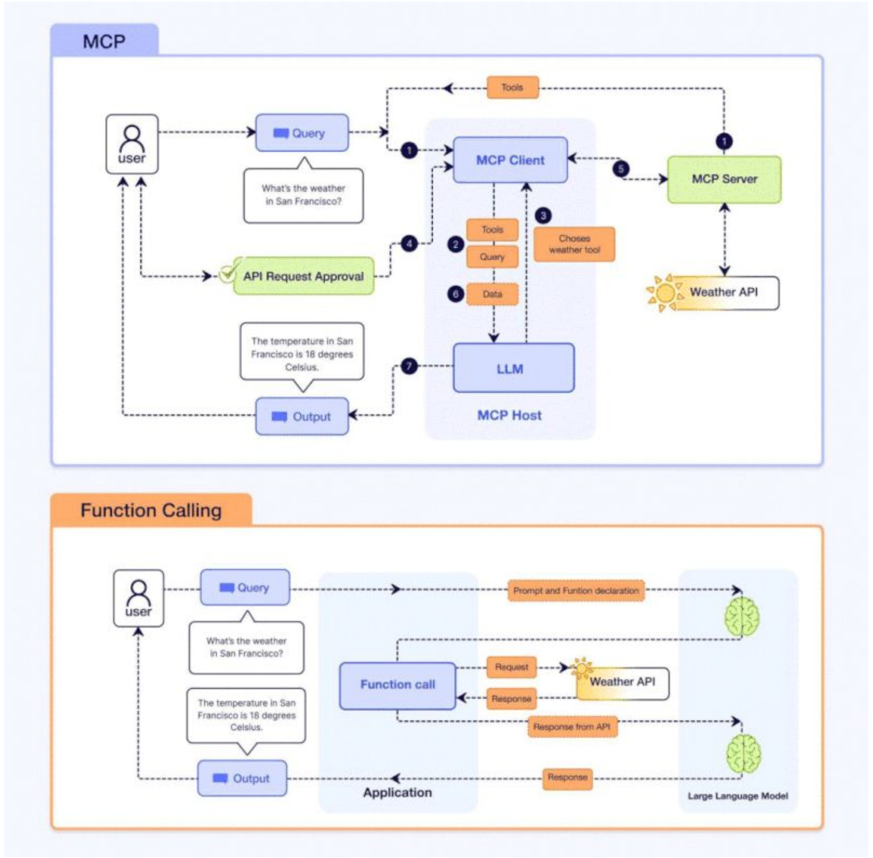
- Local: JSON-RPC over STDIO for inter-process communication
- Remote: Streamable HTTP and Server-Sent Events (SSE)
- Requires robust authentication/authorization controls

Key Considerations:

- Underlying APIs need filtering/sorting for efficient agent use
- Data format must be agent-friendly (e.g., Markdown not raw PDFs)
- Local vs remote deployment, on-demand vs batch processing
- Error handling and fallback are critical

FastMCP:

- High-level Python framework simplifying MCP server development
- Automatic schema generation from type hints and docstrings
- Supports server composition and proxying for complex systems



MCP was donated to the Linux Foundation's Agentic AI Foundation in December 2025 Modelcontextprotocol and now spans **10,000+ active servers** with **97 million monthly SDK downloads**.

MCP is like API but for AI, also moves from local to cloud

Now:

(production, managed) Composio: 500+ managed MCP servers with built-in OAuth, SOC 2 Type 2 compliant handling across services like Gmail, GitHub, Slack, and Notion with a single MCP endpoint that dynamically routes agents to the right integration at runtime.

(community-based) Smithery - de facto registry and hosting platform for MCP servers - "npm for MCP." - 2,000+ servers_with semantic search, managed hosting for remote servers, built-in OAuth.

(low-level browser automation framework used for testing and web interaction, which can be wrapped in an MCP server to allow AI to "see" and interact with websites) Microsoft Playwright MCP: exposes browser automation via accessibility tree (not screenshots), giving LLMs structured, deterministic web interaction without vision models - enabling any AI client to control a full browser.

(remote, cloud platform) Cloudflare Workers MCP enables deploying remote MCP servers on Cloudflare's global edge network - internet-accessible, authenticated, auto-scaling production endpoints. mcp-remote adapter solves a critical ecosystem gap: allowing any MCP client that only supports local connections to work with remote servers.

Future:

autonomously discovering tools, so providers of APIs and SDKs will need to make sure their tooling is easily discoverable from search and be differentiated enough for the agent to pick for a particular task.

mapping from API to tools is rarely 1:1 as tools are a higher abstraction that makes the most sense for agents at the time of task execution — instead of simply calling send_email(), an agent may opt for draft_email_and_send() function that includes multiple API calls to minimize latency. MCP server design will be scenario- and use-case-centric instead of API-centric.

new pricing model may emerge if every app becomes a MCP client and every API becomes a MCP server - could lead to market-driven tool-adoption process that picks the best-performing and the most modular tool instead of the most widely adopted one.

documentation will become a critical piece of MCP infrastructure as companies will need to design tools and APIs with clear, machine-readable formats (e.g., llms.txt) and make MCP servers a de facto artifact based on existing documentation.

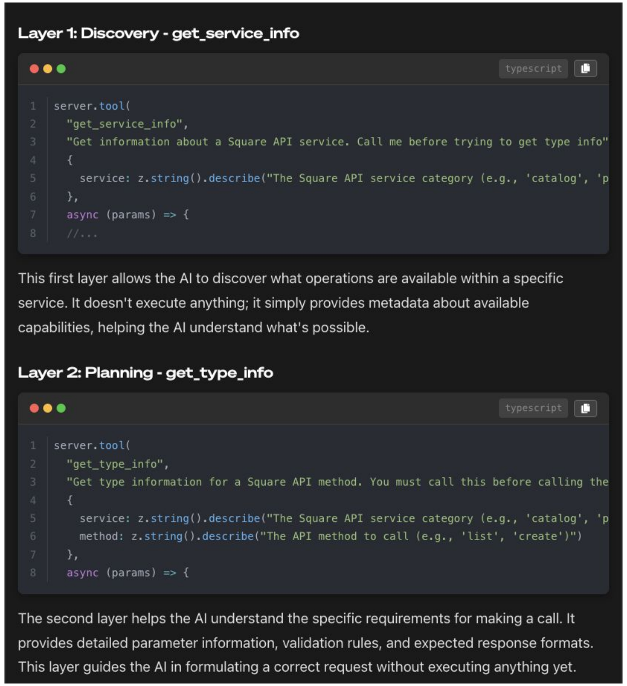
https://a16z.com/a-deep-dive-into-mcp-and-the-future-of-ai-tooling/

Block: 120+ internal MCP servers deployed company-wide in 8 weeks with Goose - open-source MCP-native agent framework

Block deployed 120+ internal MCP servers covering every internal application: Snowflake, Jira, Slack, Google Drive, and dozens more.Aviator,Block.

Notable ideas:

- "Layered Tool Pattern" (Discovery → Planning → Execution) to manage 30+ APIs and 200+ endpoints without overwhelming the LLM's context window.
- Every server passes dual review: MCP engineering review plus security team review. Employees toggle MCP servers on and off via preconfigured Goose - no JSON configs, API keys, or scopes required.
- Cloudflare: the infrastructure layer for remote MCP servers
- Built by the relevant product team around specific "jobs to be done" rather than wrapping entire APIs.



Layer 1: Discovery - get_service_info

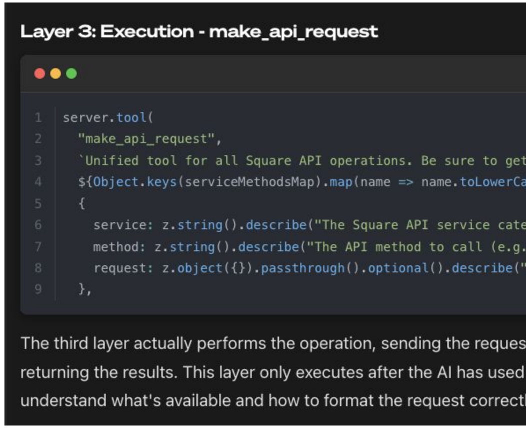
```typescript
server.tool(
  "get_service_info",
  "Get information about a Square API service. Call me before trying to get type info"
  {
    service: z.string().describe("The Square API service category (e.g., 'catalog', 'p
  },
  async (params) => {
  //...
```

This first layer allows the AI to discover what operations are available within a specific service. It doesn't execute anything; it simply provides metadata about available capabilities, helping the AI understand what's possible.

Layer 2: Planning - get_type_info

```typescript
server.tool(
  "get_type_info",
  "Get type information for a Square API method. You must call this before calling the
  {
    service: z.string().describe("The Square API service category (e.g., 'catalog', 'p
    method: z.string().describe("The API method to call (e.g., 'list', 'create')")
  },
  async (params) => {
```

The second layer helps the AI understand the specific requirements for making a call. It provides detailed parameter information, validation rules, and expected response formats. This layer guides the AI in formulating a correct request without executing anything yet.



Layer 3: Execution - make_api_request

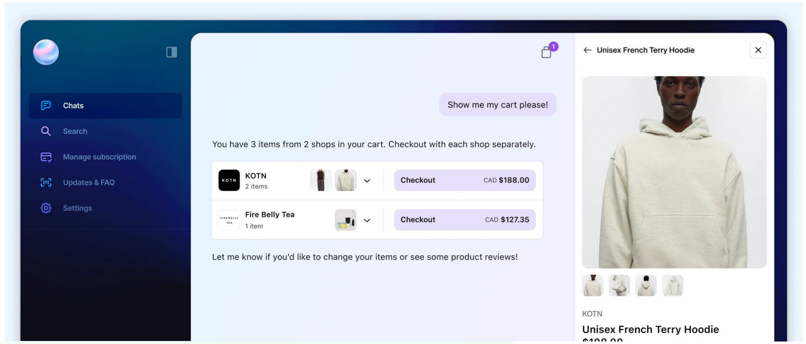
```
server.tool(
  "make_api_request",
  `Unified tool for all Square API operations. Be sure to get
  ${Object.keys(serviceMethodsMap).map(name => name.toLowerCa
  {
    service: z.string().describe("The Square API service cate
    method: z.string().describe("The API method to call (e.g.
    request: z.object({}).passthrough().optional().describe("
  },
```

The third layer actually performs the operation, sending the reques returning the results. This layer only executes after the AI has used understand what's available and how to format the request correct

(allegedly) Stripe & Shopify prepare ~~for Google introducing purchases in search results and~~ agent-driven shopping

Agentic commerce stack is already forming around MCP:

- Stripe now features its MCP server on its homepage alongside SDKs and APIs, Stripe exposing the full payments API plus knowledge base search. Stripe Order Intents — an API designed specifically for autonomous agents to complete purchases end-to-end.
- Shopify went further, building both a developer MCP server and a storefront MCP server (Agent One, Shopify Engineering), contributing MCP UI as an open-source protocol extension enabling servers to return interactive components (product cards, size selectors, add-to-cart flows) rather than just text.



Show me my cart please!

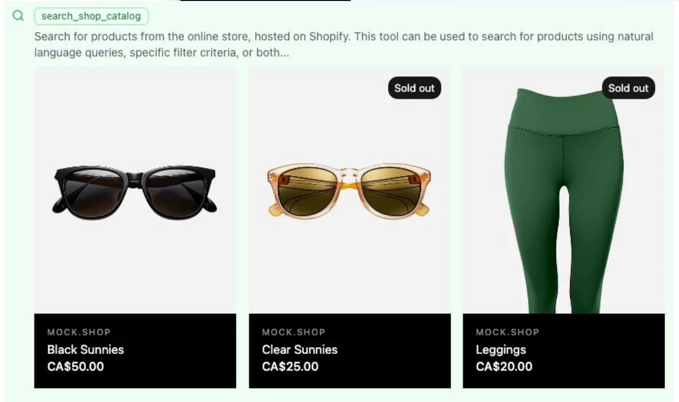
You have 3 items from 2 shops in your cart. Checkout with each shop separately.

KOTN 2 items	Checkout	CAD $188.00
Fire Belly Tea 1 item	Checkout	CAD $127.35

Let me know if you'd like to change your items or see some product reviews!

Unisex French Terry Hoodie

KOTN
Unisex French Terry Hoodie

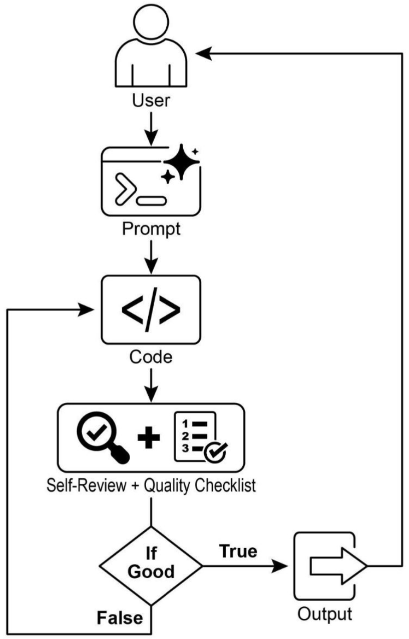

search_shop_catalog

Search for products from the online store, hosted on Shopify. This tool can be used to search for products using natural language queries, specific filter criteria, or both...

	Sold out	Sold out
MOCK.SHOP Black Sunnies CA$50.00	MOCK.SHOP Clear Sunnies CA$25.00	MOCK.SHOP Leggings CA$20.00

Chapter 11 - Goal setting ~~and monitoring~~

How do you approach goal setting for an agent?



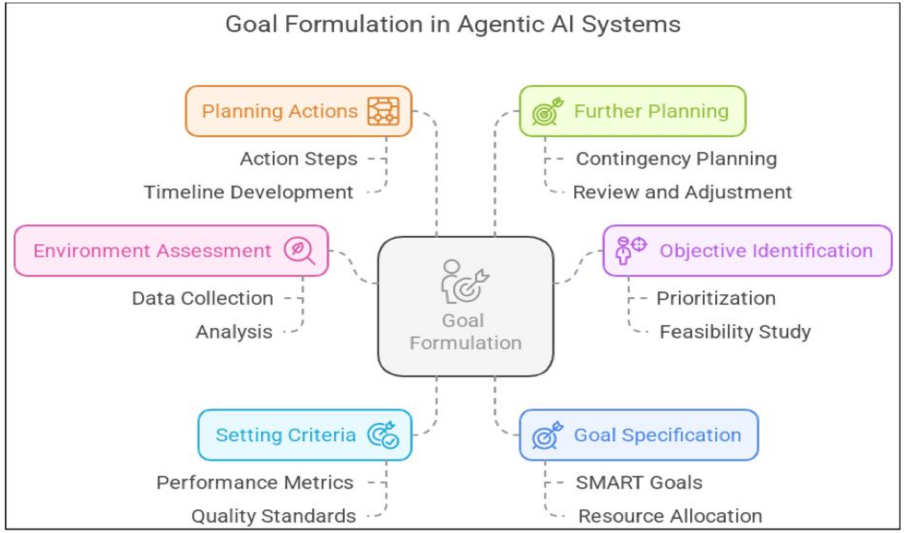
Goal setting transforms agents from reactive systems into proactive, goal-driven entities capable of autonomous, reliable operation

Agents need both a clear objective and a feedback mechanism to track progress, assess performance, and adapt when deviating from success criteria.

SMART Goals Framework: Goals should be Specific, Measurable, Achievable, Relevant, and Time-bound.

Monitoring Components: Observing agent actions, environmental states, tool outputs, and establishing feedback loops that enable agents to revise plans or escalate issues.

Implementation: In ADK, goals are conveyed through agent instructions, with monitoring accomplished through state management and tool interactions.



Goals codification beats goals description - goals are reflected in system architecture 1/2

Salesforce initially used prescriptive guardrails its own Agentforce for example, blocking any mention of competitors.

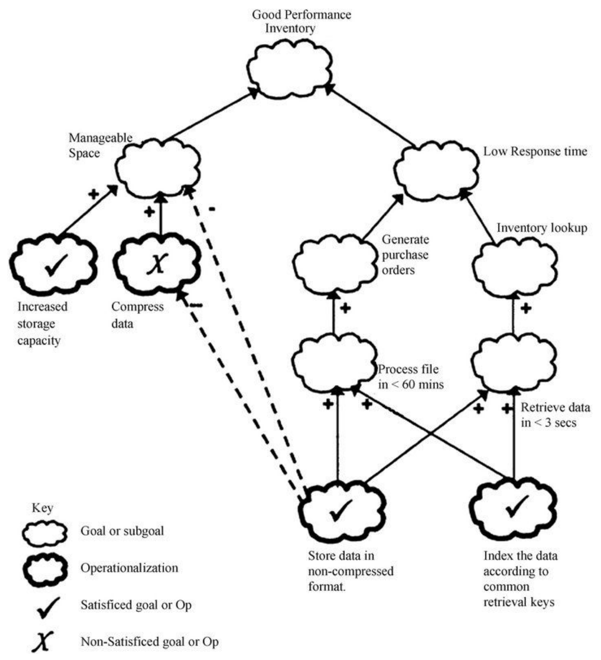
- The solution was to replace rigid rules with a goal statement — "act in Salesforce's and customers' best interest."
- a real-time deflection score that dynamically tracks sentiment and intent throughout conversations, adjusting escalation thresholds based on signals like frustration level and explicit requests for human help.

Vodafone two agents embeds business goals in the system architecture:

- Super TOBi handles complex customer interactions (billing, service activation, roaming, sales) using a Supervisor-plus-specialized-agents architecture,
- Super Agent augments human call center consultants with instant diagnostics and compliant guidance
- Business specialists author troubleshooting procedures using structured templates, which LLM agents parse into a Neo4j knowledge graph. Every agent recommendation is validated against Rule nodes encoded in Neo4j that enforce company policy

Capital One deployed Chat Concierge, a customer-facing multi-agent AI system on auto dealership websites: five specialized agents:

- a natural-language understanding agent,
- an action-planning agent,
- a policy-validation agent,
- an execution agent, Klover
- **dedicated evaluator agent trained on Capital One's regulations and policies that continuously monitors the other agents and can reject or "kick back" plans that violate guardrails**



Good Performance Inventory
Manageable Space
Low Response time
Increased storage capacity
Compress data
Generate purchase orders
Inventory lookup
Process file in < 60 mins
Retrieve data in < 3 secs
Store data in non-compressed format.
Index the data according to common retrieval keys
Key
Goal or subgoal
Operationalization
Satisficed goal or Op
Non-Satisficed goal or Op

Keywords: systems theory, NFR, KAOS (Keeping all Objectives Satisfied), GO

Goals codification beats goals description - goals are reflected in system architecture 2/2

BabyAGI pioneered the autonomous goal decomposition pattern:

- Task Creation agent decomposes a high-level objective into subtasks,
- Prioritization agent reorders them by dependency,
- Execution agent completes each task with vector-DB context feeding results back into the cycle.
-

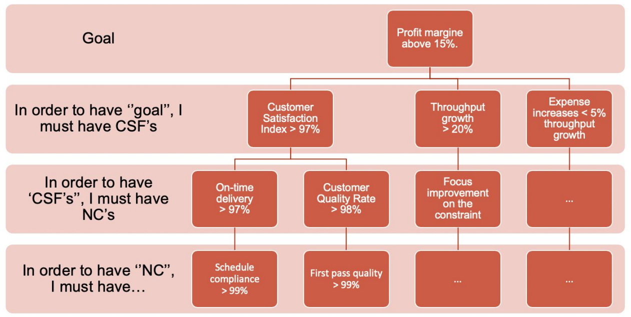
CrewAI - an agent is defined with explicit role, goal, and backstory parameters using a declarative goal specification pattern that goes well beyond SMART.:

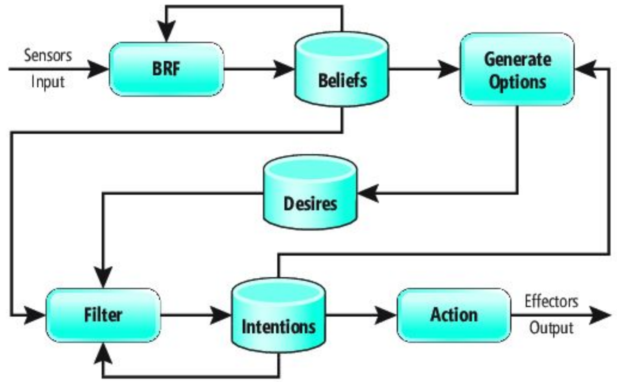
- Planning Agent generates step-by-step execution plans before any task begins
- reasoning=True flag enables agents to reflect on objectives and refine plans mid-execution.
- dual architectures: Crews for autonomous, goal-driven collaboration and Flows for event-driven stateful goal tracking with conditional branching.
- built-in memory layers (short-term, long-term, entity, contextual) enable goal-aware context retention across sessions

SPADE-BDI uses **Belief-Desire-Intention cognitive architecture**:

- Agents maintain explicit Beliefs (knowledge), Desires (goals), and Intentions (committed plans) as first-class objects, with plans written as AgentSpeak production rules triggered by goal/belief changes.

The Goal Tree


Goal
Profit margine above 15%.
In order to have "goal", I must have CSF's
Customer Satisfaction Index > 97%
Throughput growth > 20%
Expense increases < 5% throughput growth
In order to have 'CSF's", I must have NC's
On-time delivery > 97%
Customer Quality Rate > 98%
Focus improvement on the constraint
...
In order to have "NC", I must have...
Schedule compliance > 99%
First pass quality > 99%
...
...

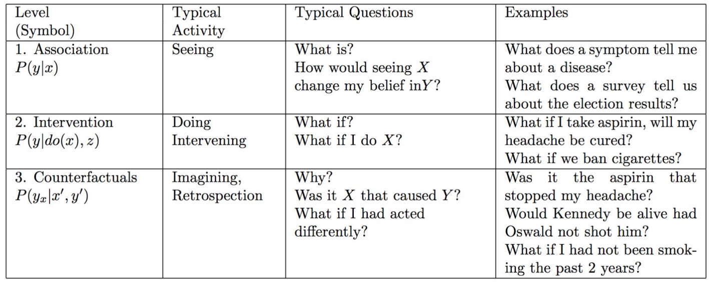


4D-ARE: Bridging the Attribution Gap in LLM Agent Requirements Engineering

When a regional manager asked "Why is Eastern region's completion rate only 80%?", the agent responded: "Eastern region's deposit completion rate is 80%. Visit frequency is 4.2 per week, which is lower than other regions. Product penetration is 24%. The team should work to improve these metrics."

The core insight: decision-makers seek attribution, not answers. Attribution concerns organize into four dimensions (Results → Process → Support → Long-term), motivated by Pearl's causal hierarchy.

The Three Layer Causal Hierarchy



Level (Symbol)	Typical Activity	Typical Questions	Examples
1. Association $P(y\|x)$	Seeing	What is? How would seeing X change my belief in Y?	What does a symptom tell me about a disease? What does a survey tell us about the election results?
2. Intervention $P(y\|do(x), z)$	Doing Intervening	What if? What if I do X?	What if I take aspirin, will my headache be cured? What if we ban cigarettes?
3. Counterfactuals $P(y_x\|x', y')$	Imagining, Retrospection	Why? Was it X that caused Y? What if I had acted differently?	Was it the aspirin that stopped my headache? Would Kennedy be alive had Oswald not shot him? What if I had not been smoking the past 2 years?

Figure 1: The Causal Hierarchy. Questions at level i can only be answered if information from level i or higher is available.

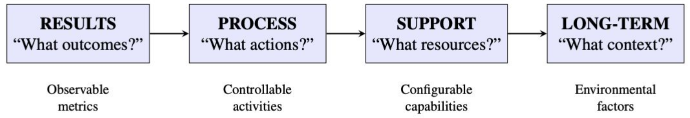
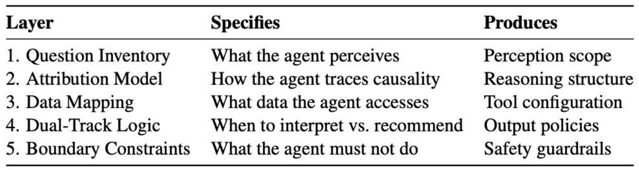


Figure 1: The Four Dimensions. When Results show a gap, attribution traces backward through Process, Support, and Long-term to identify causes.

Table 1: The Five-Layer Architecture

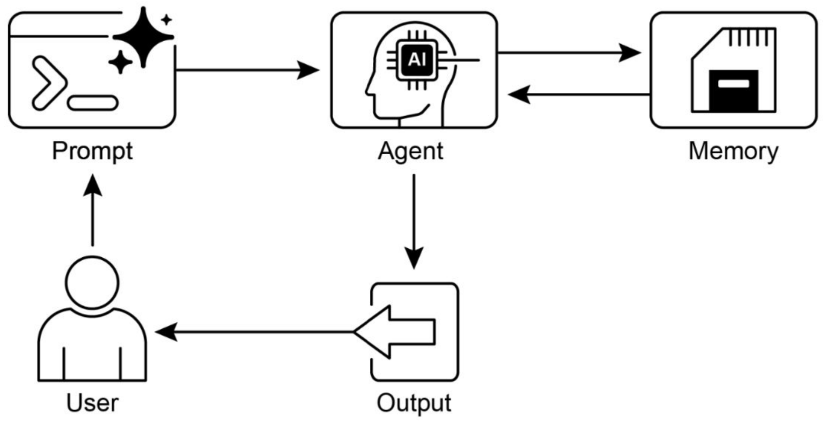


Layer	Specifies	Produces
1. Question Inventory	What the agent perceives	Perception scope
2. Attribution Model	How the agent traces causality	Reasoning structure
3. Data Mapping	What data the agent accesses	Tool configuration
4. Dual-Track Logic	When to interpret vs. recommend	Output policies
5. Boundary Constraints	What the agent must not do	Safety guardrails

Chapter 8 - Memory Management

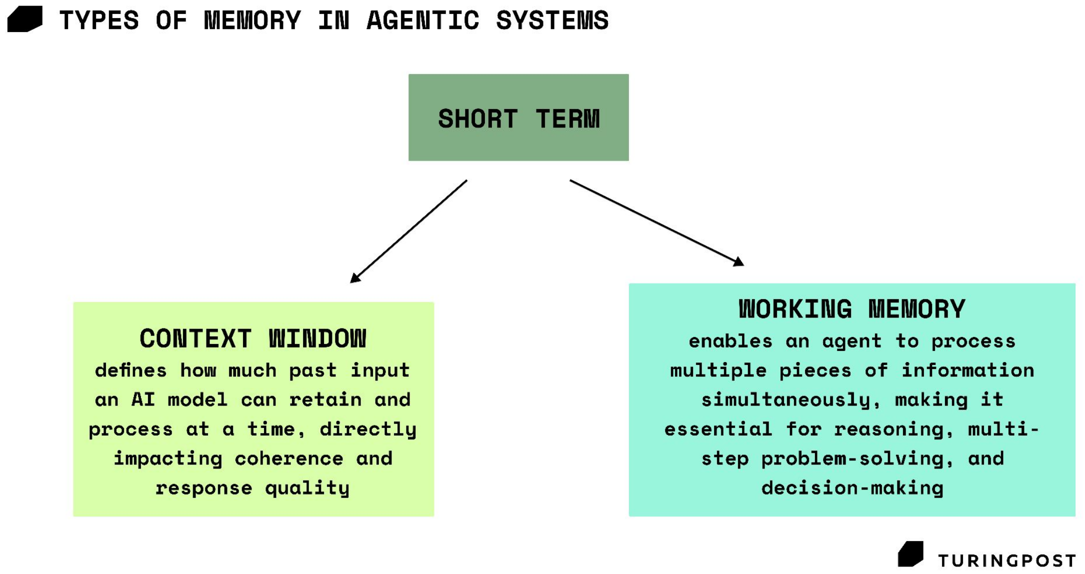
How different types of memory are working together?

How to evaluate different memory approaches?



Memory enables personalization, multi-step tasks, RAG, learning from past interactions - both short-term….

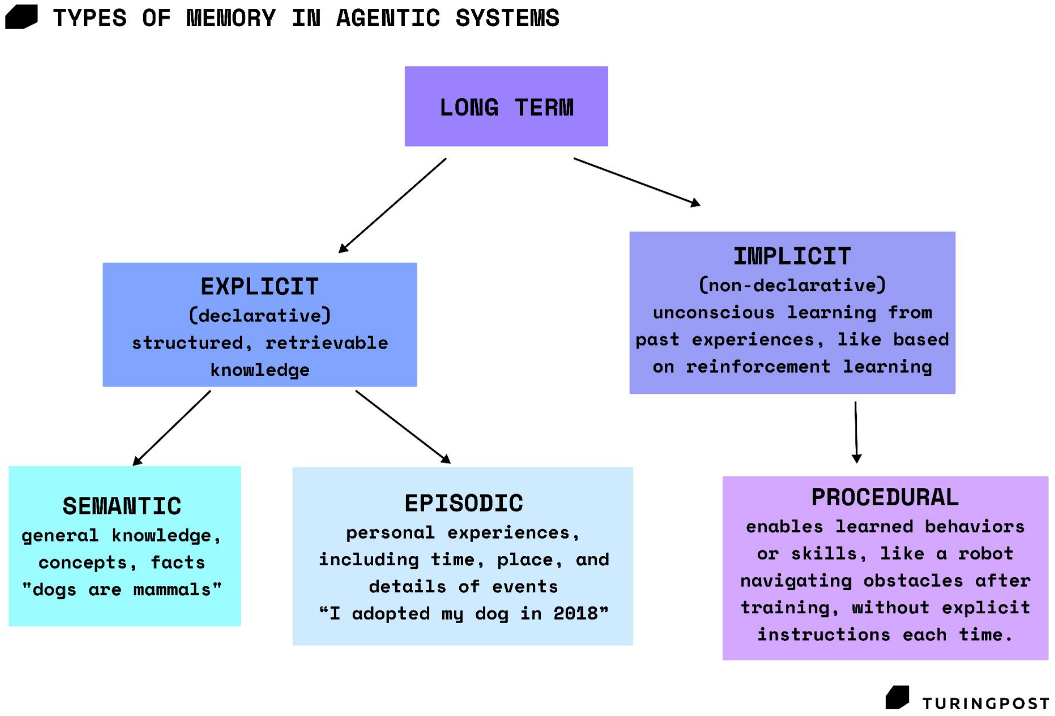
- Short-term holds recent messages, tool results, reflections - limited by context window and how working memory is organised


TYPES OF MEMORY IN AGENTIC SYSTEMS
SHORT TERM
CONTEXT WINDOW
defines how much past input an AI model can retain and process at a time, directly impacting coherence and response quality
WORKING MEMORY
enables an agent to process multiple pieces of information simultaneously, making it essential for reasoning, multi-step problem-solving, and decision-making
TURINGPOST

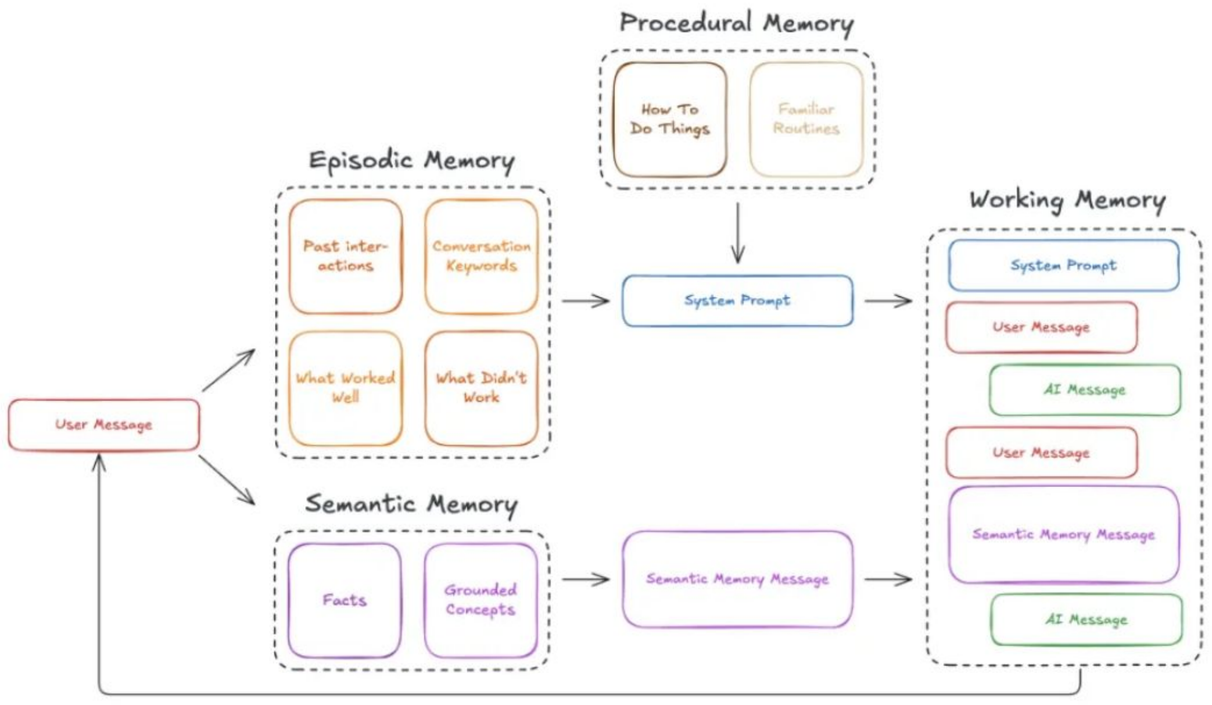
https://huggingface.co/blog/Kseniase/memory

… and long-term (persistent external storage)

- Long-term uses vector databases for semantic search across sessions (RAG) and could be one of three:
  a. Semantic (facts),
  b. Episodic (experiences),
  c. Procedural (rules)


TYPES OF MEMORY IN AGENTIC SYSTEMS
LONG TERM
EXPLICIT (declarative) structured, retrievable knowledge
IMPLICIT (non-declarative) unconscious learning from past experiences, like based on reinforcement learning
SEMANTIC general knowledge, concepts, facts "dogs are mammals"
EPISODIC personal experiences, including time, place, and details of events "I adopted my dog in 2018"
PROCEDURAL enables learned behaviors or skills, like a robot navigating obstacles after training, without explicit instructions each time.
TURINGPOST

Memory is one of the things what makes agents effective



Full Working Memory Demonstration

Generative Agents: Interactive Simulacra of Human Behavior: Memory organised as a stream, where agents continuously log their experiences in natural language.

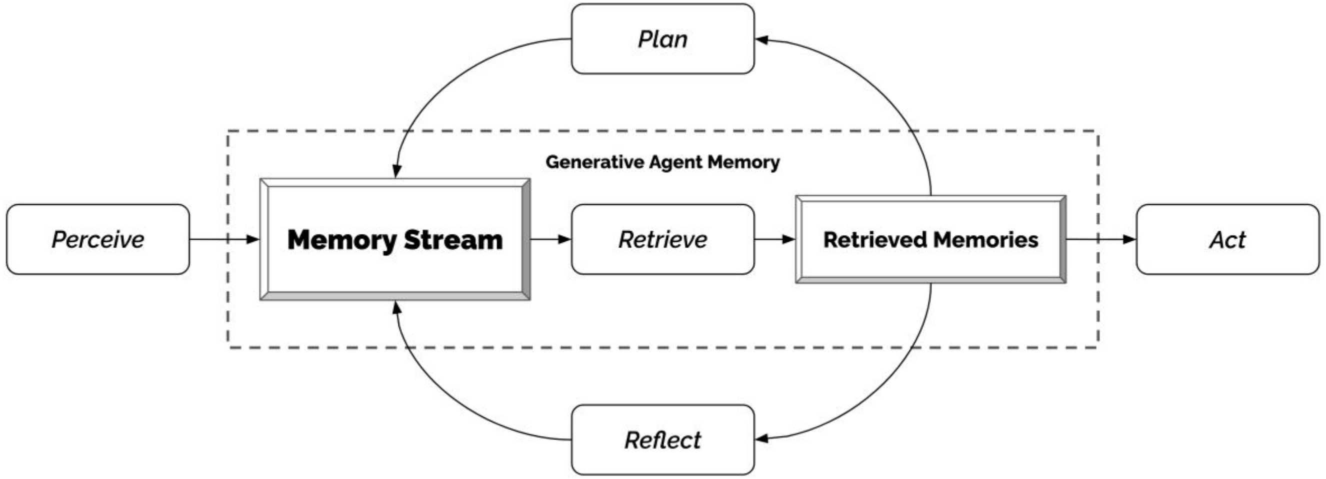
These memories are periodically retrieved and synthesized into reflections, allowing agents to draw broader conclusions about themselves, and their environment.

Memory retrieval is governed by three key factors:

- recency (recent memories are more accessible),
- importance (highly significant events are prioritized),
- relevance (only contextually relevant information is surfaced for decision-making).

Reflection allows for generalisation from their experiences, forming insights that influence future behavior. For example, an agent repeatedly working on a music composition may develop a self-perception as a passionate musician. This process enhances long-term coherence, helping agents behave in ways that align with their past interactions and evolving relationships.

Planning further integrates memory by allowing agents to anticipate future actions based on their prior experiences. Agents generate daily schedules, which they refine into detailed action sequences, recursively adjusting them based on new observations.



Most notable infrastructure layers for agentic memory

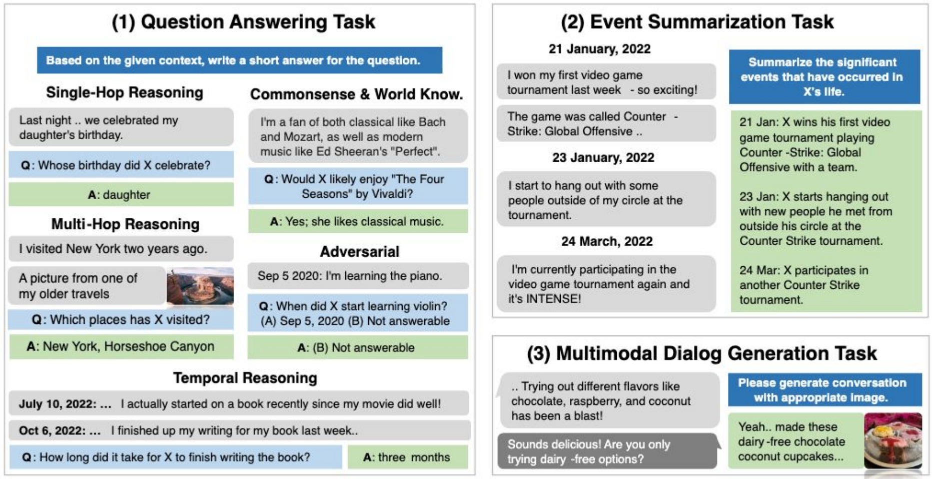
Mem0 - an universal memory layer using hybrid storage:

- vector DB, key-value store, and knowledge graph simultaneously — to give agents long-term, short-term, semantic, and episodic memory.
- When `add()` is called with messages, Mem0 extracts facts via an LLM pipeline, stores them across vector, key-value, and graph databases, then retrieves them via a scoring layer evaluating relevance, importance, and recency.
- https://snap-research.github.io/locomo/

Zep / Graphiti takes a temporally-aware knowledge graph approach where every fact carries `valid_at` and `invalid_at` timestamps, enabling agents to reason about how information changes over time - **sub-200ms retrieval latency** without agentic retrieval loops arXiv:

- existing retrieval-augmented generation (RAG) frameworks for large language model (LLM)-based agents are limited to static document retrieval, enterprise applications demand dynamic knowledge integration from diverse sources including ongoing conversations and business data.

Letta (MemGPT) models agent memory after OS virtual memory management — agents autonomously move data between in-context core memory, conversation recall memory, and vector DB-backed archival memory using tool calls.



(1) Question Answering Task

Based on the given context, write a short answer for the question.

Single-Hop Reasoning

Last night .. we celebrated my daughter's birthday.

Q: Whose birthday did X celebrate?

A: daughter

Multi-Hop Reasoning

I visited New York two years ago.

A picture from one of my older travels

Q: Which places has X visited?

A: New York, Horseshoe Canyon

Commonsense & World Know.

I'm a fan of both classical like Bach and Mozart, as well as modern music like Ed Sheeran's "Perfect".

Q: Would X likely enjoy "The Four Seasons" by Vivaldi?

A: Yes; she likes classical music.

Adversarial

Sep 5 2020: I'm learning the piano.

Q: When did X start learning violin? (A) Sep 5, 2020 (B) Not answerable

A: (B) Not answerable

Temporal Reasoning

July 10, 2022: ... I actually started on a book recently since my movie did well!

Oct 6, 2022: ... I finished up my writing for my book last week..

Q: How long did it take for X to finish writing the book? A: three months

(2) Event Summarization Task

21 January, 2022

I won my first video game tournament last week - so exciting!

The game was called Counter - Strike: Global Offensive ..

23 January, 2022

I start to hang out with some people outside of my circle at the tournament.

24 March, 2022

I'm currently participating in the video game tournament again and it's INTENSE!

Summarize the significant events that have occurred in X's life.

21 Jan: X wins his first video game tournament playing Counter -Strike: Global Offensive with a team.

23 Jan: X starts hanging out with new people he met from outside his circle at the Counter Strike tournament.

24 Mar: X participates in another Counter Strike tournament.

(3) Multimodal Dialog Generation Task

.. Trying out different flavors like chocolate, raspberry, and coconut has been a blast!

Sounds delicious! Are you only trying dairy -free options?

Please generate conversation with appropriate image.

Yeah.. made these dairy -free chocolate coconut cupcakes...

Bilt deployed 1M Letta-powered agents to deliver personalised local merchant recommendations

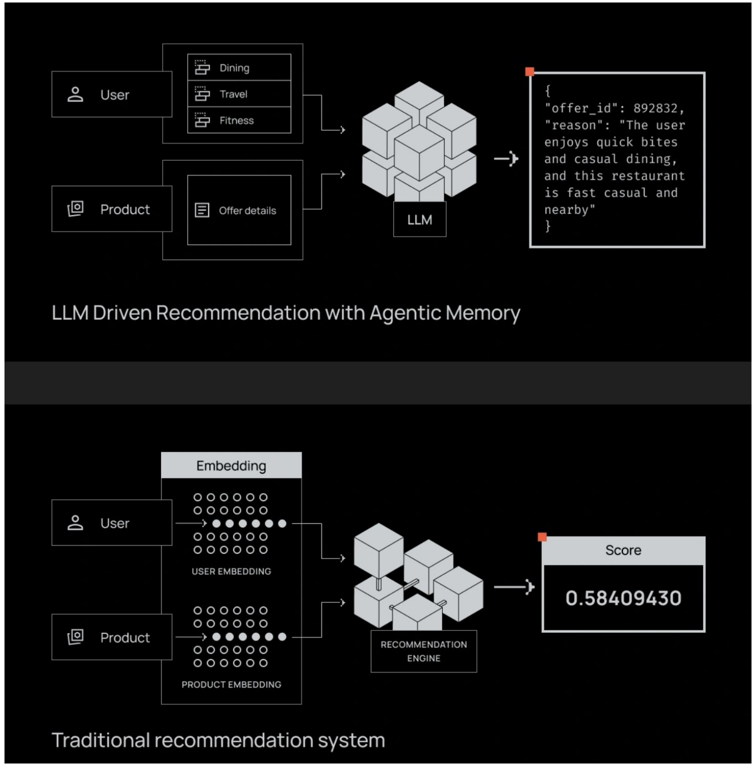
Letta Each agent maintains persistent memory blocks containing compressed summaries of user transaction patterns and platform engagement. The architecture evolved through rapid iteration: from a single agent per user to a multi-agent system with specialized dining agents, supervisors, and re-rankers.

The technical design separates memory creation from inference. Strong models (Anthropic) run asynchronously in batch mode to create rich memory summaries from unstructured data, while faster, cheaper models handle real-time serving using those memory blocks. letta The result: 99%+ of latency is pure LLM inference - Letta's memory layer adds negligible overhead.

Perhaps most notably, non-technical staff including the CEO can directly modify agent behavior through Letta's Agent Development Environment.

approach revealed important limitations. "The fine tuned model was very matter-of-fact and lost its creativity," Krauth said. "We wanted deeper inferences and more creative takes on how these benefits are all related to each other."

Zep did similar for art preferences to show them evolving overtime



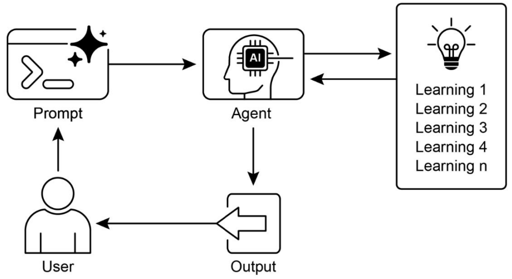
```
{
"offer_id": 892832,
"reason": "The user
enjoys quick bites
and casual dining,
and this restaurant
is fast casual and
nearby"
}
```

Chapter 9 - Learning and Adaptation

How different methods work?

Which one is better for specific limitations?

What are self-improving agents?



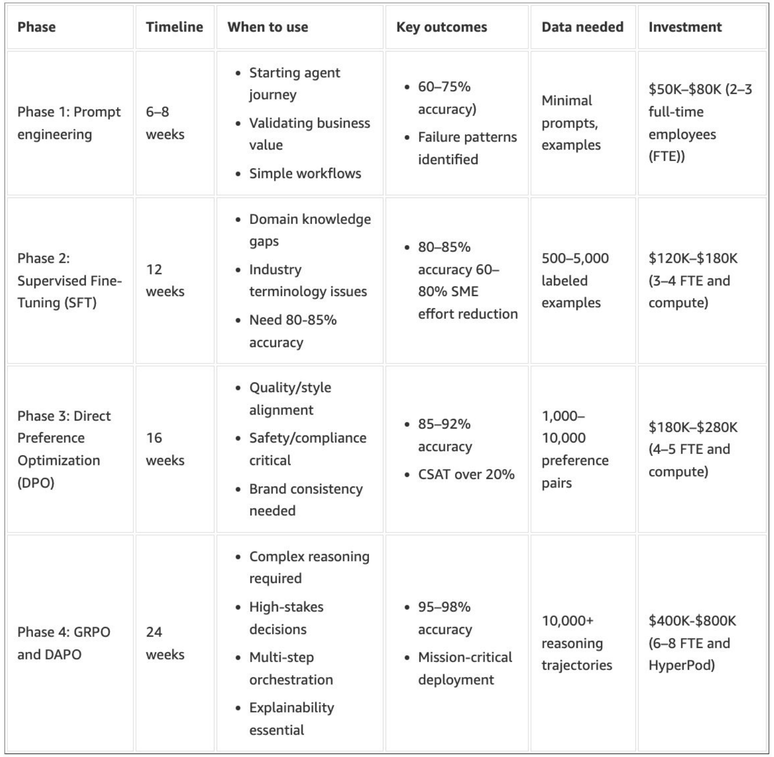
Learning and adaptation allows agents to improve autonomously through experience and environmental interaction

- **Few-Shot/Zero-Shot**: LLMs adapt with minimal/no examples
- **Online Learning**: Continuous updates from streaming data
- **Memory-Based Learning**: Using past experiences for current decisions
- **Supervised Learning**: Learning from labeled examples
- **Reinforcement Learning**: Reward/penalty signals guide behavior optimization (PPO algorithm)

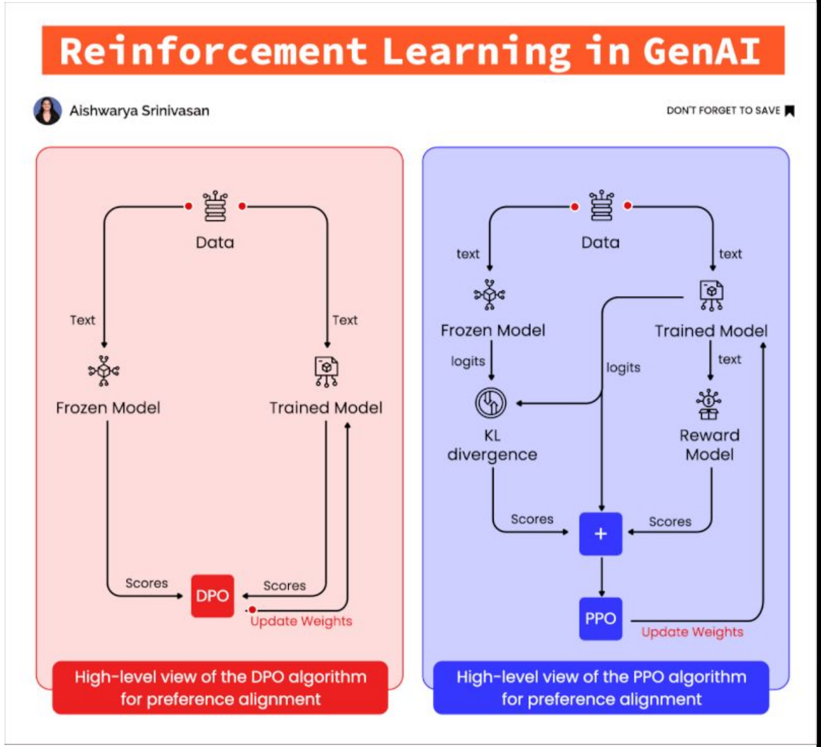
Training Methods (comparison):

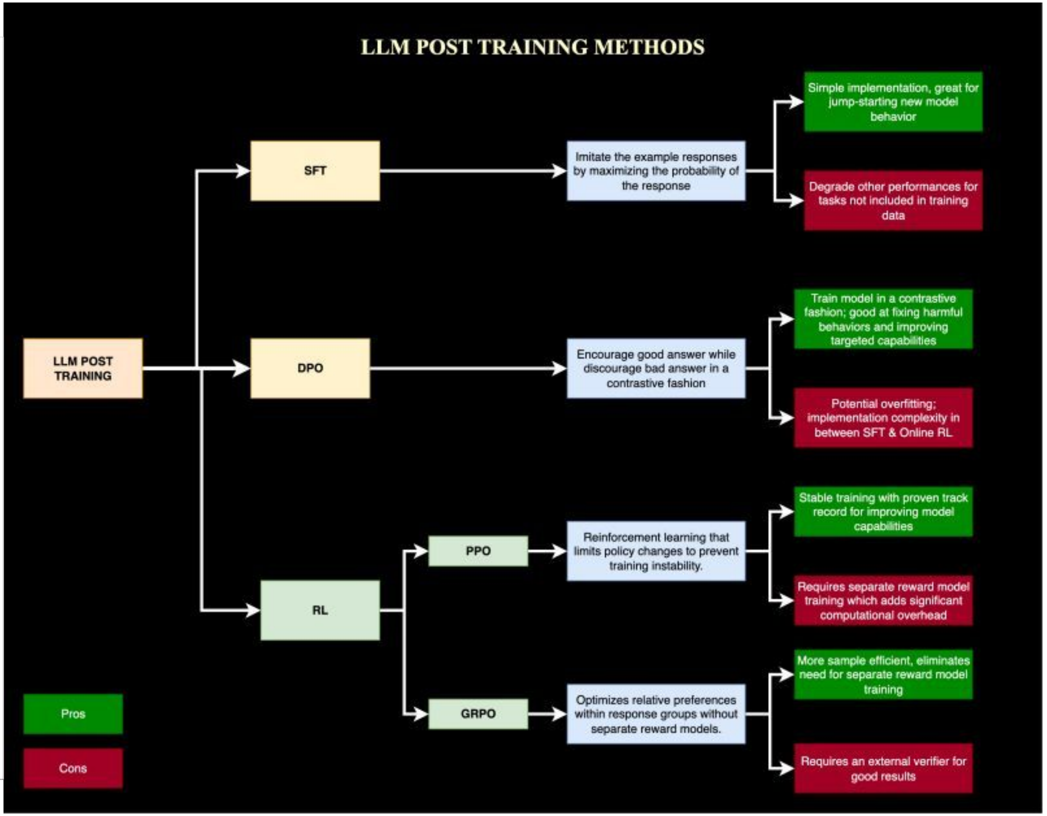
- PPO: Proximal Policy Optimization
  a. train agents in environments with a continuous range of actions, like controlling a robot's joints or a character in a game. Its main goal is to reliably and stably improve an agent's decision-making strategy, known as its policy.
- DPO: Direct Preference Optimization - simpler alternative to PPO for LLM alignment

https://aws.amazon.com/blogs/machine-learning/advanced-fine-tuning-techniques-for-multi-agent-orchestration-patterns-from-amazon-at-scale/



Phase	Timeline	When to use	Key outcomes	Data needed	Investment
Phase 1: Prompt engineering	6–8 weeks	• Starting agent journey • Validating business value • Simple workflows	• 60–75% accuracy) • Failure patterns identified	Minimal prompts, examples	$50K–$80K (2–3 full-time employees (FTE))
Phase 2: Supervised Fine-Tuning (SFT)	12 weeks	• Domain knowledge gaps • Industry terminology issues • Need 80-85% accuracy	• 80–85% accuracy 60–80% SME effort reduction	500–5,000 labeled examples	$120K–$180K (3–4 FTE and compute)
Phase 3: Direct Preference Optimization (DPO)	16 weeks	• Quality/style alignment • Safety/compliance critical • Brand consistency needed	• 85–92% accuracy • CSAT over 20%	1,000–10,000 preference pairs	$180K–$280K (4–5 FTE and compute)
Phase 4: GRPO and DAPO	24 weeks	• Complex reasoning required • High-stakes decisions • Multi-step orchestration • Explainability essential	• 95–98% accuracy • Mission-critical deployment	10,000+ reasoning trajectories	$400K-$800K (6–8 FTE and HyperPod)





Imitate the example responses by maximizing the probability of the response

Simple implementation, great for jump-starting new model behavior

Degrade other performances for tasks not included in training data

Encourage good answer while discourage bad answer in a contrastive fashion

Train model in a contrastive fashion; good at fixing harmful behaviors and improving targeted capabilities

Potential overfitting; implementation complexity in between SFT & Online RL

Reinforcement learning that limits policy changes to prevent training instability.

Stable training with proven track record for improving model capabilities

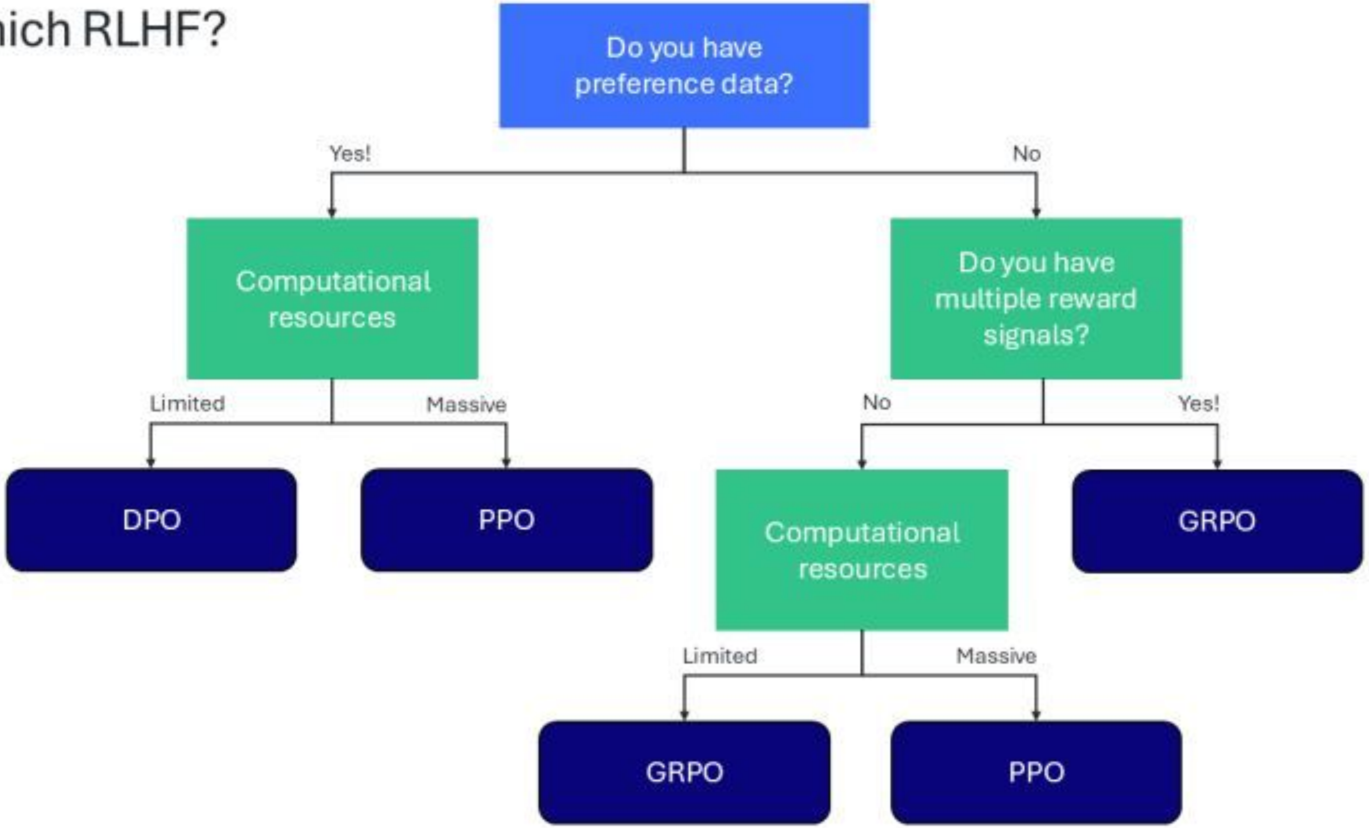
Requires separate reward model training which adds significant computational overhead

Optimizes relative preferences within response groups without separate reward models.

More sample efficient, eliminates need for separate reward model training

Requires an external verifier for good results https://huggingface.co/reasoning-course

Which RLHF?

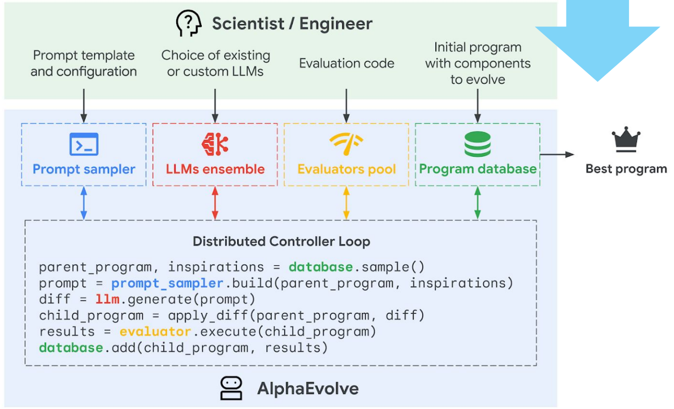


Self-improving coding agents were the first concepts to be implemented using new frameworks

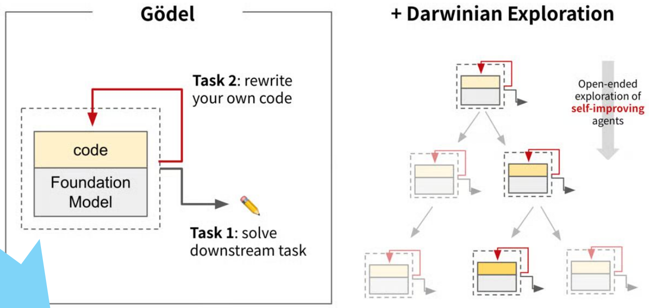
Self-Improving Coding Agent (SICA):

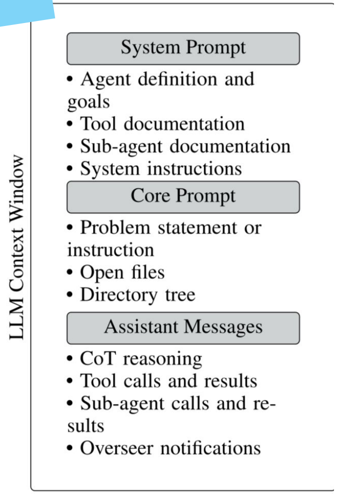
- Modular architecture with specialized sub-agents (coding, problem-solving, reasoning)
- Asynchronous overseer LLM monitors for pathological behavior
- Docker containerization for security
- Interactive webpage visualizes event bus and callgraph
- Challenge: Prompting LLM to propose novel, feasible modifications

https://deepmind.google/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/


Scientist / Engineer
Prompt template and configuration
Choice of existing or custom LLMs
Evaluation code
Initial program with components to evolve
Prompt sampler
LLMs ensemble
Evaluators pool
Program database
Best program
Distributed Controller Loop
AlphaEvolve

```
parent_program, inspirations = database.sample()
prompt = prompt_sampler.build(parent_program, inspirations)
diff = llm.generate(prompt)
child_program = apply_diff(parent_program, diff)
results = evaluator.execute(child_program)
database.add(child_program, results)
```


Gödel
Task 2: rewrite your own code
code
Foundation Model
Task 1: solve downstream task
+ Darwinian Exploration
Open-ended exploration of self-improving agents



System Prompt
- Agent definition and goals
- Tool documentation
- Sub-agent documentation
- System instructions

Core Prompt
- Problem statement or instruction
- Open files
- Directory tree

Assistant Messages
- CoT reasoning
- Tool calls and results
- Sub-agent calls and results
- Overseer notifications

Agents learn without retraining using RL training libraries, compilation-based optimizers, and embodied learning agents

DSPy replaces manual prompt engineering with compilation-based optimization. Its optimizers (MIPROv2, BootstrapFewShot, GEPA) automatically discover better prompts, few-shot examples, and fine-tuning weights to maximize a given metric: **ReAct agent scores jumped from 24% to 51%** on HotPotQA after DSPy optimization of gpt-4o-mini.

TRL is full-stack library for post-training LLMs with reinforcement learning. It implements SFT, GRPO (used to train DeepSeek R1), DPO, and PPO trainers in a unified API.

OpenRLHF specializes in distributed, high-throughput RL training for 70B+ parameter models using a Ray + vLLM architecture. It supports PPO, GRPO, REINFORCE++, and DAPO..

Voyager lifelong learning agent, operating in Minecraft without any parameter fine-tuning, it uses three interlocking mechanisms: an automatic curriculum where GPT-4 proposes progressively harder tasks based on current capabilities, an ever-growing skill library storing verified code programs indexed by semantic description for future retrieval, and iterative self-verification with environment feedback.

DSPy enables durable deployments through better prompts, decreases model costs and flexible pipelines

JetBlue Airways deployed DSPy-optimized LLM pipelines for customer feedback classification and RAG-powered predictive maintenance chatbots:

- single words in manually crafted prompts could make or break deployment quality.
- DSPy signature optimizers and in-context learning optimizers automatically determine optimal examples to include in prompts

DSPy enables 10x cheaper models to at least match GPT-4 accuracy - Gradient AI optimises prompts

Self-improving agentic email generation for outbound sales, using DSPy with a human-in-the-loop feedback loop:

- BootstrapFewShot for under 20 samples,
- BootstrapFewShot with Random Search for ~50 samples
- MIPROv2 for 200+ samples.
- An "Approval Required" setting pauses agents at key checkpoints; human-approved outputs feed directly back into DSPy training data,
- Evaluation uses SemanticF1 scoring (LLM-based semantic precision and recall), and optimized programs are cached in a knowledge base for efficient reuse.

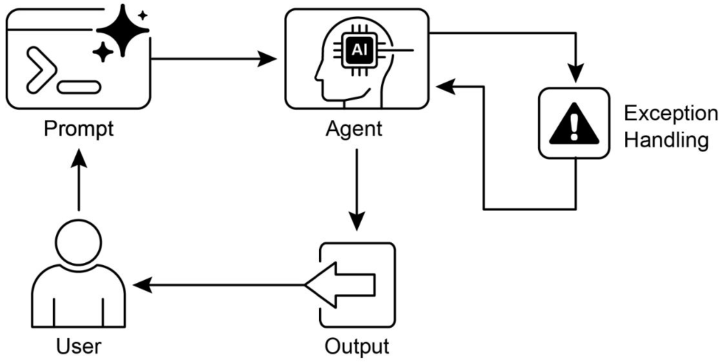
Hyper-personalized promotional emails for e-commerce brands:

- execution logs from production serve as labeled data for model fine-tuning, with prompt updates deployed independently from application code
- This decoupled approach, built on the Vellum AI development platform, delivered 50% faster debugging through the observability layer and enables real-time A/B testing of prompts and models.

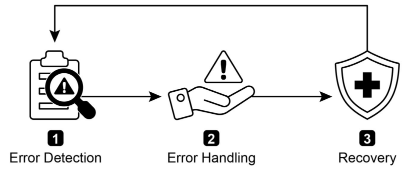
Chapter 12 - Exception Handling and Recovery

What are different scenarios for exception handling and recovery?

How could a generic exception handling and recovery process look like?



Exception handling and recovery are a must for building production-grade agents that maintain operational integrity



Three-Phase Pattern:

1. **Error Detection:** Identifying invalid tool outputs, API errors (404, 500), timeouts, malformed responses, and anomalies
2. **Error Handling:** Logging for diagnostics, retries for transient failures, fallbacks to alternative methods, graceful degradation (partial functionality), notifications to operators
3. **Recovery:** State rollback (reversing transactions), diagnosis and self-correction, replanning or adjusting strategy, escalation to humans when needed

Implementation Pattern in the book: Sequential agent architecture with primary handler, fallback handler, and response agent ensuring ordered execution and layered recovery.

Engineering Principles: Modularity (specialized agents collaborating), checkpoint/rollback systems, fault tolerance, and treating agents as complex software systems requiring traditional engineering rigor.

```
from google.adk.agents import Agent, SequentialAgent

# Agent 1: Tries the primary tool. Its focus is narrow and clear.
primary_handler = Agent(
   name="primary_handler",
   model="gemini-2.0-flash-exp",
   instruction="""
Your job is to get precise location information.
Use the get_precise_location_info tool with the user's provided
address.
   """,
   tools=[get_precise_location_info]
)

# Agent 2: Acts as the fallback handler, checking state to decide its
action.
fallback_handler = Agent(
   name="fallback_handler",
   model="gemini-2.0-flash-exp",
   instruction="""
Check if the primary location lookup failed by looking at
state["primary_location_failed"].
- If it is True, extract the city fro
use the get_general_area_info tool.
- If it is False, do nothing.
   """,
```

```
   tools=[get_general_area_info]
)

# Agent 3: Presents the final result from the state.
response_agent = Agent(
   name="response_agent",
   model="gemini-2.0-flash-exp",
   instruction="""
Review the location information stored in state["location_result"].
Present this information clearly and concisely to the user.
If state["location_result"] does not exist or is empty, apologize
that you could not retrieve the location.
   """,
   tools=[] # This agent only reasons over the final state.
)

# The SequentialAgent ensures the handlers run in a guaranteed order.
robust_location_agent = SequentialAgent(
   name="robust_location_agent",
   sub_agents=[primary_handler, fallback_handler, response_agent]
)
```

EH must be introduced at all levels of agentic architecture

The most effective production pattern stacks these tools in defense-in-depth layers, each handling a specific failure mode:

- **Validation layer** — NeMo Guardrails or Guardrails AI intercepts malformed inputs/outputs before they propagate
- **Gateway layer** — LiteLLM handles model fallbacks, rate limiting, and cooldowns across providers
- **Circuit breaker layer** — PyBreaker prevents cascading failures from downed LLM providers
- **Retry layer** — Tenacity handles transient failures with exponential backoff and jitter
- **Orchestration layer** — Temporal provides durable execution, crash recovery, and saga-pattern rollback

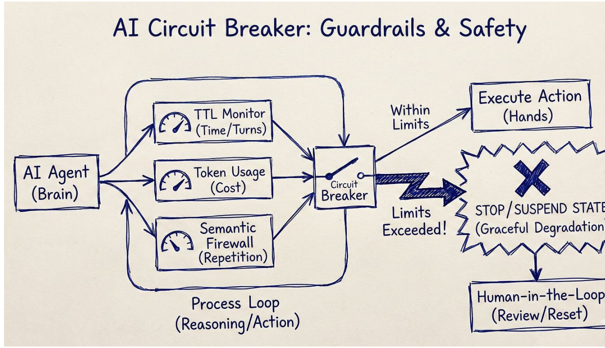
For goal monitoring, Langfuse sits across all layers, tracing every call and feeding evaluation metrics back into the system—closing the loop between Chapter 11's goal tracking and Chapter 12's error recovery.

Tenacity used by LangChain or completion_with_retry(). Custom callbacks (before, after, before_sleep) enable observability hooks.

LiteLLM - comprehensive resilience layer purpose-built for LLM calls:
- Router object implements multi-model fallbacks (if gpt-4o fails, automatically try claude-3-haiku),
- context window fallbacks (switch to larger-context models when input exceeds limits)
- content policy fallbacks (separate chains for policy violations).
- RetryPolicy offers per-exception-type retry counts (AuthenticationErrorRetries, RateLimitErrorRetries, TimeoutErrorRetries).

Temporal - each LLM call or tool execution becomes a Temporal Activity with granular retry policies configurable per exception type - meaning if a worker dies mid-LLM-call, processing resumes on another worker without re-running completed steps.

NVIDIA NeMo programmable middleware layer between application code and LLMs:
- input rails (reject/modify before reaching LLM),
- dialog rails (enforce conversation flows),
- retrieval rails (filter RAG chunks),
- execution rails (validate tool call inputs/outputs
- output rails (validate/modify responses).


AI Circuit Breaker: Guardrails & Safety
AI Agent (Brain)
TTL Monitor (Time/Turns)
Token Usage (Cost)
Semantic Firewall (Repetition)
Circuit Breaker
Process Loop (Reasoning/Action)
Within Limits
Execute Action (Hands)
Limits Exceeded!
STOP/SUSPEND STATE (Graceful Degradation)
Human-in-the-Loop (Review/Reset)