Agentic Design Patterns: ReAct, Plan-then-Execute, ReWOO, LLMCompiler, Reflexion, and Multi-Agent Architectures

Recording: Coffee Chat - Agentic Design Patterns - 2025/11/07 13:44 CET - Recording

Transcription: Coffee Chat - Agentic Design Patterns - 2025/11/07 13:44 CET - Notes by Gemini

Motivation: Why We're Exploring This

Our current ReAct agent struggles with complex, multi-step queries that require comprehensive analysis and actionable recommendations. Two recent experiences illustrate this:

❌ Example 1: Simple query, inadequate response

- **Query**: "What actions can I take to reduce churn?"
- **Result**: Agent refused to provide recommendations, citing role limitations. Offered only to "analyze data" or provide "general HR knowledge with disclaimer."
- **Problem**: User received no actionable insights despite having relevant data available. Testing session link

✅ Example 2: Heavy prompt engineering → great results

- **Session**: Testing session link
- **Approach**: Manually guided agent through structured workflow:
  a. Asked agent to map out a comprehensive analysis plan
  b. Executed each step explicitly, validating intermediate results
  c. Had agent ask clarifying questions throughout the process
  d. Finally requested actionable recommendations based on complete analysis
- **Result**: High-quality, data-driven action plan with detailed supporting evidence.

The Gap: Example 2 required significant user effort to orchestrate what should be the agent's natural capability. Planning patterns (Plan-then-Execute, ReWOO) could bridge this gap by building structured workflows directly into the agent's reasoning process, delivering Example 2 quality with Example 1 simplicity.

Pattern deep dives

ReAct (Reason + Act) — Currently Used

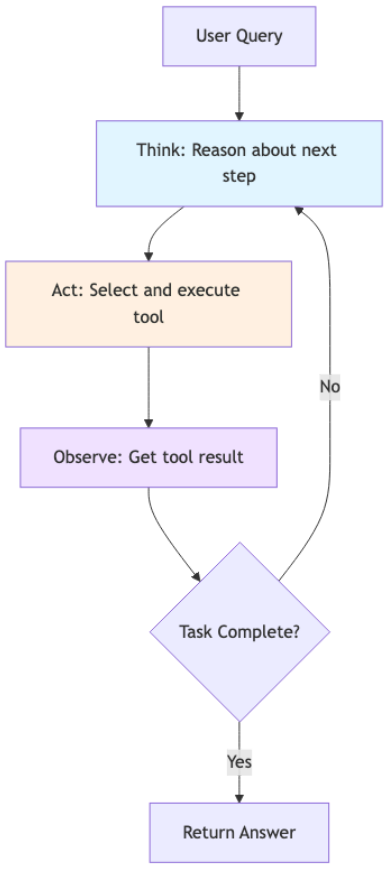
What

ReAct augments LLM action spaces by interleaving explicit reasoning traces (thoughts) with environment interactions (actions and observations). The agent generates natural language thoughts explaining its reasoning, executes corresponding actions (search, calculate, navigate), receives observations from the environment, and repeats until task completion.

Technical innovation: Extends the agent's action space from just environmental actions to include a language space for reasoning traces. Each thought decomposes goals into subgoals, tracks progress, injects commonsense knowledge, and handles exceptions—all visible to humans.

Our implementation: LangChain `create_agent` uses ReAct by default with 7 MCP servers (EDA, Sentiment, Org Psych, Answers, Questions, Questionnaire Info, Custom Info). Agent reasons → selects tool → observes result → reasons again → repeats (capped at `recursion_limit=10`).

Architecture Flow



Strengths

- **Maximum adaptability**: Handles unknown task complexity and ambiguous queries by discovering requirements during exploration
- **Transparent reasoning**: Every tool call has an associated thought trace, enabling debugging, compliance auditing, and human-in-the-loop intervention
- **Uncertainty handling**: Adapts strategy based on intermediate results—if initial search fails, can try alternative approaches
- **Multi-hop reasoning**: Naturally chains information across 3+ sources (HotpotQA: 27.4% accuracy, ALFWorld: 71% success vs 45% for action-only)
- **Reduced hallucination**: Grounds reasoning in external knowledge sources (6% false positives vs 14% for Chain-of-Thought alone)
- **Interactive environments**: Excels in text-based games, web navigation, and embodied AI tasks requiring environmental feedback

Limitations

- **Token inefficiency**: Requires LLM call for EACH tool invocation (typically 3-7 steps), increasing both latency and API costs significantly

- **Myopic planning**: Only plans for 1 sub-problem at a time without upfront global reasoning, leading to sub-optimal trajectories
- **Tool selection overload**: Performance degrades with 7+ tools—our 7 MCP servers at upper limit for single-agent ReAct (calendar scheduling drops to 2% with 7+ domains)
- **Higher reasoning errors**: 47% vs 16% failure rate compared to pure Chain-of-Thought due to structural constraints of interleaving
- **Repetitive loops**: Can generate same action sequence repeatedly, unable to break out until hitting token limits (23% of HotpotQA failures from poor search results)
- **Search dependency**: When retrieval returns poor information, reasoning derails with difficult recovery
- **Requires large models**: Models smaller than 62B parameters show poor performance

Core limitations that planning patterns address:

- Requires an LLM call for EACH tool invocation (expensive)
- Only plans for 1 sub-problem at a time (myopic, can lead to sub-optimal trajectories)
- No explicit global reasoning about the full task

When to use ReAct

- **Exploratory tasks** where complexity is unknown upfront and maximum adaptability needed
- **Dynamic scenarios** where next steps depend on previous results or require trial-and-error
- **Interpretability critical** for debugging, compliance auditing, or human-in-the-loop oversight
- **Interactive environments** like text games, web navigation, or embodied AI requiring environmental feedback

When to avoid ReAct

- **Cost-sensitive high-volume** systems where token efficiency is critical (use ReWOO or LLMCompiler for 80% cost reduction)
- **Well-scoped repeatable workflows** with predictable steps (use Plan-then-Execute for deterministic execution)
- **Speed-critical applications** requiring fastest execution with parallelizable tasks (use LLMCompiler)
- **Large tool sets** (>7 tools) where tool selection degrades performance (use Multi-Agent Supervisor)

Demo Example

```
Query: "What measures can I take to reduce churn?" (Alpenblick
hospitality group)

1. Think: "Need to identify churn drivers first"
2. Act: get_sentiment_statistics()
3. Observe: 38% negative sentiment, "Workload" top concern

4. Think: "Check org psych for engagement issues"
5. Act: get_org_psych_analysis(theory="engagement")
6. Observe: Engagement 2.4/5.0, autonomy 2.1/5.0

7. Think: "Synthesize into actionable recommendations"
8. Return: "Reduce churn by: (1) Address workload—38% negative
sentiment
   (2) Boost autonomy—critically low at 2.1/5.0
   (3) Improve engagement—2.4/5.0 indicates disengagement"
```

Plan-then-Execute (P-t-E)

What

Plan-then-Execute is a full agentic framework with three distinct components: **Planner**, **Executor**, and **Replanner**. The planner generates initial multi-step plans as structured lists. The executor (typically a ReAct agent) carries out

individual steps using available tools. The replanner examines completed steps and decides whether to continue with remaining steps, generate a revised plan, or respond with final results.

Inspired by Plan-and-Solve Prompting: The architecture draws inspiration from Wang et al.'s Plan-and-Solve (PS/PS+) prompting technique (ACL 2023), which improved zero-shot arithmetic reasoning from 70.4% to 76.7% by having LLMs explicitly plan before solving. However, P-t-E extends this from a simple prompting technique into a production agentic framework with actual tool execution and dynamic replanning capabilities.

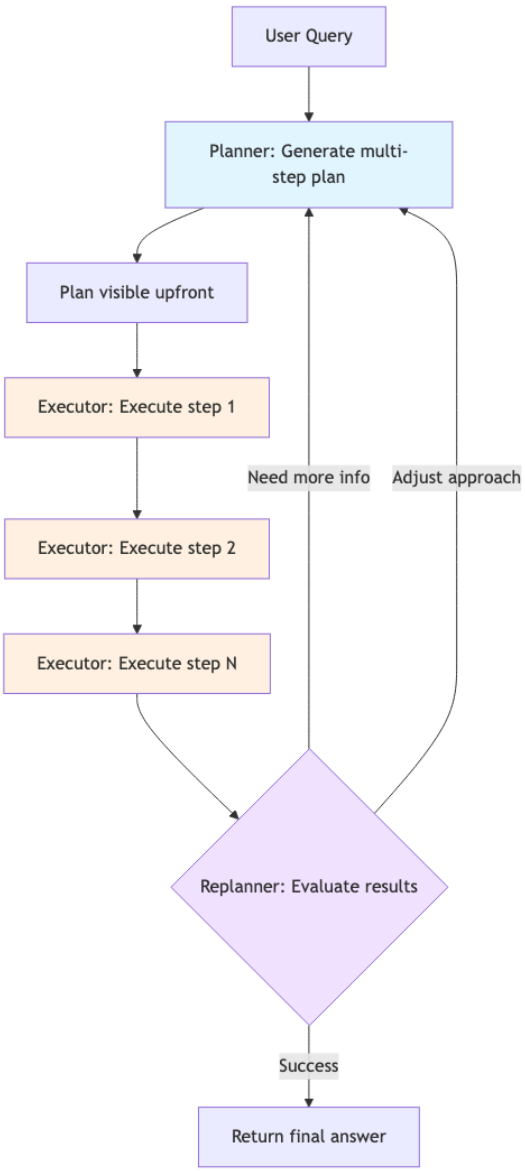
The original **Plan-and-Solve+ (PS+) prompt**:

> "Let's first understand the problem, extract relevant variables and their numerals, make a complete plan, calculate intermediate variables with attention to correct numerical calculation and commonsense, solve step by step, and show the answer."

This prompt reduced calculation errors from 7% to 5% and missing-step errors from 12% to 7% by explicitly instructing variable extraction and intermediate calculation attention.

Key mechanism: After execution completes, the replanner decides whether to finish with a response or generate a follow-up plan if the initial plan didn't achieve desired results. This enables adaptive recovery from plan failures while maintaining the efficiency benefits of upfront planning.

Architecture Flow



Strengths

- **Speed advantage**: Multi-step workflows execute faster since the large planning LLM is only called during planning and replanning phases rather than after every action
- **Cost optimization**: Can use more sophisticated models for planning and smaller models for execution. Plus fewer tool calls.
- **Quality improvement**: Forces planner to "think through" ALL steps upfront creating more coherent multi-step solutions
- **Deterministic and auditable**: Plan is visible upfront before any execution, making it easier to test, debug, and validate against business requirements

Limitations

- **Sequential execution bottleneck**: Tasks execute one after another (ReWOO and LLMCompiler address this)
- **Planning rigidity**: Brittle if initial plan is wrong; limited adaptability if user query needs different tools mid-flight (requires costly replanning)

- **Planning overhead**: Not justified for simple single-step queries where ReAct or direct function calling would be faster
- **Context window limitations**: Performance degrades as domain and tool counts increase; models like o3-mini show steeper performance drops as context expands
- **Replanning ambiguity**: Deciding when to replan versus respond lacks clear criteria, adding overhead
- **Dynamic environment challenges**: Highly dynamic environments where plans quickly become obsolete, creative open-ended tasks, and ambiguous requirements all struggle with upfront planning

When to use Plan-then-Execute

- **Multi-step complex tasks** with 5+ decomposable reasoning steps (research, data pipelines, long-horizon analysis)
- **Arithmetic reasoning** where explicit planning reduces missing-step and calculation errors
- **Repeatable workflows** with predefined procedures (customer support, batch reports)
- **Audit-critical scenarios** requiring deterministic, visible-upfront plans for validation
- **Cost optimization priority** using model tiering (larger model for planning, smaller model for execution)
- **Stable environments** where plans remain valid during execution and accuracy outweighs latency

When to avoid Plan-then-Execute

- **Simple single-step queries** where planning overhead isn't justified (use direct function calling)
- **Highly dynamic environments** where plans quickly become obsolete or need constant revision
- **Exploratory tasks** without clear structure requiring adaptive discovery (use ReAct)
- **Speed-critical applications** with parallelizable tasks (use LLMCompiler for faster execution)

Demo Example

```
Query: "What measures can I take to reduce churn?" (Alpenblick hospitality group)

Planner Phase:
┌──────────────────────────────────────────────────────────────┐
| PLAN:                                                        |
| 1. Identify churn reasons                                    |
|    get_summarized_eda_data(topics=["Churn", "Exit"])         |
| 2. Find subgroups where churn is predominant                 |
|    get_sentiment_by_team(department="all")                   |
| 3. Look at warning signs in surveys                          |
|    search_text_answers(keywords=["quit", "leave", "burnout"])  |
| 4. Look at company profile                                   |
|    get_custom_info(category="Company Profile")               |
| 5. Check if any actions were taken in the past               |
|    search_text_answers(keywords=["retention", "action plan"])  |
| 6. Draft possible actions                                    |
| 7. Return final response                                     |
└──────────────────────────────────────────────────────────────┘

Executor Phase (sequential execution):
Step 1: EDA shows 28% mention "Better opportunities", 19% "Workload"
Step 2: Engineering team 47% negative sentiment, Operations 41% negative
Step 3: 34 mentions of "burnout", 12 mentions of "leaving soon"
Step 4: Company: 150 employees, hospitality sector, 23% annual turnover
Step 5: No previous retention initiatives found in survey responses
Step 6: LLM drafts: Career paths, workload reduction, burnout prevention

Replanner: ✅ Complete, respond

Final Response:
"Churn Analysis Summary:
```

```
- Primary drivers: Career stagnation (28%), workload (19%)
- High-risk teams: Engineering (47% negative), Operations (41%)
- Warning signs: 34 burnout mentions, 12 'leaving soon' signals
- Context: 23% annual turnover, no prior retention efforts
- Recommended actions:
  1. Career development program (addresses 28% top driver)
  2. Engineering/Operations workload audit (targets high-risk teams)
  3. Burnout prevention initiative (addresses 34 warning signals)"
```

ReWOO (Reasoning Without Observation)

What

ReWOO introduces a three-module architecture that **completely separates planning from execution**. The Planner generates a complete multi-step plan before any tool execution using "foreseeable reasoning"—predicting needed information without observing actual results. Plans use variable placeholders (#E1, #E2, #E3) to reference future evidence, enabling subsequent steps to depend explicitly on prior results without waiting for actual observations.

Three-module architecture:

1. **Planner**: Generates complete reasoning graph with variable placeholders (#E1, #E2, #E3) before any tool execution. Plans what information is needed without seeing actual results.
2. **Worker**: Executes tools based on the Planner's blueprint, populating evidence variables with actual results—this phase involves no LLM reasoning, just pure execution.
3. **Solver**: Receives the complete plan plus all evidence and synthesizes the final answer, prompted to use evidence "with caution" to handle potential errors. Can partially compensate for Planner or Worker failures.

Critical innovation: **Variable substitution**—planning occurs using placeholders rather than actual tool outputs, eliminating the need to wait for observations during the reasoning phase. Tasks can reference previous outputs using syntax like `#E2` (e.g., `Search[Stats for #E2]`).

Token efficiency: On HotpotQA, ReWOO consumed **1,986 tokens** vs ReAct's 9,795 tokens (**5× token efficiency**), translating to $3.97 per 1,000 queries vs $19.59 for ReAct (**80% cost reduction**).

Architecture Flow

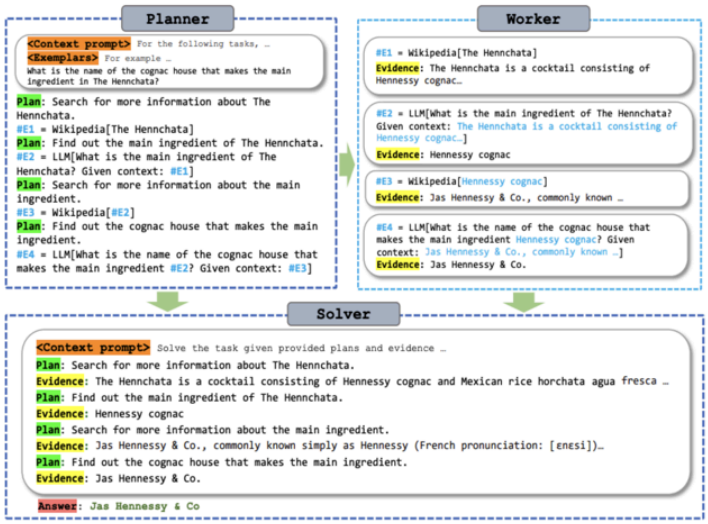


Figure 1: Workflow of ReWOO . Given a question, Planner composes a comprehensive blueprint of interlinked plans prior to tool response. The blueprint instructs Worker to use external tools and collect evidence. Finally, plans and evidence are paired and fed to Solver for the answer.

Strengths

- **Dramatic token efficiency**: 5× token reduction (1,986 vs 9,795 tokens on HotpotQA), **80% cost reduction** ($3.97 vs $19.59 per 1,000 queries) [Xu et al., 2023] (Focused context per task: Each task has only required context (input + variable values) rather than full history
-)
- **Improved accuracy**: HotpotQA 42.4% vs 40.8% for ReAct, TriviaQA 66.6% vs 59.4%, StrategyQA 66.6% vs 64.6%, SOTUQA 70.2% vs 64.8% (**8% absolute improvement**)
- **Eliminates prompt redundancy**: Question and context fed only twice (Planner and Solver) vs every step in ReAct—no observations clutter prompts since Planner never sees them
- **Robustness under tool failure**: 29.2% accuracy drop vs ReAct's 40.8% drop when tools fail, actually **saves 110 tokens during failure** while ReAct consumes 851 additional tokens attempting recovery
- **Explicit dependency tracking**: Variable flow through #E references makes reasoning traceable and debugging straightforward

Limitations

- **Sequential execution bottleneck**: Tasks execute one after another (total time = sum of tool times)—LLMCompiler addresses this with parallelization
- **Planning rigidity**: Once committed to a plan, execution proceeds regardless of observations; cannot dynamically switch tools or revise approaches mid-execution
- **Initial plan blind spots**: Reasoning happens before any observation, might miss edge cases that would be discovered during exploration
- **Tool count sensitivity**: Performance degraded from 42% with 2 tools to 37% with 7 tools—**our 7 MCP servers at the degradation threshold**
- **Real-time interactive applications**: Not suitable for scenarios requiring adaptive strategy based on intermediate results
- **No dynamic replanning**: Unlike Plan-then-Execute, ReWOO doesn't have a replanner component—some production implementations hybrid with ReAct fallback

When to use ReWOO

- Predictable multi-hop question answering where information dependencies are clear ("Find X, then use X to find Y")
- Complex multi-theory org psych queries requiring synthesis across multiple MCP servers
- High-volume production systems where **cost is critical** (80% cost reduction vs ReAct)
- Curated tool environments with **2-5 well-defined complementary tools** (our 7 MCP servers at upper limit)

When to avoid ReWOO

- **Exploratory tasks** requiring adaptive discovery or trial-and-error (use ReAct)
- **Large tool sets** with >5 options (our 7 servers borderline)
- **Dynamic environments** needing adaptive strategy based on intermediate results
- **Real-time interactive applications** or scenarios with highly uncertain tool reliability

Demo Example

```
Query: "What measures can I take to reduce churn?" (Alpenblick
hospitality group)

Planner Phase (foreseeable reasoning):
Plan: Get sentiment to identify pain points
#E1 = get_sentiment_statistics()

```

```
Plan: Check engagement levels
#E2 = get_org_psych_analysis(theory="engagement")

Plan: Analyze workload themes (likely driver based on experience)
#E3 = get_summarized_eda_data(topics=["Workload"])

Worker Phase (sequential execution, no LLM reasoning):
#E1 = {38% negative, "Workload" top concern (45%)}
#E2 = {Engagement: 2.4/5.0, Autonomy: 2.1/5.0 critical}
#E3 = {72% understaffing, 51% overtime pressure}

Solver Phase (synthesize with caution):
"Churn reduction measures for Alpenblick:
1. Staffing increase—72% report understaffing (#E3)
2. Autonomy programs—2.1/5.0 critical low (#E2)
3. Workload management—45% cite as concern (#E1)
Evidence: #E1, #E2, #E3"
```

LLMCompiler (Parallel Function Calling)

What

LLMCompiler draws inspiration from classical compiler design to optimize agent execution through parallel function calling. The framework decomposes user queries into **Directed Acyclic Graphs (DAGs)** representing tasks with explicit inter-dependencies, then executes independent tasks concurrently. This extends beyond ReWOO's sequential execution to achieve true parallelization while maintaining dynamic replanning capabilities.

Three-component architecture:

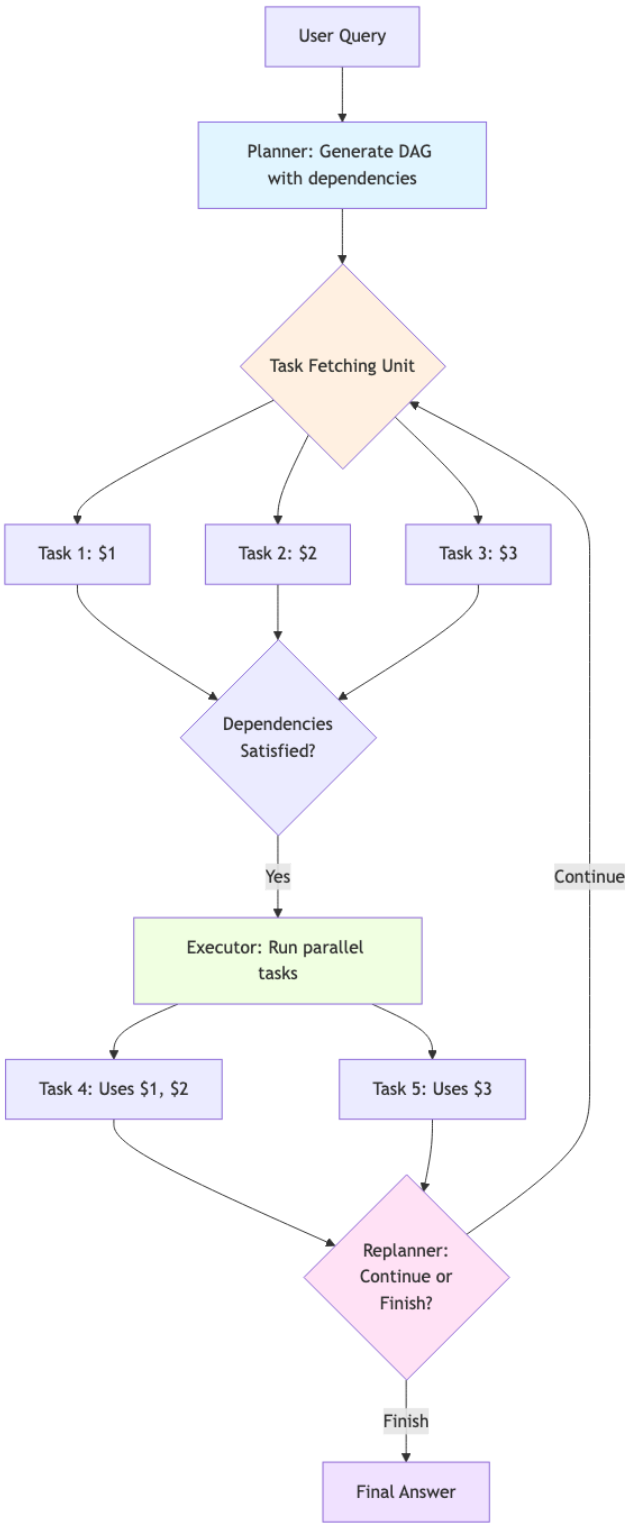
1. **Planner**: Generates task sequences with dependencies forming a DAG, identifying necessary tasks, input arguments, and inter-dependencies using placeholder variables ($1, $2, $3). Can **stream tasks as they're generated**, hiding planning latency behind tool execution through instruction pipelining.
2. **Task Fetching Unit**: Schedules and dispatches tasks as soon as dependencies are satisfied using a greedy policy. Replaces placeholder variables with actual outputs from completed tasks without requiring dedicated LLM calls.
3. **Executor**: Receives independent tasks and runs them asynchronously in parallel, with each task having dedicated memory for intermediate outcomes.

Critical innovation vs ReWOO: LLMCompiler supports **two key capabilities explicitly**: (1) parallel function calling reducing latency and cost, and (2) dynamic replanning for problems whose execution flow cannot be determined statically upfront.

Performance claims: Up to **3.7× latency speedup** (Movie Recommendation: 5.47s vs 20.47s for ReAct), **6.73× cost reduction** on some benchmarks, and **35% faster execution than OpenAI's proprietary parallel function calling**.

Our context: With 7 MCP servers involving data retrieval, sentiment analysis, and org psych theories, many queries have parallelizable operations (e.g., fetching sentiment + org commitment + workload data simultaneously).

Architecture Flow



Strengths

- **Dramatic performance efficiency**: Up to **3.7× latency speedup** (Movie Recommendation: 5.47s vs 20.47s) and **3-7× cost reduction** ($1.47 vs $5.00 per 1K queries on HotpotQA) through parallel execution—total time equals longest single tool per dependency level rather than sum of all tools
- **Quality improvements from upfront planning**: DAG planning prevents common ReAct failure modes including premature early stopping (85% of cases), repetitive loops (10% of cases), and context pollution from intermediate

observations—enabling **~9% accuracy improvement** (ParallelQA: 68.14% vs 59.59%)

- **Instruction pipelining optimization**: Streaming task generation hides planning latency behind tool execution, with Task Fetching Unit dispatching tasks as soon as dependencies are satisfied without waiting for complete plan generation
- **Dynamic replanning capability**: Unlike ReWOO's rigid commit-and-execute, supports replanning for problems whose execution flow cannot be determined statically, combining parallelization benefits with adaptability
- **Architecture flexibility**: Model-agnostic design demonstrated across multiple model families enabling cost-quality trade-offs through model tiering

Limitations

- **Parallelization limitations (Amdahl's Law)**: Planner overhead (~1.88s) can't be parallelized, straggler effects mean slowest task determines completion time (1.13s vs 0.61s average), and speedup is highly workload-dependent (3.7× for 8-way parallel vs 1.8× for 2-way)—preventing ideal N× scaling for N parallel tasks
- **Implementation and debugging complexity**: Requires DAG scheduler, task fetching logic, and parallel execution infrastructure making it significantly more complex than ReAct/P-t-E—parallel execution complicates error tracing and requires careful dependency identification upfront
- **Requires parallelizable workflows**: Sequential dependencies or complex causal chains see minimal benefit (Game of 24: 2.89× vs Movie Recommendation: 3.7×)—pattern only valuable when tasks have substantial independent operations
- **Production readiness concerns**: Newer pattern (Dec 2023) with less battle-testing than ReAct, unknown tool count sensitivity at 7+ tools (ReWOO showed 42%→37% degradation), and higher memory overhead from dedicated context per parallel task
- **Replanning overhead trade-off**: Dynamic replanning adds latency compared to ReWOO's commit-and-execute approach—flexibility comes at execution time cost

When to use LLMCompiler

- **Speed-critical applications** demanding fastest execution (1.8-3.7× faster with parallel workflows)
- **Embarrassingly parallel workflows** with multiple independent data fetches running concurrently
- **Cost-sensitive high-volume** systems where 3-6× cost reduction matters at scale
- **Clear task dependencies** where dependency graphs are predictable (sentiment + commitment + workload)

When to avoid LLMCompiler

- **Sequential workflows** where tasks have long dependency chains (minimal parallelization benefit)
- **Exploratory tasks** with unpredictable dependencies making DAG planning difficult (use ReAct)
- **Resource-constrained environments** unable to support parallel execution infrastructure
- **Immature tooling concerns** if production validation and battle-testing are critical

Demo Example

```
Query: "Create comprehensive churn risk report: analyze Engineering
sentiment, org commitment, and workload themes"

Step 1: Planner Phase (generates DAG with dependencies)
Planner streams task generation:
┌──────────────────────────────────────────────────────────────┐
│ DAG PLAN (with explicit dependencies):                       │
│                                                               │
│ $1 = get_team_info(team_name="Engineering")                  │
│   Dependencies: [] (no dependencies, can execute immediately)  │
│                                                               │
│ $2 = get_sentiment_statistics(team_ids=[$1])                 │
```

```
|   Dependencies: [$1] (wait for team info)                        |
|                                                                   |
| $3 = get_org_psych_analysis(                                     |
|        question="commitment",                                     |
|        theory="organizational_commitment",                        |
|        team_ids=[$1]                                              |
|      )                                                            |
|   Dependencies: [$1] (wait for team info)                        |
|                                                                   |
| $4 = get_summarized_eda_data(                                    |
|        topics=["Workload"],                                       |
|        team_ids=[$1]                                              |
|      )                                                            |
|   Dependencies: [$1] (wait for team info)                        |
|                                                                   |
| $5 = synthesize_churn_report(                                    |
|        sentiment=$2,                                              |
|        commitment=$3,                                             |
|        workload=$4                                                |
|      )                                                            |
|   Dependencies: [$2, $3, $4] (wait for all analyses)             |
└──────────────────────────────────────────────────────────────────┘

Step 2: Task Fetching Unit (schedules based on satisfied dependencies)
Iteration 1:
  - $1 has no dependencies → DISPATCH to Executor
  - $2, $3, $4 waiting for $1
  - $5 waiting for $2, $3, $4

Step 3: Executor (parallel execution)
Execute $1: get_team_info(team_name="Engineering")
  → Result: {team_id: 42, size: 87 members, department: "Technology"}
  → $1 COMPLETE

Step 4: Task Fetching Unit (re-check dependencies)
Iteration 2:
  - $1 complete, replace placeholder with actual value
  - $2, $3, $4 now have all dependencies satisfied
  - DISPATCH $2, $3, $4 IN PARALLEL ⚡
```

Reflexion (Self-Reflective Iterative Improvement)

What

Reflexion introduces **verbal self-reflection and iterative refinement** to agent architectures. After generating an initial solution, the agent reflects on failures by producing natural language feedback about what went wrong and how to improve. This reflection is stored in an **episodic memory buffer** and provided as context for subsequent trials, enabling the agent to learn from mistakes within a task without parameter updates.

Three-component architecture:

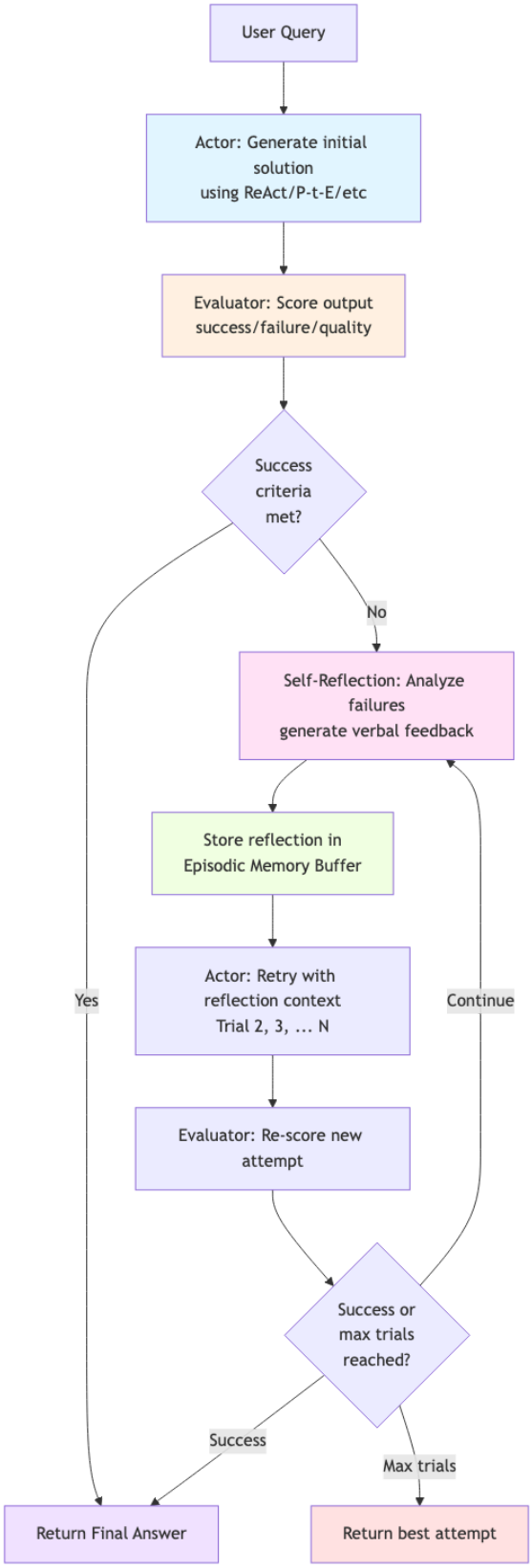
1. **Actor**: Generates text and actions based on state observations and reflection memory (typically a ReAct agent)
2. **Evaluator**: Scores outputs using task-specific heuristics, learned reward models, or binary success/failure signals. Provides feedback on what worked and what didn't.
3. **Self-Reflection**: Generates verbal reinforcement cues from evaluation signals and trajectory history. Creates natural language summaries of failure patterns (e.g., "Search query was too specific, try broader terms" or "Missed checking sentiment before analyzing commitment")

Critical innovation: Unlike traditional RL which updates model weights through backpropagation, Reflexion stores verbal reflections in episodic memory and provides them as additional context. This enables **learning within a task** through language-based feedback rather than requiring retraining.

Performance claims: **20-25% success rate improvement** on complex tasks (ALFWorld: 97% vs 75% baseline), but at **3-12× cost increase** due to multiple trial iterations. Game of 24 improved from 4% to 74% with 3 reflections, HumanEval code generation reached 91% pass@1 (vs 80% without reflection).

Our context: Could add reflection layer on top of current ReAct agent—after generating churn risk report, agent critiques its own analysis ("Did I check all relevant sentiment dimensions? Should I have examined more team segments?") and improves on retry.

Architecture Flow



Strengths

- **Dramatic quality improvements through iterative refinement**: **20-70 percentage point success rate gains** (Game of 24: 4%→74%, ALFWorld: 75%→97%) by learning from failures across trials—particularly effective for long-horizon tasks requiring 50+ steps where single-pass approaches struggle

- **Verbal self-reflection with episodic memory**: Core innovation enabling learning within a task without parameter updates—agent generates human-interpretable failure analyses (e.g., "Search too narrow, try broader terms") stored in memory buffer, preventing repeated mistakes in subsequent trials
- **Complementary architecture wrapper**: Unlike other patterns that replace ReAct/P-t-E, Reflexion wraps around any existing Actor pattern as a quality-enhancing meta-layer—can add reflection to ReAct, Plan-then-Execute, or ReWOO without changing core architecture
- **Adaptive learning from evaluation signals**: Supports flexible evaluation approaches including task-specific heuristics, learned reward models, or LLM-as-evaluator—reflections guide Actor toward more promising action spaces rather than random exploration based on feedback quality

Limitations

- **Multi-trial cost-latency multiplication**: **3-12× cost increase** and proportional latency impact (3 trials = 3× execution time) makes it incompatible with real-time requirements or cost-constrained high-volume systems—must set max_trials budget (typically 3-5) to prevent runaway costs, though diminishing returns emerge (Game of 24: +35% trial 1→2, only +35% trial 2→3)
- **Evaluation quality dependency**: Requires reliable evaluator providing meaningful signals—weak evaluators produce poor reflections leading to no improvement or degradation, and misaligned reward functions cause optimization toward wrong objectives. Verbal feedback quality depends on LLM's diagnostic accuracy (models may generate plausible but incorrect reflections)
- **No cross-task generalization**: Reflections are task-specific ephemeral learning—agent doesn't improve at new tasks unlike fine-tuning which generalizes across problem types, must start from scratch on each query with no transfer learning benefits
- **Only valuable for failure-prone tasks**: ROI exists only when baseline success rate <80%—high-success tasks see minimal benefit from reflection overhead, making pattern overkill for queries that simpler patterns solve correctly in one pass
- **Production deployment challenges**: Long reflection histories consume context window requiring pruning strategies, unknown interaction with large tool sets (our 7 MCP servers), and memory buffer management complexity adds operational overhead

When to use Reflexion

- **Quality-critical applications** where accuracy/completeness outweigh cost (executive reports, compliance docs)
- **High first-attempt failure rate** (<80% success) where reflection enables learning from mistakes
- **Complex multi-dimensional analysis** where missing aspects is common failure mode
- **Latency-tolerant scenarios** like batch processing or overnight report generation

When to avoid Reflexion

- **Cost-constrained applications** where 3-12× cost increase is unacceptable for high-volume queries
- **Real-time requirements** where multiple trial latency is incompatible with user-facing interactions
- **High baseline success rate** (>80%) where marginal benefit doesn't justify cost
- **No quality evaluator** available—requires reliable scoring mechanism for meaningful reflections

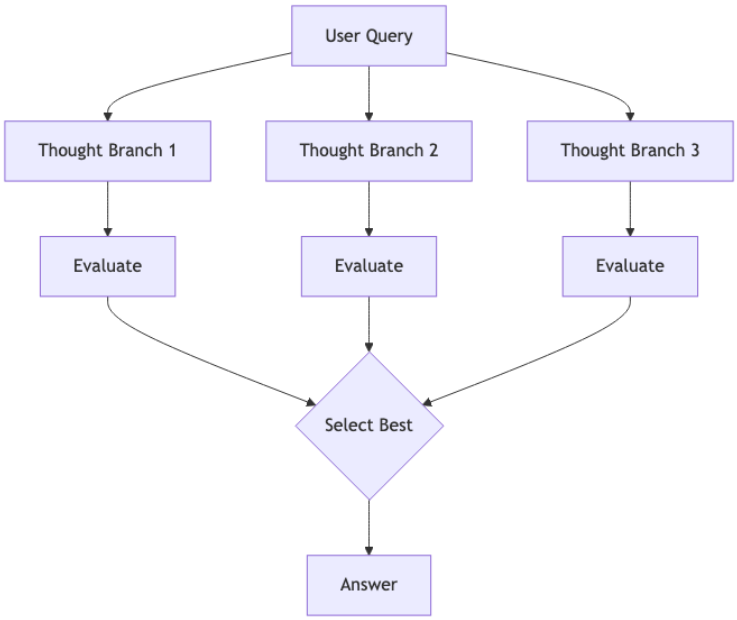
Demo Example

```
Query: "What measures can I take to reduce churn?" (Alpenblick
hospitality group)

TRIAL 1 (ReAct Actor):
1. get_sentiment_statistics() → 38% negative
```

```
2. Return: "Address negative sentiment"
Evaluator: 1.5/5.0 (too vague, no specifics) → FAIL

Self-Reflection:
"Failure: Only checked sentiment, no root cause analysis.
Next trial: Check workload themes + org psych engagement drivers."

TRIAL 2 (with reflection context):
1. get_sentiment_statistics() → 38% negative, "Workload" top (45%)
2. get_org_psych_analysis(theory="engagement") → 2.4/5.0, autonomy
2.1/5.0
3. get_summarized_eda_data(topics=["Workload"]) → 72% understaffing
4. Return: "Measures: (1) Staffing—72% report understaffing
            (2) Autonomy—2.1/5.0 critical (3) Workload—45% cite"
Evaluator: 4.5/5.0 (specific, actionable, complete) → SUCCESS ✅

Cost: 2.5× | Quality: +200% improvement
```

Other Notable Patterns (For Awareness)

- **Tree-of-Thought/Graph-of-Thought**: Explore multiple reasoning branches; pick best via scoring. *Use case: "Generate 3 different churn risk hypotheses" → evaluate each → pick best.*



How to choose: decision framework for org psych analysis

Why Planning Patterns Matter (vs ReAct):

Planning patterns (Plan-then-Execute, ReWOO, LLMCompiler) promise improvements over traditional ReAct-style agents:

1. ⏰ **Speed**: Execute multi-step workflows faster since the large agent doesn't need to be consulted after each action. Sub-tasks can be performed without additional LLM calls (or with calls to lighter-weight LLMs).
2. 💸 **Cost**: Significant cost savings over ReAct. If LLM calls are used for sub-tasks, they can be made to smaller, domain-specific models. The larger model is only called for (re-)planning steps and to generate the final response.
   - **Model tiering**: Use larger models for planning, smaller models for execution → **30-50% cost reduction** without accuracy loss
3. 🏆 **Quality**: Can perform better overall (task completion rate and quality) by forcing the planner to explicitly "think through" all steps required to accomplish the entire task. Generating full reasoning steps is a tried-and-true prompting technique. Subdividing the problem permits more focused task execution.

Pattern Comparison Table

Pattern	Best For	Speed	Cost	Quality	Complexity	Tool Limit
ReAct (current)	Exploratory, unknown complexity, adaptive workflows	Moderate (3-7 steps)	Moderate	Good	Low	**7 MCP servers at limit**
Plan-then-Execute	Well-scoped repeatable workflows, compliance/audit	Fast (fewer LLM calls)	Low (40-50% reduction)	Good-Excellent	Medium	Unknown
ReWOO	Predictable multi-hop, token efficiency critical	Fast (no reasoning loops)	Very Low (80% reduction)	Good-Excellent	Medium	**7 tools at degradation threshold**
LLMCompiler	Speed-critical, parallel workflows, clear dependencies	Fastest (1.8-3.7× speedup)	Very Low (3-6× reduction)	Good-Excellent	High	Unknown
Reflexion	Quality-critical, failure-prone tasks, batch processing	Slowest (2-10 trials)	Very High (3-12× increase)	Excellent	Medium	Unknown

Multi-Agent Supervisor	10 tools, domain specialization needed	Moderate	Moderate	Excellent	High	Specialist: 3-5 tools each

Key insights for our 7 MCP server context:

- **ReAct**: At upper limit (research shows 2% performance with 7+ domains in calendar scheduling)
- **ReWOO**: At degradation threshold (42% → 37% performance with 2 → 7 tools)
- **LLMCompiler**: Unknown tool count sensitivity; research focused on smaller tool sets
- **Multi-Agent Supervisor**: Best option if expanding beyond 7 tools—split into specialists

Updated Decision Matrix

Query Type	ReAct	P-t-E	ReWOO	LLMCompiler	Reflexion
Exploratory ("Find red flags")	✅ Best	❌ Too rigid	⚠️ Might miss edges	⚠️ Hard to plan DAG	⚡ Add for quality
Multi-theory ("Analyze commitment + conflict + satisfaction")	⚠️ Expensive, redundant	❌ Serial slow	✅ Variable assignment	⚡ Best (parallel)	⚡ Add for completeness
Well-scoped ("Q3 churn report")	⚠️ Overkill	✅ Predictable	⚠️ Overkill	⚡ Best (parallel)	⚠️ If first-pass fails

Ambiguous ("Employee satisfaction")	✅ Adapts	❌ Can't plan	⚠️ Might plan wrong	❌ Hard to plan	⚡ Add for quality
Speed-critical ("Real-time dashboard")	⚠️ Variable latency	⚠️ Predictable	✅ Fast	⚡ Fastest	❌ Too slow
Quality-critical ("Executive report")	⚠️ Good enough	✅ Auditable	✅ Good	✅ Good	⚡ Best

Legend: ✅ Recommended | ⚡ Optimal | ⚠️ Works with caveats | ❌ Not recommended

Performance comparison (from LangChain blog):

- ⏰ **Speed**: LLMCompiler > ReWOO ≥ P-t-E > ReAct > Reflexion
- 💸 **Cost**: ReWOO ≥ LLMCompiler > P-t-E > ReAct >>> Reflexion
- 🏆 **Quality**: Reflexion > LLMCompiler ≥ ReWOO ≥ P-t-E ≥ ReAct

Quick decision cues:

- **Ambiguity high, info unknown** → **ReAct** (current) ✅ — adapts during exploration
- **Workflow known, repeatable** → **Plan-then-Execute** (e.g., batch churn reports) — predictable + cost-efficient
- **Complex reasoning with predictable operations** → **ReWOO** (multi-theory queries) — 80% cost reduction, 5× token efficiency
- **Speed critical, clear task dependencies** → **LLMCompiler** (1.8-3.7× speedup via parallel DAG execution)
- **Quality critical, time flexible** → **Reflexion** on top of any pattern — 20-70% success rate improvement at 3-12× cost
- **Exploration and solution diversity needed** → **Tree/Graph-of-Thought** (hypothesis generation)
- **>7 tools, domain specialization** → **Multi-Agent Supervisor** — split into specialists with 3-5 tools each

Emerging Hybrid Approaches

Real-world production systems increasingly **combine multiple patterns** rather than using them in isolation. These hybrid architectures leverage complementary strengths while mitigating individual weaknesses:

1. ReWOO + ReAct Fallback (Graceful Degradation)

Pattern: Start with ReWOO for efficiency; fallback to ReAct on failure

- **Trigger**: If ReWOO plan execution returns empty results or evaluator scores output as low-quality
- **Benefit**: Get 80% cost reduction on successful cases, full adaptability on edge cases
- **Use case**: Predictable org psych queries (95% success with ReWOO) with ReAct handling outliers

- **Implementation**: Wrap ReWOO in try-catch; on failure, invoke ReAct with full context

2. LLMCompiler + Reflexion (Speed + Quality)

Pattern: Use LLMCompiler for fast parallel execution; add Reflexion layer for quality-critical outputs

- **Benefit**: 1.8-3.7× speedup with 20-25% quality improvement on complex analyses
- **Use case**: Executive reports requiring both speed (user-facing) and quality (accuracy-critical)
- **Trade-off**: 1st trial fast (LLMCompiler), 2nd trial expensive (full reflection) but only on failures
- **Implementation**: LLMCompiler as Actor in Reflexion framework; evaluator triggers re-trial if needed

3. Plan-then-Execute with Multi-Agent Workers (Scale + Structure)

Pattern: Planner generates structured plan; route steps to specialized worker agents; replanner coordinates

- **Benefit**: Handles >10 tools by domain specialization while maintaining deterministic workflows
- **Use case**: Comprehensive org analysis requiring Literature Search agent + Data Analysis agent + Report Synthesis agent
- **Tool distribution**:
  - Worker 1 (Literature Agent): 3 tools (EDA, Sentiment, Questions)
  - Worker 2 (Analysis Agent): 3 tools (Org Psych, Answers, Custom Info)
  - Worker 3 (Synthesis Agent): 1 tool (Questionnaire Info) + report formatting
- **Implementation**: Planner identifies which specialist per step; supervisor routes to workers; replanner evaluates

4. Reflexion with Model Diversity (X-MAS Pattern)

Pattern: Each Reflexion trial uses different LLM for ensemble quality

- **Benefit**: 70% vs 23.33% accuracy from heterogeneous models (X-MAS research, 2025)
- **Use case**: Critical analyses where consensus across models increases confidence
- **Trade-off**: 3-5 trials × 3 models = 9-15× cost, but dramatic quality improvement
- **Implementation**: Different frontier model per trial with cross-model reflections for ensemble learning

5. ReWOO + Dynamic Tool Loading (Adaptive Efficiency)

Pattern: ReWOO Planner generates plan; Worker dynamically loads only required MCP servers

- **Benefit**: Mitigates tool selection degradation (42% → 37% with 7 tools) by reducing active tool count per query
- **Use case**: Multi-domain analysis where different queries need different tool subsets
- **Tool loading**: Query about "Engineering churn" loads only [Sentiment, Org Psych, EDA]; "Survey design" loads [Questionnaire Info, Questions, Custom Info]
- **Implementation**: Planner identifies required tools; Worker initializes only subset; Solver synthesizes

6. Hierarchical Planning (Two-Level P-t-E)

Pattern: Strategic Planner creates high-level phases; Tactical Planner details each phase; Executor runs steps

- **Benefit**: Addresses planning rigidity by allowing phase-level replanning without full plan regeneration
- **Use case**: Long-horizon analyses (quarterly reviews, annual reports) with evolving requirements
- **Example flow**:
  - Strategic Plan: [Phase 1: Data Collection] → [Phase 2: Analysis] → [Phase 3: Report Synthesis]
  - Tactical Plan for Phase 1: [Get surveys] → [Extract sentiment] → [Segment by team]
  - After Phase 1: Tactical Replanner adjusts Phase 2 based on Phase 1 outcomes
- **Implementation**: Nested P-t-E agents; strategic replanner decides whether to continue/revise next phase

7. ReAct with Cached Plans (Learning Pattern Library)

Pattern: ReAct agent builds episodic memory of successful reasoning traces; retrieves similar patterns for new queries

- **Benefit**: Combines ReAct's adaptability with P-t-E's efficiency through learned templates
- **Use case**: Recurring query types ("churn risk for X team") that follow similar trajectories
- **Memory structure**: Vector database storing {query_embedding, successful_tool_sequence, outcome_quality}
- **Retrieval**: New query → find top-3 similar past queries → inject their tool sequences as "suggested approach" → ReAct adapts if needed
- **Implementation**: LangChain Memory + vector store; inject retrieved sequences into system prompt

Choosing hybrid approaches:

- **Production maturity critical** → ReWOO + ReAct Fallback (battle-tested components)
- **Budget available, quality paramount** → LLMCompiler + Reflexion (best of both worlds)
- **Tool count >10** → P-t-E + Multi-Agent Workers (specialization at scale)
- **Mission-critical decisions** → Reflexion + Model Diversity (consensus across LLMs)
- **Recurring query patterns** → ReAct + Cached Plans (learn from experience)
- **Long-horizon workflows** → Hierarchical Planning (phase-level adaptation)

References

Core Pattern Papers

- **ReAct**: "ReAct: Synergizing Reasoning and Acting in Language Models" (Yao et al., ICLR 2023)
  - Introduces interleaved reasoning and acting with 27.4% HotpotQA, 71% ALFWorld, 6% hallucination rate
  - Establishes baseline for modern agentic systems with thought-action-observation cycle
- **Plan-and-Solve**: "Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models" (Wang et al., ACL 2023)
  - PS+ prompting technique: 70.4% → 76.7% average accuracy on arithmetic reasoning
  - MultiArith: 83.8% → 91.8%, GSM8K: 56.4% → 59.3%
  - Inspiration for Plan-then-Execute agentic framework
- **ReWOO**: "ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models" (Xu et al., 2023)
  - 5× token efficiency (1,986 vs 9,795 tokens), 80% cost reduction ($3.97 vs $19.59 per 1K queries)
  - Variable substitution (#E1, #E2, #E3) enables foreseeable reasoning without observations
  - HotpotQA: 42.4% vs 40.8% ReAct with dramatic cost savings
- **LLMCompiler**: "An LLM Compiler for Parallel Function Calling" (Kim et al., UC Berkeley, ICML 2024)
  - Up to 3.7× latency speedup, 6.73× cost reduction through DAG-based parallel execution
  - Beats OpenAI parallel function calling by ~35% through instruction pipelining
  - HotpotQA: 1.80× speedup with 3.37× cost reduction
- **Reflexion**: "Reflexion: Language Agents with Verbal Reinforcement Learning" (Shinn et al., NeurIPS 2023)
  - Self-reflection with episodic memory: ALFWorld 97% vs 75% baseline (+22%)
  - Game of 24: 4% → 74% with 3 reflections, HumanEval: 91% vs 80%
  - Verbal feedback without model fine-tuning, 3-12× cost increase

Foundational Techniques

- **Chain-of-Thought**: "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" (Wei et al., NeurIPS 2022)
  - Establishes step-by-step reasoning as core prompting technique
  - Foundation for ReAct's reasoning traces and Plan-and-Solve improvements
- **Tree-of-Thoughts**: "Tree of Thoughts: Deliberate Problem Solving with Large Language Models" (Yao et al., NeurIPS 2023)
  - Explores multiple reasoning branches with backtracking
  - Game of 24: 4% → 74% through search-based reasoning

Multi-Agent & Advanced Architectures

- **Multi-Agent Collaboration**: LangGraph documentation on supervisor patterns
  - Hub-and-spoke topology with specialist agents handling 3-5 tools each
  - 50% performance improvement when tools properly grouped by domain
  - Addresses tool selection overload at 7+ tools
- **X-MAS (Heterogeneous Multi-Agent Systems)**: "X-MAS: Solving Math Word Problems via Cross-Model Augmented Self-Correction" (2025)
  - 70% accuracy with heterogeneous models vs 23.33% homogeneous (+46.67 points)
  - Model diversity through ensemble of different frontier models
  - Validates Reflexion with Model Diversity hybrid approach

Security & Reliability

- **Plan-then-Execute Security**: "Architecting Resilient LLM Agents: A Guide to Secure Plan-then-Execute Implementations" (Del Rosario et al., 2025)
  - Control-flow integrity through planning-execution separation
  - Defense-in-depth strategies: least privilege, sandboxing, human-in-the-loop
  - Resilience to indirect prompt injection attacks

Benchmarking & Evaluation

- **AI Agents That Matter**: (Princeton & Allen Institute for AI, July 2024)
  - Simple baselines often outperform complex architectures when cost-controlled
  - Emphasizes importance of rigorous evaluation and fair comparison
  - Challenges inflated performance claims in agent research
- **τ-bench**: (Sierra AI, 2024)
  - Industry standard for realistic agent evaluation with retail/airline scenarios
  - <50% success rates reveal gap between research benchmarks and production reality
  - Emphasizes need for practical, grounded agent assessment

Implementation Resources

- **LangChain Blog**: "Planning Agents" (Feb 2024)
  - Compares Plan-and-Execute, ReWOO, and LLMCompiler implementations
  - Production insights on speed/cost/quality tradeoffs
  - Model tiering strategies for 40-50% cost reduction
- **LangGraph Tutorials** (Official Documentation):

    - ReAct Agent from Scratch
    - Plan-and-Execute - Full implementation with state tracking
    - ReWOO - Variable substitution examples
    - LLMCompiler - DAG scheduling implementation
- **BabyAGI**: GitHub Repository (Nakajima, 2023)
    - Early autonomous agent with task management and prioritization
    - Inspiration for task decomposition patterns

Evaluation Tooling

- **τ-bench**: (Sierra AI, 2024) - Realistic retail/airline agent evaluation
- **LangSmith**: LangChain's observability and testing platform for agent traces
- **LangFuse**: Open-source LLM observability and monitoring
- **HumanEval/MBPP**: Code generation benchmarks (used for Reflexion evaluation)
- **HotpotQA**: Multi-hop question answering requiring 2-3 Wikipedia passages
- **ALFWorld**: Embodied AI tasks in text-based household environments (134 tasks)
- **WebShop**: E-commerce navigation with 1.18M real products
- **GSM8K**: Grade-school math word problems (2-8 reasoning steps)

Production Insights

- **LangSmith Production Trends** (End of 2024):
    - LangGraph adoption: 43% of organizations
    - ReAct pattern: 39.8% of production implementations
    - Top concern: Quality/performance (45.8%), Cost second (22.4%)
- **Tool Selection Research** (2024):
    - Performance degrades with 10+ tools even with capable models
    - Calendar scheduling: 2% success with 7+ domains (GPT-4o)
    - Best practice: 5-10 tools per agent, multi-agent for larger tool sets