MATH 289 · AI for Mathematical Research

Agents

Sentence-completion machine → Research assistant



Shurui Liu

Fall 2026



Plan for today

0	**Overview**	pipeline	10 min
1	**Tools**	why and how	16 min
2	**Memory**	why and how	15 min
3	**Prompt & context**	why and how	15 min
4	**Orchestration**	multi-agents	16 min
	Wrap-up	and a design exercise	8 min

Plan for today

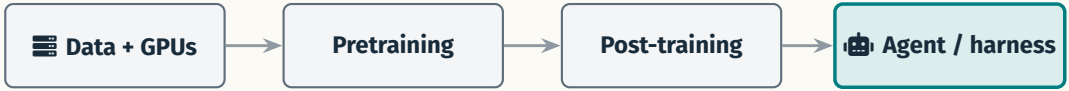
0	**Overview**	pipeline	10 min
1	**Tools**	why and how	16 min
2	**Memory**	why and how	15 min
3	**Prompt & context**	why and how	15 min
4	**Orchestration**	multi-agents	16 min
	Wrap-up	and a design exercise	8 min

> Agent = LLM (sentence-completion machine) + harness (sentence-completion→task-resolving)

The pipeline of modern AI

previous lectures: deep learning, LLM architecture, reasoning & RL

today



text, code, math…

Minimize next-token cross-entropy.

⇒ a *base model* that continues a sentence.

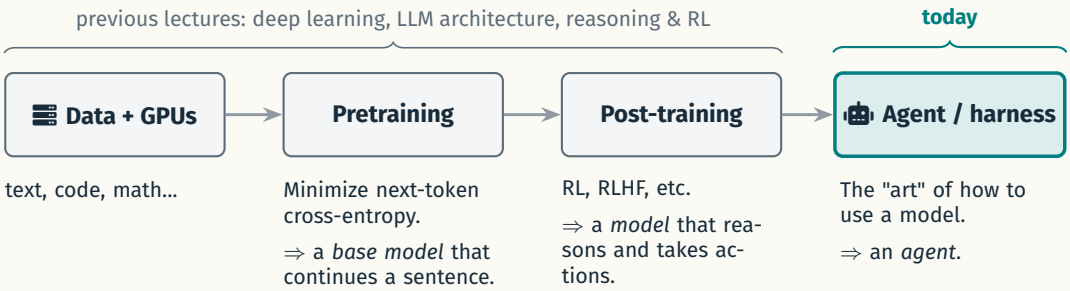
RL, RLHF, etc.

⇒ a *model* that reasons and takes actions.

The "art" of how to use a model.

⇒ an *agent*.

The pipeline of modern AI



Everything today happens **at inference time**. We do not update the weights θ.

What we have after training: a machine that completes text

What we have after training: a machine that completes text

- Recall **definition** from previous lecture: an **LLM** is a function

$$p_\theta : \{[t_1, \ldots, t_n] : t_i \in \mathcal{V}\} \to \mathbb{P}(\mathcal{V})$$

mapping a sequence of input tokens to a probability distribution over all possible tokens, where θ is the parameters controlling **linear part** of this function.

What we have after training: a machine that completes text

- Recall **definition** from previous lecture: an **LLM** is a function

$$p_\theta : \{[t_1, \ldots, t_n] : t_i \in \mathcal{V}\} \to \mathbb{P}(\mathcal{V})$$

  mapping a sequence of input tokens to a probability distribution over all possible tokens, where θ is the parameters controlling **linear part** of this function.
- **Generation**: sample $x_{t+1} \sim p_\theta(\cdot \mid x_{\leq t})$, append it, repeat.

What we have after training: a machine that completes text

- Recall **definition** from previous lecture: an **LLM** is a function

$$p_\theta : \{[t_1, \ldots, t_n] : t_i \in \mathcal{V}\} \to \mathbb{P}(\mathcal{V})$$

  mapping a sequence of input tokens to a probability distribution over all possible tokens, where θ is the parameters controlling **linear part** of this function.
- **Generation**: sample $x_{t+1} \sim p_\theta(\cdot \mid x_{\leq t})$, append it, repeat.
- **Pre-training**: update θ to learn **how to guess next token** from the training texts, i.e. minimize cross-entropy loss of predicting the next token.

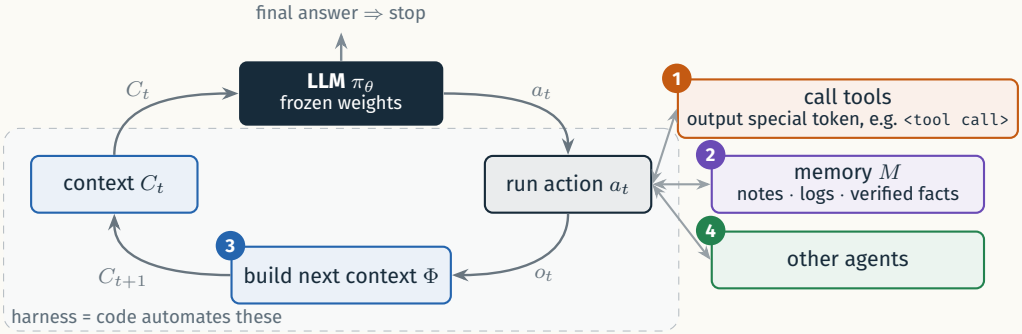
What we have after training: a machine that completes text

- Recall **definition** from previous lecture: an **LLM** is a function

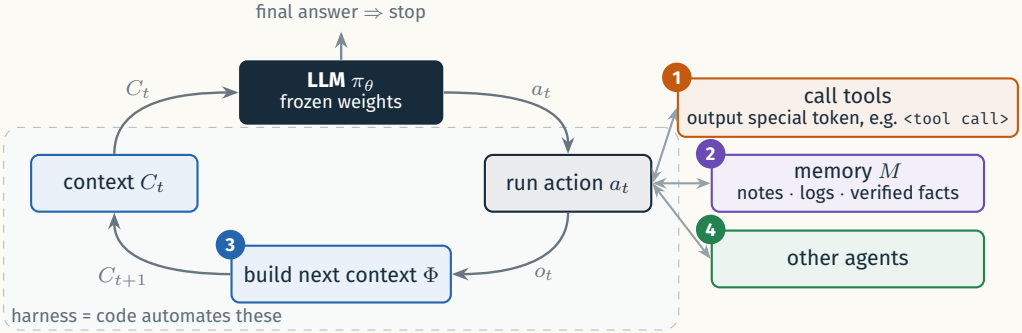
$$p_\theta : \{[t_1, \ldots, t_n] : t_i \in \mathcal{V}\} \to \mathbb{P}(\mathcal{V})$$

  mapping a sequence of input tokens to a probability distribution over all possible tokens, where θ is the parameters controlling **linear part** of this function.
- **Generation**: sample $x_{t+1} \sim p_\theta(\cdot \mid x_{\leq t})$, append it, repeat.
- **Pre-training**: update θ to learn **how to guess next token** from the training texts, i.e. minimize cross-entropy loss of predicting the next token.
- **Post-training**: update θ to learn **how to think, plan and act** instead of blurting out an answer word-by-word from interactions and rewards. (topic of next week)

Agent = model + harness (Overview picture)



Agent = model + harness (Overview picture)



$$a_t \sim \pi_\theta(\cdot \mid C_t), \qquad o_t = T(a_t), \qquad M_{t+1} = \mathrm{write}(M_t, a_t, o_t), \qquad C_{t+1} = \Phi(C_t, a_t, o_t, M_{t+1})$$

a_t is trained via RL, to be discussed in the next lecture.

Why should a mathematician care?

An agent can, for instance:

Why should a mathematician care?

An agent can, for instance:

- **compute**: write and run Sage / Python / GAP code; test a conjecture on 10^4 cases

Why should a mathematician care?

An agent can, for instance:

- **compute**: write and run Sage / Python / GAP code; test a conjecture on 10^4 cases
- **look things up**: arXiv, MathSciNet, zbMATH, OEIS, LMFDB — with sources

Why should a mathematician care?

An agent can, for instance:

- **compute**: write and run Sage / Python / GAP code; test a conjecture on 10^4 cases
- **look things up**: arXiv, MathSciNet, zbMATH, OEIS, LMFDB — with sources
- **check**: call Lean + Mathlib; cross-examine a draft for gaps

Why should a mathematician care?

An agent can, for instance:

- **compute**: write and run Sage / Python / GAP code; test a conjecture on 10^4 cases
- **look things up**: arXiv, MathSciNet, zbMATH, OEIS, LMFDB — with sources
- **check**: call Lean + Mathlib; cross-examine a draft for gaps
- **remember**: notation, lemmas, dead ends — across weeks

Why should a mathematician care?

An agent can, for instance:

- **compute**: write and run Sage / Python / GAP code; test a conjecture on 10^4 cases
- **look things up**: arXiv, MathSciNet, zbMATH, OEIS, LMFDB — with sources
- **check**: call Lean + Mathlib; cross-examine a draft for gaps
- **remember**: notation, lemmas, dead ends — across weeks
- **explore in parallel**: many attempts, with a verifier keeping the good ones

Why should a mathematician care?

An agent can, for instance:

- **compute**: write and run Sage / Python / GAP code; test a conjecture on 10^4 cases
- **look things up**: arXiv, MathSciNet, zbMATH, OEIS, LMFDB — with sources
- **check**: call Lean + Mathlib; cross-examine a draft for gaps
- **remember**: notation, lemmas, dead ends — across weeks
- **explore in parallel**: many attempts, with a verifier keeping the good ones

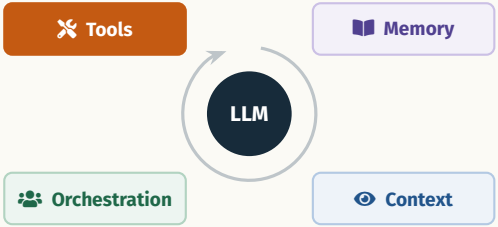
> Examples: Codex CLI, Claude Code, Cursor...
> Other examples: Rethlas, Danus/Pharos, OSVerify ...



1

Tools

Equip the model with real world resources



Why tools? Some things should not be predicted

Demo: GPT-2

Why tools? Some things should not be predicted

Demo: GPT-2

Some type of tasks should be provided as external "resources", for example:

Why tools? Some things should not be predicted

Demo: GPT-2

Some type of tasks should be provided as external "resources", for example:

1. What date is it today? What is the weather today?

Why tools? Some things should not be predicted

Demo: GPT-2

Some type of tasks should be provided as external "resources", for example:

1. What date is it today? What is the weather today?
2. $40392 + 201353 = ?$

Why tools? Some things should not be predicted

Demo: GPT-2

Some type of tasks should be provided as external "resources", for example:

1. What date is it today? What is the weather today?
2. $40392 + 201353 = ?$
3. arXiv papers released after the model was trained.

Why tools? Some things should not be predicted

Demo: GPT-2

Some type of tasks should be provided as external "resources", for example:

1. What date is it today? What is the weather today?
2. $40392 + 201353 = ?$
3. arXiv papers released after the model was trained.
4. Private data, non-language enviroment (Lean4, databases, browser, etc.)

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.
2. **Fill in** typed arguments.

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.
2. **Fill in** typed arguments.
3. **Execute**: the harness runs the call.

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.
2. **Fill in** typed arguments.
3. **Execute**: the harness runs the call.
4. **Return**: the output is appended.

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.
2. **Fill in** typed arguments.
3. **Execute**: the harness runs the call.
4. **Return**: the output is appended.

WHAT THE MODEL SEES: THE DESCRIPTION

```
name:        oeis_search
description: Search the OEIS for sequences
             containing the given terms.
input:       terms: list of integers
```

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.
2. **Fill in** typed arguments.
3. **Execute**: the harness runs the call.
4. **Return**: the output is appended.

WHAT THE MODEL SEES: THE DESCRIPTION

```
name:        oeis_search
description: Search the OEIS for sequences
             containing the given terms.
input:       terms: list of integers
```

WHAT THE MODEL WRITES

```
oeis_search(terms=[1, 2, 5, 14, 42, 132])
```

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.
2. **Fill in** typed arguments.
3. **Execute**: the harness runs the call.
4. **Return**: the output is appended.

WHAT THE MODEL SEES: THE DESCRIPTION

```
name:        oeis_search
description: Search the OEIS for sequences
             containing the given terms.
input:       terms: list of integers
```

WHAT THE MODEL WRITES

```
oeis_search(terms=[1, 2, 5, 14, 42, 132])
```

WHAT COMES BACK

```
A000108  Catalan numbers: binomial(2n,n)/(n+1)
```

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

What is a tool?

Definition (Wang et al., 2024)

A **tool** is a program that runs *outside* the model, with a function interface. The model writes the call and its arguments; the harness executes it.

1. **Select** a tool by its description.
2. **Fill in** typed arguments.
3. **Execute**: the harness runs the call.
4. **Return**: the output is appended.

WHAT THE MODEL SEES: THE DESCRIPTION

```
name:        oeis_search
description: Search the OEIS for sequences
             containing the given terms.
input:       terms: list of integers
```

WHAT THE MODEL WRITES

```
oeis_search(terms=[1, 2, 5, 14, 42, 132])
```

WHAT COMES BACK

```
A000108  Catalan numbers: binomial(2n,n)/(n+1)
```

Example: Claude Code, Codex CLI: bash.

Z. Wang et al., *What Are Tools Anyway? A Survey from the Language Model Perspective*, 2024.

CodeAct vs ReAct

“Bash is the one tool to rule them all.”

X. Wang et al., *Executable Code Actions Elicit Better LLM Agents* (CodeAct), ICML 2024; Terminal-Bench and the Pi harness.

CodeAct vs ReAct

“Bash is the one tool to rule them all.”

The minimalist harness *Pi* has just 4 tools: `read`, `write`, `edit`, `bash`.

X. Wang et al., *Executable Code Actions Elicit Better LLM Agents* (CodeAct), ICML 2024; Terminal-Bench and the Pi harness.

CodeAct vs ReAct

“**Bash is the one tool to rule them all**.”

The minimalist harness *Pi* has just 4 tools: `read`, `write`, `edit`, `bash`.

The terminal is really powerful.

X. Wang et al., *Executable Code Actions Elicit Better LLM Agents* (CodeAct), ICML 2024; Terminal-Bench and the Pi harness.

CodeAct vs ReAct

"**Bash is the one tool to rule them all**."

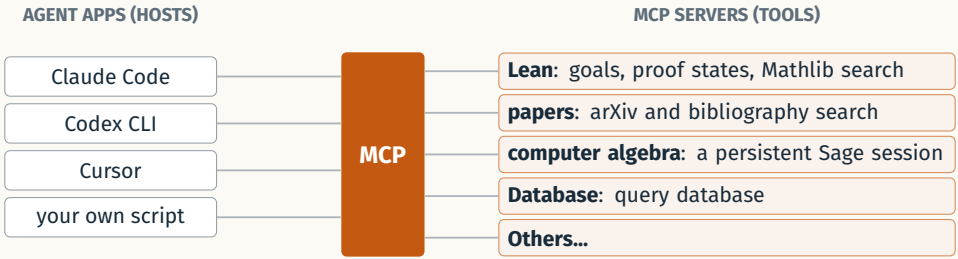
The minimalist harness *Pi* has just 4 tools: `read`, `write`, `edit`, `bash`.

The terminal is really powerful.

Warning: it can also run `rm -rf`. Sandboxes and permissions matter.

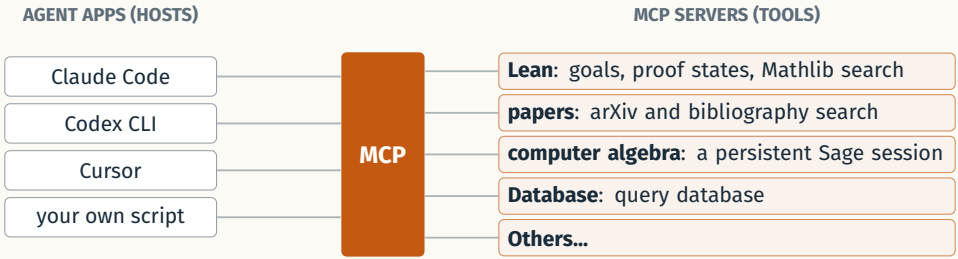
X. Wang et al., *Executable Code Actions Elicit Better LLM Agents* (CodeAct), ICML 2024; Terminal-Bench and the Pi harness.

One plug for every tool: the Model Context Protocol (MCP)



Model Context Protocol, an open standard introduced by Anthropic in 2024: https://modelcontextprotocol.io.

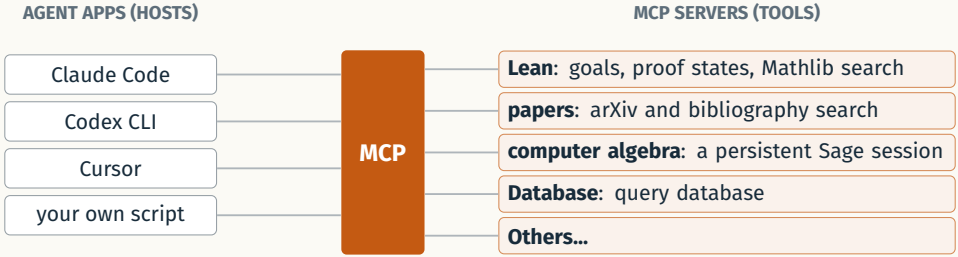
One plug for every tool: the Model Context Protocol (MCP)



- Without a standard protocol, every app needs glue for every tool ($m \times n$); with one, $m + n$.

Model Context Protocol, an open standard introduced by Anthropic in 2024: https://modelcontextprotocol.io.

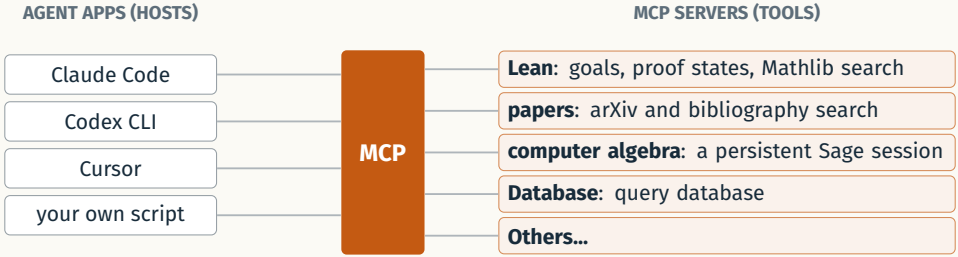
One plug for every tool: the Model Context Protocol (MCP)



- Without a standard protocol, every app needs glue for every tool ($m \times n$); with one, $m + n$.
- MCP makes capabilities *pluggable* and permissions *explicit*.

Model Context Protocol, an open standard introduced by Anthropic in 2024: https://modelcontextprotocol.io.

One plug for every tool: the Model Context Protocol (MCP)

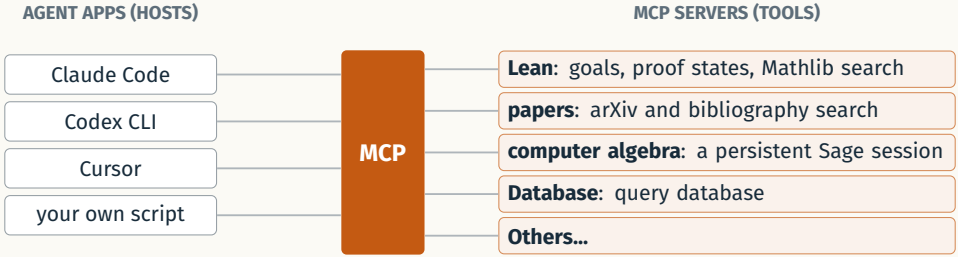


- Without a standard protocol, every app needs glue for every tool ($m \times n$); with one, $m + n$.
- MCP makes capabilities *pluggable* and permissions *explicit*.

- Example: Lean4 (e.g. lean-lsp-mcp), Slack.

Model Context Protocol, an open standard introduced by Anthropic in 2024: `https://modelcontextprotocol.io`.

One plug for every tool: the Model Context Protocol (MCP)



- Without a standard protocol, every app needs glue for every tool ($m \times n$); with one, $m + n$.
- MCP makes capabilities *pluggable* and permissions *explicit*.

- Example: Lean4 (e.g. lean-lsp-mcp), Slack.
- Example: Danus manages fact-graph memory via MCP.

Model Context Protocol, an open standard introduced by Anthropic in 2024: https://modelcontextprotocol.io.

When tool use goes wrong

Single call

Several steps

Trust

When tool use goes wrong

Single call

- wrong tool

Several steps

Trust

When tool use goes wrong

Single call

- wrong tool
- imistyped arguments

Several steps

Trust

When tool use goes wrong

Single call

- wrong tool
- imistyped arguments
- hallucination of tools/arguments

Several steps

Trust

When tool use goes wrong

Single call

- wrong tool
- imistyped arguments
- hallucination of tools/arguments

Several steps

- wrong order

Trust

When tool use goes wrong

Single call

- wrong tool
- imistyped arguments
- hallucination of tools/arguments

Several steps

- wrong order
- retries the same failing call forever

Trust

When tool use goes wrong

Single call

- wrong tool
- imistyped arguments
- hallucination of tools/arguments

Several steps

- wrong order
- retries the same failing call forever
- misreads a long or truncated output

Trust

When tool use goes wrong

Single call

- wrong tool
- imistyped arguments
- hallucination of tools/arguments

Several steps

- wrong order
- retries the same failing call forever
- misreads a long or truncated output

Trust

- *prompt injection*: dangerous!

When tool use goes wrong

Single call

- wrong tool
- imistyped arguments
- hallucination of tools/arguments

Several steps

- wrong order
- retries the same failing call forever
- misreads a long or truncated output

Trust

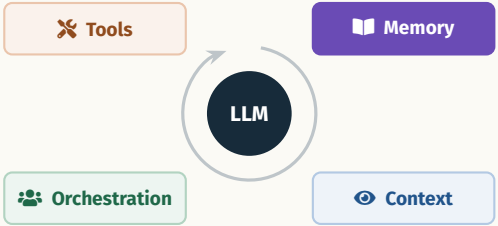
- *prompt injection*: dangerous!
- wrong actions: deletes, overwrites, sends

2

Memory

Persistent Memory

Overcome lost attention and limited context window



The model has no memory of its own

The model has no memory of its own

- Weights θ are frozen at inference time.

The model has no memory of its own

- Weights θ are frozen at inference time.
- The context window is finite and is gone when the session ends.

The model has no memory of its own

- Weights θ are frozen at inference time.
- The context window is finite and is gone when the session ends.
- So without memory, every session starts from zero. Lost:

The model has no memory of its own

- Weights θ are frozen at inference time.
- The context window is finite and is gone when the session ends.
- So without memory, every session starts from zero. Lost:
  - **knowledge**: notation, conventions, lemmas already established.

The model has no memory of its own

- Weights θ are frozen at inference time.
- The context window is finite and is gone when the session ends.
- So without memory, every session starts from zero. Lost:
  - **knowledge**: notation, conventions, lemmas already established.
  - **experience**: how we should do things in this project.

The model has no memory of its own

- Weights θ are frozen at inference time.
- The context window is finite and is gone when the session ends.
- So without memory, every session starts from zero. Lost:
  - **knowledge**: notation, conventions, lemmas already established.
  - **experience**: how we should do things in this project.
  - **progress**: what was tried, and why it failed.

The model has no memory of its own

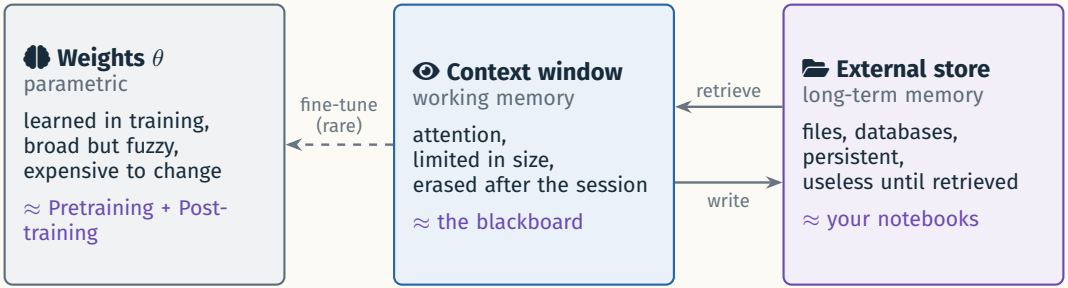
- Weights θ are frozen at inference time.
- The context window is finite and is gone when the session ends.
- So without memory, every session starts from zero. Lost:
  - **knowledge**: notation, conventions, lemmas already established.
  - **experience**: how we should do things in this project.
  - **progress**: what was tried, and why it failed.



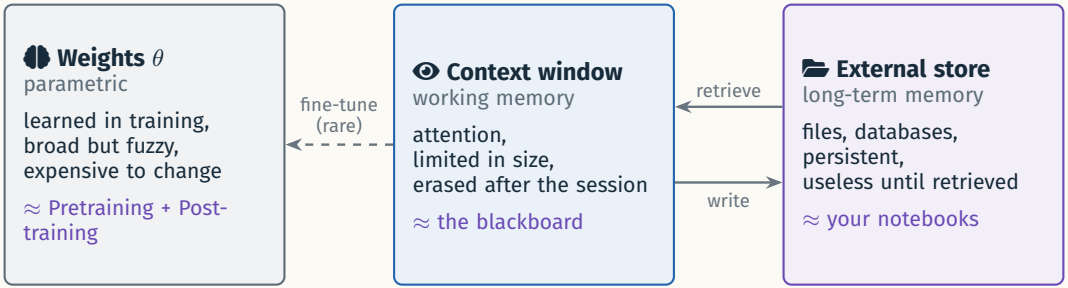
We take notes, write papers, and remind ourselves from time to time.

Agent bahaves similarly.

Where can information live?

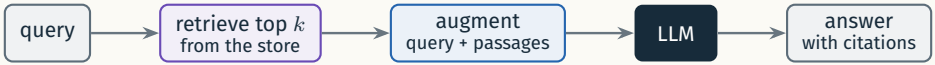


Where can information live?



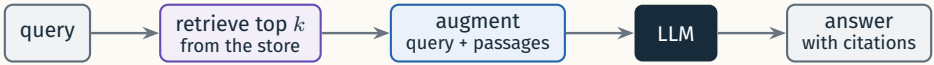
Agent memory = an external store + a **write** policy + a **read** policy.
The design questions: *what* content/format to store, *when*, and *how* to retrieve.

Retrieval: bringing the right memory back into context



Lewis et al., *Retrieval-Augmented Generation*, NeurIPS 2020; Robertson & Zaragoza, *The Probabilistic Relevance Framework: BM25 and Beyond*, 2009; CS329Z Lecture 3.

Retrieval: bringing the right memory back into context



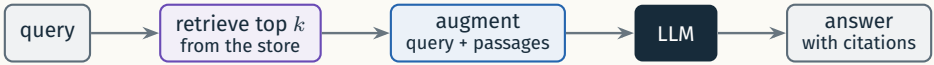
Lexical: match words (BM25)

Score a document by the query words it contains, weighting *rare* words most.

Good for exact strings: “Kazhdan–Lusztig”, “Lemma 4.2”, a piece of notation.

Lewis et al., *Retrieval-Augmented Generation*, NeurIPS 2020; Robertson & Zaragoza, *The Probabilistic Relevance Framework: BM25 and Beyond*, 2009; CS329Z Lecture 3.

Retrieval: bringing the right memory back into context



Lexical: match words (BM25)

Score a document by the query words it contains, weighting *rare* words most.

Good for exact strings: "Kazhdan–Lusztig", "Lemma 4.2", a piece of notation.

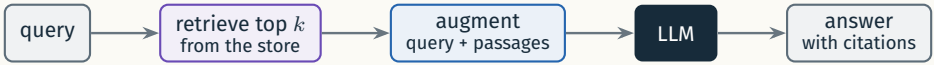
Dense: match meaning (embeddings)

Embed text as vectors, E : text $\to \mathbb{R}^n$; return the documents D with the largest $\langle E(q), E(D) \rangle$.

Good for paraphrase: "line bundle" $\approx$ "cartier divisor".

Lewis et al., *Retrieval-Augmented Generation*, NeurIPS 2020; Robertson & Zaragoza, *The Probabilistic Relevance Framework: BM25 and Beyond*, 2009; CS329Z Lecture 3.

Retrieval: bringing the right memory back into context



Lexical: match words (BM25)

Score a document by the query words it contains, weighting *rare* words most.

Good for exact strings: "Kazhdan–Lusztig", "Lemma 4.2", a piece of notation.

Dense: match meaning (embeddings)

Embed text as vectors, E : text $\to \mathbb{R}^n$; return the documents D with the largest $\langle E(q), E(D) \rangle$.

Good for paraphrase: "line bundle" $\approx$ "cartier divisor".

- **Agentic retrieval**: Self-RAG, Search-R1, etc.

Lewis et al., *Retrieval-Augmented Generation*, NeurIPS 2020; Robertson & Zaragoza, *The Probabilistic Relevance Framework: BM25 and Beyond*, 2009; CS329Z Lecture 3.

In practice, memory is often a folder of text files

```
example-project/
  AGENTS.md            goal, notation, rules
  notes/
    definitions.md
    facts.md
    attempts.md
    todo.md
  report.jsonl
  scripts/
  draft/main.tex
```

In practice, memory is often a folder of text files

```
example-project/
  AGENTS.md            goal, notation, rules
  notes/
    definitions.md
    facts.md
    attempts.md
    todo.md
  report.jsonl
  scripts/
  draft/main.tex
```

- Coding agents read `AGENTS.md` (or `CLAUDE.md`, …) at the start of every session

In practice, memory is often a folder of text files

```
example-project/
  AGENTS.md            goal, notation, rules
  notes/
    definitions.md
    facts.md
    attempts.md
    todo.md
  report.jsonl
  scripts/
  draft/main.tex
```

- Coding agents read `AGENTS.md` (or `CLAUDE.md`, …) at the start of every session
- Reading and writing memory uses `ls`, `grep`, read, edit. Version it with `git`.

In practice, memory is often a folder of text files

```
example-project/
  AGENTS.md            goal, notation, rules
  notes/
    definitions.md
    facts.md
    attempts.md
    todo.md
  report.jsonl
  scripts/
  draft/main.tex
```

- Coding agents read `AGENTS.md` (or `CLAUDE.md`, …) at the start of every session
- Reading and writing memory uses `ls`, `grep`, read, edit. Version it with `git`.
- Some are human-readable: you can audit and correct what the agent “believes”.

In practice, memory is often a folder of text files

```
example-project/
  AGENTS.md            goal, notation, rules
  notes/
    definitions.md
    facts.md
    attempts.md
    todo.md
  report.jsonl
  scripts/
  draft/main.tex
```

- Coding agents read `AGENTS.md` (or `CLAUDE.md`, …) at the start of every session
- Reading and writing memory uses `ls`, `grep`, read, edit. Version it with `git`.
- Some are human-readable: you can audit and correct what the agent "believes".

Plan Mode in Claude Code/Codex CLI: simply a prompt + updating a `plan.md` file, enforced by a sandbox policy.

Procedural memory: skills

Anthropic, *Agent Skills* (2025); G. Wang et al., *Voyager* (2023); skill from github.com/frenzymath/Rethlas (Apache-2.0).

Procedural memory: skills

- A **skill** is a folder: `SKILL.md` (name, description, instructions) plus scripts and references.

Anthropic, *Agent Skills* (2025); G. Wang et al., *Voyager* (2023); skill from github.com/frenzymath/Rethlas (Apache-2.0).

Procedural memory: skills

- A **skill** is a folder: `SKILL.md` (name, description, instructions) plus scripts and references.
- **Invoked like tools**: only the name and description stay in the context. The agent opens the rest only when it decides to use the skill.

Anthropic, *Agent Skills* (2025); G. Wang et al., *Voyager* (2023); skill from github.com/frenzymath/Rethlas (Apache-2.0).

Procedural memory: skills

- A **skill** is a folder: `SKILL.md` (name, description, instructions) plus scripts and references.
- **Invoked like tools**: only the name and description stay in the context. The agent opens the rest only when it decides to use the skill.

Example (from Rethlas)

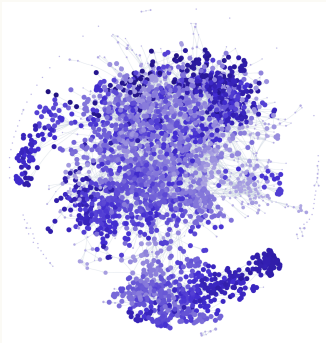
```
---
name: construct-counterexamples
description: Construct candidate counterexamples to
  test a proposed conjecture, lemma, or intermediate
  claim by keeping the assumptions true while making
  the claimed conclusion fail. [...]
---
## Procedure
1. Identify the assumptions that must hold and the
   conclusion to fail.
2. Use reasoning, decomposition, and retrieval to
   search for standard obstructions, pathological
   constructions, or known counterexamples.
3. Decide status: refuted | not_refuted | inconclusive
   [...]
6. If no counterexample is found, treat that only as
   evidence that the claim may be correct, not as a
   proof.
```

Anthropic, *Agent Skills* (2025); G. Wang et al., *Voyager* (2023); skill from github.com/frenzymath/Rethlas (Apache-2.0).

Example: Memory management

A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:



The fact graph of an actual Danus investigation (matroid tangent classes).

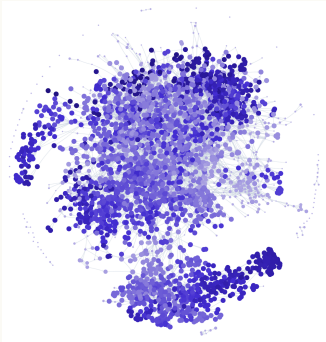
Danus, arXiv:2607.06447; `https://github.com/frenzymath/Danus`.

Example: Memory management

A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:

- **local**: each worker’s private scratch work;



The fact graph of an actual Danus investigation (matroid tangent classes).

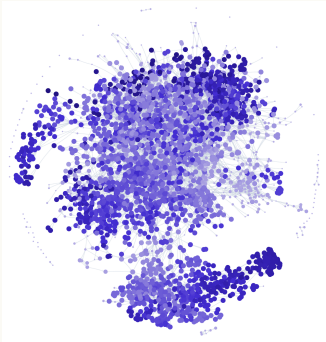
Danus, arXiv:2607.06447; `https://github.com/frenzymath/Danus`.

Example: Memory management

A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:

- **local**: each worker’s private scratch work;
- **global**: shared plans, findings, dead ends;



The fact graph of an actual Danus investigation (matroid tangent classes).

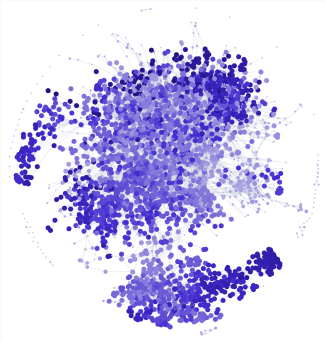
Danus, arXiv:2607.06447; `https://github.com/frenzymath/Danus`.

Example: Memory management

A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:

- **local**: each worker’s private scratch work;
- **global**: shared plans, findings, dead ends;
- **fact graph**: only statements accepted by the verifier.



The fact graph of an actual Danus investigation (matroid tangent classes).

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

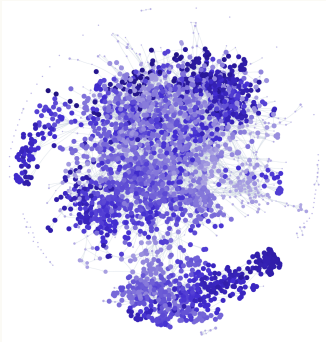
Example: Memory management

A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:

- **local**: each worker’s private scratch work;
- **global**: shared plans, findings, dead ends;
- **fact graph**: only statements accepted by the verifier.

Remark: this is similar to human collaboration:



The fact graph of an actual Danus investigation (matroid tangent classes).

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

Example: Memory management

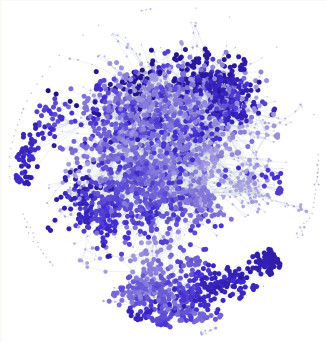
A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:

- **local**: each worker’s private scratch work;
- **global**: shared plans, findings, dead ends;
- **fact graph**: only statements accepted by the verifier.

Remark: this is similar to human collaboration:

- each person has his/her own drafts;



The fact graph of an actual Danus investigation (matroid tangent classes).

Danus, arXiv:2607.06447; `https://github.com/frenzymath/Danus`.

Example: Memory management

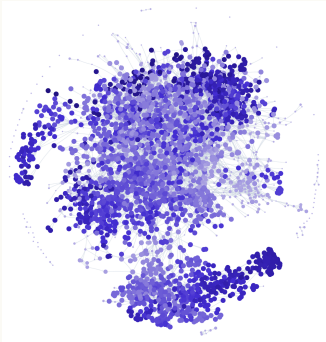
A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:

- **local**: each worker’s private scratch work;
- **global**: shared plans, findings, dead ends;
- **fact graph**: only statements accepted by the verifier.

Remark: this is similar to human collaboration:

- each person has his/her own drafts;
- a shared informal knowledge about findings, plans, bitter lessons;



The fact graph of an actual Danus investigation (matroid tangent classes).

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

Example: Memory management

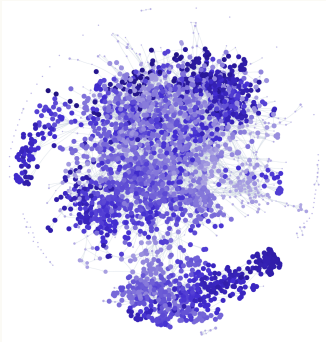
A wrong “lemma” in memory silently poisons everything built on it.

Danus (a multi-agent math system) keeps three tiers:

- **local**: each worker’s private scratch work;
- **global**: shared plans, findings, dead ends;
- **fact graph**: only statements accepted by the verifier.

Remark: this is similar to human collaboration:

- each person has his/her own drafts;
- a shared informal knowledge about findings, plans, bitter lessons;
- a draft paper tex file which only contains verified statements.



The fact graph of an actual Danus investigation (matroid tangent classes).

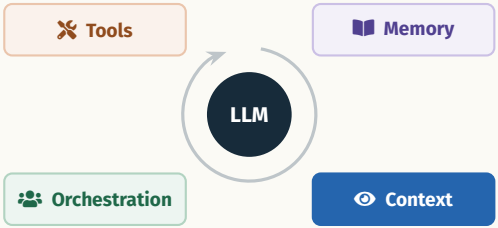
Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

3

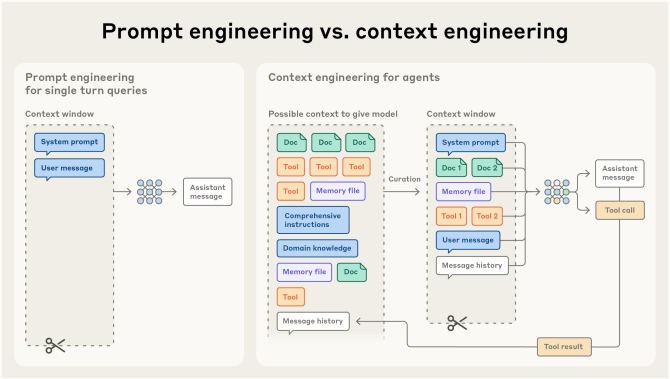
Prompt & context

What the model sees

Imagine: you are meeting your PhD advisor, you should let him know the exact context to give meaningful suggestions.



From prompt engineering to context engineering

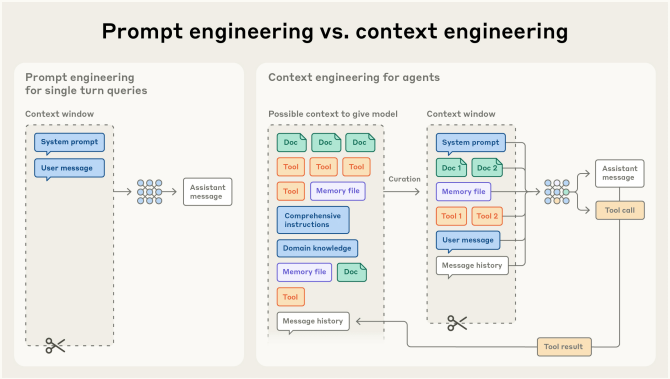


Prompt engineering: find a good phrasing of the same content.

Metaphor: "This food tastes disgusting" vs "This is good food but I am not used to it"

Anthropic, *Effective context engineering for AI agents* (2025).

From prompt engineering to context engineering



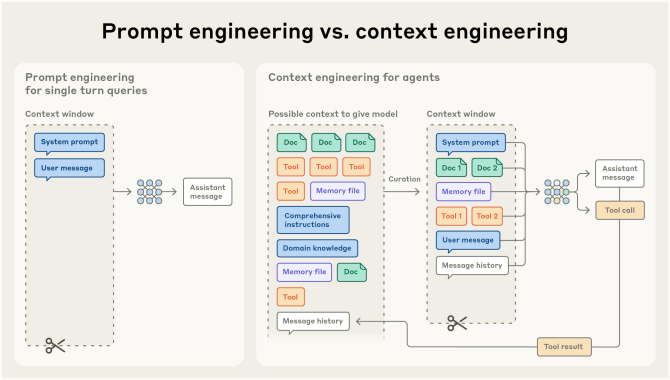
Prompt engineering: find a good phrasing of the same content.

Metaphor: "This food tastes disgusting" vs "This is good food but I am not used to it"

Context engineering: decide which tokens (instructions, tool descriptions, memories, etc.) go into the context window.

Anthropic, *Effective context engineering for AI agents* (2025).

From prompt engineering to context engineering



Prompt engineering: find a good phrasing of the same content.

Metaphor: "This food tastes disgusting" vs "This is good food but I am not used to it"

Context engineering: decide which tokens (instructions, tool descriptions, memories, etc.) go into the context window.

Personal feeling: At this moment, prompt engineering doesn't matter much anymore. But context engineering matters.

Anthropic, *Effective context engineering for AI agents* (2025).

Quiz: which instruction gives the highest accuracy?

Grade-school math word problems (GSM8K), model PaLM 2-L, zero-shot. The instruction is placed at the start of every answer.

A “Let’s think step by step.”

B “Take a deep breath and work on this problem step-by-step.”

C “Let’s combine our numerical command and clear thinking to quickly and accurately decipher the answer.”

C. Yang et al., *Large Language Models as Optimizers* (OPRO), ICLR 2024; quiz from CS329Z Lecture 9.

Quiz: which instruction gives the highest accuracy?

Grade-school math word problems (GSM8K), model PaLM 2-L, zero-shot. The instruction is placed at the start of every answer.

A	"Let's think step by step."	**71.8%**
B	"Take a deep breath and work on this problem step-by-step."	**80.2%**
C	"Let's combine our numerical command and clear thinking to quickly and accurately decipher the answer."	**74.5%**

C. Yang et al., *Large Language Models as Optimizers* (OPRO), ICLR 2024; quiz from CS329Z Lecture 9.

Context engineering: two principles, five moves

Two principles

Focus: keep only what the current step needs.
Continuity: never lose goal, constraints, definitions, progress.

Context engineering: two principles, five moves

Two principles

Focus: keep only what the current step needs.
Continuity: never lose goal, constraints, definitions, progress.

1. **Compaction**: summarize old history.

Context engineering: two principles, five moves

Two principles

Focus: keep only what the current step needs.
Continuity: never lose goal, constraints, definitions, progress.

1. **Compaction**: summarize old history.
2. **Truncation**: truncate stale tool outputs.

Context engineering: two principles, five moves

Two principles

Focus: keep only what the current step needs.
Continuity: never lose goal, constraints, definitions, progress.

1. **Compaction**: summarize old history.
2. **Truncation**: truncate stale tool outputs.
3. **Handoff**: write results to files.

Context engineering: two principles, five moves

Two principles

Focus: keep only what the current step needs.
Continuity: never lose goal, constraints, definitions, progress.

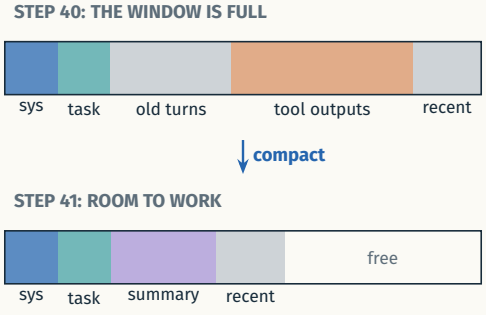
1. **Compaction**: summarize old history.
2. **Truncation**: truncate stale tool outputs.
3. **Handoff**: write results to files.
4. **Isolation**: a sub-agent, a fresh window.

Context engineering: two principles, five moves

Two principles

Focus: keep only what the current step needs.
Continuity: never lose goal, constraints, definitions, progress.

1. **Compaction**: summarize old history.
2. **Truncation**: truncate stale tool outputs.
3. **Handoff**: write results to files.
4. **Isolation**: a sub-agent, a fresh window.

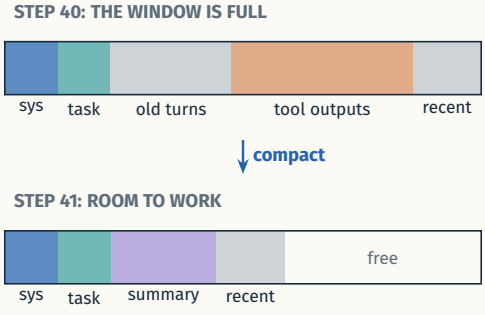


Context engineering: two principles, five moves

Two principles

Focus: keep only what the current step needs.
Continuity: never lose goal, constraints, definitions, progress.

1. **Compaction**: summarize old history.
2. **Truncation**: truncate stale tool outputs.
3. **Handoff**: write results to files.
4. **Isolation**: a sub-agent, a fresh window.



Personal experience (valid till 2026-10):

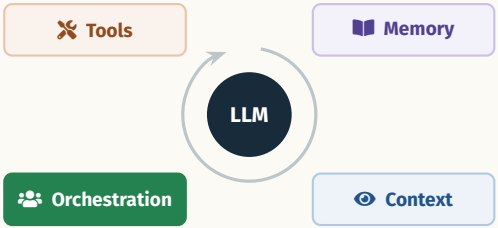
1. Performance of Claude Code / Codex CLI becomes unstable after two/three (default) compactions.
2. Estimation of when to compactify can be wrong (observed in Codex CLI), results in full context window.

4

Orchestration

Workflows, agents, and teams of agents

Multi-agents: mimic a research team/organization





Five building blocks

1. Prompt chaining



statement → proof plan → proof → LaTeX

Anthropic, *Building effective agents* (2024)

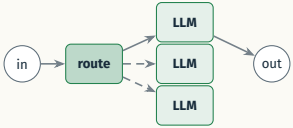
Five building blocks

1. Prompt chaining



statement → proof plan → proof → LaTeX

2. Routing



Similar idea as MOE

Anthropic, *Building effective agents* (2024)

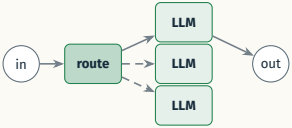
Five building blocks

1. Prompt chaining



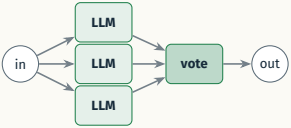
statement → proof plan → proof → LaTeX

2. Routing



Similar idea as MOE

3. Parallelization



k independent solutions, then a majority vote

Anthropic, *Building effective agents* (2024)

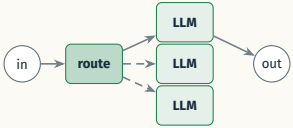
Five building blocks

1. Prompt chaining



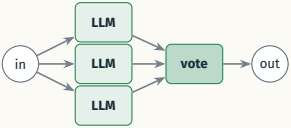
statement → proof plan → proof → LaTeX

2. Routing



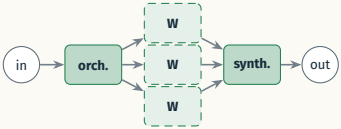
Similar idea as MOE

3. Parallelization



k independent solutions, then a majority vote

4. Orchestrator–workers



Orchestrator assigns tasks to workers

Anthropic, *Building effective agents* (2024)

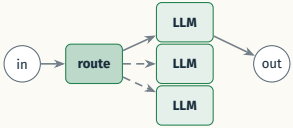
Five building blocks

1. Prompt chaining



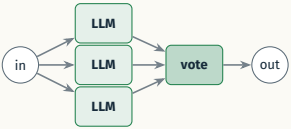
statement → proof plan → proof → LaTeX

2. Routing



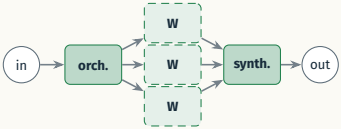
Similar idea as MOE

3. Parallelization



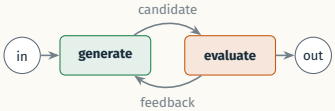
k independent solutions, then a majority vote

4. Orchestrator–workers



Orchestrator assigns tasks to workers

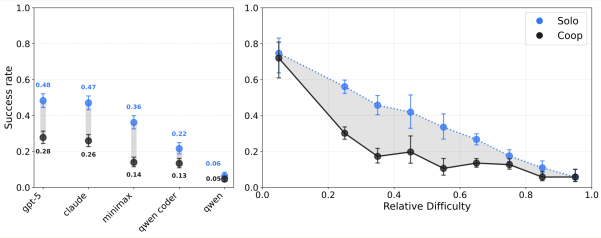
5. Evaluator–optimizer



Underlying design of Rethlas and OSVerify

Anthropic, *Building effective agents* (2024)

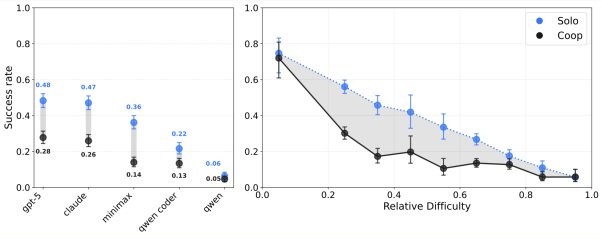
More agents $\neq$ better results



CooperBench. One agent doing two related coding tasks (*solo*) versus two agents cooperating on them: cooperating is markedly worse for every model tested.

Khatua et al., *CooperBench*, arXiv:2601.13295; Kim et al., *Towards a Science of Scaling Agent Systems*, arXiv:2512.08296; Zou, *Patterns and Problems in Emerging Multi-Agent Systems* (2026).

More agents $\neq$ better results

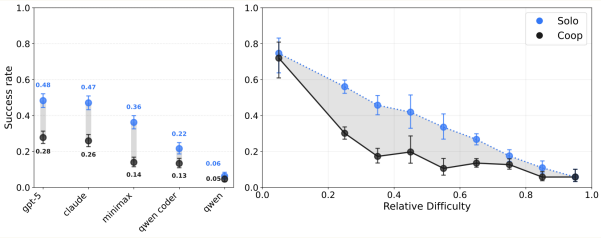


CooperBench. One agent doing two related coding tasks (*solo*) versus two agents cooperating on them: cooperating is markedly worse for every model tested.

- Across model families, multi-agent variants helped for some and hurt for others (Kim et al., 2026).

Khatua et al., *CooperBench*, arXiv:2601.13295; Kim et al., *Towards a Science of Scaling Agent Systems*, arXiv:2512.08296; Zou, *Patterns and Problems in Emerging Multi-Agent Systems* (2026).

More agents ≠ better results

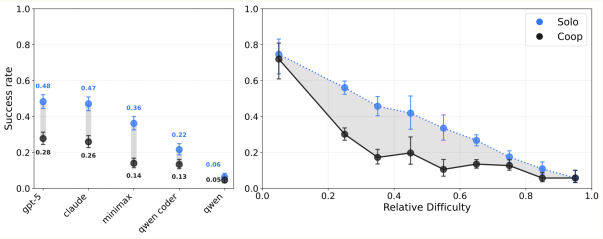


CooperBench. One agent doing two related coding tasks (*solo*) versus two agents cooperating on them: cooperating is markedly worse for every model tested.

- Across model families, multi-agent variants helped for some and hurt for others (Kim et al., 2026).
- Without orchestration, multiple agents may conflict with each other and harm the performance.

Khatua et al., *CooperBench*, arXiv:2601.13295; Kim et al., *Towards a Science of Scaling Agent Systems*, arXiv:2512.08296; Zou, *Patterns and Problems in Emerging Multi-Agent Systems* (2026).

More agents $\neq$ better results

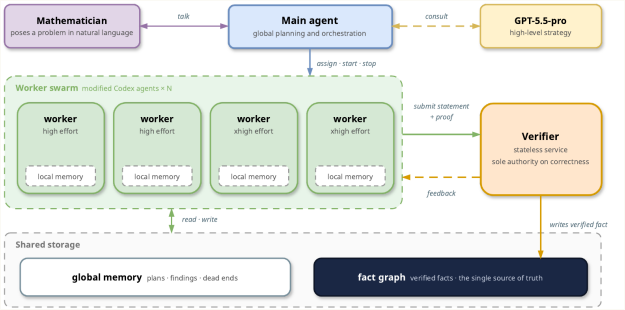


CooperBench. One agent doing two related coding tasks (*solo*) versus two agents cooperating on them: cooperating is markedly worse for every model tested.

- Across model families, multi-agent variants helped for some and hurt for others (Kim et al., 2026).
- Without orchestration, multiple agents may conflict with each other and harm the performance.
- Orchestration really depends on the base model/agent capability (eg. Danus→Pharos)

Khatua et al., *CooperBench*, arXiv:2601.13295; Kim et al., *Towards a Science of Scaling Agent Systems*, arXiv:2512.08296; Zou, *Patterns and Problems in Emerging Multi-Agent Systems* (2026).

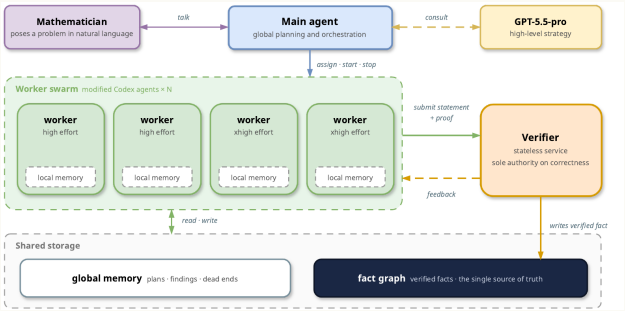
Example: Danus/Pharos, a multi-agent system for mathematics



Danus

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

Example: Danus/Pharos, a multi-agent system for mathematics

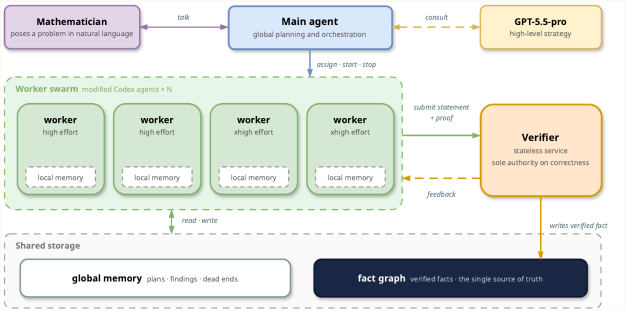


Danus

4 **Orchestration**: the main agent plans and assigns but never does the mathematics; each worker is a single-agent prover.

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

Example: Danus/Pharos, a multi-agent system for mathematics



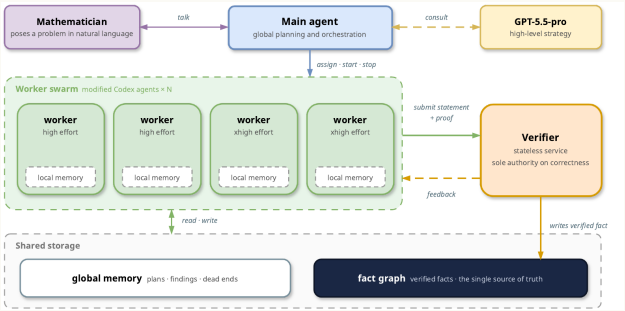
Danus

4 **Orchestration**: the main agent plans and assigns but never does the mathematics; each worker is a single-agent prover.

1 **Tools**: Different roles have different tools. Main agent can not submit to fact graph

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

Example: Danus/Pharos, a multi-agent system for mathematics

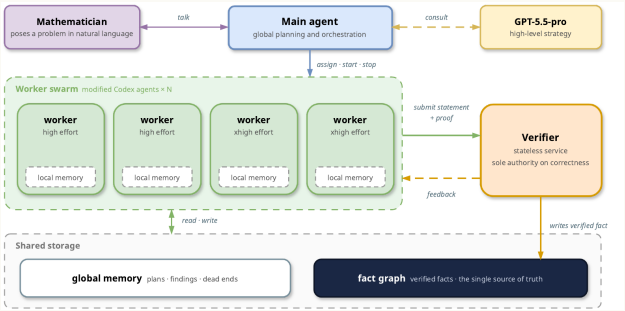


Danus

4. **Orchestration**: the main agent plans and assigns but never does the mathematics; each worker is a single-agent prover.
1. **Tools**: Different roles have different tools. Main agent can not submit to fact graph
2. **Memory**: local, global, fact graph.

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

Example: Danus/Pharos, a multi-agent system for mathematics



Danus

4. **Orchestration**: the main agent plans and assigns but never does the mathematics; each worker is a single-agent prover.
1. **Tools**: Different roles have different tools. Main agent can not submit to fact graph
2. **Memory**: local, global, fact graph.
3. **Context**: papers and reports are written by a fresh agent with a clean context; verifer is stateless; orchestration is separated from operator (Pharos)

Danus, arXiv:2607.06447; https://github.com/frenzymath/Danus.

Other topics

agent = LLM + loops + memory + context + orchestration

If we have time, we may talk about the following extra topics:

Other topics

agent = LLM + loops + memory + context + orchestration

If we have time, we may talk about the following extra topics:

- How Codex CLI / Pharos manages long term memory, context, etc, in practice, on engineering level.

Other topics

agent = LLM + loops + memory + context + orchestration

If we have time, we may talk about the following extra topics:

- How Codex CLI / Pharos manages long term memory, context, etc, in practice, on engineering level.
- Limitation of harness, and how to improve harness as base model improves?

Other topics

agent = LLM + loops + memory + context + orchestration

If we have time, we may talk about the following extra topics:

- How Codex CLI / Pharos manages long term memory, context, etc, in practice, on engineering level.
- Limitation of harness, and how to improve harness as base model improves?
- How to train certain harness in postraining (dynamic workflow)?

Other topics

agent = LLM + loops + memory + context + orchestration

If we have time, we may talk about the following extra topics:

- How Codex CLI / Pharos manages long term memory, context, etc, in practice, on engineering level.
- Limitation of harness, and how to improve harness as base model improves?
- How to train certain harness in postraining (dynamic workflow)?
- Evaluation and benchmark of multi-agent systems?

Other topics

agent = LLM + loops + memory + context + orchestration

If we have time, we may talk about the following extra topics:

- How Codex CLI / Pharos manages long term memory, context, etc, in practice, on engineering level.
- Limitation of harness, and how to improve harness as base model improves?
- How to train certain harness in postraining (dynamic workflow)?
- Evaluation and benchmark of multi-agent systems?
- Final projects related to agents: Understanding/Writing, Human-AI collaboration.

References

Stanford CS329Z, *Engineering AI Agents*, Fall 2026 (D. Yang, M. Ryan, J. Yang), Lectures 1–10.

Yao et al., ReAct, ICLR 2023.
Schick et al., Toolformer, NeurIPS 2023.
Z. Wang et al., What Are Tools Anyway?, 2024.
X. Wang et al., CodeAct, ICML 2024.
Dziri et al., Faith and Fate, NeurIPS 2023.
Model Context Protocol, modelcontextprotocol.io.

Sumers et al., CoALA, TMLR 2024.
Lewis et al., Retrieval-Augmented Generation, NeurIPS 2020.
Robertson and Zaragoza, BM25 and Beyond, 2009.
Shinn et al., Reflexion, NeurIPS 2023.
G. Wang et al., Voyager, 2023.
Anthropic, Agent Skills, 2025.

Rethlas, arXiv:2604.03789, github.com/frenzymath/Rethlas.
Danus, arXiv:2607.06447, github.com/frenzymath/Danus.

Anthropic, Effective context engineering for AI agents, 2025.
N. Liu et al., Lost in the Middle, TACL 2024.
OOLONG, arXiv:2511.02817.
C. Yang et al., Large Language Models as Optimizers (OPRO), ICLR 2024.
METR, Recent frontier models are reward hacking, 2025.

Anthropic, Building effective agents, 2024.
Romera-Paredes et al., FunSearch, Nature 2024.
Novikov et al., AlphaEvolve, 2025.
Chen et al., Evaluating LLMs Trained on Code (pass@k), 2021.
Yao et al., τ-bench (pass^k), 2024.
Khatua et al., CooperBench, arXiv:2601.13295.
Kim et al., Towards a Science of Scaling Agent Systems, arXiv:2512.08296.
Zou, Patterns and Problems in Emerging Multi-Agent Systems, 2026.
N. Carlini, C compiler with parallel agents, Anthropic Engineering, 2026.

Shen and Tamkin, AI assistance and coding skills, Anthropic, 2026.