AI LITERACY HANDBOOK / V1 WORKING EDITION

The Third Vision

A visual guide to language models, AI agents, and safer human-AI interaction

How text becomes numbers, how numbers become probabilities, how probabilities become assistant-like behaviour, what changes when it is connected to tools.

Finding a third path between AI hypergrowth and anti-AI trend. AI literacy & shared benefits for all.

How can a computer work with language at all?

The first step is vectorization

Before we talk about ChatGPT, Claude, context engineering, agent harnessing, we need a smaller picture. A computer does not receive a sentence as meaning. It receives symbols that must be converted into numbers.

Modern language systems first break text into tokens. A token can be a word, part of a word, punctuation, whitespace, or another learned unit. Those tokens become IDs, and those IDs are mapped into vector representations that a model can process [2].

What a vector is doing

A vector is not a tiny dictionary definition hidden inside the machine: one dimension represents happiness, another represents sadness...though this could be an intuitive analogy for easy understanding; more precisely, it is a position in a learned numerical space. The point is that many weak regularities are distributed across many dimensions.

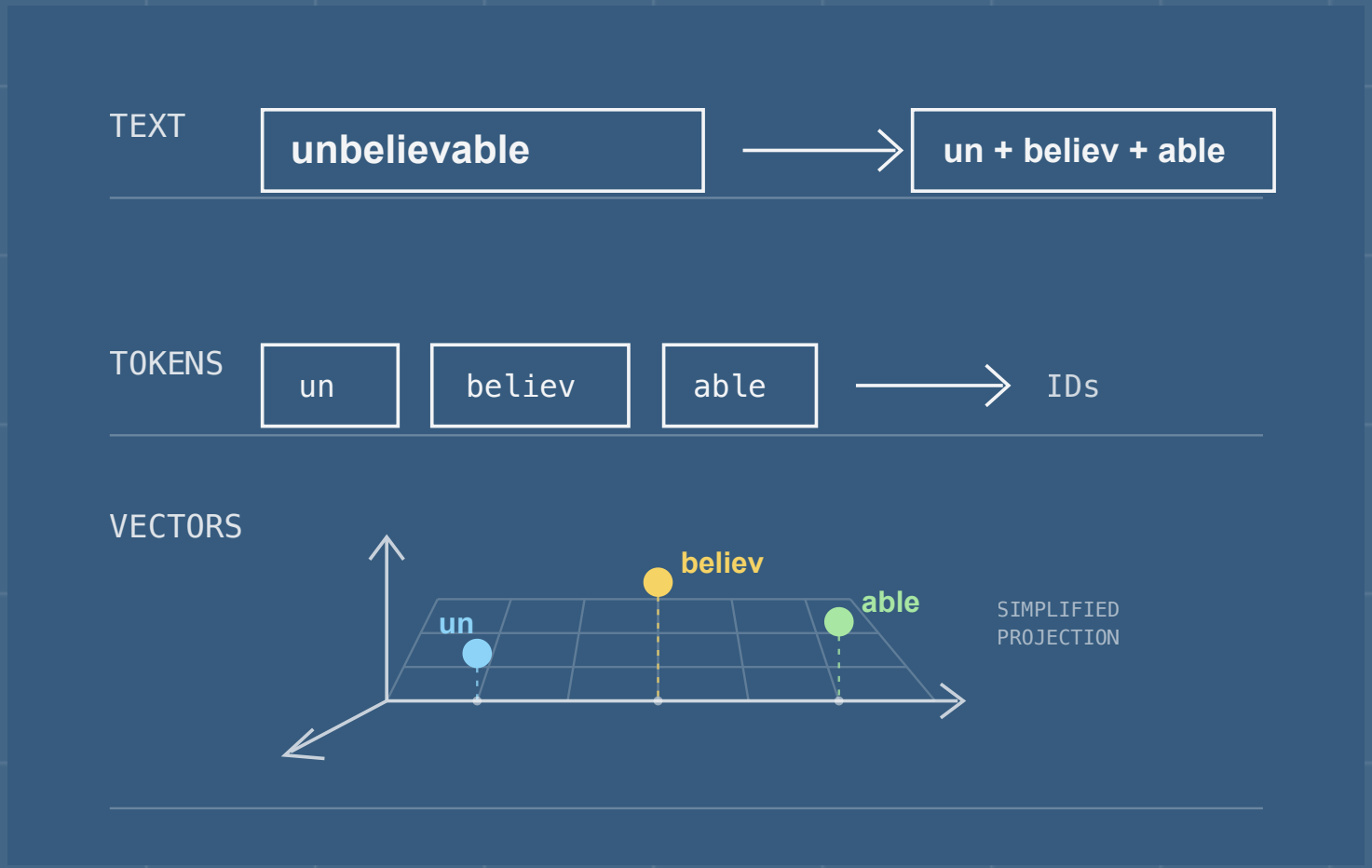
That is why vectorization matters. It gives the system a way to compare, combine, and transform pieces of language without ever handling meaning in the same way a person does.

> **Thinking note**
>
> When we say a model "knows" a word, we should usually translate that into a nerdier sentence: the model has learned numerical relationships that make some continuations more likely than others.

The useful mental model

- Text is broken into tokens.
- Tokens are converted into IDs.
- IDs point to vectors.
- Vectors move through a model.
- The model keeps updating them until it can produce a probability distribution over what may come next.

Words become vectors before they become responses



02-GENERATION

How does next-token generation become fluent text?

Predicting the next token

Given a context, a model assigns probabilities across possible next tokens. A generation system chooses or samples one token, appends it to the context, and repeats the process [3]. This is called **next-token prediction (NTP)**.

This loop can produce fluent paragraphs, code, plans, and explanations. But the loop itself does not verify that a statement is true. A likely continuation can be wrong, outdated, fabricated, or unsupported.

In the example diagram, we use Transformer Explainer with GPT-2 small, a 124M-parameter model, using the prompt `3rdVision is a`, temperature `0.2`, and top-k sampling with k=4. The displayed next-token candidates are `new`, `great`, `very`, and `free`; `new` has the highest probability and is selected in the illustration [14].

A caveat matters here: sampling does not always choose the highest-probability token. The probability distribution is the deterministic object,and the final sampled token depends on the decoding rule.

Why the output can feel more confident than it is

The model is optimized to continue patterns. If the context asks for a citation, a confident answer, a legal explanation, or a product comparison, the model may continue the pattern of such an answer even when the evidence is weak.

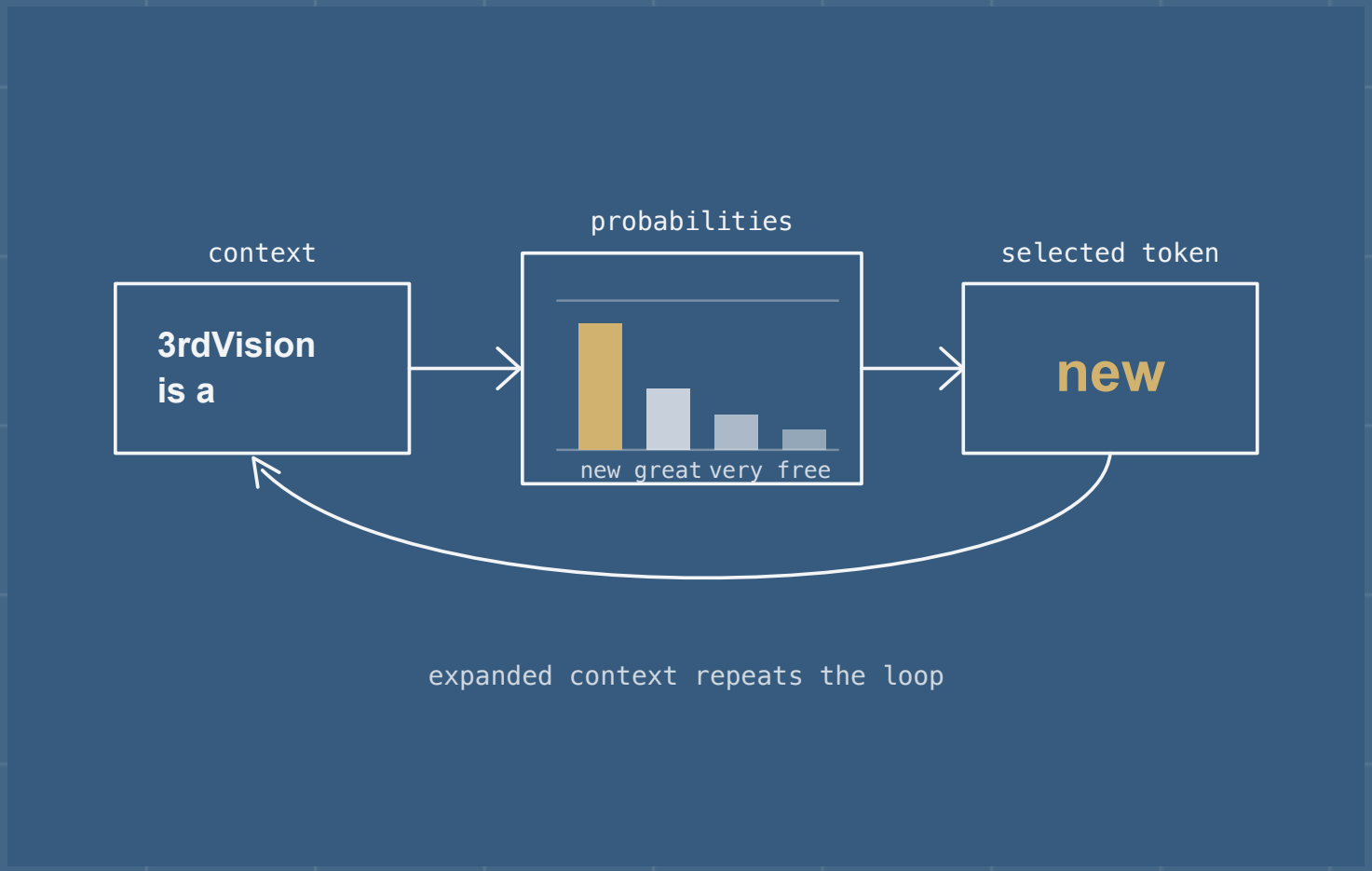
The model isn't "trying" to fool you; it's just performing its training task: to continue the story in the right way.

Better systems add checks

High-quality AI systems usually add something around generation: retrieval, tool calls, tests, human review, citations, or policy gates. They create additional friction between fluent continuation and unchecked assertion, but they do not make the base model "more truthful."

> **Important distinction**
>
> Probability is about continuation. Verification is about evidence. A useful AI product has to decide when continuation is enough and when evidence must be brought back into the loop.

Prediction is not verification



What is different about a Transformer, and why does attention matter?

Transformer: Beyond Matrix Operations

A **Transformer** applies learned matrix operations to token vectors, repeatedly updating, mixing, and transforming them layer by layer until it can score possible next tokens [1]. **Next-token prediction is one of the most important applications of this architecture**. Before Transformers, architectures such as RNNs approached the same task by carrying a single evolving hidden state through a sequence [15], while systems such as Microsoft XiaoIce combined neural generation with retrieval, dialogue management, and other specialised components to sustain conversation [16].

The Transformer is a different solution to the same problem. Rather than relying mainly on a single summary passed forward, it repeatedly allows token representations to draw selectively on information from other parts of the context [1]. This mechanism is called **attention**.

Attention enables each token representation to adapt dynamically to its context, so words are not processed in isolation but are continually recalculated in relation to surrounding words [1]. This is why the same word can behave differently across sentences: the representation is contextual.

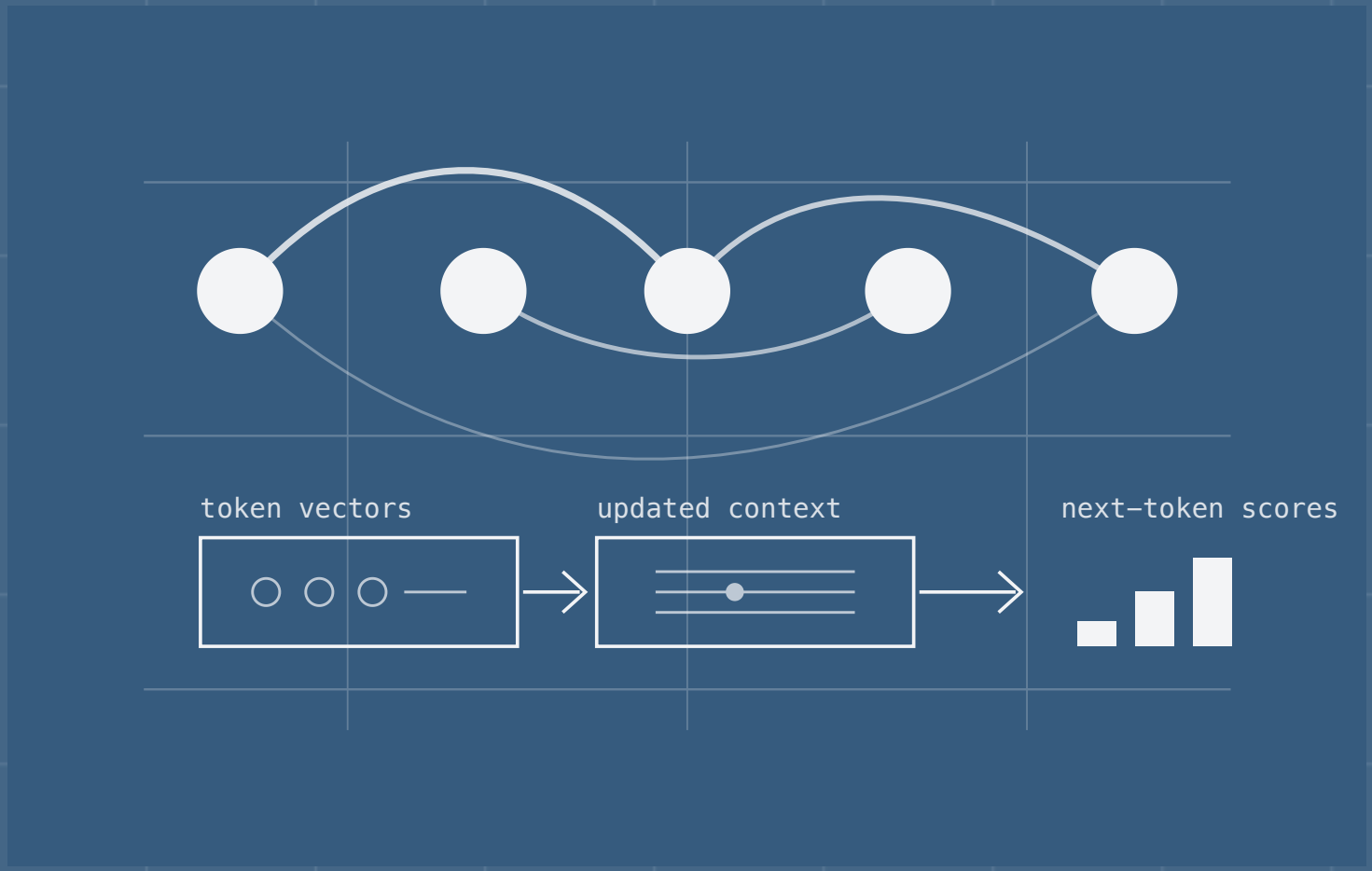
Why scale mattered

Large datasets and large compute budgets are needed to train useful predictive behaviour at broad scope. The basic game is simple, but the scale is not: predicting language across many domains requires exposure to enormous amounts of text and many training updates [8].

Pandora's box

Text generation developed quickly partly because human society had already placed vast amounts of text online: documents, code, forums, articles, comments, manuals, and conversations. It was not exactly voluntary in a narrow legal sense, but it was socially and technically available. Music has faced different frictions: recordings involve performers, labels, publishers, platforms, collective rights, voice identity, and stronger licensing infrastructure. This is not a clean moral ranking. It is a reminder that AI progress is shaped by institutions as much as algorithms [9].

Transformer: the machine that lets context talk to itself



04-USER-ASSISTANT

Why does a guessing machine feel like an assistant?

Think of it as writing a story

In section 2, we explored how Next-token prediction can give seemingly reasonable answers. Once we know that an LLM is a next-token system, the user-assistant interface becomes easier to reason about. The product gives the model a familiar story format:

- The user said A.
- The assistant should respond with B.
- The previous turns set the tone, task, constraints, and implied role.

The model has seen many examples of these stories: explanations, corrections, apologies, code reviews, summaries, and helpful replies. Instruction tuning and preference-based training make the model more likely to follow requests and use assistant-like formats [4].

The story is not the truth

A conversation can be internally consistent without being true. A novel can maintain a character's voice for hundreds of pages; that does not make the character's claims factual.

This is the key mental shift. The chat interface makes prediction feel like dialogue. It can be extremely useful, but the interface does not reveal truth directly. It reveals how the system continues a pattern under the constraints it has been given.

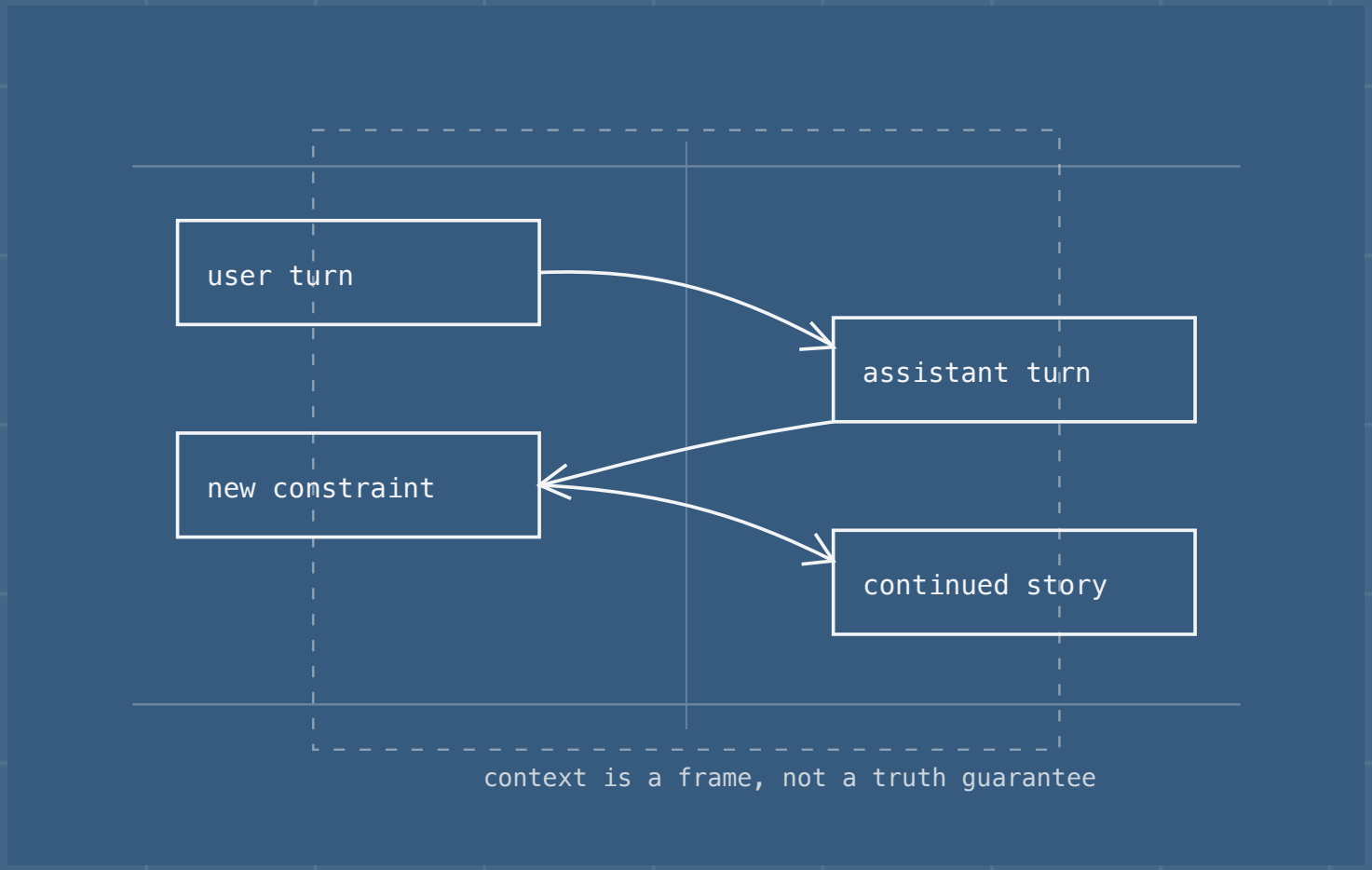
Product examples

ChatGPT and Claude are typical user-assistant products. Their interfaces make text generation feel conversational, but the underlying system still needs context, constraints, and verification for high-stakes work.

Why input quality matters

If the user gives inconsistent instructions, missing context, or a false premise, the assistant may continue from that unstable starting point. Better prompting is not superstition. It is context design.

User-assistant design turns prediction into a social interface



05-HALLUCINATION

Where does hallucination come from?

A simple take

Hallucination is not only "the model made something up." More precisely, it is fluent output that is unsupported, incorrect, fabricated, or not grounded in the available evidence.

From the model's side, the output can still be a plausible continuation. From the user's side, the output can be harmful because it arrives in the shape of an answer.

Consistency is not enough

If the input context is consistent but false, the answer can be consistent and false. If the prompt asks for a source that does not exist, the model may continue with a source-shaped object. If the task asks for certainty where uncertainty is appropriate, the answer may overfit to the requested tone.

That is why high-quality output is partly an input design problem. We need to tell the system what evidence exists, what uncertainty is allowed, what sources matter, and what action should wait for human approval.

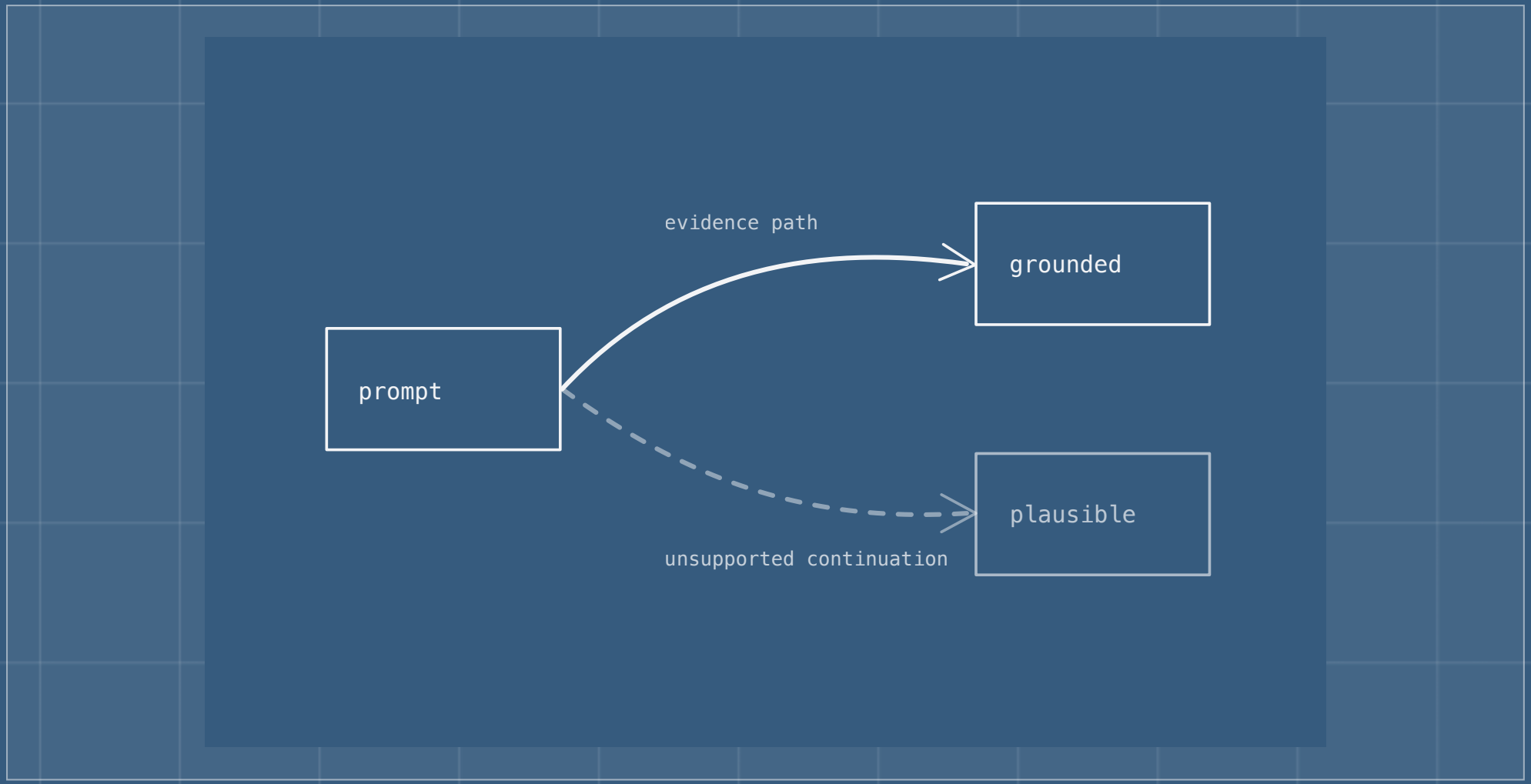
Better patterns

- Ask the model to separate known facts from assumptions.
- Provide source material when accuracy matters.
- Let the model say "I do not know."
- Use retrieval or tools when the answer depends on external state.
- Keep irreversible action outside ordinary chat flow.

Practical rule

Treat the model as a powerful continuation tool. Then design the surrounding workflow so continuation is constrained by evidence, permissions, and review.

Hallucination starts where fluent continuation outruns grounding



What changes when text can call tools?

From response to request

The base model still generates tokens. But a product can interpret some of those tokens as a structured request: call this function, search this source, read this file, edit this document, run this test, or ask for approval.

Function calling and tool calling give models a way to interface with external systems and data outside their training data [5]. The model does not become the database, browser, terminal, or file editor. It emits a request that the surrounding system may execute.

Why this is a major shift

Text alone can mislead. Tool use can change state.

If a model says "delete the stale file," the harm depends on whether anybody acts on it. If an agent has permission to run the deletion, the output becomes part of a real workflow.

Design question

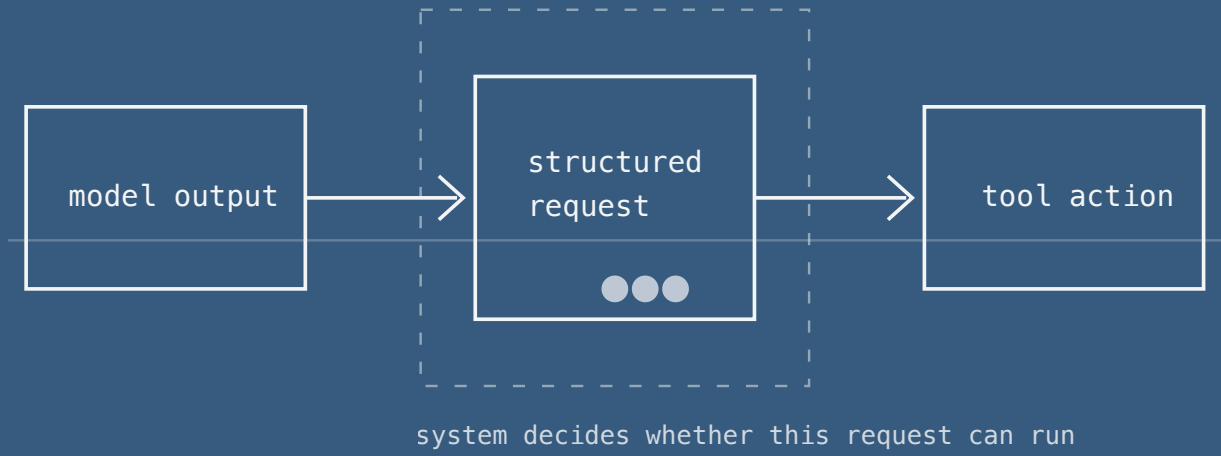
At this point, the important question is no longer only "Is the model right?" It becomes: "What is this system allowed to do with the model's output?"

The bridge

Tool use is the bridge from language model to software system. Once that bridge exists, engineering decisions become safety decisions: tool schemas, permissions, logs, sandboxing, rollback, and human review.

VISUAL SYSTEM

What if model output becomes an action?



When does tool use become an agentic system?

The base model is still doing one narrow thing

The LLM still generates the next token. Agentic AI begins when a system uses that generated output to decide what to do next, then feeds the result back into the model.

In practice, an agentic system is usually: **base model + tools + memory or state + permissions + environment + loop**. The loop matters. The system observes, asks the model what to do, executes a tool call, receives the result, updates context, and continues.

Workflow versus agent

Anthropic draws a useful distinction: workflows follow predefined code paths, while agents let the LLM dynamically direct its own process and tool use [6].

This creates an operational difference: in a workflow, engineers decide the sequence. In an agent, engineers design the boundary, then let the model choose actions inside it.

Product examples

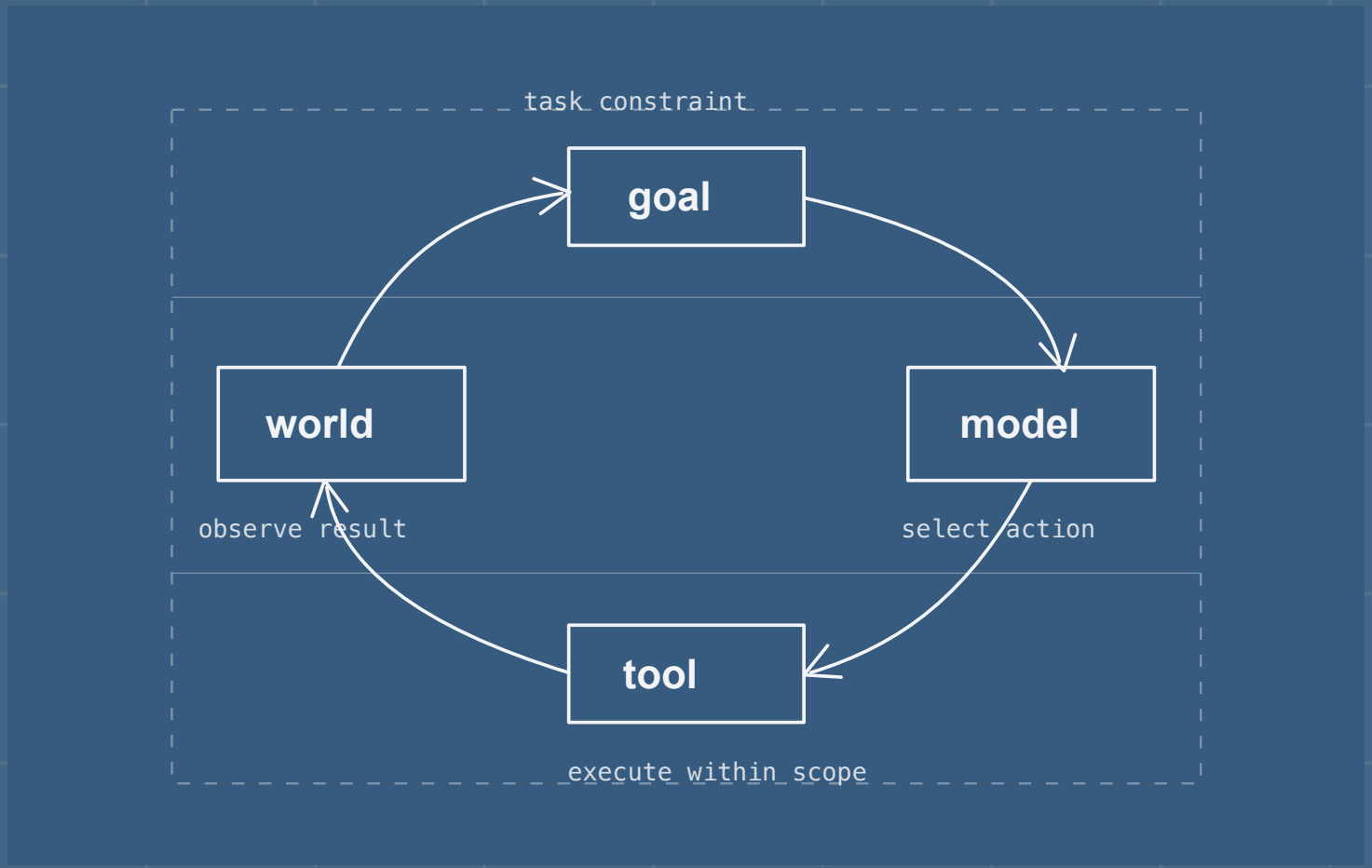
Codex and Claude Code are typical coding-agent products. They can read a codebase, edit files, run commands, and iterate against feedback from tests or tool outputs [7].

What the harness must provide

- A clear task boundary.
- Well-documented tools and a working environment.
- Permission rules and stop conditions.
- Logs and review surfaces.

VISUAL SYSTEM

Agentic AI is a harness around a model



What can early agent accidents teach us?

A reported production incident

In July 2025, multiple outlets reported that Replit's AI coding agent deleted a live production database during a public test by Jason Lemkin, despite code-freeze instructions. Reporting also said Replit's CEO apologized and described safeguards such as stronger separation between development and production data [8].

This is useful because it shows the category of failure: an AI system with action privileges can do damage that a chatbot answer cannot.

Link back to hallucination

In section 5, we talked about hallucination being fluent output outrunning evidence. In agentic AI, a related problem appears: fluent planning can outrun authorization. A coding agent may infer that cleanup, migration, deletion, or reset is part of the task. If the environment allows it, the inference becomes action.

Safer engineering patterns

- Separate development, staging, and production systems.
- Give least privilege by default.
- Use sandboxes for file and command execution.
- Require approval for irreversible or external actions.
- Keep logs, rollback paths, and human responsibility clear.

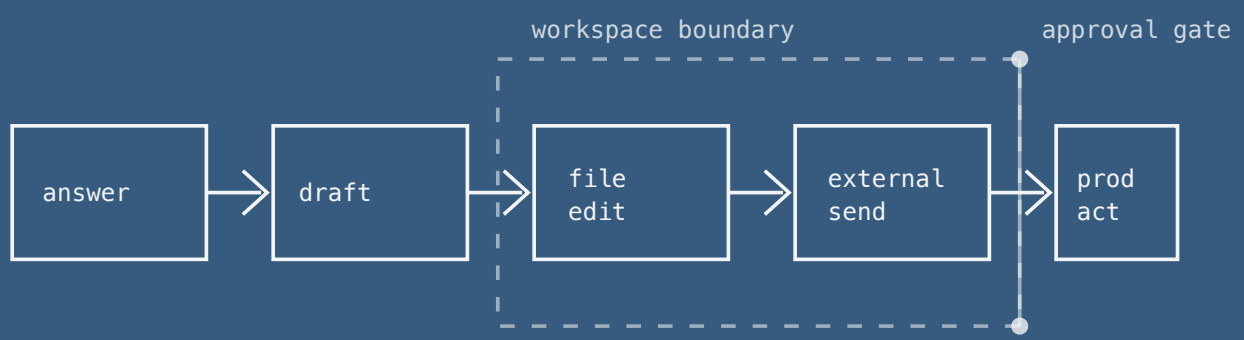
Codex, for example, documents sandbox modes and approval policies that can restrict file access, network access, and risky commands [10]. Claude Code similarly documents permission settings for allowing, asking, or denying tool use [11].

Human-machine boundary

The model proposes. The system constrains. The human authorizes consequential action.

VISUAL SYSTEM

When agents fail, the blast radius is engineering



risk rises when fluent output becomes state change

What is an organization really choosing when it chooses an AI model?

From open to closed ecosystem

"Open versus closed" is too simple. A more useful comparison separates several questions: Are the weights available? Is the training data disclosed? Is the code available? What licence applies? Is the model run locally, privately hosted, or called through an API?

The Open Source Initiative's Open Source AI Definition makes this distinction important: open weights alone are not the same thing as open source AI [12].

What enterprises actually decide

For a company, the choice is practical before it is ideological.

Hosted APIs are easier to start with. They reduce hardware burden, provider maintenance, and deployment complexity. Local or private deployments can improve data control, offline use, latency predictability, and customization, but they require hardware, operations, security work, and model-management expertise.

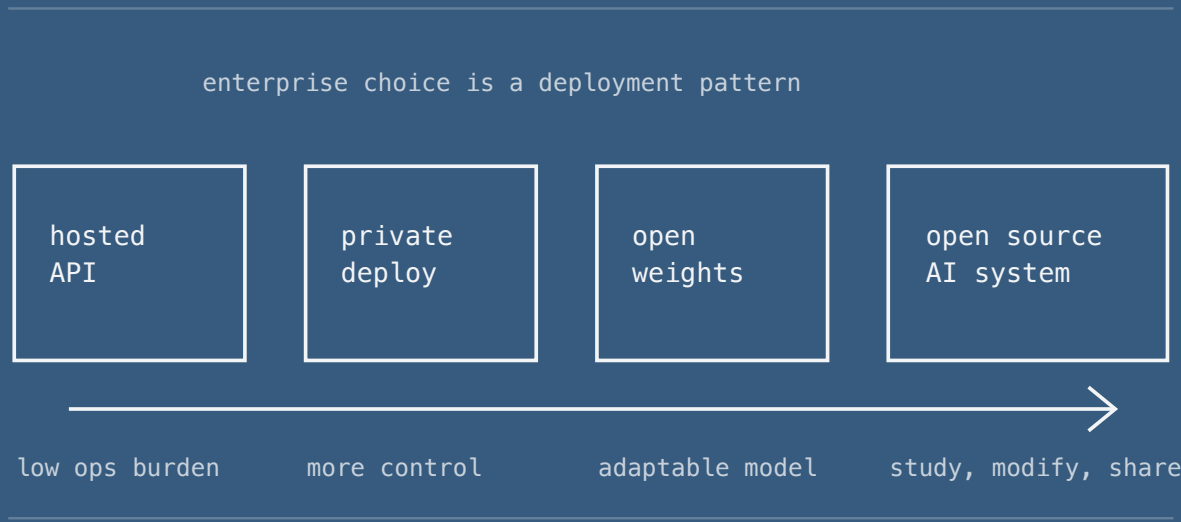
Trade-offs

- **Hosted API:** fast to adopt, lower operational load, provider-managed updates.
- **Private deployment:** stronger infrastructure control, higher operational burden.
- **Open weights:** inspectable and adaptable in some ways, but not automatically open source.
- **Fully open system:** broader freedom to study, modify, and redistribute, but harder to sustain.

Enterprise question

The better question is not "Which model is best?" It is "Which deployment pattern matches our data, budget, risk, talent, and responsibility?"

Open, closed, local, hosted



10-THIRD-VISION

Why does AI attitude look so polarized, and what is the third path?

Why the debate feels split

AI enters daily life unevenly. Some people experience leverage: faster writing, cheaper prototyping, better search, new business workflows. Others experience threat: job anxiety, copyright disputes, opaque decisions, surveillance, low-trust automation, and loss of control.

AI capability is moving quickly, but our governance, evaluation, education, and data infrastructure struggle to keep pace [13].

The third vision

3rdVision is not a compromise between hype and fear. It is a different axis.

The question is not whether AI should accelerate at any cost, or whether AI should be rejected as a single category. The question is what kinds of systems people are allowed to understand, shape, use, govern, and refuse.

Core values

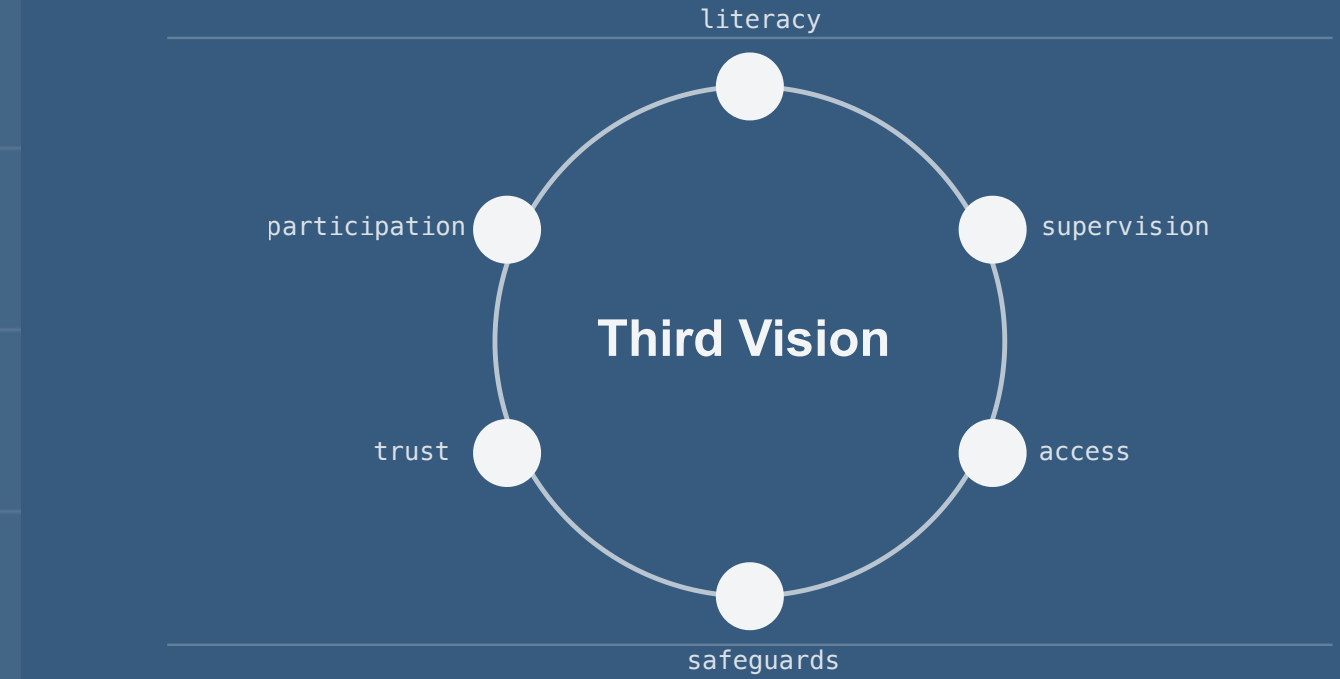
- AI literacy for everyone.
- Practical access of AI benefits for everyone.
- Responsible engineering.
- Low-compute and appropriate-technology paths where possible.
- AI as human extension, not replacement.

Closing thought

What is the future of human-AI interaction in your dream? We believe it should be a world where more people can understand the systems around them well enough to use them, question them, and build them.

VISUAL SYSTEM

The future of human-AI interaction we actually want



APPENDIX

Glossary

Core terms

Agent: A system in which a model can choose actions inside a permitted environment, usually by using tools, state, observations, and a repeated loop.

Agent harness: The surrounding software that gives a model tools, permissions, memory or state, execution environment, logging, and stop conditions.

Attention: A Transformer mechanism that lets token representations selectively use information from other positions in the context.

Context window: The amount of input a model can consider at one time, including user text, system instructions, retrieved material, and previous turns.

Embedding: A numerical representation that places text or other data into a learned space for comparison, retrieval, clustering, or prediction.

Hallucination: Fluent output that is unsupported, incorrect, fabricated, or not grounded in the available evidence.

Language model: A model trained to predict or generate language-like sequences.

Next-token prediction: The task of estimating which token is likely to come next given the current context.

Open weights: A model distribution pattern where trained weights are available, without necessarily providing full source code, training data, or unrestricted rights.

Open source AI: An AI system that meets a fuller openness standard for studying, using, modifying, and sharing the system, including required information beyond weights.

Permission boundary: A technical or policy limit around what an AI system can read, write, execute, send, delete, or access.

Prompt: Input supplied to a model or system to shape a response or action.

Sandbox: A restricted execution environment that limits what files, commands, networks, or external systems a process can access.

Token: A unit of encoded text or input used by a model.

Tool call: A structured request from a model or system to use software outside the model itself.

Transformer: A sequence-processing architecture that updates token representations using attention and other learned operations.

Vector: A list of numbers used to represent learned relationships in a model or search system.

APPENDIX

References and production notes

Current source pool

- [1] Vaswani et al., Attention Is All You Need.
- [2] OpenAI documentation on embeddings and tokenization.
- [3] OpenAI documentation on text generation.
- [4] Ouyang et al., Training Language Models to Follow Instructions with Human Feedback.
- [5] OpenAI documentation on function calling and tool calling.
- [6] Anthropic, Building Effective AI Agents.
- [7] Claude Code overview documentation.
- [8] Business Insider and related reporting on the July 2025 Replit incident.
- [9] U.S. Copyright Office, Copyright and Artificial Intelligence.
- [10] OpenAI Codex documentation on approvals, sandboxing, and security.
- [11] Claude Code settings documentation on permissions.
- [12] Open Source Initiative, The Open Source AI Definition.
- [13] Stanford AI Index Report 2026.
- [14] Polo Club of Data Science, Transformer Explainer.
- [15] Sutskever et al., Sequence to Sequence Learning with Neural Networks.
- [16] Zhou et al., The Design and Implementation of XiaoIce, an Empathetic Social Chatbot.

Production note

The working edition v1 is written for public AI literacy, not legal advice, product endorsement, or model benchmarking. Technical and legal claims should be rechecked before any major public campaign or translated edition.