Analysis Report

Sanyog Mishra

March 2, 2025

PAPERS STUDIED:

1. ReAct: Synergizing Reasoning and Acting in Language Models
2. ReST meets ReAct: Self-Improvement for Multi-Step Reasoning LLM Agent
3. Toolformer: Language Models Can Teach Themselves to Use Tools
4. Chain of Tools: Large Language Model is an Automatic Multi-tool Learner
5. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models

1 Conceptual Map

The conceptual map illustrates how Language Models (LMs) reason and use external tools across the five papers (ReAct (Yao et al., 2023), ReST meets ReAct (Aksitov et al.), Toolformer (Schick et al.), Chain of Tools (Shi et al.), and Language Agent Tree Search (LATS) (Zhou et al., 2024)), their core workflows and interaction with external environment.

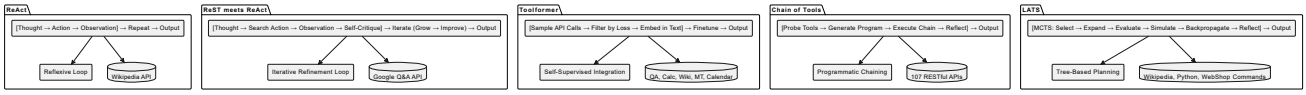

Figure 1: Representations of all five approaches

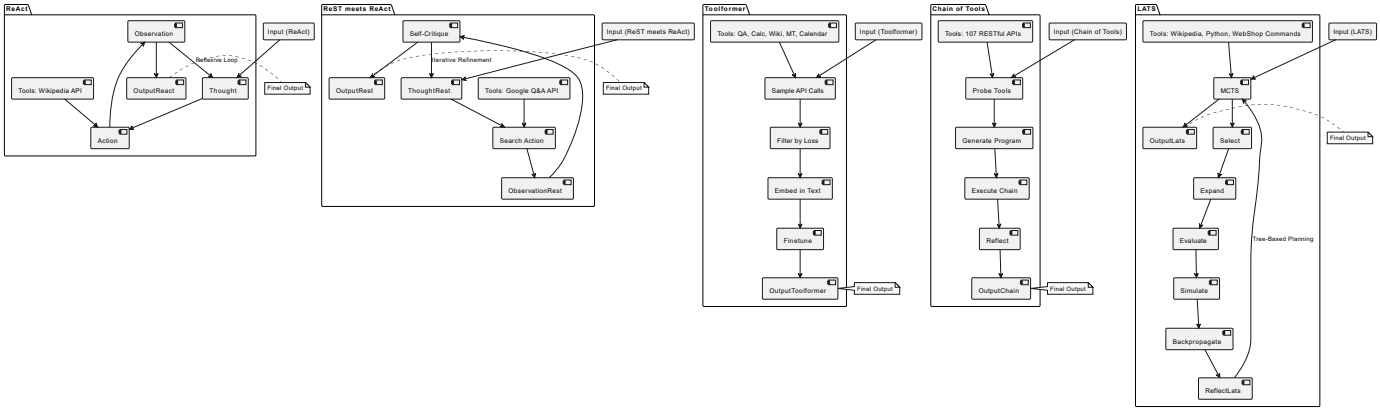

Figure 2: Illustrative workflow demonstrations of all five approaches

1.1 Explanation

1.1.1 ReAct

Workflow: A linear, reflexive cycle where the LLM generates Thoughts (reasoning steps), executes Actions (e.g., search[entity]), and receives observations from an external environment (Wikipedia API). This repeats sequentially until an output is produced.
Interaction: Direct, one-step tool calls with immediate feedback looped back into the context.

1.1.2 ReST meets ReAct

Workflow: Extends ReAct's loop with a state machine: Thought → Search Action → Observation → Self-Critique (relevance, grounding checks). Iterates via ReST's grow (trajectory sampling) and improve (AI ranking) phases, refining the agent over multiple cycles.
Interaction: Structured search loop with Google Q&A API, capped at 10 searches, producing long-form answers.

1.1.3 Toolformer

Workflow: A two-phase process:
(1) Pretraining—samples API calls, filters by loss reduction, augments text;
(2) Inference—embeds calls (e.g., [QA(question)]) into generation. No explicit reasoning trace; tool use is implicit in prediction.
Interaction: Multiple tools (QA, Calculator, Wiki Search, MT, Calendar) are called autonomously during decoding, triggered by special tokens.

1.1.4 Chain of Tools (ATC)

Workflow: Generates a Python program chaining tools, executes it, and reflects on errors. Black-box probing pre-learns tool protocols, enabling dynamic toolset expansion.
Interaction: Executes multi-tool sequences (e.g., Tool A → Tool B) within a single program, using 107 RESTful APIs with data flow dependencies.

1.1.5 LATS

Workflow: Uses Monte Carlo Tree Search (MCTS) with six operations: Select (UCT), Expand (sample actions), Evaluate (LM + self-consistency), Simulate (to terminal state), Backpropagate (update values), Reflect (on failures). Builds a tree of trajectories.
Interaction: Task-specific tools (Wikipedia, Python, WebShop commands) are explored combinatorially, with feedback shaping the search.

1.2 Connections Across Papers

ReAct ↔ ReST meets ReAct: ReAct is foundation for others, emphasizing external feedback integration. ReST enhances ReAct's loop with iteration and self-critique, both relying on external feedback but differing in refinement scope.
ReAct ↔ LATS: LATS expands ReAct's reflexiveness into a tree-based search, sharing tool-use reliance but adding planning.
Toolformer ↔ Chain of Tools: Both aim for autonomous tool use; Toolformer embeds tools implicitly, while Chain of Tools programs them explicitly with probing.
Toolformer ↔ LATS: Toolformer's self-supervision contrasts LATS's explicit planning, though both leverage external tools.

Chain of Tools ↔ LATS: Both handle multi-tool scenarios, but Chain of Tools uses static programs, while LATS explores dynamically via MCTS.

2 Analysis

This analysis compares agent design, reasoning steps, and tool use across the five papers.

2.1 Summary of Each Approach

2.1.1 ReAct

Design: It is a framework where LMs interleave reasoning (thoughts) and actions with external feedback for solving tasks reflexively. Uses a prompt-based approach with interleaved thought-action-observation steps, relying on few-shot examples, i.e., a simple, reflexive loop of thought-action-observation steps, driven by few-shot prompts (e.g., 1-2 examples). Tested on PaLM-540B.
Reasoning: Multi-step verbal reasoning as natural language (e.g., "I need to search X").
Actions: Single-step actions (e.g., search[entity]) that interact with external environment, and receive observations to update context. Actions are chosen greedily based on current context. Actions are executed sequentially.
Tool Use: Predefined APIs, primarily Wikipedia. Executed sequentially without planning.
Methodology: Reflexive, exploits LM's reasoning with external feedback for grounding.

2.1.2 ReST meets ReAct

Design: Extends ReAct by adding iterative self-improvement through a ReST-like (Reinforced Self-Training) process, refining a Search Agent with AI feedback. It uses a state machine with self-critique steps and Python code format prompts for reasoning and action. Tested on PaLM 2 (XS/S/L).
Reasoning: Enhanced multi-step reasoning through interleaved thought-action loops, along with self-critique (relevance, grounding checks) to refine answers.
Actions: Actions via web search (Google Q&A API), giving long-form, attributable answers, with a fixed search loop (up to 10 searches).
Learning: ReST's grow-improve loop, grows trajectories by sampling, improves via AI-ranked filtering, iteratively finetuning the model (2 iterations).
Tool Use: Google Q&A API, returning top 3 snippets within a 10 search loop.
Methodology: Iterative self-improvement via AI feedback, focusing on long-form QA.

2.1.3 Toolformer

Design: A GPT-J (6.7B) model finetuned self-supervisedly to embed API calls, using CCNet data. No task-specific prompts. Decides tool use dynamically, inference uses modified greedy decoding (k=10) to trigger API calls.
Reasoning: No explicit reasoning trace. Implicit in token prediction; tools are called to resolve factual or computational gaps (e.g., facts, math) without explicit traces.
Actions: Embeds API calls (e.g., [QA(question)]) directly into text generation. Executed during inference when triggered by special tokens.
Learning: Self-supervised finetuning: samples API calls via in-context learning, filters by loss reduction, and finetunes on useful calls, preserving generality.
Tool Use: Multiple tools like QA (Atlas), Calculator, Wiki Search (BM25), Machine Translation (NLLB), Calendar. These tools address LM weaknesses (facts, math, time, language). Called autonomously during decoding.

Methodology: Self-supervised sampling and filtering by loss reduction, preserving LM generality.

2.1.4 Chain of Tools (ATC)

Design: A programmatic framework generating Python code to chain multiple tools sequentially, with attributable reflection correcting runtime errors by pinpointing faulty calls. Black-box probing discovers tool protocols via instance generation and schema extraction. Chain probing for interdependent tools. Tested on GPT-3.5/GPT-4.
Reasoning: Programmatic, planning tool sequences with data flow dependencies.
Actions: Executes multi-tool chains in a single program (e.g., tool A $\rightarrow$ tool B), and handles complex workflows with interdependent tool calls.
Learning: Black-box probing learns tool protocols (input-output schemas) via self-generated instances; no iterative training, reflection corrects runtime errors.
Tool Use: 107 RESTful APIs (ToolFlow), executed in a single program, learned via probing.
Methodology: Structured chaining with active tool learning. Suitable for complex workflows.

2.1.5 LATS

Design: A Monte Carlo Tree Search-based (MCTS) agent integrating reasoning, acting, and planning. Tested on GPT-3.5/GPT-4. Uses external feedback and self-reflection to explore and refine trajectories deliberately. Adapts ReAct into a combinatorial search space.
Reasoning: Multi-step, tree-structured reasoning evaluated by an LM-powered value function.
Actions: Executes multi-step actions (e.g., Wikipedia searches, Python solutions) within an MCTS-guided trajectory, informed by environmental observations.
Planning: MCTS planning with six operations (selection, expansion, evaluation, simulation, backpropagation, reflection), to balance exploration and exploitation.
Tool Use: Task-specific (Wikipedia, Python, WebShop). Explored combinatorially via MCTS.
Methodology: Deliberate planning with feedback and reflection, balancing exploration and exploitation.

2.2 Comparative Analysis

2.2.1 Agent Design

ReAct: Simplest, a linear loop with minimal structure, relying on prompts.
ReST meets ReAct: Adds complexity with a state machine and iteration, improving robustness but still linear in action scope.
Toolformer: Minimalist, embedding tools into the LM's core prediction process, maximizing autonomy but lacks explicit structure.
Chain of Tools: Structured, using Python programs for multi-tool workflows, with probing adding adaptability.
LATS: Most complex, a tree-based search with six operations, offering deliberate exploration and adaptability.

2.2.2 Reasoning Steps

ReAct: Explicit verbal steps (thought $\rightarrow$ action), sequential and reflexive, exposed to error propagation.
ReST meets ReAct: Explicit steps, but augmented by self-critique, iteratively refined for accuracy.
Toolformer: Implicit, no distinct steps; reasoning comes from tool-augmented predictions.
Chain of Tools: Programmatic, reasoning encoded in code logic, executed as a single plan.
LATS: Explicit and multi-step within a tree, evaluated and refined via MCTS, reduced error propagation.

2.2.3 Tool Use

ReAct: Single-tool (Wikipedia), predefined, executed step-by-step.
ReST meets ReAct: Single-tool (Google Q&A), iteratively used, capped at 10 calls.
Toolformer: Multi-tool, autonomously selected, embedded in text generation.
Chain of Tools: Multi-tool, chained programmatically, scaled via probing.
LATS: Task-specific, multi-step, explored via tree search, specific to domain needs.

2.2.4 Contrast of Methodologies

Reflexive vs. Iterative vs. Planning: ReAct uses a reflexive loop whereas, ReST is iterative refinement and LATS employs MCTS planning. Toolformer's uses self-supervised embedding and Chain of Tools is about programmatic chaining.
Self-Supervision vs. Prompting: Toolformer and Chain of Tools lean on self-supervision (loss filtering, probing), while ReAct and ReST rely on prompting (few-shot, code-based) and LATS blends prompting with planning.
Tool Scope: ReAct and ReST focus on single tools, Toolformer and Chain of Tools expand to multiple tools, and LATS adapts tools per task.

2.3 Real-World Applicability

ReAct: Applicable for quick, interactive tasks (e.g., general purpose QA bots).
ReST meets ReAct: Suited for educational or research QA systems requiring long-form, refined answers.
Toolformer: Ideal for versatile, zero-shot applications (e.g., chatbots, content generation), though non-interactive tools limit dynamic use cases.
Chain of Tools: Fits real-world automation (e.g., e-commerce, logistics) with multi-tool needs, but requires robust error handling.
LATS: Best for autonomous agents in hybrid domains (e.g., programming assistants, web navigation), but high computational cost may limit applications.

3 Open Questions

3.1 Deployment Challenges

3.1.1 Scalability

ReAct: Scaling to multiple tools or larger tasks requires extensive prompt changes.
ReST meets ReAct: Iterative training scales badly with more tools or complex tasks.
Toolformer: Sample inefficiency (e.g., few calculator calls from millions) causes issues in large-scale tool integration.
Chain of Tools: Probing 107+ tools is computationally intensive.
LATS: High token cost (173,290 HotpotQA) restricts scalability in scarce resource settings.

3.1.2 Adaptability

ReAct: Fixed prompts limit adaptation to new domains or tools.
ReST meets ReAct: Single-tool focus limits adaptability to different tasks.
Toolformer: Prompt sensitivity affects tool use consistency.
Chain of Tools: Static programs lack runtime adaptability to new tools or contexts.
LATS: Reversion assumption limits use in non-revertible environments (e.g., live systems, robots in irreversible environments).

3.1.3 Errors

ReAct: Reasoning errors (47%) and hallucination (6%) persist without recovery mechanisms.
ReST meets ReAct: Stochasticity increases variance, discouraging consistent error correction.
Toolformer: Non-interactive tools miss error feedback loops.
Chain of Tools: False successes (wrong outputs) go undetected beyond syntax checking.
LATS: Generic reflections reduce error correction performance.

3.1.4 Tool Integration

ReAct: Limited to predefined APIs, lacking multi-tool involvement.
ReST meets ReAct: Single-tool design hinders broader tool integration.
Toolformer: No chaining restricts integration of interdependent tools.
Chain of Tools: Probing integrates new tools, but manual prompt design persists.
LATS: Task-specific tools limit general integration frameworks.

3.2 Proposed Improvements and Future Research

3.2.1 Scalability

Improvement: For ReAct and ReST, we could develop modular prompt libraries for multi-tool scaling. Toolformer could use iterative sampling to boost efficiency. Chain of Tools and LATS could be improved with use of pruning (e.g., to limit MCTS depth).
Research: Experimenting lightweight planning (e.g., A* for LATS) and dynamic tool sampling methods.

3.2.2 Adaptability

Improvement: ReAct could be improved by adopting dynamic prompt generation. ReST might extend to multi-tool ReST loops. Chain of Tools could utilise runtime program adaptation. LATS could be improved by relaxing reversion via memory-based state tracking.
Research: Exploring meta-learning for domain adaptation or multimodal tool integration (e.g., image + text).

3.2.3 Errors

Improvement: ReAct could improve from integrating error backtracking. Toolformer might add interactive feedback loops. Chain of Tools could validate outputs semantically.
Research: Development of robust error detection (e.g., semantic validators) or self-correcting mechanisms across frameworks.

3.2.4 Integration

Improvement: ReAct and ReST could adopt Chain of Tools' chaining or Toolformer's multi-tool embedding. Toolformer could support chaining via sequential finetuning. Chain of Tools might automate prompt design. LATS could improve by generalizing tool interfaces.
Research: Exploiting unified tool APIs or hybrid frameworks combining planning (LATS) with autonomy (Toolformer).