Reasoning and Tool Use

Luke Zettlemoyer

Slides from Robert Minneker

Part 1: Chain-of-Thought and Structured Reasoning

State Change: Beyond Direct Answers

Standard prompting asks for a final answer. Chain-of-thought prompting elicits intermediate reasoning steps, dramatically improving performance on complex tasks.

Chain-of-thought prompting decomposes complex tasks into explicit intermediate steps

CoT biases the model toward generating a *reasoning trace* rather than a single direct answer:

Example: Arithmetic Word Problem

Prompt:

```
"If Alice has 3 apples and buys 2 more,
how many does she have? Let's think
step by step."
```

→

LLM Output:

```
"Alice starts with 3. She buys 2 more.
3 + 2 = 5. Alice has 5 apples."
```

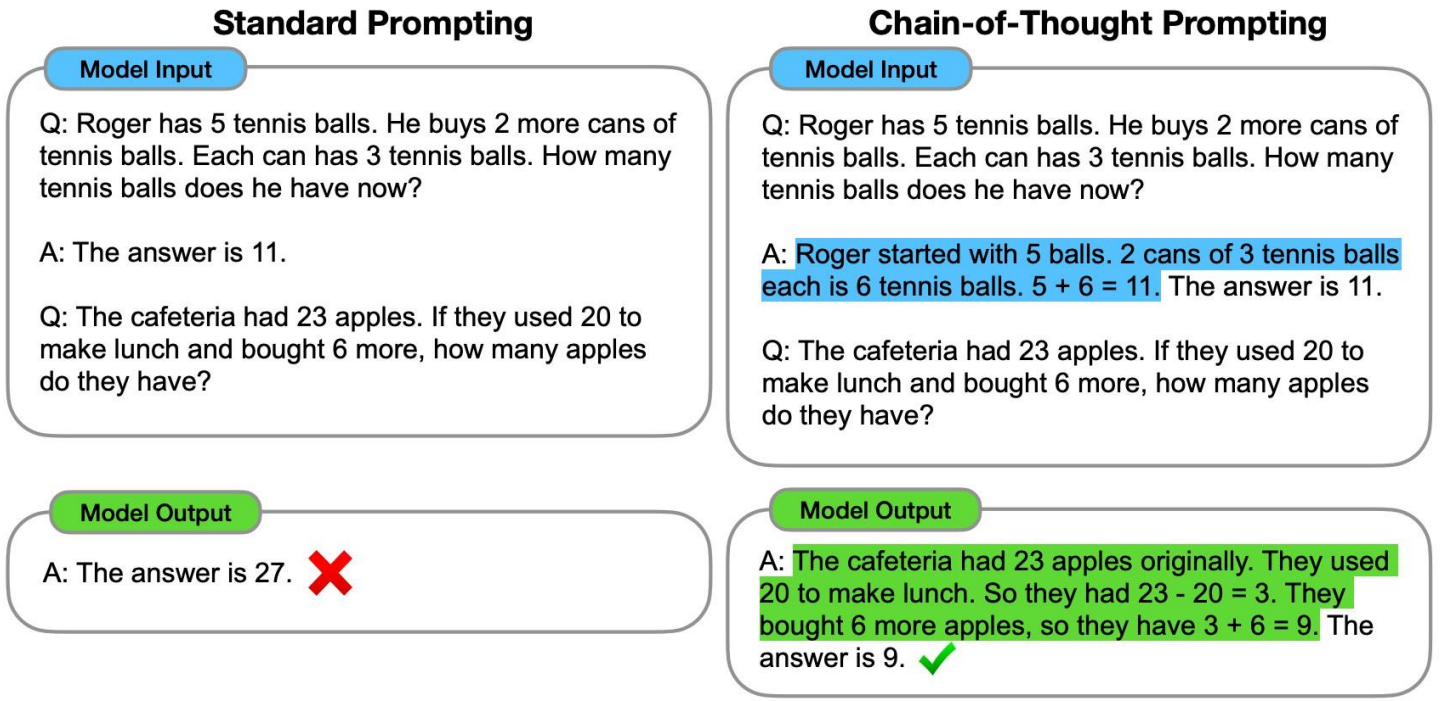


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Standard vs. chain-of-thought prompting from Wei et al. (2022)

- The model breaks complex problems into tractable subproblems, avoiding shallow heuristics
- CoT is most effective for tasks requiring multi-step logic: arithmetic, symbolic reasoning, multi-hop QA

Zero-shot CoT adds a trigger phrase while few-shot CoT provides worked examples

Zero-shot CoT

Append *"Let's think step by step."* to the prompt

Leverages the model's learned prior for multi-step explanations

Few-shot CoT

Include exemplars with full reasoning traces

Provides direct inductive bias via demonstration

- On GSM8K, GPT-3 accuracy rises from below 20% (direct) to over 50% with zero-shot CoT
- Even a single trigger phrase dramatically changes model behavior due to learned priors

Adding "Let's think step by step" produces dramatic accuracy gains across reasoning tasks (Kojima et al., 2022)

	Arithmetic					
	SingleEq	AddSub	MultiArith	GSM8K	AQUA	SVAMP
zero-shot	74.6/**78.7**	**72.2/77.0**	17.7/22.7	10.4/12.5	22.4/22.4	58.8/58.7
zero-shot-cot	**78.0/78.7**	69.6/74.7	**78.7/79.3**	**40.7/40.5**	**33.5/31.9**	**62.1/63.7**
	Common Sense		Other Reasoning Tasks		Symbolic Reasoning	
	Common SenseQA	Strategy QA	Date Understand	Shuffled Objects	Last Letter (4 words)	Coin Flip (4 times)
zero-shot	**68.8/72.6**	12.7/**54.3**	49.3/33.6	31.3/29.7	0.2/-	12.8/53.8
zero-shot-cot	64.6/64.0	**54.8**/52.3	**67.5/61.8**	**52.4/52.9**	**57.6**/-	**91.4/87.8**

- A single phrase appended to the prompt consistently doubles accuracy on math reasoning benchmarks
- The effect is most pronounced on tasks requiring multi-step arithmetic

Where would this help?

CoT Helps

✓ Multi-step arithmetic
✓ Multi-hop reasoning
✓ Logical deduction
✓ Complex triage decisions

CoT Doesn't Help (or Hurts)

✗ Simple factual recall
✗ Single-label classification
✗ Pattern matching / extraction
✗ Tasks where the answer is obvious

⚠ **GPT-4 and the Bar Exam**

GPT-4 scored in the 90th percentile on the Bar Exam (vs. GPT-3.5's 10th percentile). **Takeaway:** Benchmarks can overstate reasoning ability — multiple-choice format may not reflect true open-ended legal reasoning.

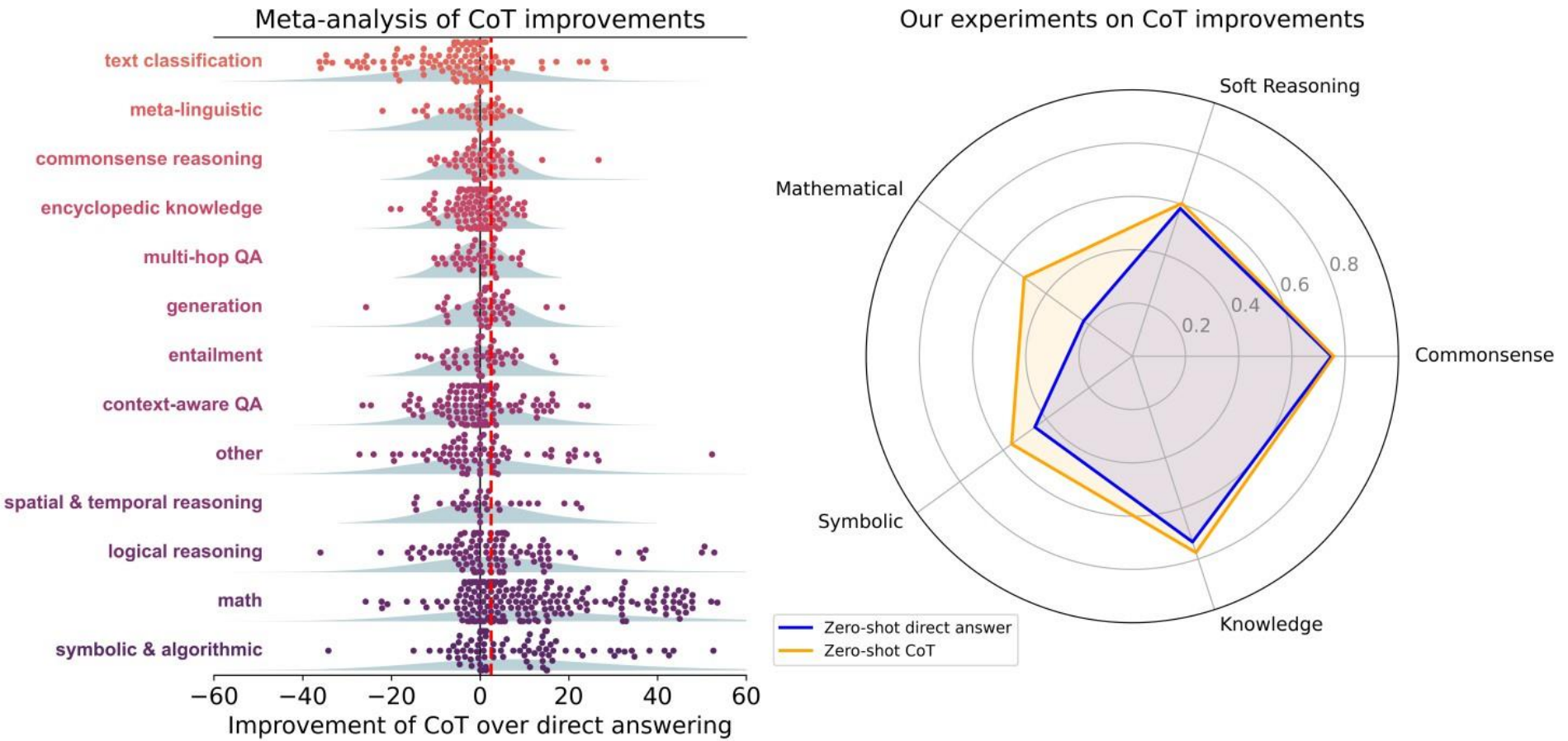


Figure 1: Left: meta-analysis of CoT literature; each point is a reported delta of CoT over direct answering for some (LLM, task) pair. Right: average performance of using zero-shot CoT v.s. direct answer prompts across five general reasoning categories, covering 20 datasets with 14 LLMs evaluated on each. In both sets of results, math and other kinds of symbolic reasoning are the domains that consistently see substantial improvements from CoT (red dotted line indicates the mean improvement from CoT across experiments).

What can we do to get better reasoning chains?

Self-Consistency

Sample N chains, take majority vote

$\hat{y} = \text{mode}\{f(C_i)\}$

Tree-of-Thought

Branch-and-explore reasoning paths with backtracking

BFS over reasoning states

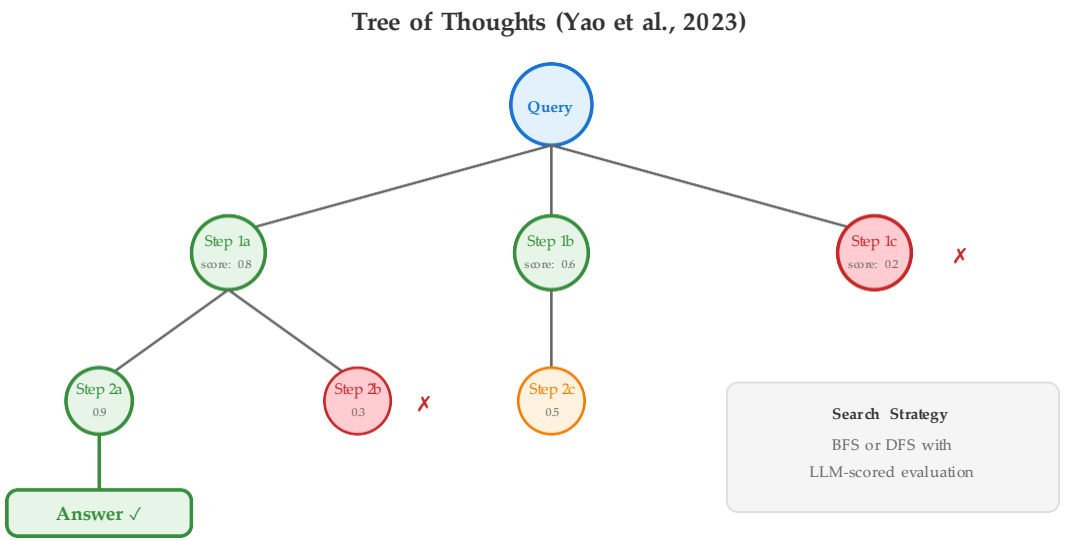
Least-to-Most

Decompose into ordered sub-questions, solve bottom-up

Hierarchical decomposition

- Self-consistency reduces stochasticity by aggregating over multiple reasoning chains
- These methods trade compute for accuracy — more reasoning paths yield more reliable answers

Tree-of-Thought explores multiple reasoning branches and backtracks from dead ends



- Unlike linear CoT, ToT can backtrack from unpromising paths and explore alternatives
- The LLM serves as both the generator (proposing steps) and the evaluator (scoring them)

Choosing a reasoning strategy involves compute-accuracy tradeoffs

Strategy	Compute Cost	Robustness	Best For
Basic CoT	1×	Low	Simple multi-step problems
Self-Consistency	N×	High	When you need reliable answers (triage!)
Tree-of-Thought	N×M×	Highest	Creative/open-ended exploration
Least-to-Most	K×	Medium	Hierarchical decomposition

Concept Check

Why does self-consistency improve over a single chain-of-thought? Under what conditions might it fail? Think about the relationship between answer diversity and aggregation quality.

Discussion (2 min)

Consider a multi-hop science question: "What element has the highest electronegativity, and what compounds does it commonly form?" Would you use basic CoT, self-consistency, or least-to-most prompting? Why?

Hallmark Discoveries in Prompting and Reasoning

In-Context Learning as a Scale-Era Behavior

Large language models can perform new tasks from a few demonstrations in the prompt — no gradient updates required. This emergent capability fundamentally changed how practitioners interact with models.

Chain-of-Thought as Eliciting Latent Computation Traces

Wei et al. (2022) showed that prompting models to "show their work" unlocks reasoning capabilities already present in the weights — intermediate tokens serve as a scratchpad that transforms single-step prediction into multi-step computation.

Zero-Shot CoT Triggers as Steering Priors

Kojima et al. (2022) demonstrated that a single phrase — "Let's think step by step" — activates latent reasoning without any demonstrations, revealing that models encode reasoning priors that can be unlocked through simple prompt design.

Part 2: Agentic Workflows

State Change: From Answering to Acting

Prompting strategies so far produce text. Agentic workflows enable LLMs to reason, plan, and invoke external tools to complete tasks in the real world.

The ReAct framework interleaves reasoning traces with tool-calling actions

LLMs alternate between generating intermediate thoughts and invoking external tools:

$$\pi : (x, h) \rightarrow (r, a)$$

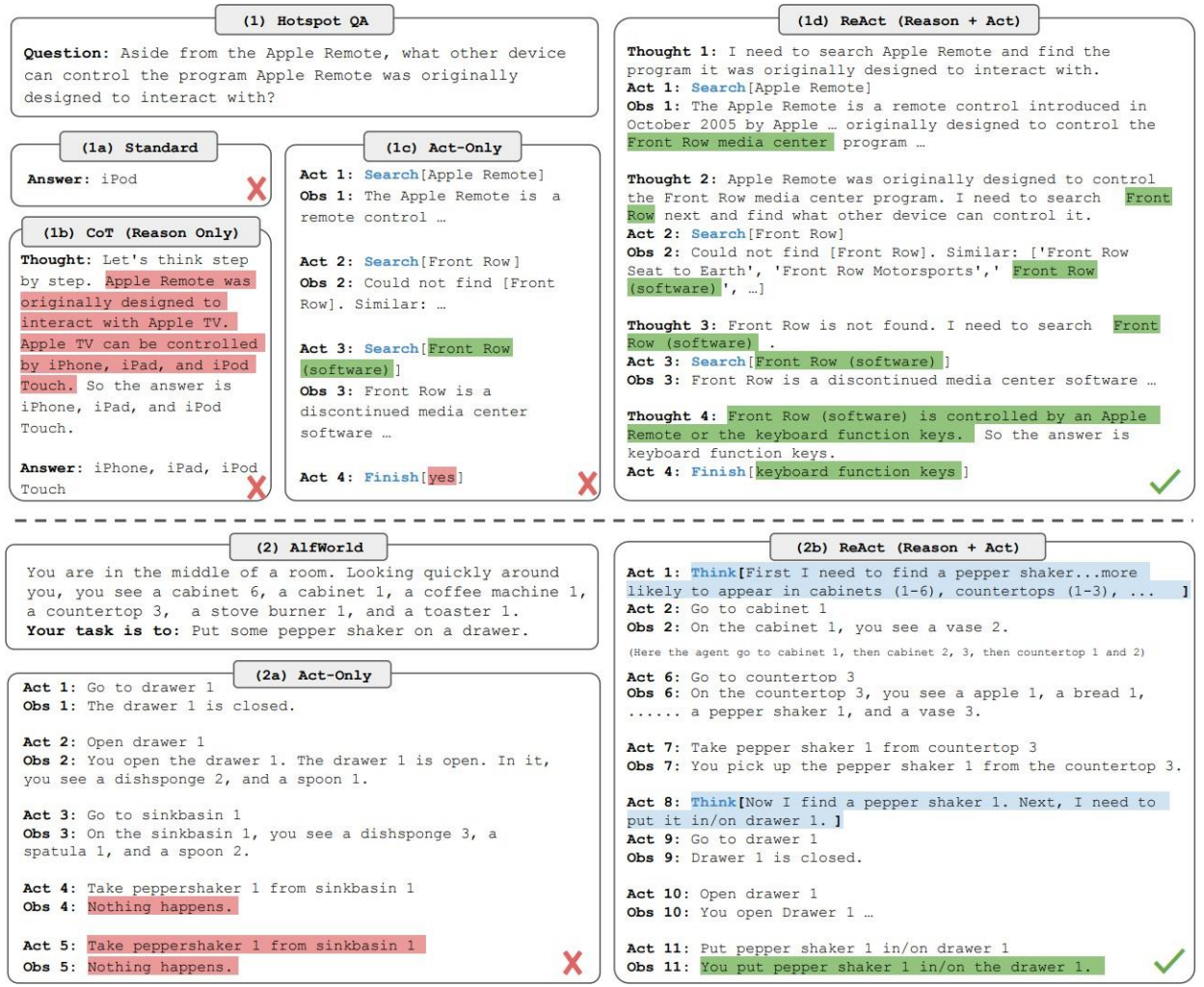
where r is a reasoning step, a is an action (API/tool call), and h is the interaction history.

ReAct Loop Example

Reason `"To answer, I need the current weather in Paris."`

Act `weather_api("Paris") → {"temp": "12°C", "condition": "cloudy"}`

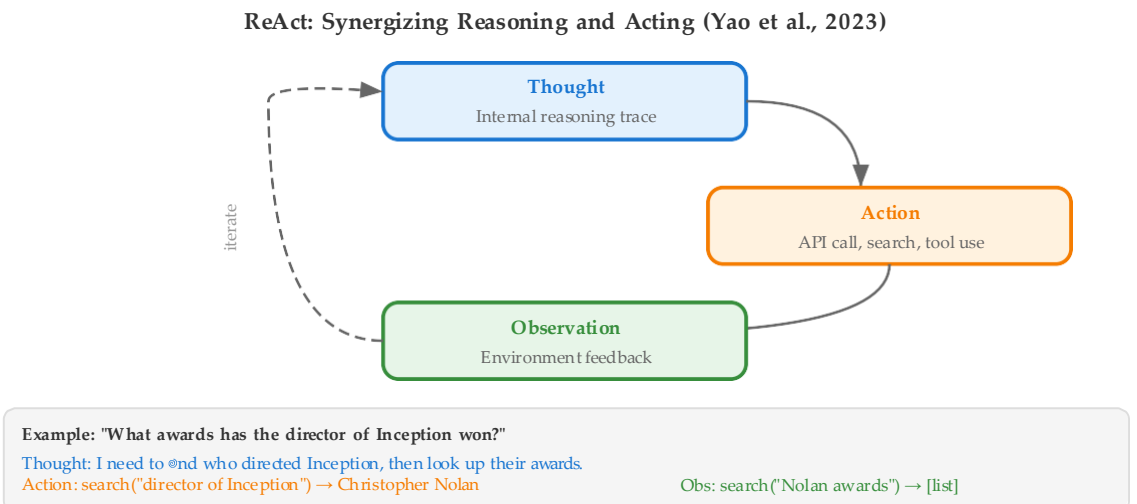
Answer `"It is currently 12°C and cloudy in Paris."`



ReAct framework comparison from Yao et al. (2023): reasoning-only vs. action-only vs. ReAct

- Tool calls are generated as output tokens, extending LLM capacity beyond language modeling

ReAct grounds each reasoning step in observable actions and environmental feedback



- ReAct agents outperform pure reasoning (CoT) and pure acting (tool-only) approaches by combining both

Agents combine planning, tool use, and reflection

- **Planning:** Decompose goals into sub-tasks with explicit success criteria
- **Tool use:** Invoke APIs, run code, do search, query databases
- **Observe-act loop:** Agents iterate on environment feedback (ReAct pattern)

What are the key performance challenges?

For a pipeline of n steps where each step has error probability p_i:

$$P_{\text{failure}} = 1 - \prod_{i=1}^{n}(1 - p_i)$$

Compounding Errors

Early mistakes cascade through all downstream steps

Hallucination

Plausible but incorrect facts or invalid reasoning steps

Prompt Injection

Adversarial inputs override system instructions and safety guardrails

Concept Check

If each step in a 5-step agentic pipeline has a 10% error rate, what is the overall failure probability? What does this imply about designing agentic systems?

Confidence thresholds and fallback

The router must know when *not* to use a tool — wrong tool is worse than no tool:

```
def route_query(query, tools, threshold=0.7):
    scores = router.score(query, tools)          # {tool_name: confidence}
    best_tool = max(scores, key=scores.get)
    if scores[best_tool] < threshold:
        return "no_tool"    # fallback: answer from parametric knowledge
    return best_tool
```

- If confidence is below threshold → **don't use a tool** (fall back to parametric knowledge) If
- two tools score similarly → **ask user to clarify** or call both and compare

Calibration matters: Uncalibrated confidence scores give false safety. A router that outputs 0.95 for every query won't catch misroutes. Calibrate on held-out tool-selection data so thresholds are meaningful.

Failure taxonomy: retrieval and tool-selection failures

Most failures occur at **system boundaries**: Router ↔ Tool, Retriever ↔ Generator, Generator ↔ Verifier.

Failure type	Symptom	First diagnostic
Retrieval miss	Wrong answer, no relevant docs in context	Check Recall@k on gold query set
Wrong tool selection	Used calculator when should have searched	Log router decisions; check tool selection accuracy
Bad arguments	Tool call fails or returns garbage	Schema validation rejection rate

Part 4: Learning Tool Use

How tool use is learned: a spectrum

1. **No learning** — tools hardcoded in pipeline (classical RAG)
2. **In-context learning** — tool descriptions in prompt, LLM selects via few-shot (ReAct, function calling)
3. **Finetuned on API docs** — model trained on (query, API call) pairs (Gorilla, Patil et al. 2023)
4. **Self-supervised** — model discovers when tools help via perplexity reduction (Toolformer)

Key distinction: Approaches 1–2 are **inference-time** solutions (no training changes). Approaches 3–4 require **training-time** investment but produce models that intrinsically know *when* to use tools.

This spectrum shows how autonomy can be shifted from inference-time control to training-time internalization.

Adaptive retrieval and self-supervised tool use

The LLM decides *when* to retrieve:

- **Self-RAG** (Asai et al., 2023): The model emits **reflection tokens** — `[Retrieve]`, `[IsRel]`, `[IsSup]`, `[IsUse]` — trained via distillation from GPT-4 judgments. The model learns to self-assess at inference time without an external verifier.
- **Corrective RAG** (Yan et al., 2024): A lightweight evaluator scores each retrieved document; if confidence is low, the system triggers web search as a fallback — the retriever corrects itself mid-pipeline.

Toolformer insight (Schick et al., 2023): LLMs can learn to use tools **without explicit tool-use training data.** Toolformer inserts candidate API calls into text, keeps only those that reduce perplexity on the next token, and finetunes on the augmented data. The model learns *when* a tool call would help — not from human demonstrations, but from the self-supervised signal of "did this tool call make my prediction better?"

Discussion (2 min)

When should the agent stop tool use and answer? Propose:

1. One explicit **stopping rule** (e.g., "stop after N steps," "stop when confidence > threshold," "stop when no new information gained")
2. One **failure tradeoff** of your stopping rule (e.g., "stopping too early = incomplete answer; stopping too late = wasted compute / compounding errors")

AutoGPT: the autonomy experiment and what it taught us

AutoGPT / BabyAGI (2023) — the first widely-deployed fully autonomous agents:

- **Architecture:** LLM generates a multi-step plan → executes each step via tools → loops until "done" — no human in the loop
- **BabyAGI's decomposition:** task creation agent + prioritization agent + execution agent — the first viral multi-agent system

Why it mostly fails in practice

- **No verification** → errors compound silently (the 0.9^n problem)
- **No stopping criterion** → loops diverge or cycle
- **No approval gates** → side-effectful actions execute unchecked
- **Plans generated all-at-once** → no adaptation to intermediate observations (unlike ReAct)

Community benchmarking: AutoGPT completed **<30%** of multi-step tasks end-to-end, despite each individual step being ~85% accurate — compounding error in action.

AutoGPT is what happens when you have **planning + execution but no verification**. The P/E/V pattern we'll see next adds the missing piece.

Four paradigms for tool-using LLMs

These paradigms differ mainly in who controls tool use and how verification is handled:

	RAG	ReAct	Toolformer	AutoGPT / BabyAGI
Who selects tools	Hardcoded (always retrieval)	LLM per-step via prompt	LLM learned via self-supervision	LLM + autonomous planner
Loop control	Single-shot, no loop	LLM-driven thought-action loop	Inline during generation	External orchestrator + LLM
Planning horizon	None (one step)	Reactive (next step only)	None (token-level)	Full plan upfront, then execute
Human oversight	N/A (read-only)	Optional	None	None (by design)
Key paper	Lewis et al. 2020	Yao et al. 2023	Schick et al. 2023	Significant Gravitas 2023

The autonomy spectrum: RAG (no autonomy) → ReAct (step-by-step autonomy) → AutoGPT (full autonomy). More autonomy = more capability but exponentially more failure modes. All paradigms are attempts to manage compounding error while increasing autonomy.