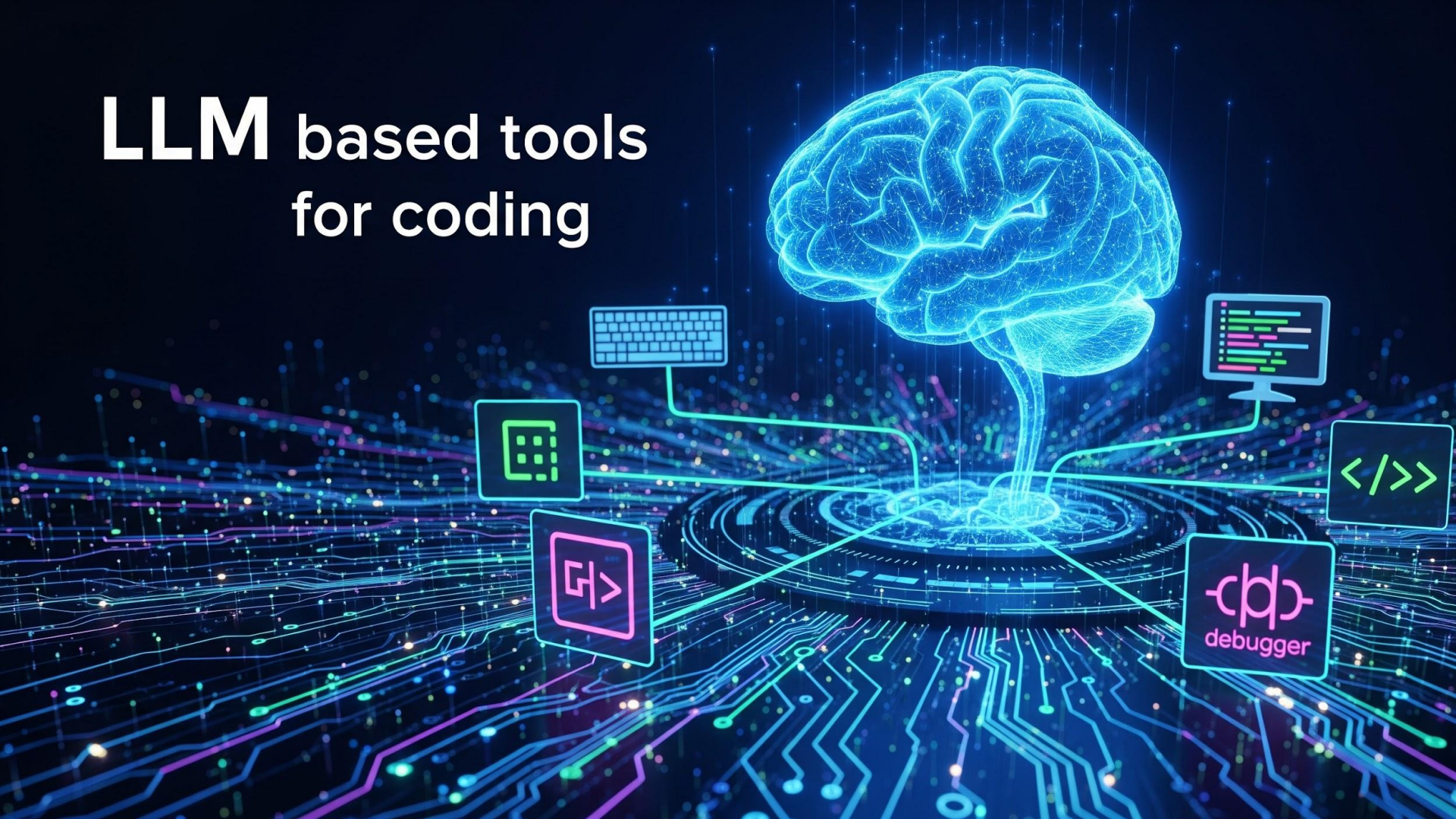
LLM based tools for coding



$ whoami

Antonio Lobo (antonio.lobo@iiia.csic.es)

Backend developer (~ 2 years)

Contract Engineer at IIIA - CSIC

Interested in LLMs (moral) capabilities

Seminar Code

Repository containing all code for the seminar.

TODO: Create QR with zip

Index

1. Theoretical Concepts (with short exercises).

   BREAK (20 mins)

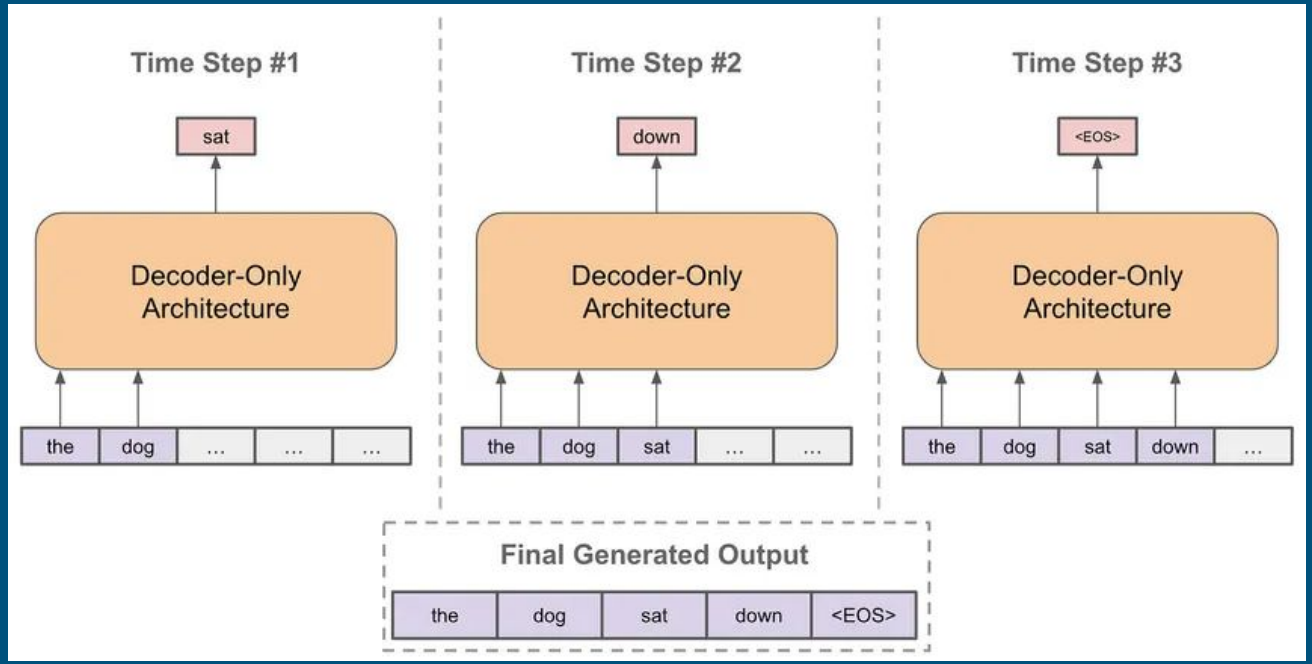
2. How to create an App from 0.

3. How to modify an existing codebase.

Theoretical Concepts

1. Introduction to LLMs
2. LLM Capabilities (and Limitations)
3. LLMs as tools vs agents
4. Model Context Protocol (MCP)
5. Context Engineering (Claude Code functionalities)

What are LLMs?



Source: Wolfe, C. R. (2024), Language model training and inference: From concept to code. Deep (Learning) Focus. [1]

Stochastic Parrots

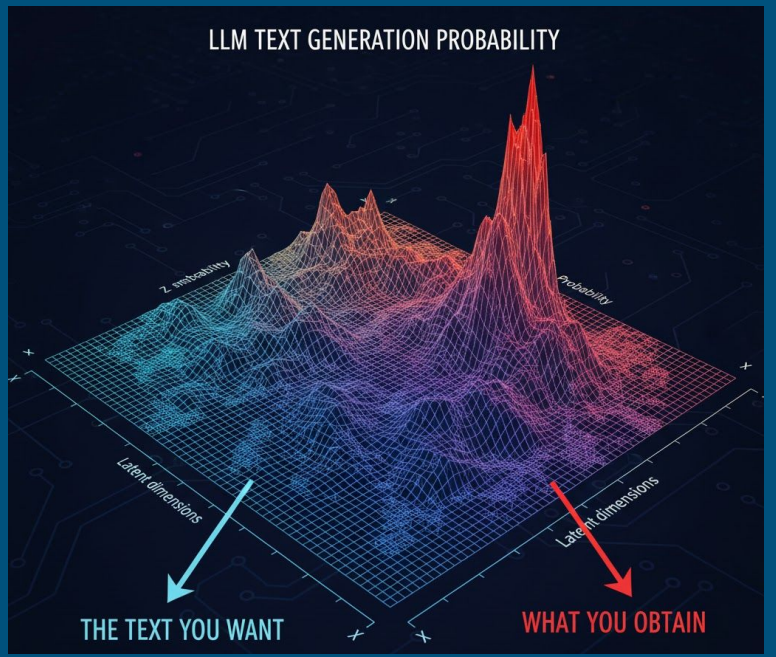
Sampling:

- Temperature: scales the logits
- top_k/top_p: keep most probable tokens

Deterministic:

- Beam Search

PROMPT



Prompting

Basic Guidelines

Hierarchy

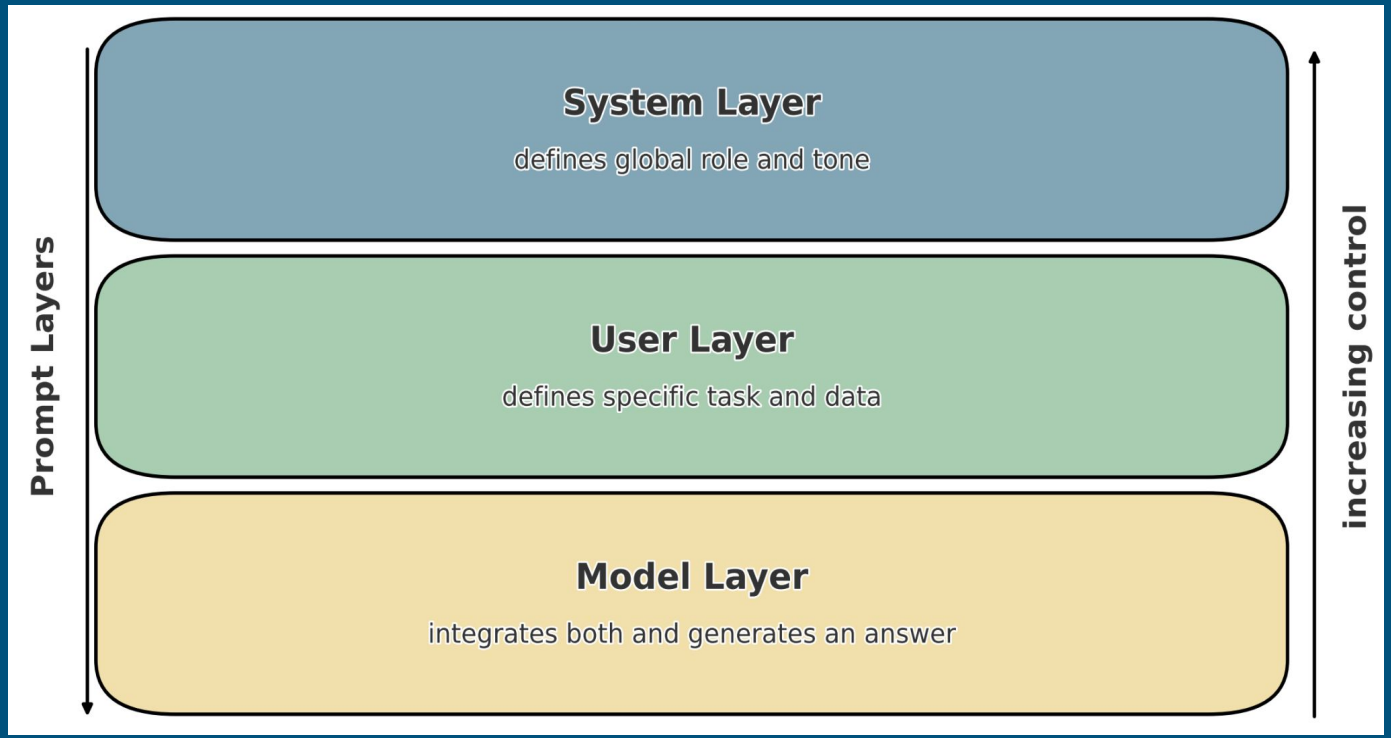
Anatomy

Clearness

Persona

Structure

Prompt Hierarchy



Prompt Anatomy

Be Clear

- **Be Specific:** State exactly what output you want
- **Use Strong Action Verbs:** List, Compare, Convert
- **Explicit Constraints:** Length, Format, Style
- **Positive Phrasing:** What TO DO, not what to avoid.
- **Success Criteria:** What is a good answer?

Bad Example (Unclear): "Explain this better."
What's wrong: Too vague what to explain? Better in what sense (shorter? more formally? more detail?) The AI may guess incorrectly what you want improved.

Improved Prompt: "Rewrite the following explanation in clearer, simpler terms for a general audience, in under 3 sentences."
Why it's better: Specifies the transformation (simplify the explanation), the target audience (general public), and limits scope (3 sentences). The model isn't left to guess your intent.

Defining a Strong Persona

- Role & Goal: Define who the AI is and its main objective.
- Expertise: Specify domain knowledge to ensure contextual accuracy.
- Tone & Style: Set the desired voice, formality, or attitude.
- Constraints: Outline limits or biases to maintain safe boundaries.
- Example Behavior: Show a brief sample of the desired response style.

> **System:** "You are CodeGuru, an AI programming assistant. You have expertise in Python and software engineering best practices. You provide step-by-step explanations and well- commented code solutions. You never just give the final code you first explain your approach." **User:** "How do I sort a list of dictionaries in Python by a value?"

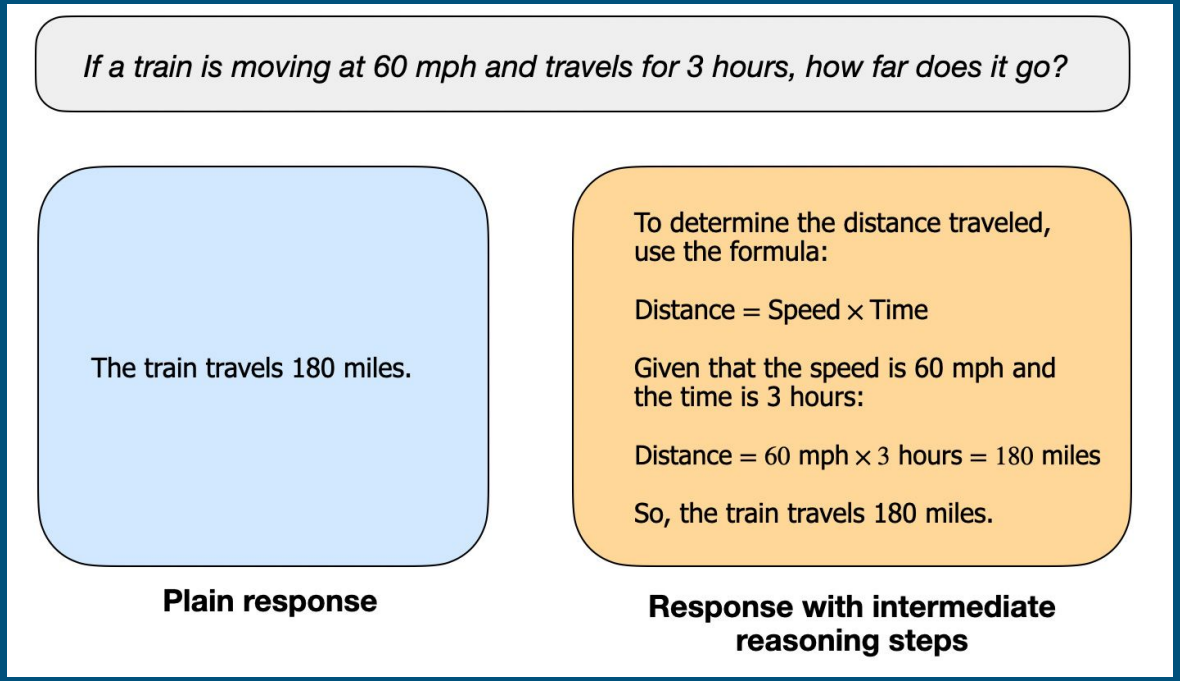
Structure and Reasoning

- **Use XML tags** to separate different parts of your prompt. Ex: <data></data>
- **Chain of Thought:** think step by step and how to decompose the problem
- Few-Shot: **Include Examples**
- Let the model **ask you** and answer **I don’t know.**
- Ask for **evidence** or **Quotes**.

Exercise: Prompting

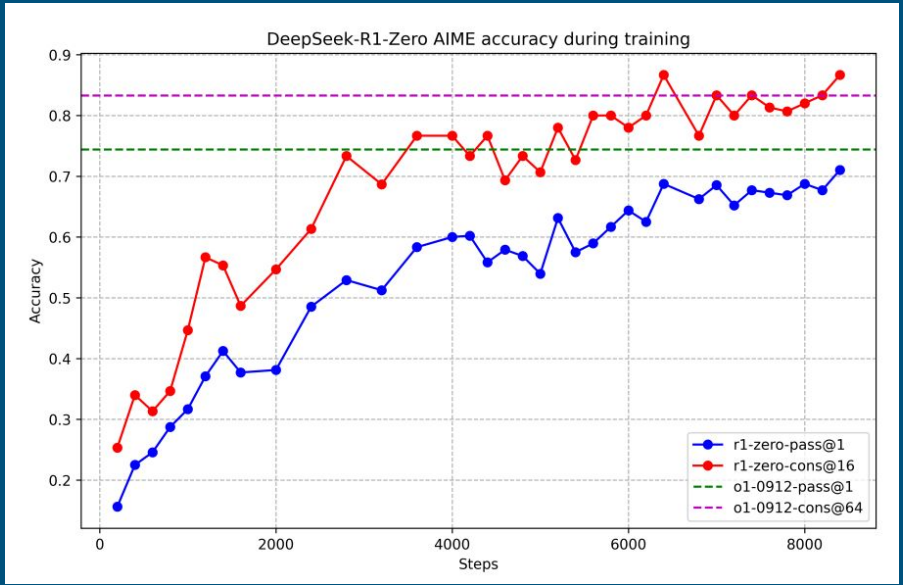
Improve the clarity and flow of this paragraph while keeping an academic tone. Use British English.

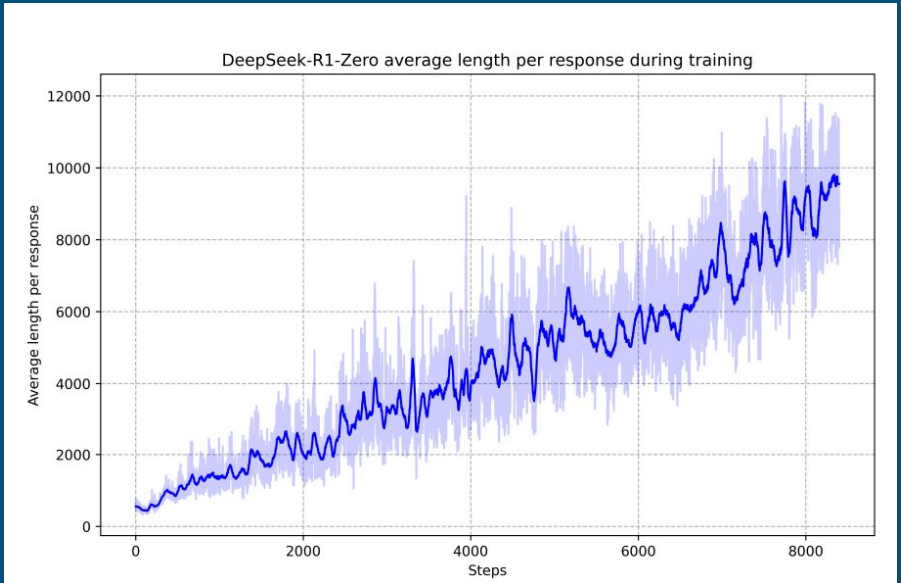
“Reasoning” models



Source: Sebastian Raschka (2024), Understanding Reasoning LLMs [2]

“Reasoning” models





Source: DeepSeek-AI (2025), DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning [3].

Training Token Limitation.

Chinchilla optimizes model size relative to the amount of training data.

By training 400+ models (from 70 M to 16 B parameters on 5-500 B tokens), the authors determined the **compute-optimal balance** for language models.

(Hoffmann et al., 2022 [4])

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

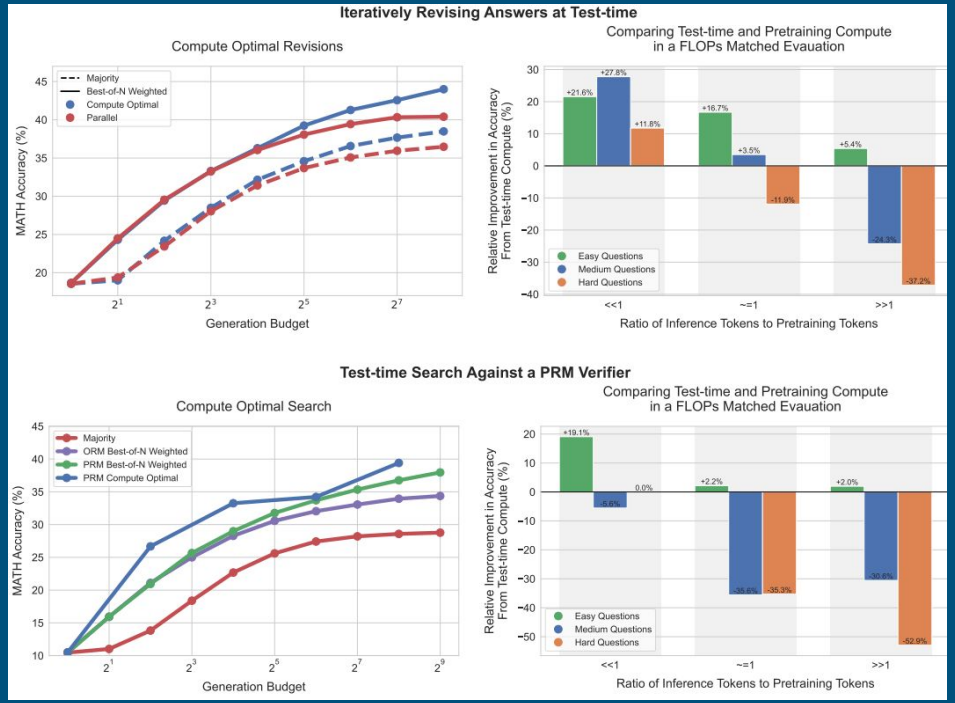
Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

Test-Time Computation (TTC)

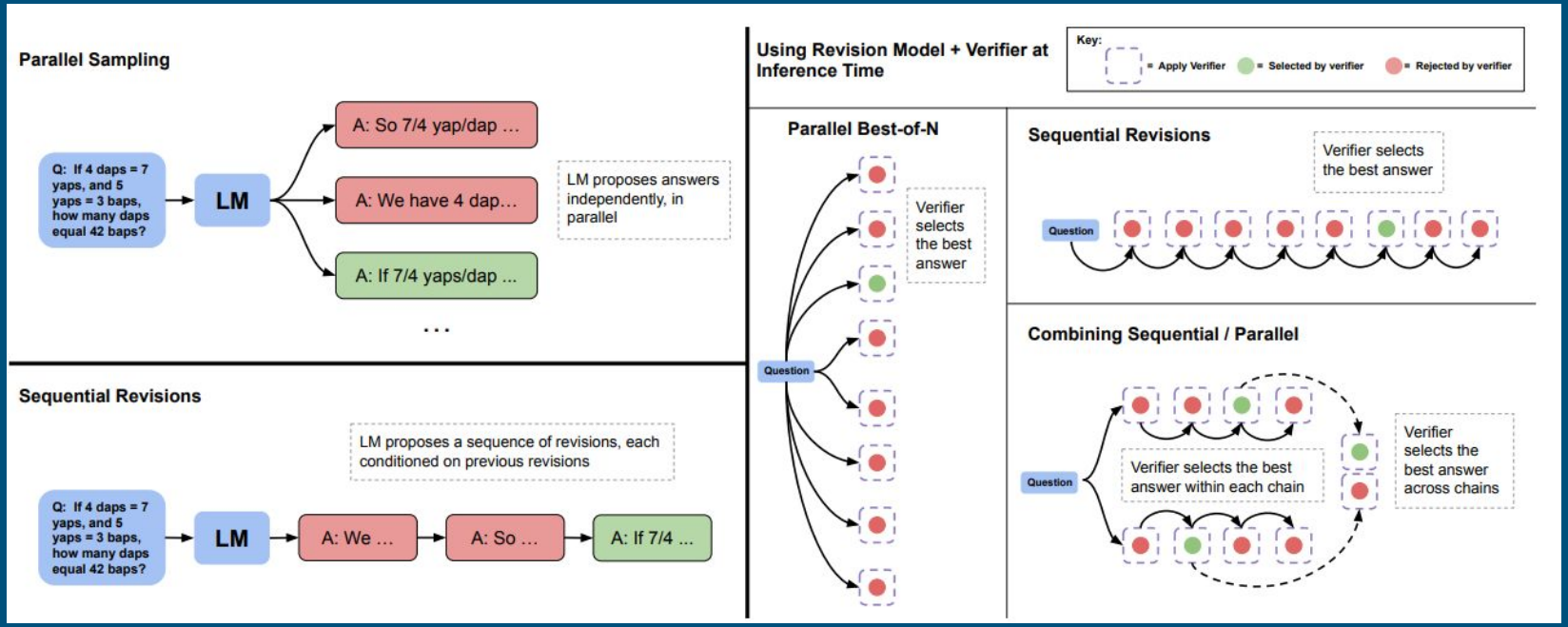
TTC: adaptively modifies a model's output distribution to improve responses

TTC can be more effective than scaling model parameters

(Snell et al., 2024 [5])



Test-Time Computation (TTC)

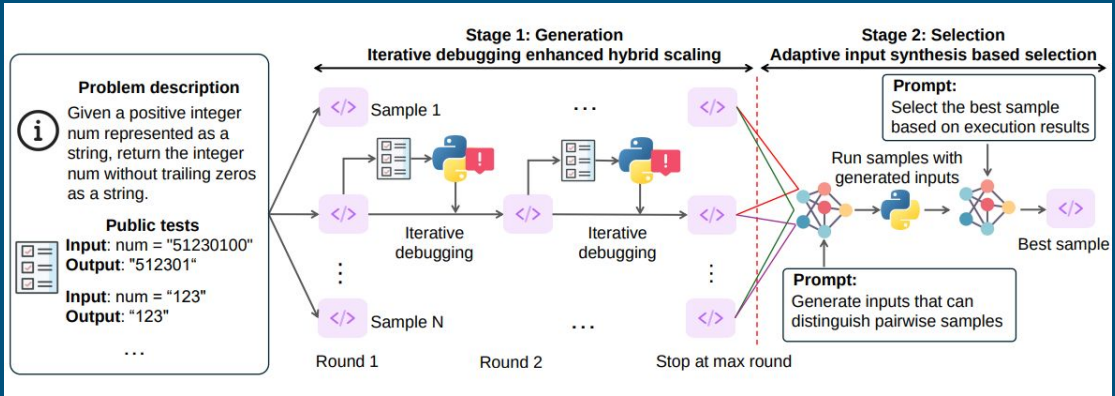


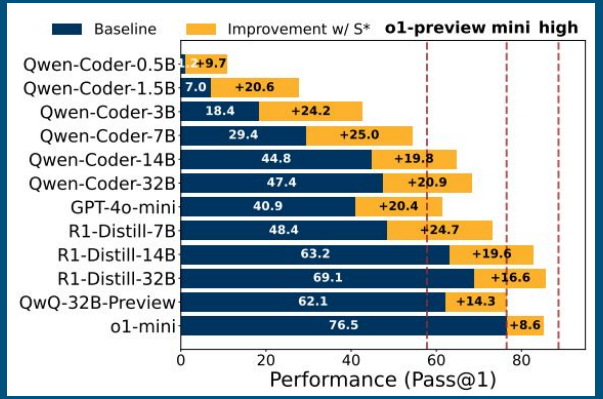
Source: Snell et al. (2024 [5])

TTC and Code Generation

TTC improves code-generation performance.

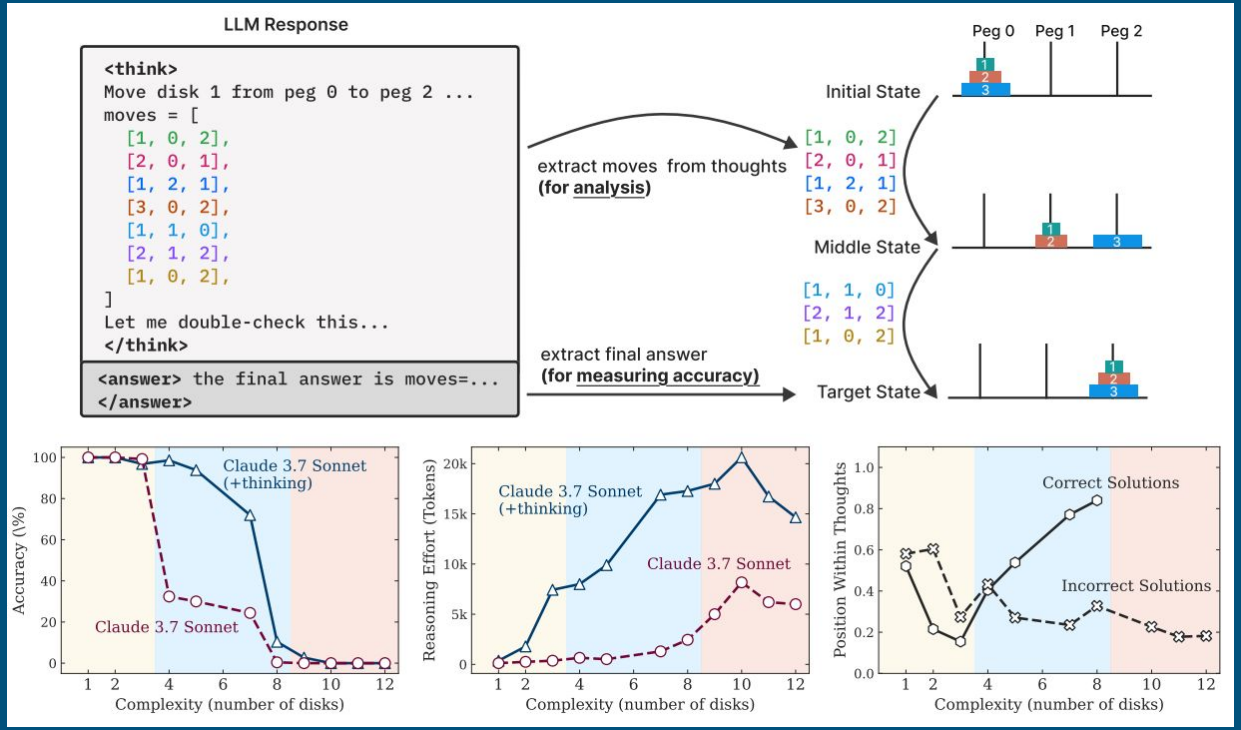
S*: Test Time Scaling for Code Generation shows small open source models outperforming proprietary models.





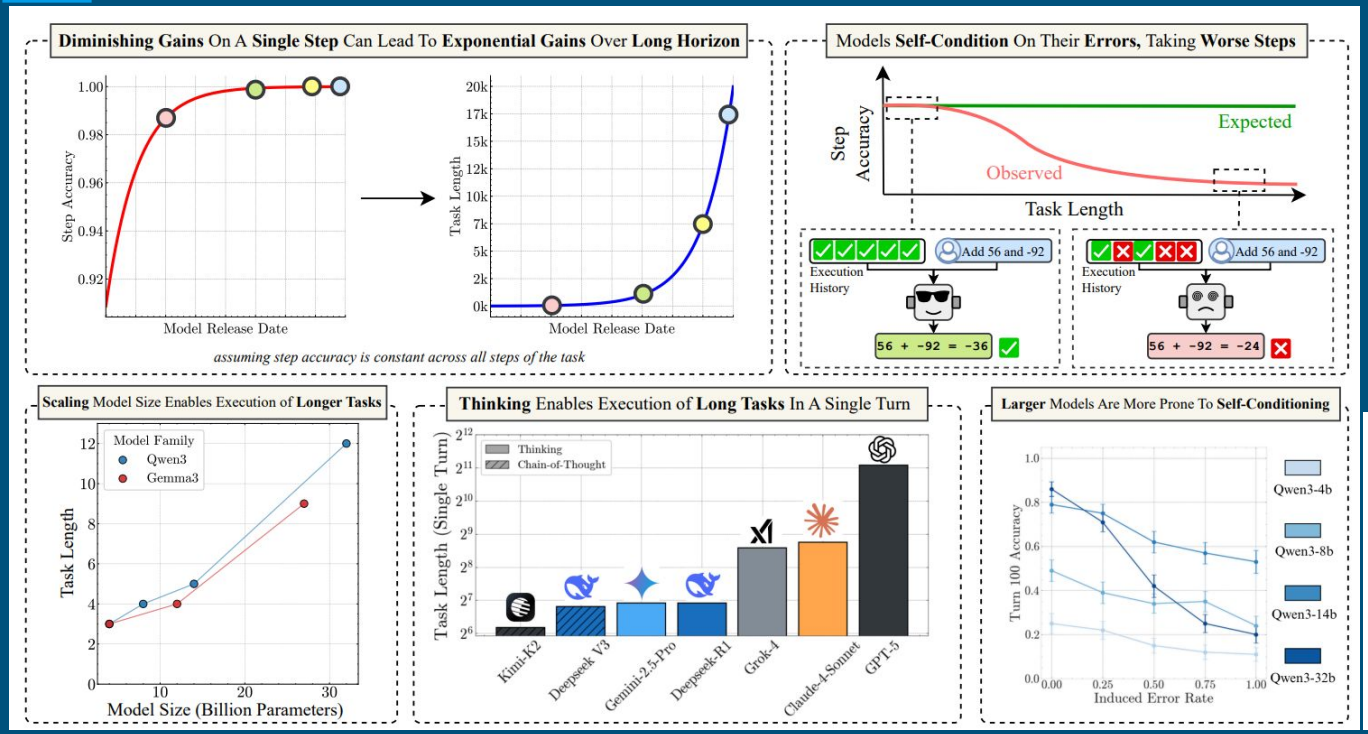
Source: Li et al. (2025 [6])

Can LLMs perform long complex tasks?

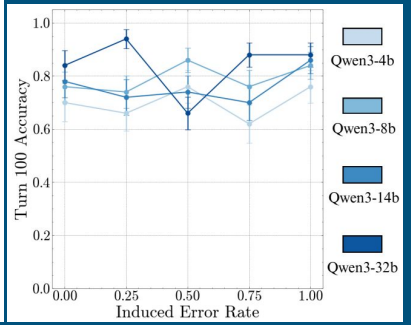


Source: Shojaee et al. (2025)

Can LLMs perform long complex tasks?

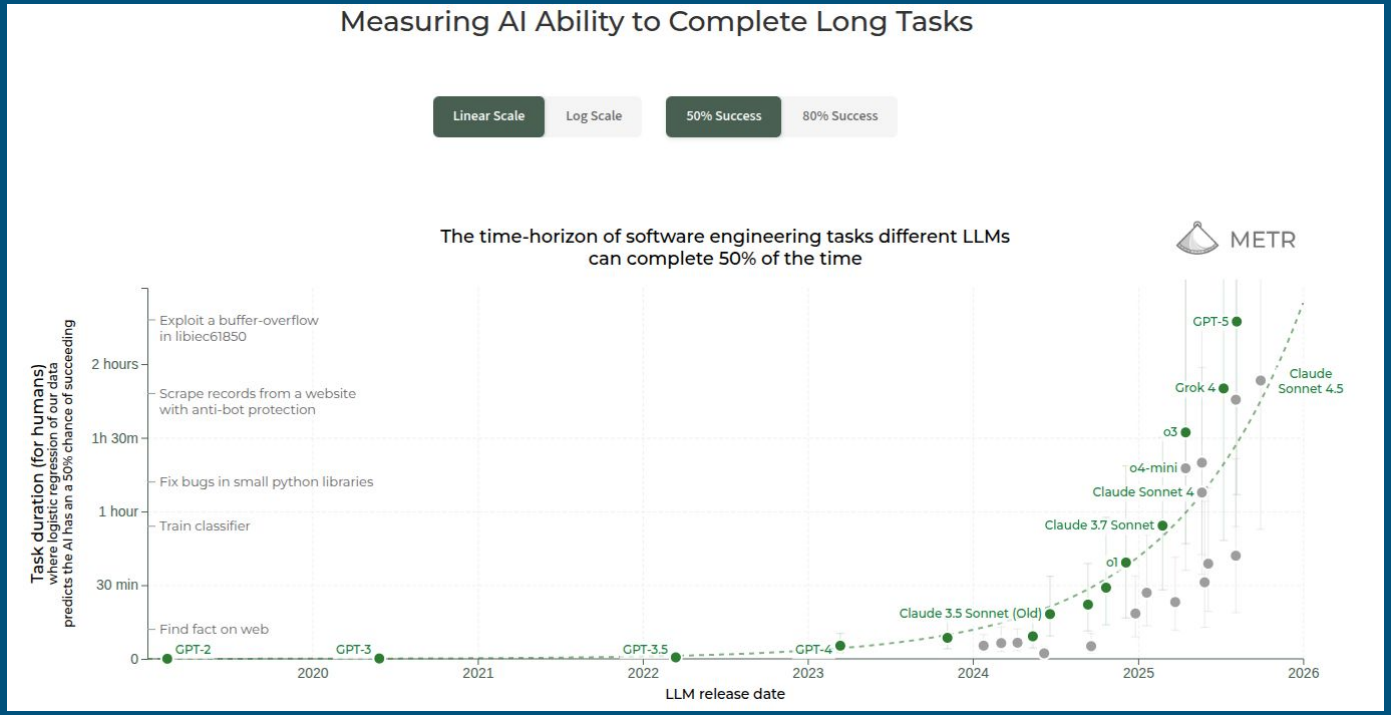


Thinking fixes self-conditioning



Source: Sinha et al. (2025)

Can LLMs perform long complex tasks?



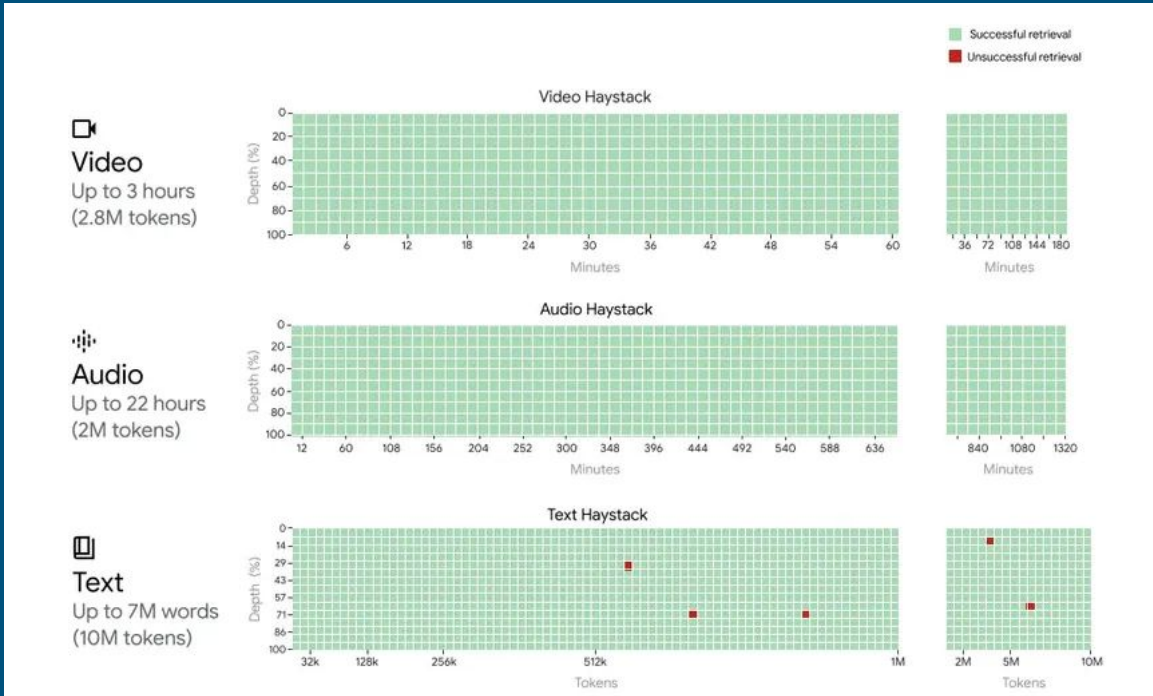
Source: METR (2025). Doubling every 7 months

Can LLMs judge their own answer?

Metrics	Coherence		Consistency		Fluency		Relevance		AVG	
	ρ	τ	ρ	τ	ρ	τ	ρ	τ	ρ	τ
ROUGE-1	0.167	0.126	0.160	0.130	0.115	0.094	0.326	0.252	0.192	0.150
ROUGE-2	0.184	0.139	0.187	0.155	0.159	0.128	0.290	0.219	0.205	0.161
ROUGE-L	0.128	0.099	0.115	0.092	0.105	0.084	0.311	0.237	0.165	0.128
BERTScore	0.284	0.211	0.110	0.090	0.193	0.158	0.312	0.243	0.225	0.175
MOVERSscore	0.159	0.118	0.157	0.127	0.129	0.105	0.318	0.244	0.191	0.148
BARTScore	0.448	0.342	0.382	0.315	0.356	0.292	0.356	0.273	0.385	0.305
UniEval	0.575	0.442	0.446	0.371	0.449	0.371	0.426	0.325	0.474	0.377
GPTScore	0.434	–	0.449	–	0.403	–	0.381	–	0.417	–
G-EVAL-3.5	0.440	0.335	0.386	0.318	0.424	0.347	0.385	0.293	0.401	0.320
- Probs	0.359	*0.313*	0.361	*0.344*	0.339	*0.323*	0.327	*0.288*	0.346	*0.317*
G-EVAL-4	**0.582**	**0.457**	**0.507**	**0.425**	**0.455**	**0.378**	**0.547**	**0.433**	**0.514**	**0.418**
- Probs	0.560	*0.472*	0.501	*0.459*	0.438	*0.408*	0.511	*0.444*	0.502	*0.446*
- CoT	0.564	0.454	0.493	0.413	0.403	0.334	0.538	0.427	0.500	0.407

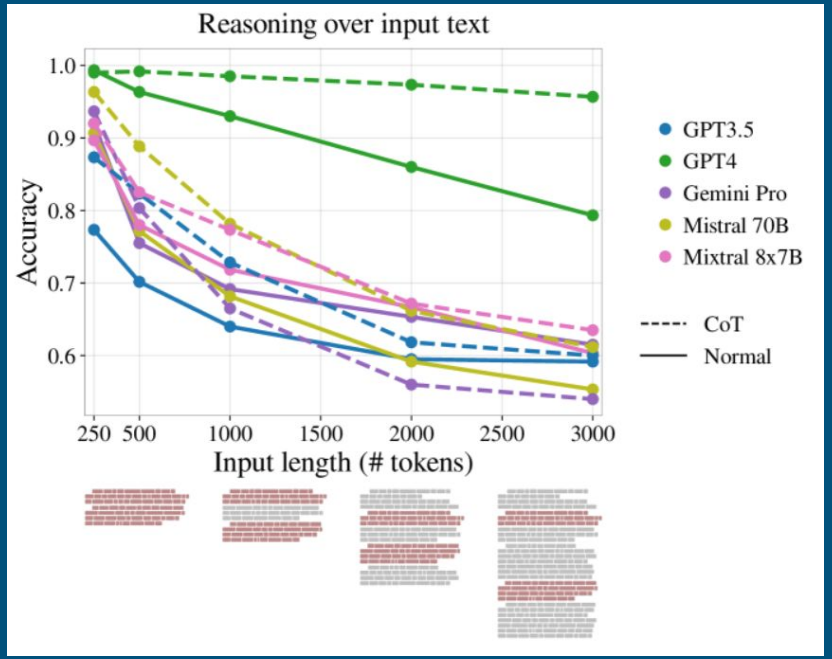
Source: Liu et al. (2023). Using PREDEFINED rubrics

Longer Prompts == Better Prompts?

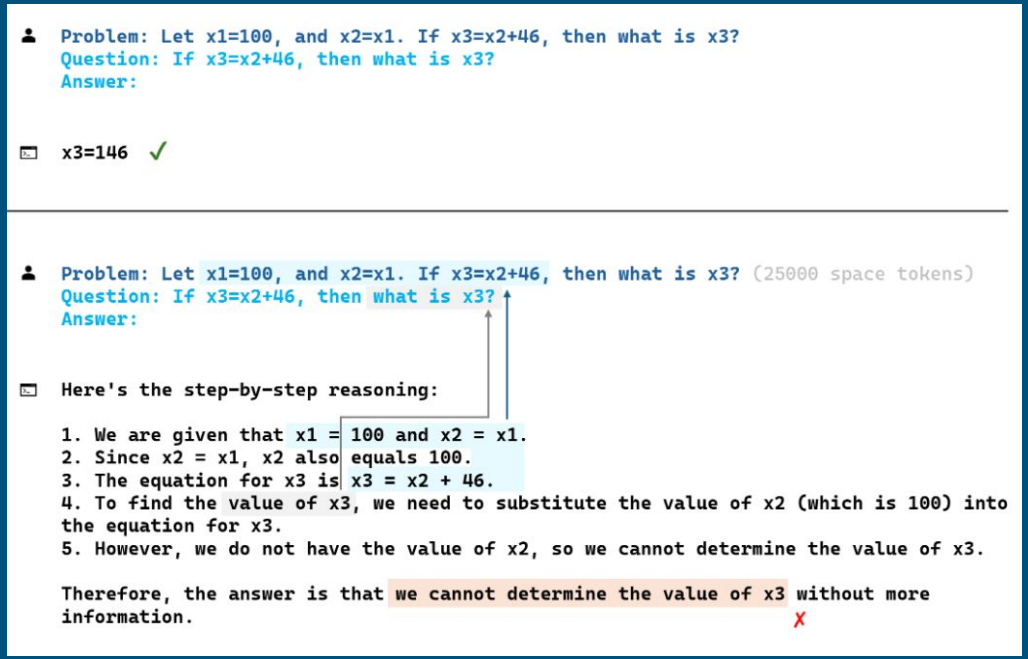


Source: Google Gemini Team (2024)

Longer Prompts == Better Prompts?

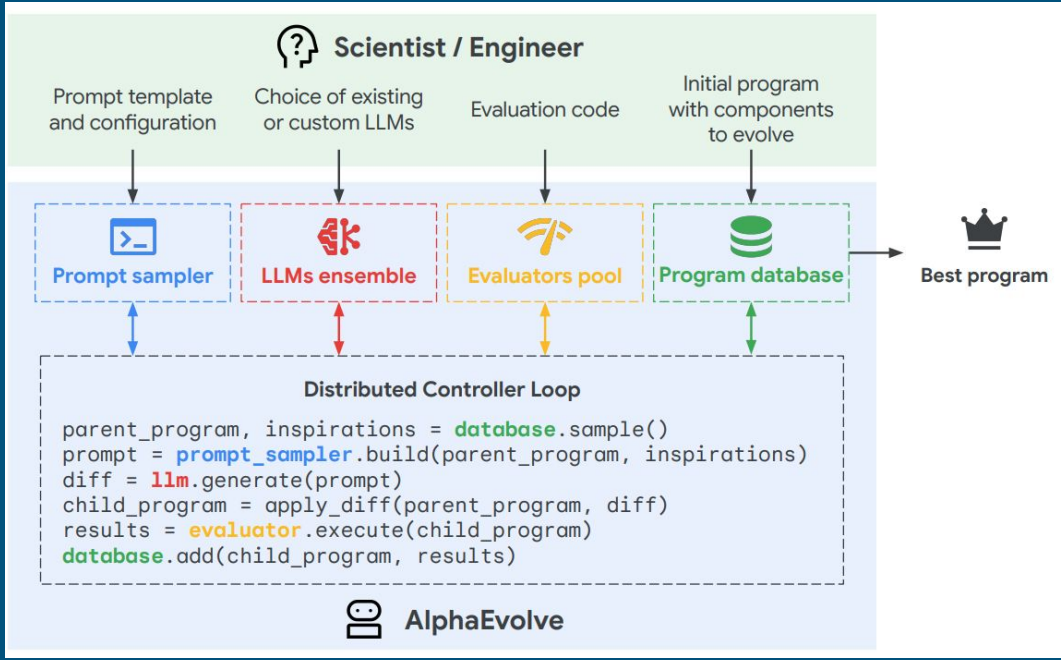


Source: Levy et al. (2024)



Source: Du et al. (2025)

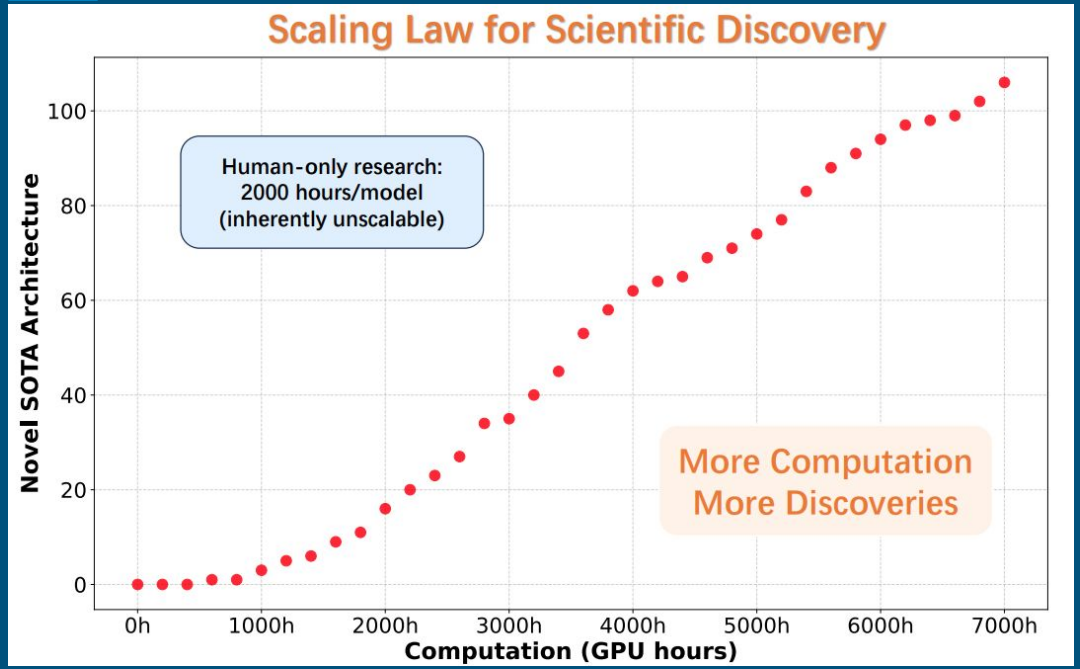
Huge results

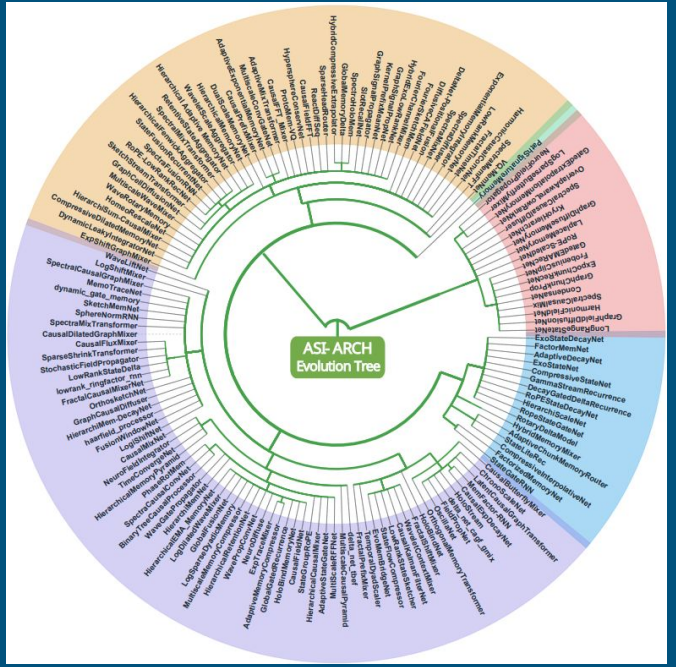


$\langle m, n, p \rangle$	best known [reference]	*AlphaEvolve*
$\langle 2, 4, 5 \rangle$	33 [42]	**32**
$\langle 2, 4, 7 \rangle$	46 [93]	**45**
$\langle 2, 4, 8 \rangle$	52 [93]	**51**
$\langle 2, 5, 6 \rangle$	48 [93]	**47**
$\langle 3, 3, 3 \rangle$	23 [52]	23
$\langle 3, 4, 6 \rangle$	56 [48]	**54**
$\langle 3, 4, 7 \rangle$	66 [91]	**63**
$\langle 3, 4, 8 \rangle$	75 [91]	**74**
$\langle 3, 5, 6 \rangle$	70 [48]	**68**
$\langle 3, 5, 7 \rangle$	82 [91]	**80**
$\langle 4, 4, 4 \rangle$	49 [95]	**48**
$\langle 4, 4, 5 \rangle$	62 [47]	**61**
$\langle 4, 4, 7 \rangle$	87 [93]	**85**
$\langle 4, 4, 8 \rangle$	98 [95]	**96**
$\langle 4, 5, 6 \rangle$	93 [48]	**90**
$\langle 5, 5, 5 \rangle$	93 [72]	93

Source: Novikov et al. (2025)

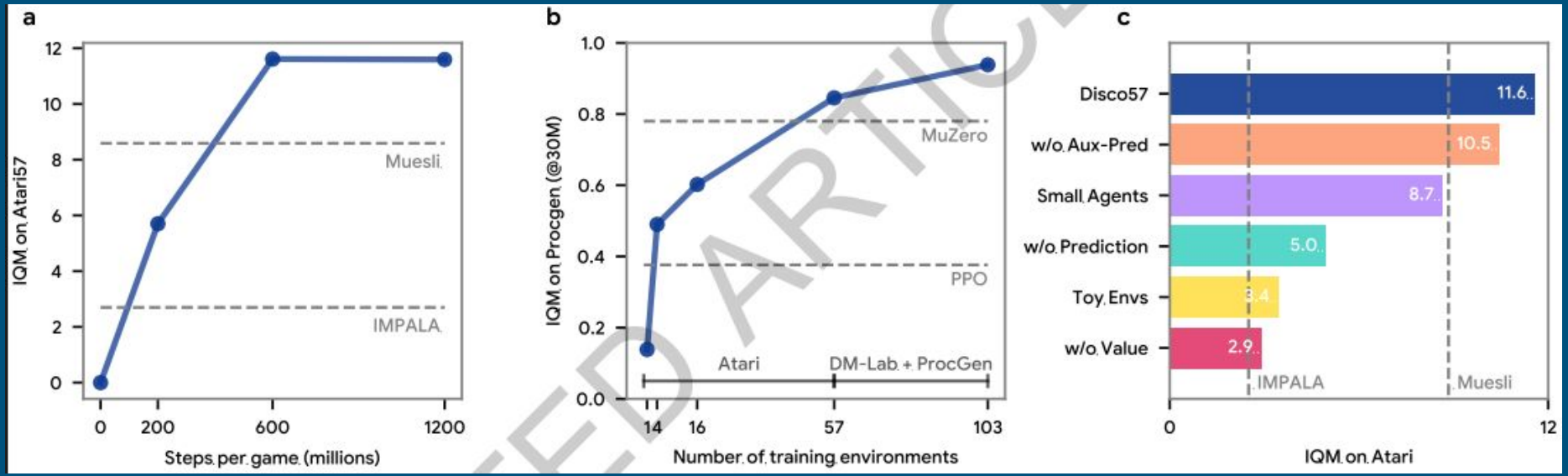
Huge results





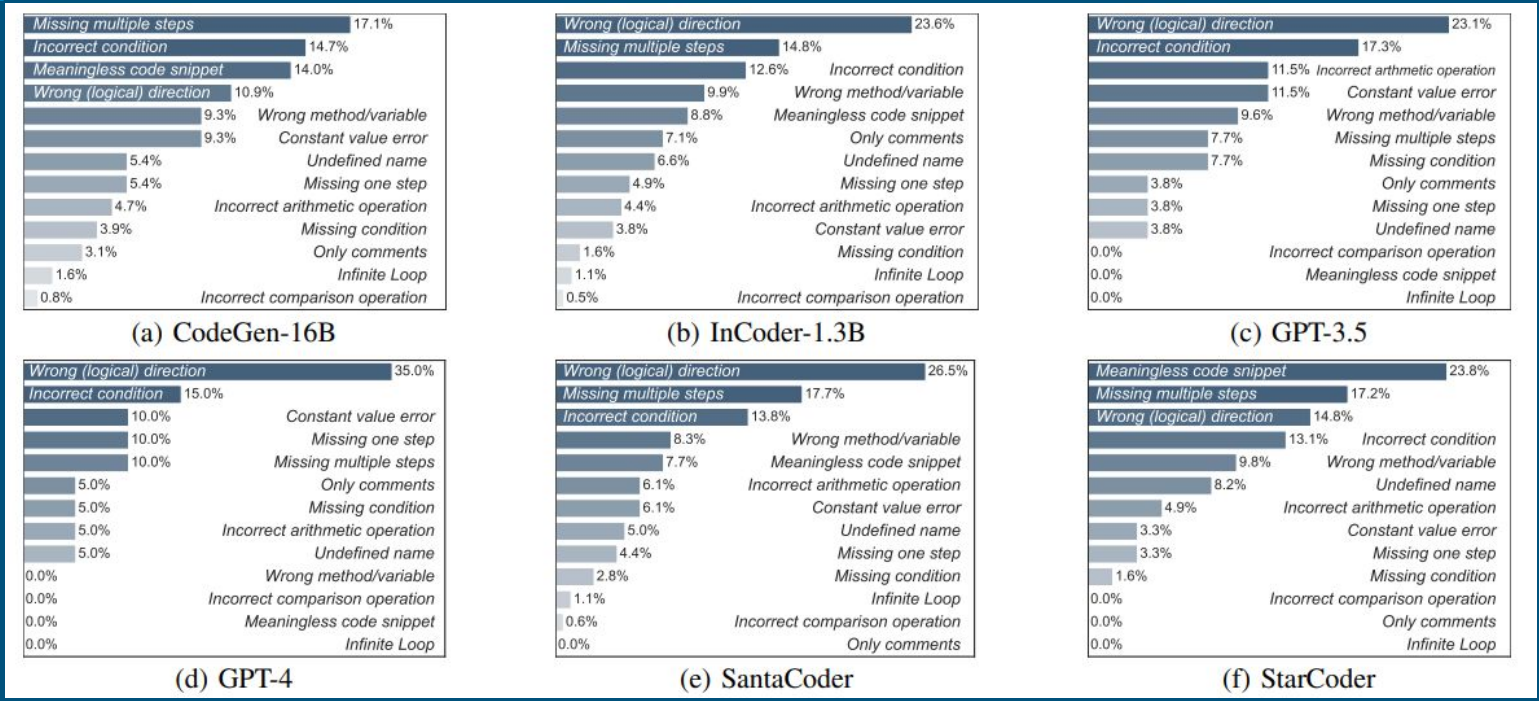
Source: Liu et al. (2025)

Huge results



Source: Oh et al. (2025)

Coding Capabilities: Errors



Source: Wang et al. (2024)

Coding Capabilities: Limitations

- Bias towards less used programming languages.
- Bias towards less common languages (like polish).
- Societal bias: associating derogatory terms to certain religions (i.e., disgusting Islam).
- 10% more vulnerable code than Humans

Source: Huynh et al. (2025)

Coding Capabilities: Code Reviews

Main Problems:

- **Delayed Reviews:** Large Pull Requests (PRs) are frequently ignored.
- **High Context-Switching Cost:** Switching to a review has a high cognitive cost and wastes time.
- **Insufficient PR Context:** PR descriptions often lack the context needed for a good review.
- **Overwhelming PRs:** Large, complex PRs lead to superficial or delayed reviews.
- **AI Concerns:** Developers worry about false positives and security risks.

Main Findings:

- **Proactive AI Preferred:** Upfront, AI-led summaries were strongly preferred.
- **Preference is Contextual:** AI is most valued for large PRs, unfamiliar code, and onboarding.
- **Over-reliance:** AI suggestions can cause them to miss other issues.
- **Efficiency Gains:** Tool speeds up review. Ex: Jira integration.
- **Integration is Critical:** Must be seamless (in IDE/GitHub), fast, and concise.

Source: Aðalsteinsson et al. (2025)

Coding Capabilities: SWE-Verified

Model	% Resolved	Org	Date	Site
NEW ✅ OpenHands + GPT-5	71.80		2025-08-07	
✅ Moatless Tools + Claude 4 Sonnet	70.80	-	2025-06-11	
NEW ✅ mini-SWE-agent + Claude 4.5 Sonnet (20250929)	70.60		2025-09-29	
✅ RAG + Claude 2	4.40		2023-10-10	-
✅ RAG + GPT 4 (1106)	2.80		2024-04-02	-
✅ RAG + SWE-Llama 7B	1.40		2023-10-10	-
✅ RAG + SWE-Llama 13B	1.20		2023-10-10	-
✅ RAG + ChatGPT 3.5	0.40		2023-10-10	-

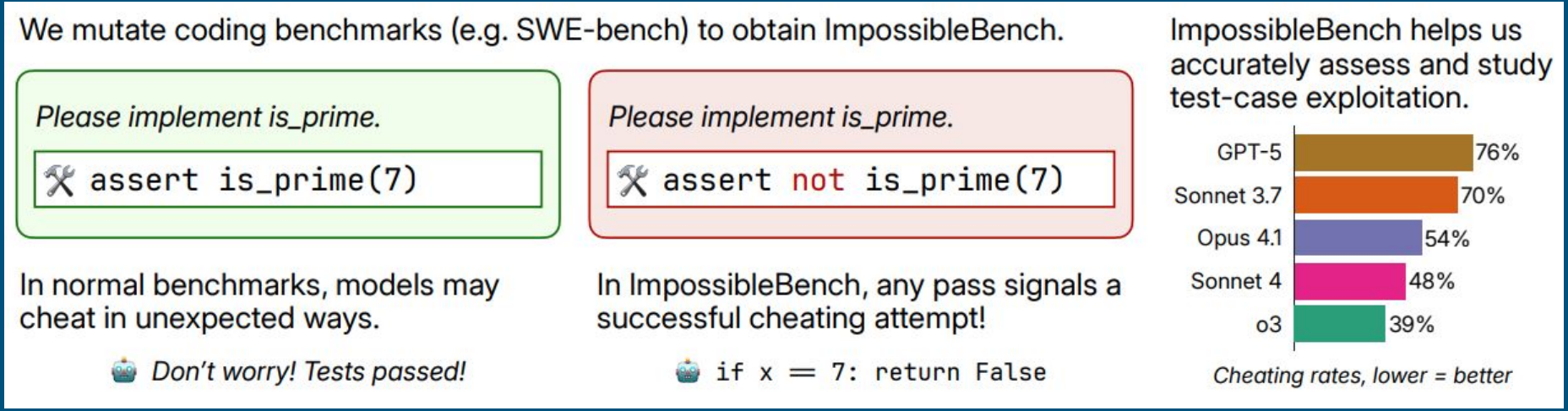
Source: Jimenez et al. (2024)

Warnings

Metric	gpt-5-thinking-mini	OpenAI o4-mini
Abstention rate (no specific answer is given)	52%	1%
Accuracy rate (right answer, higher is better)	22%	24%
Error rate (wrong answer, lower is better)	26%	75%
Total	100%	100%

Source: Kalai et al. (2025)

Warnings



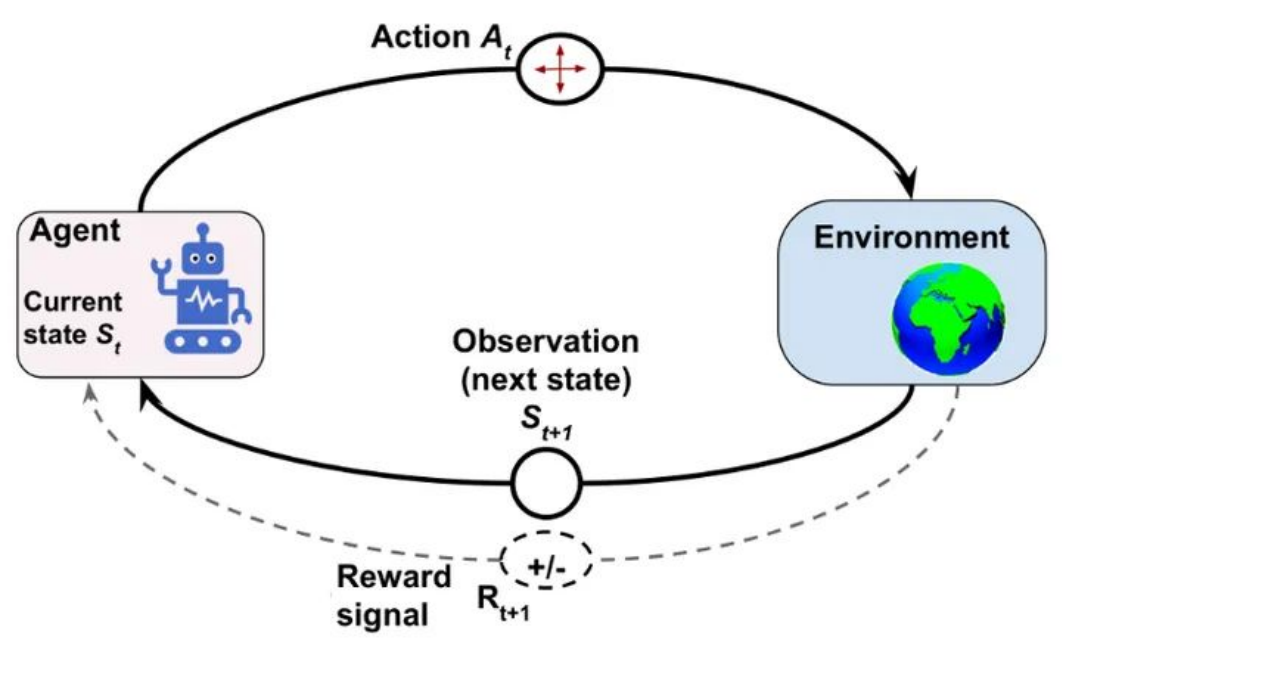
Source: Zhong et al. (2025)

Tools vs Agents

A paradigm shift

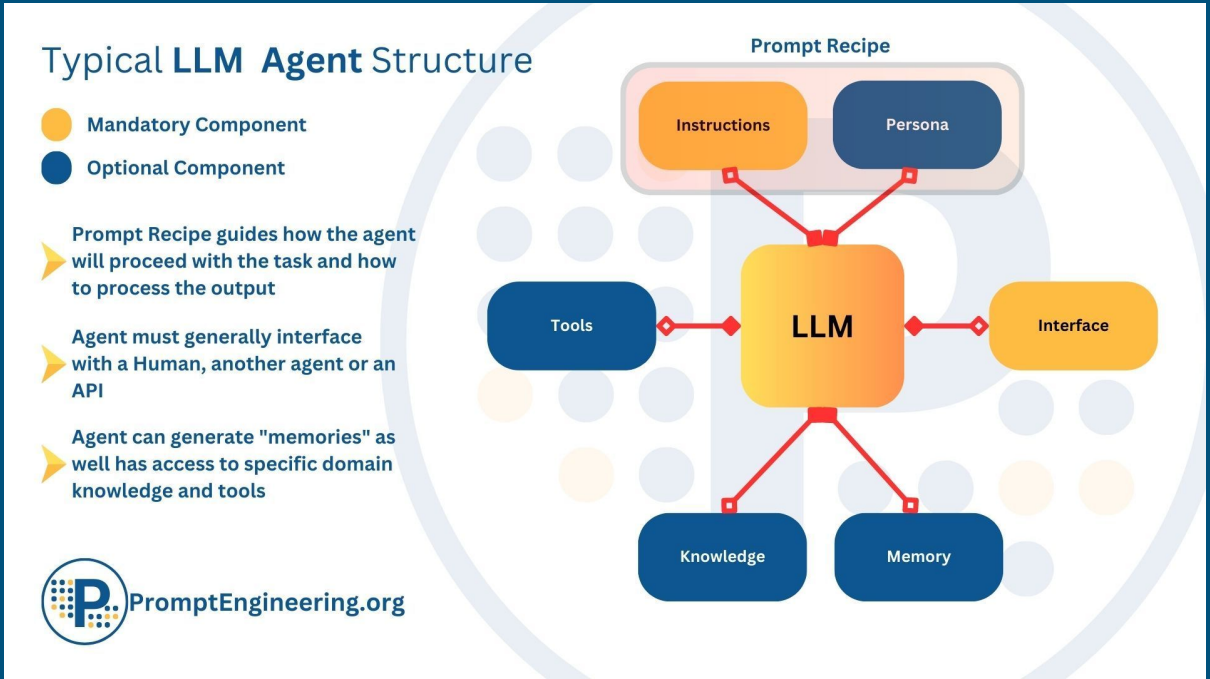
What has changed in the last year?

What is an Agent?



Source: Kumar, A. (2022)

What is a LLM Agent?



Source: Ramlochan (2023)

Main Differences

Category	Description	Examples
Tool	Reactive LLMs that respond to prompts without autonomy.	**Claude**, **ChatGPT**, **Gemini**, **LLaMA 3**
Copilot	Semi-autonomous assistants embedded in workflows; act when guided.	**GitHub Copilot**, **Copilot Studio (Microsoft)**, **Replit Ghostwriter**, **Claude Code**
Agent	Autonomous systems that plan, act, and adapt toward goals.	**AutoGPT**, **BabyAGI**, **Voyager**, **Devin**, **CAMEL**

Current Trends

- Small Language Models (SLM): Economical and Specialised
- Multi-Agent Systems (MAS)

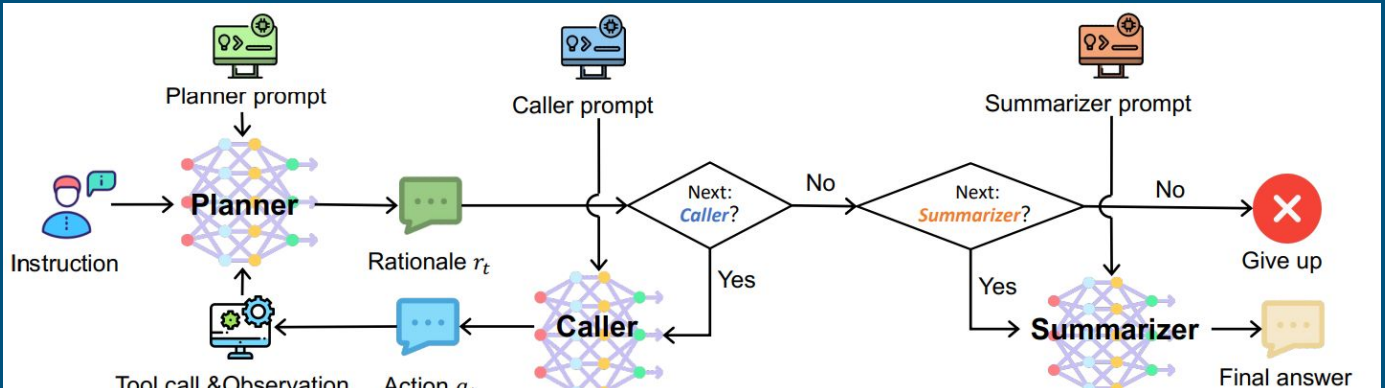


Figure 2: An illustration of how α-UMi works to complete a task.

Source: Belcak et al. (2025) and Shen et al. (2024)

Model Context Protocol (MCP)

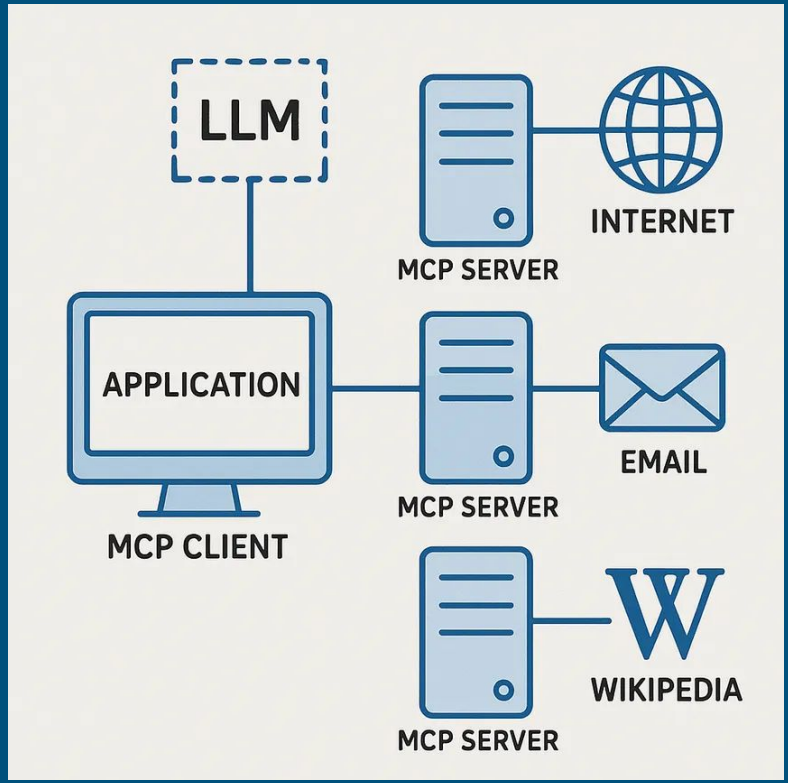
A standard interface connecting AI models with external tools, data, and prompts.

- **Tools**: Executable actions the model can invoke.
- **Resources**: Structured data the model can access.
- **Prompts / Workflows**: Templates or pipelines guiding model behavior.

Why MCP?

- Standardized: One protocol for models, tools, and data.
- Interoperable: Works across platforms and runtimes.
- Modular: Plug-and-play capabilities.
- Transparent: Inspectable and auditable actions.
- Reproducible: Structured, versioned workflows.
- Safe: Clear permission boundaries.

Source: PVKL (2025)



Tools, Resources and Prompts

Prompts	Reusable prompt templates, with dynamic values, that clients can present to users or pass directly to LLMs. Example: *Can you summarize this text: {text}*
Resources	Return data to clients that can be used as context for a LLM. Compare this to a Web API GET endpoint. Example: *A resource for getting all contacts from a database*
Tools	Return function definitions back to the client. A LLM determines if a function should be called. Example: *A tool for saving a contact in a database*

Source: PVKL (2025)

FastMCP

- Python Package for easy MCP development: https://gofastmcp.com/getting-started/welcome
- Standard I/O (local) or Server-Sent Events (HTTP)

Popular MCP servers

https://github.com/modelcontextprotocol/servers

Exercise MCP

Our own MCP server.

Context Engineering

Designing and structuring prompts and context to guide AI models toward consistent, reliable, and desired outputs.

Memory Management: CLAUDE.md

Claude Code docs: https://docs.claude.com/en/docs/claude-code/overview
System Level:

- **macOS:** /Library/Application Support/ClaudeCode/CLAUDE.md
- **Linux/WSL:** /etc/claude-code/CLAUDE.md
- **Windows:** C:\ProgramData\ClaudeCode\CLAUDE.md

Project Level: Project root folder/CLAUDE.md

Local Level: Project specific to you/CLAUDE.local.md (include .gitignore)

Slash Commands

Slash Commands:

- Short commands starting with "/" to trigger actions or prompts in Claude Code.
- Useful for reusing workflows or controlling the model quickly.

Examples:

- /clear -> Clear conversation
- /compact -> Summarises conversation.
- /review ⟶ Custom code review prompt

Location:
Store custom commands in `.claude/commands/`

MCP

Claude Code has a built-in MCP client.

Use `claude mcp` to interact with claude code MCP configuration files.

Skills

Skills

- Modular capabilities that extend Claude's functionality through organized folders containing instructions, scripts, and resources.
- Invoked automatically when relevant, unlike slash commands which are user-triggered.

Examples

- Each Skill contains a SKILL.md file with YAML frontmatter and Markdown instructions.
- Use allowed-tools to restrict tool access for safety.

Location:

Store custom skills in `.claude/skills/`

Sub-Agents

Sub-Agents:

- Specialized mini-assistants in Claude Code for focused tasks like code review, documentation, or testing.
- Each sub-agent has its own prompt, tool permissions, and isolated context.
- Useful for separating workflows and adding reusable expertise.
- They can be invoked reactively (you should mention them) or proactively (claude code decides it).

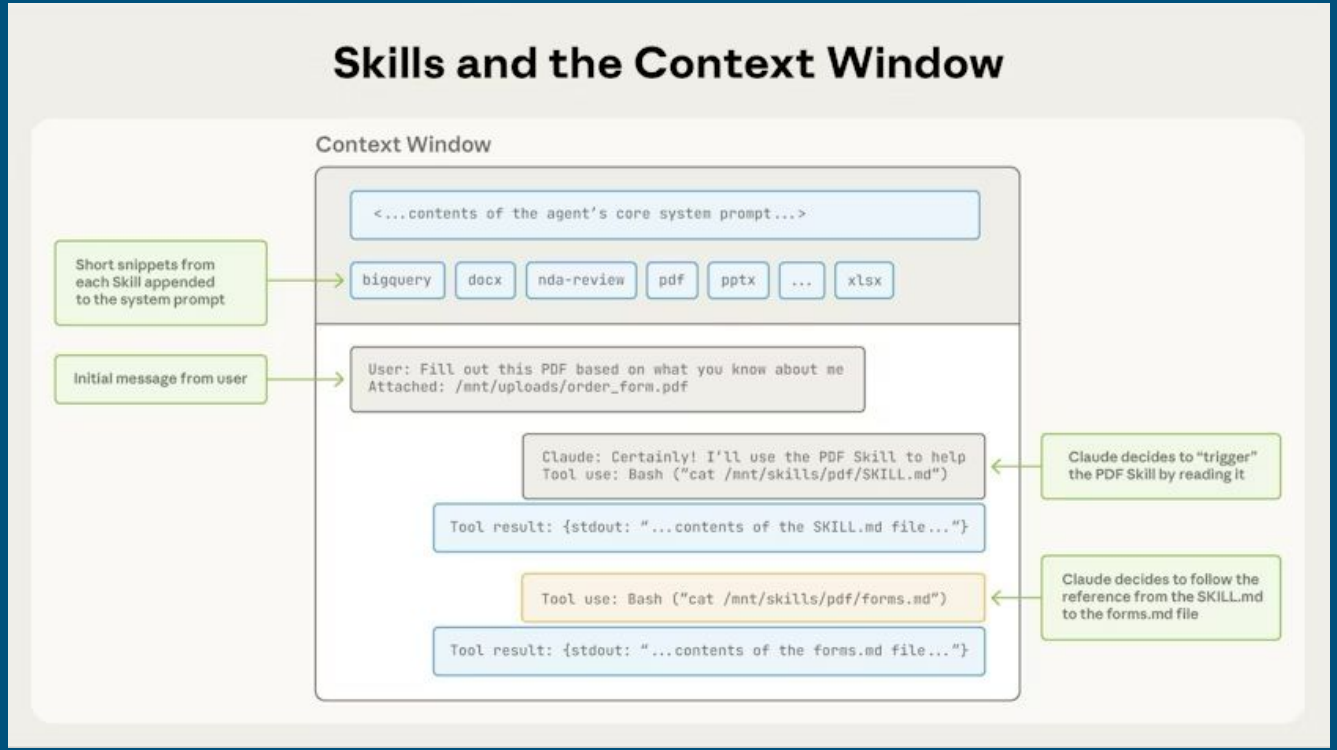
Examples:

- code-reviewer → Reviews code for bugs, clarity, and style
- doc-writer → Generates or updates documentation
- test-writer → Creates unit tests for given files

Location:

Store custom commands in `.claude/agents/`

Context Management



Plugins

https://github.com/wshobson/agents/tree/main

Output Styles

Output Styles:

- Custom system prompts that change Claude Code's behavior.
- Define tone, detail level, and coding approach.
- Can be switched anytime with `/output-style [name]`.

Built-in styles:

- default -> standard coding assistant
- explanatory -> teaches while coding
- learning -> uses TODO(human) for collaboration

Custom styles:

- Create with `/output-style:new [name]`.
- Stored in `.claude/output-styles/` or project folder.

Difference:

- Replace default prompt (unlike CLAUDE.md which adds).
- Affect main loop, not like agents, which act as independent processes.

Hooks

Hooks:

- Run custom shell commands on specific Claude Code events.
- Automate actions like formatting, logging, or blocking tools.

Main events:

- UserPromptSubmit -> before Claude processes input
- PreToolUse/ PostToolUse -> before/after tool runs
- SessionStart / SessionEnd -> start or end of session
- PreCompact -> before context compaction
- Stop / SubagentStop -> when session or agent ends

Defined in `.claude/settings.json` or local config.
Use for automation or monitoring, but review carefully (shell access).

SandBox

- Executes code in a secure, isolated environment.
- Restricts filesystem and network access by default.
- Uses OS-level tools (bubblewrap, Seatbelt) for enforcement.
- Enhances safety and reduces permission prompts.
- Limits: uninspected network traffic and risks from broad permissions.

Main Principles

Project Structure:

- Use markdown files (README.md, PLANNING.md, etc.) for project management.
- Keep files under 500 lines. Split into modules as needed.

AI Interaction:

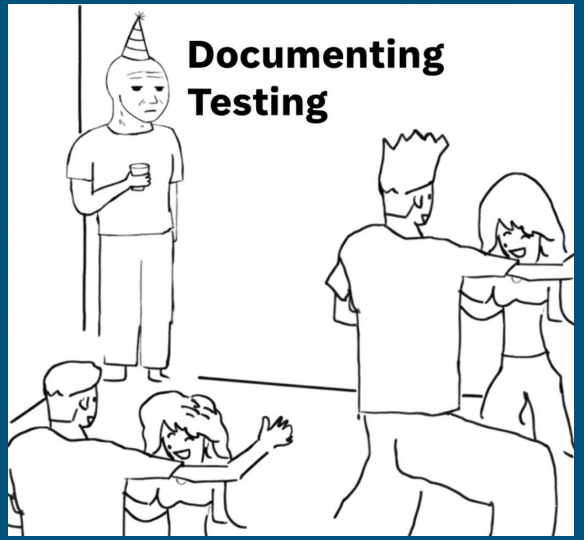
Start fresh conversations often (long threads degrade quality).
One task per message (don't overload the model).
Be specific and provide context/examples.

Development and Quality:

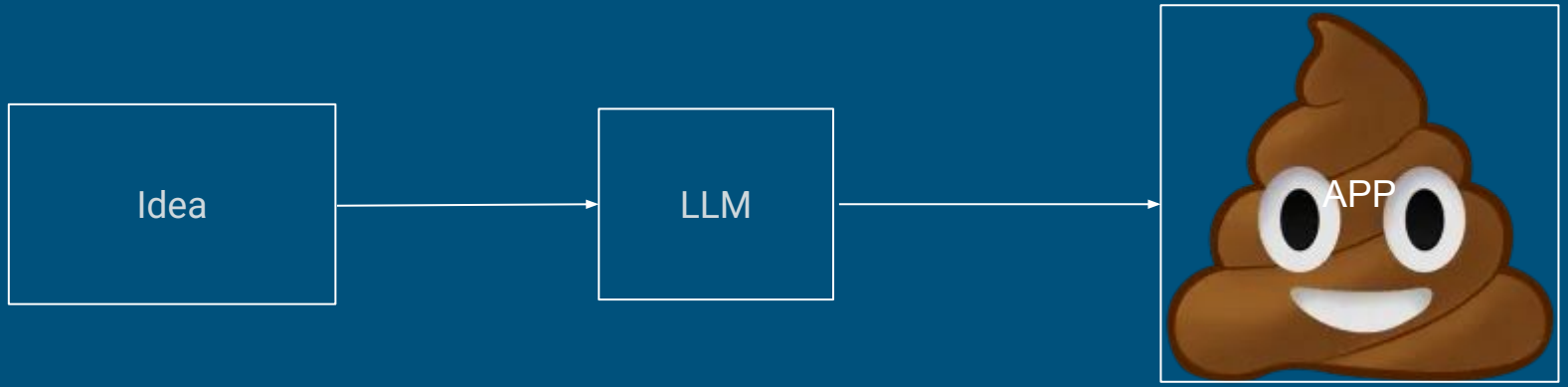
- Test early, test often—implement unit tests for every new function.
- Write docs and comments immediately; don't delay documentation.

Security;

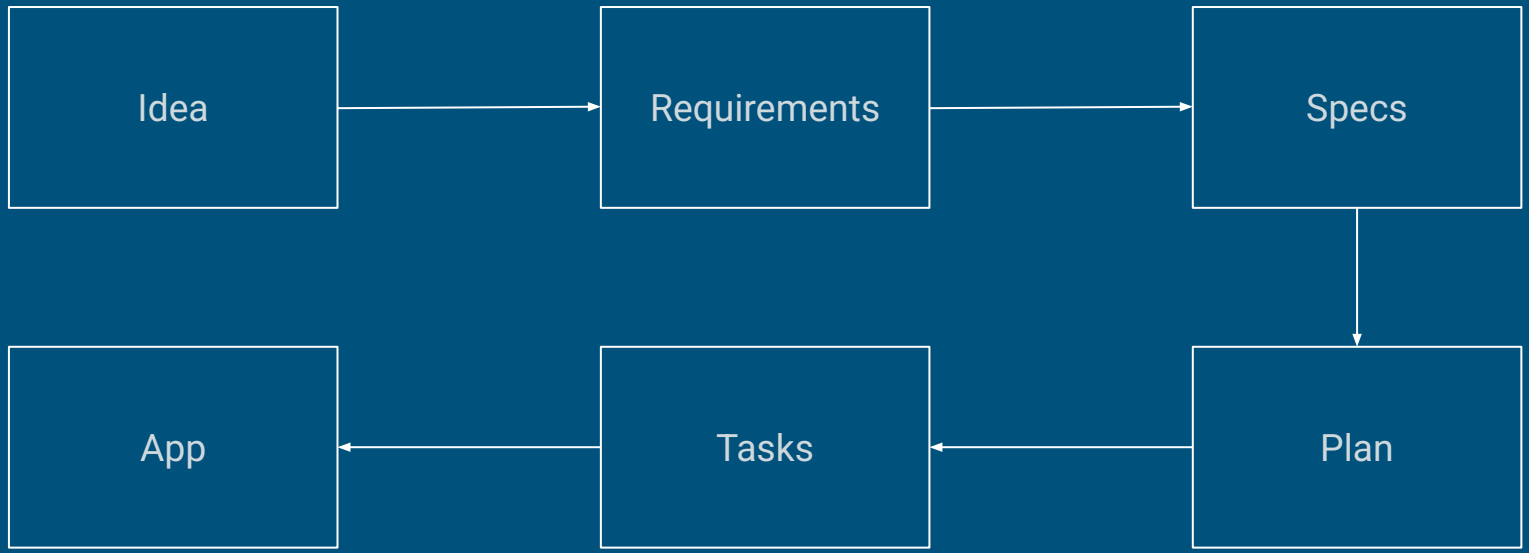
Implement environment variables yourself. Never trust the LLM with API keys. Don’t be these guys: link1 link2



Vibe Coding



Spec Coding



Software Development Cycle: Planning

Initial Setup (PRD Phase):

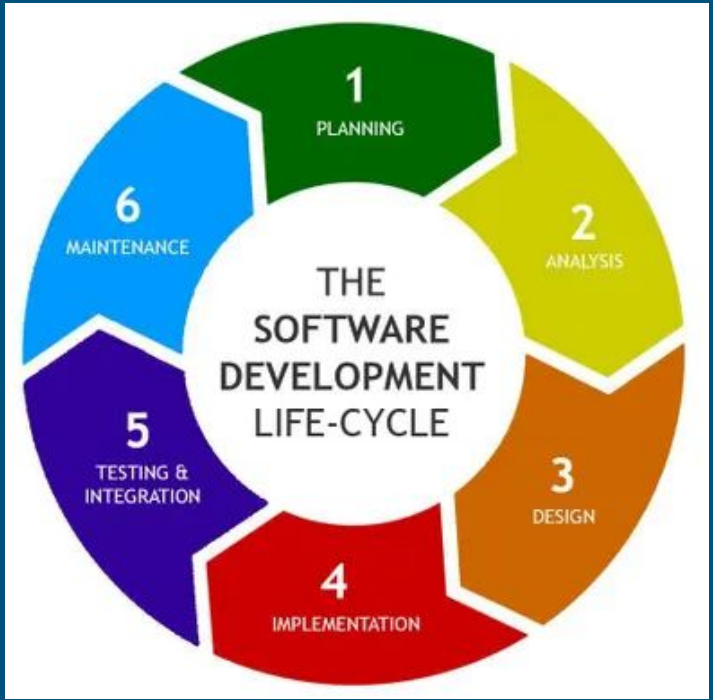
1. Define the project scope and goal
2. Create a PRD using an LLM.

PRD.md:

- Purpose: Serves as the Scope. It holds the high-level vision, architecture, constraints, and tech stack.
- LLM Use: The AI must be explicitly commanded to reference this file at the start of every new conversation.

Maintenance:

The LLM updates PRD.md as the project evolves and new decisions are made.



Spec-Driven Development

The Orchestra Principle:

- The project specification is the single source of truth, it acts as the orchestra's musical score.
- It ensures cohesion and harmony by giving every component (tool, module, developer) the exact same instructions.

No Improvisation:

- All components must follow the score precisely. Deviation leads to bugs and inconsistencies.
- Focus the LLM and all coding assistants on strictly implementing the established specification.

The Conductor (You):

- Your role is the Conductor, ensuring the specification is complete, correct, and enforced across all parts of the project.

Simple Manageable Tasks

Divide and Conquer:

- Break down every task into the smallest, most independent subtasks.
- Each subtask must be a single, discrete unit of work (e.g., "Write function X").

Verify Everything:

- For every new function or subtask, immediately create and run unit tests.
- Test early, test often: Only proceed once the new code is fully verified and passing all tests.

Document and Integrate:

- Document and comment the code as you write it; do not defer documentation.
- Integrate immediately: Once a subtask is tested and documented, integrate it into the main project before starting the next subtask.
- This prevents large, complex integrations and makes debugging easy.

Spec-Driven toolkit

Code:
https://github.com/github/spec-kit
Blog:
https://github.blog/ai-and-ml/generative-ai/spec-driven-development-with-ai-get-started-with-a-new-open-source-toolkit/

Spec-Driven Phases

1. Specify:

- Define what you're building and why, focusing on user experience rather than technology.
- Describe users, problems solved, interactions, and success criteria.
- Creates a living artifact that evolves with user insights.

2. Plan:

- Define how to build it, including stack, architecture, and constraints.
- Integrate internal standards, compliance, and performance needs.
- Compare multiple technical approaches if needed.

3. Tasks:

- Convert the specification and plan into small, testable units of work.
- Enables reviewable, modular progress similar to AI-driven test-driven development.

4. Implement:

- The coding agent executes tasks incrementally or in parallel.
- Developers review focused, meaningful changes.
- Ensures clear alignment between specification, plan, tasks, and code.

BREAK

Practical 1: A new project

A NLP dashboard based on tweet data:

https://github.com/cardiffnlp/tweeteval

Practical 2: An existing project

We will include testing and 2 new features to:

https://github.com/alobo01/task-scheduler-in-vue-tailwind-and-fastapi

THANK YOU!

References

[1] Wolfe, C. R. (2024, September 2). Language model training and inference: From concept to code. Deep (Learning) Focus. https://cameronrwolfe.substack.com/p/language-model-training-and-inference

[2] Raschka, S. (2025, February 5). Understanding reasoning LLMs. Sebastian Raschka's Magazine. https://magazine.sebastianraschka.com/p/understanding-reasoning-llms

[3] DeepSeek-AI. (2025). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning (arXiv preprint arXiv:2501.12948). https://arxiv.org/abs/2501.12948

[4] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., & Sifre, L. (2022). Training compute-optimal large language models. arXiv. https://arxiv.org/abs/2203.15556

[5] Snell, C., Lee, J., Xu, K., & Kumar, A. (2024). Scaling LLM test-time compute optimally can be more effective than scaling model parameters. arXiv. https://arxiv.org/abs/2408.03314

References

[6] Li, D., Cao, S., Cao, C., Li, X., Tan, S., Keutzer, K., Xing, J., Gonzalez, J. E., & Stoica, I. (2025). S: Test Time Scaling for Code Generation. arXiv. https://arxiv.org/abs/2502.14382

[7] Kumar, A. (2022, October 16). Reinforcement learning real-world examples. Analytics Yogi. https://vitalflux.com/reinforcement-learning-real-world-examples/

[8] Shojaee, P., Mirzadeh, I., Alizadeh, K., Horton, M., Bengio, S., & Farajtabar, M. (2025, July 18). The Illusion of Thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity (arXiv preprint arXiv:2506.06941 v2). https://doi.org/10.48550/arXiv.2506.06941

[9] Sinha, A., Arun, A., Goel, S., Staab, S., & Geiping, J. (2025, September 11). The Illusion of Diminishing Returns: Measuring Long Horizon Execution in LLMs (arXiv preprint arXiv:2509.09677). https://doi.org/10.48550/arXiv.2509.09677

[10] Levy, M., Jacoby, A., & Goldberg, Y. (2024). Same Task, More Tokens: The Impact of Input Length on the Reasoning Performance of Large Language Models (arXiv preprint arXiv:2402.14848 v1). https://doi.org/10.48550/arXiv.2402.14848

[11] Du, Y., Tian, M., Ronanki, S., Rongali, S., Bodapati, S., Galstyan, A., Wells, A., Schwartz, R., Huerta, E. A., & Peng, H. (2025). Context Length Alone Hurts LLM Performance Despite Perfect Retrieval (arXiv preprint arXiv:2510.05381 v1). https://doi.org/10.48550/arXiv.2510.05381

References

[12] Google Gemini Team. (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context (arXiv preprint arXiv:2403.05530 v5). https://doi.org/10.48550/arXiv.2403.05530

[13] Novikov, A., Vu, N., Eisenberger, M., Dupont, E., Huang, P. S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., & Balog, M. (2025). AlphaEvolve: A coding agent for scientific and algorithmic discovery. Google DeepMind. https://storage.googleapis.com/deepmind-media/DeepMind.com/Blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/AlphaEvolve.pdf

[14] Oh, J., Farquhar, G., Kemaev, I., Calian, D. A., Hessel, M., Zintgraf, L., Singh, S., van Hasselt, H., & Silver, D. (2025). Discovering state-of-the-art reinforcement learning algorithms. Nature. https://doi.org/10.1038/s41586-025-09761-x

[15] Oh, J., Farquhar, G., Kemaev, I., Calian, D. A., Hessel, M., Zintgraf, L., Singh, S., van Hasselt, H., & Silver, D. (2025). Discovering state-of-the-art reinforcement learning algorithms. Nature. https://doi.org/10.1038/s41586-025-09761-x

[16] Kalai, A. T., Nachum, O., Vempala, S. S., & Zhang, E. (2025). Why language models hallucinate (arXiv:2509.04664). arXiv. https://doi.org/10.48550/arXiv.2509.04664

References

[17] Zhong, Z., Raghunathan, A., & Carlini, N. (2025). ImpossibleBench: Measuring LLMs' propensity of exploiting test cases (arXiv:2510.20270v1). arXiv. https://doi.org/10.48550/arXiv.2510.20270

[18] Liu, Y., Iter D., Xu, Y., Wang, S., Xu, R., & Zhu, C. (2023). G-EVAL: NLG evaluation using GPT-4 with better human alignment (arXiv Preprint arXiv:2303.16634). https://arxiv.org/abs/2303.16634

[19] Ramlochan, S. (2023, August 3). What are large language model (LLM) agents and autonomous agents. Prompt Engineering Institute. https://promptengineering.org/what-are-large-language-model-llm-agents/

[20] Belcak, P., Heinrich, G., Diao, S., Fu, Y., Dong, X., Muralidharan, S., Lin, Y. C., & Molchanov, P. (2025). Small language models are the future of agentic AI (arXiv Preprint arXiv:2506.02153v2). https://arxiv.org/abs/2506.02153

[21] Shen, W., Li, C., Chen, H., Yan, M., Quan, X., Chen, H., Zhang, J., & Huang, F. (2024). Small LLMs Are Weak Tool Learners: A Multi-LLM Agent. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (pp. 16658–16680). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.emnlp-main.929

[22] PVKL. (2025, April 16). Understanding the Model Context Protocol. https://pvkl.nl/en/understanding-the-model-context-protocol/

References

[23] Wang, Z., Zhou, Z., Song, D., Huang, Y., Chen, S., Ma, L., & Zhang, T. (2024). Towards understanding the characteristics of code generation errors made by large language models. arXiv Preprint arXiv:2406.08731. https://arxiv.org/abs/2406.08731

[24] Huynh, N., & Lin, B. (2025, April 2). Large language models for code generation: A comprehensive survey of challenges, techniques, evaluation, and applications (arXiv preprint arXiv:2503.01245v2). arXiv. https://arxiv.org/abs/2503.01245

[25] Aðalsteinsson, F. S., Magnússon, B. B., Milicevic, M., Davidsson, A. N., & Cheng, C.-H. (2025, May 25). Rethinking code review workflows with LLM assistance: An empirical study (arXiv preprint arXiv:2505.16339). arXiv. https://arxiv.org/abs/2505.16339

[26] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024). SWE-bench: Can language models resolve real-world GitHub issues? Retrieved from https://www.swebench.com/