INSIDE MODERN AI

How Frontier Models Turn Intelligence Into Work

With GPT-6 Astra and Claude Fable 5.1 as technical case studies

Prasad Kukkala
TechiesJournal

Inside Modern AI

How Frontier Models Turn Intelligence Into Work

Published through TechiesJournal · techiesjournal.com

Frontier-model specifications and pricing change quickly. Time-sensitive technical values in this edition were checked against primary vendor sources on 9 September 2026 unless another date is stated.

About This Book

Most people who work with modern AI can already get useful results from it. Far fewer can explain why a system that reads a million tokens still misses a detail, why an agent that ran for three hours fails on the fourth, or why the same model behaves differently in two products. This book is about those questions. It explains the engineering problems a frontier model must solve when it works with very large amounts of information, reasons through difficult tasks, uses tools, operates software, continues work for a long time, recovers from failure and acts within limits.

It is the advanced companion to *Navigating the AI World*. That book explains the foundation: what AI is, how to think about it, where it fits in work and how to learn with direction. You do not need to have read it first. If the basic ideas in Part I feel unfamiliar, it is the better place to start, and this book will point you back to it where that helps.

General AI concepts appear here only as short reminders. The purpose is to go deeper without making the language harder. Where a term is needed, it is explained in plain words first and then used consistently.

GPT-6 Astra and Claude Fable 5.1 appear throughout as real technical case studies. They are evidence for the book's argument, not its subject. The book does not try to declare a winner. It uses what each vendor has publicly documented to show how modern AI systems are actually built, and it says clearly where the documentation stops. Most of the book concerns digital work; one chapter follows the same engineering pattern out into the physical world to show what changes when an AI action has consequences beyond the screen.

Frontier-model specifications and prices change quickly. Time-sensitive values in this edition were checked against primary vendor sources on the date shown in the back matter and should be rechecked before they are relied on.

Contents

Who This Book Is For and How Deeply to Read It

This book is for people who already use modern AI systems and want to understand why they succeed, why they fail, and why so much engineering sits around the model. It is written for working technical people, but it does not assume you build models for a living. If *Navigating the AI World* was about deciding where AI fits, this book is about how the systems hold together when the work gets hard.

Not every chapter deserves the same attention from every reader. The chapters use a simple scale. **Know** means you understand the idea, the terms and the limits well enough to follow a design discussion. **Use** means you can build or operate with it as part of your normal work. **Master** means you make the design decisions, own the failures and guide other people. These describe the depth a role needs, not a ladder everyone is expected to climb.

Six reader roles are used throughout: Developer, Platform/DevOps, Architect, Security, Leader and Student.

Developer. Chapters 6, 7 and 9–12 at Use; the rest at Know. No Master requirement unless you are responsible for production-agent design. Chapter 6 is the one to read twice; almost everything you will build on top of a frontier model is a version of that loop.

Platform/DevOps. Chapters 6, 7 and 9–12 at Master; Chapters 5, 8 and 13–15 at Use; the rest at Know. Chapters 9 and 12 describe problems you already own under other names: state, recovery, idempotency and least privilege.

Architect. Chapters 5 and 12–15 at Master; Chapters 1–3, 6, 7, 9–11 and 16 at Use; Chapters 4 and 8 at Know. The five-layer lens in the next section is the tool you will reuse most, because it lets you place any vendor claim at the layer it actually belongs to.

Security. Chapters 6, 7, 12, 13 and 14 at Master. Chapters 5, 8, 9, 11 and 15 at Use, the rest at Know. The recurring theme for you is that authority must be enforced outside the model.

Leader. Chapter 5 at Use; Chapters 1–4 and 6–16 at Know. You do not need the mechanisms, but you need to know which promised outcomes depend on which layer, and who owns that layer.

Student / early-career reader. Chapter 6 at Use; Chapters 1–5 and 7–16 at Know, with Chapter 6 taken up once the ideas in Part I are comfortable. If the concepts in Part I are still unfamiliar, read *Navigating the AI World* before going deeper into Parts II–IV.

Chapters 1–16 each end with the same short guide, "How much do you need?", listing the roles for whom that chapter rises above Know. Chapter 17 turns the whole scale into a first project for each role.

How to Read Vendor Claims

Frontier AI attracts confident explanations. A diagram appears online showing an exact hidden architecture. A thread states a parameter count. Someone explains precisely how a model's private memory system works. Much of that information has never been publicly disclosed, and a technically serious book has to know when not to pretend. This section sets the rules the rest of the book follows, so that when a later chapter says "Anthropic does not disclose this" or "OpenAI has not documented that mechanism," you read it as a boundary of evidence rather than a gap in research.

Four evidence levels

Every model-specific claim in this book fits one of four categories.

Documented. The vendor or another authoritative primary source directly states the fact. Example: OpenAI documents Astra's 1,050,000-token context window.

Observed. A behavior can be reproduced or measured from the public system, even if the internal reason is unknown. This means a reproducible observation, either our own or a published one, not an online anecdote. Example: a model repeatedly succeeds at a particular tool workflow under controlled testing.

Reasonable inference. Available evidence supports a likely explanation, but the mechanism is not directly confirmed. Inference is labeled as inference, never rewritten as fact.

Unknown. The information is not publicly available, or the evidence is too weak. Unknown is a valid technical answer, and some of the most important questions about these systems currently sit here.

The five-layer lens

The word "model" is used loosely in most discussion of AI. In this book it is used precisely, because the thing you interact with is a system of five layers, and a claim about one layer is often mistaken for a claim about another.

1. **Underlying model capability** — the capability provided by the trained model itself, before surrounding product layers are considered.
2. **Post-training behavior** — how further training has shaped what it tends to do.
3. **Tools and harness** — the software around it that retrieves, executes, checks and stores.
4. **Safety and policy controls** — the rules and monitors that limit or route what it may do.
5. **Product experience** — the interface, defaults and pricing through which a person meets all of the above.

The model is one layer of the complete AI system

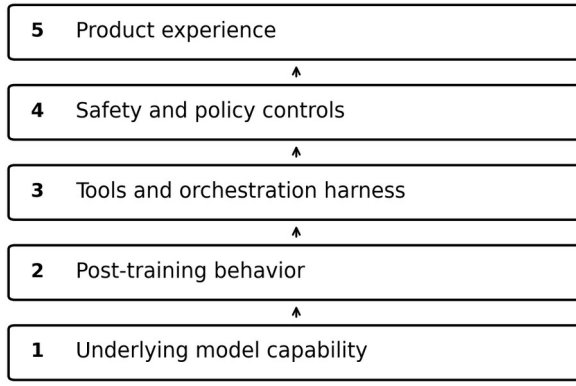

Figure 1. The underlying model is only one layer of the complete AI system.

When a benchmark improves, a feature ships or a request is refused, the first question is which layer changed. Chapter 5 develops this idea in full; every model-specific chapter after it uses the lens.

Why the obvious numbers mislead

People ask how many parameters a frontier model has because it feels like a simple measure of size. Even when the number is available, it explains little on its own. Practical capability also depends on architecture, training data, training and inference compute, post-training, tool access, context management, serving optimizations and the safety and routing layers. A single

number can create the appearance of understanding without explaining behavior.

Benchmarks have a similar limit. A result tells you how a system performed under one defined setup. It does not by itself tell you why it succeeded, whether the result carries to a different environment, whether a tool or harness did much of the work, or how the system will behave after the next product update. This is why the technical references in this book carry a date and a source.

Behavior does not uniquely reveal mechanism either. If a model recalls a detail after hours of work, that could be because the detail stayed in active context, was retrieved again, was kept in a state store, survived a compacted summary, or was rebuilt by the product from earlier records. Seeing the right answer does not tell you which of those happened unless the system exposes it.

Diagrams carry labels too

This book still draws diagrams. Each technical diagram is labeled according to the evidence behind it: documented architecture, directly supported by vendor sources; system-level conceptual architecture, which explains the observable components without claiming hidden neural detail; or reasonable inference, clearly marked. Most diagrams here are the second kind. The rule is to explain the system the evidence supports and not to decorate uncertainty until it looks like architecture.

Why vendors do not disclose everything

There are legitimate reasons for limited disclosure: competitive advantage, security risk, misuse prevention, systems that change too fast to document, and complexity that does not reduce to a public diagram. At the same time, limited disclosure makes independent evaluation harder. Both are true, and the book does not pretend otherwise.

Seven questions for any frontier-model claim

1. Who is making the claim?
2. Is it about the model, the product or the complete system?
3. Is the mechanism directly documented or merely inferred?
4. Can the behavior be reproduced?

5. What tool, harness and safeguard conditions were present?
6. What has changed since the claim was measured?
7. What would falsify the explanation?

Those seven questions are worth more than any memorized architecture diagram. They are also the questions this book asks of its own case studies.

Documented behavior is not the same as documented mechanism. Where the evidence ends, this book says so.

Part I — The New Capability Layer

This part examines what changes inside the frontier capability itself: very large working context, variable reasoning effort, multimodal problem solving, and AI participation in research.

Chapter 1

Long Context Is Not Memory

What changes when frontier models work with enormous amounts of information

A customer sends a large project: requirements, meeting notes, architecture diagrams, support tickets, source code, spreadsheets and previous decisions. The AI reads all of it. Later, someone asks about one small decision buried deep in that material, and the answer sounds confident but is wrong.

If the model can hold so much information, why can this still happen?

A Short Reminder: What Context Means

Context is the information available to the model while it is working on the current request. It can contain instructions, conversation history, documents, code, tool results and other task information. The context window is the maximum working space available for that information during a single model inference.

A large context window means the model can work with a large amount of information at once. It does not mean the model permanently remembers every fact or will use every detail correctly. Keeping those two ideas apart is the whole subject of this chapter.

Why One Million Tokens Changes the Problem

OpenAI currently documents GPT-6 Astra with a 1,050,000-token context window and a maximum output of 128,000 tokens. That is large enough to change how applications can be designed. Instead of breaking every long document or codebase into small

pieces immediately, a system can sometimes give the model a much larger working set.

But a larger working set creates a new engineering problem. The system is no longer only trying to fit the information; it must make sure the model can find and use the important part at the right time.

Capacity Is Not Attention

Imagine a model receives hundreds of documents, and the answer to the user's question is present somewhere inside them. The information being present does not guarantee that the model will treat it as important. Relevant evidence may be surrounded by large amounts of unrelated material, earlier instructions may compete with newer information, and similar passages may point in different directions.

The engineering problem changes from "Can the information fit?" to "Can the system keep the right information useful?"

Why Retrieval Still Matters

A very large context window does not remove the need for retrieval. Retrieval means keeping information outside the active model request and bringing in the pieces that are relevant to the current problem.

If a user asks why an authentication design was rejected, the application can search decision records, meeting notes and architecture documents, then bring the strongest evidence into the active context. That is often more reliable than asking the model to search mentally through every document that happens to fit inside a huge window.

Long Tasks Need More Than Context

The problem becomes harder when the model is not answering one question but working for hours. A long-running agent may read files, call tools, edit code, run tests, receive new instructions, wait for an external system, recover from an error and continue. If every event is simply appended forever, the working context becomes increasingly expensive and noisy.

This is where state becomes important. State is the durable record of where the task stands: what has been completed, what

changed, what is waiting, what failed, and what the next step needs to know. Context is what the model sees now; state is what the system keeps.

Compaction: Keeping the Meaning Without Keeping Everything

OpenAI documents compaction support for Astra. Compaction reduces older working material into a smaller representation so a long task can continue without carrying every earlier detail in full.

That sounds simple, but it creates a trade-off. Compression saves space by removing detail, and a fact that appeared unimportant at step 20 may become important at step 80. A robust system therefore needs a way to return to original sources when a compacted summary is not enough.

Prompt Caching and Persisted Reasoning

Astra also supports prompt caching and persisted reasoning, which solve different parts of the long-task problem. Prompt caching avoids repeatedly processing an unchanged prefix of a large request. Persisted reasoning allows useful reasoning state to continue across related work.

Neither should be treated as magical memory. They are engineering mechanisms that reduce repeated work and help continuity.

Where Claude Fable 5.1 Enters the Picture

Anthropic describes Claude Fable 5.1 as its most advanced model for coding and knowledge work, designed for demanding and long-running tasks. For this book, the interesting question is not whether Fable has a bigger number in one specification. It is how the Claude system keeps a difficult task coherent while it spans many steps, tools and possibly several applications.

Anthropic does not publicly disclose every internal mechanism behind that continuity. Where the design is not documented, this book says so rather than filling the gap with assumptions.

How a Long-Context Agent Actually Works

Put the pieces together and a modern long-context agent is a loop rather than a single act of reading:

1. Load the current task state.

2. Retrieve the information needed for the next decision.
3. Assemble the active context.
4. Run model inference.
5. Produce an answer or choose a tool action.
6. Execute the action outside the model.
7. Validate what happened.
8. Update durable state.
9. Build the next context and continue.

Reliable long-running work does not depend on the model remembering the whole project by itself. The surrounding system continually rebuilds the information the model needs. This loop is a system-level conceptual pattern, not a description of any vendor's hidden implementation.

Where It Can Still Fail

Relevant information is present but underused. The model sees the evidence but gives more weight to other material.

Retrieval brings back the wrong evidence. The correct source exists, but the surrounding system selects weaker or unrelated information.

Compaction removes a detail too early. A summary loses something that later becomes important.

Task state becomes stale. The stored record says one thing while the external system has already changed.

The model simply reasons incorrectly. Better context reduces one class of failure; it does not remove ordinary model error.

The move to very large context windows is important, but the deeper change is architectural. Frontier AI systems are becoming combinations of model inference, retrieval, caching, state, compaction, tools and orchestration. A million-token model can read much more, but a well-designed system must still decide what matters, what should survive, what can be compressed, and what must be fetched again.

Large context is capacity. Reliable continuity is a system-design problem.

Three Things to Remember, One Thing to Do

1. Large context is not the same as permanent or perfect memory.
2. Long-running AI needs state, retrieval and compaction in addition to context.
3. When an AI appears to forget, the failure may be in the model or in the system around it.

One thing to do. Take one long-document task you already give to AI. Before pasting everything in, write down which facts must be in the active context, which should be retrieved only when needed, and which belong in a durable record outside the conversation. Then run it both ways and compare the answers.

How much do you need? Architect: Use. Everyone else: Know.

Chapter 2

Reasoning Is More Than Thinking Longer

Why frontier models spend different amounts of computation on different problems

A developer asks an AI model to rename a field in a configuration file. A few seconds of reasoning is enough. The next request is different: inspect a production failure, connect logs from several services, understand a recent code change, find the likely root cause, propose a fix and explain the risk of deploying it.

Treating both requests as the same kind of inference would be wasteful. One is routine; the other may require the model to explore several explanations, reject weak ones, check evidence and revise its conclusion.

Modern reasoning is not only about whether a model can solve a hard problem. It is also about how much computation should be spent, when, and whether the extra work is worth its cost.

A Short Reminder: What Reasoning Means Here

In this book, reasoning means the additional internal computation a model uses to work through a problem before producing or committing to an answer or action. It does not mean the model reasons exactly like a person, and it does not mean every internal step is visible to the user.

One distinction matters throughout. The answer we see is an output. The model's internal computation is not the same thing as the explanation it later writes about how it reached that answer.

The Same Model Can Spend Different Effort

OpenAI exposes this directly in GPT-6 Astra. Its API supports reasoning-effort levels named low, medium, high, xhigh and max, and OpenAI allows the level to be changed during an ongoing conversation without rebuilding the unchanged beginning of the prompt.

That tells us something important about frontier-model design: reasoning is increasingly treated as a controllable resource. A routine task can use a lower setting, and a difficult investigation can be given more inference-time effort. The model remains the same model, but the amount of work performed for the request changes.

Why More Reasoning Can Help

Consider a software incident with three plausible causes: a database migration introduced an incompatible schema change, a network timeout caused retries that duplicated work, or an authentication change rejected requests from one service. A shallow answer may notice the most obvious error message and stop there. More reasoning effort lets the model compare hypotheses, look for contradictory evidence, inspect the order of events, and ask whether the explanation accounts for all the symptoms.

This is especially useful when a problem has several properties at once: multiple steps, incomplete evidence, dependencies between decisions, or a high cost if the answer is wrong.

But More Compute Is Not the Same as More Intelligence

If a model can be asked to reason more, it is tempting to assume maximum reasoning should always be used. That would be poor system design. Deeper inference costs latency and tokens, some problems do not improve after additional work, a model can spend effort exploring possibilities that do not matter, and more reasoning does not guarantee that a wrong starting assumption will be corrected.

The goal is not maximum reasoning. It is sufficient reasoning for the difficulty and risk of the task.

Reasoning Becomes a Scheduling Problem

Once reasoning effort is variable, an AI system has to make a scheduling decision. A support workflow might classify a case as routine, begin with lower reasoning, then escalate when logs disagree, security is involved or confidence remains low.

Reasoning is then no longer just a property of the model. It becomes part of orchestration: the surrounding system decides when to spend additional intelligence.

Reasoning Can Be Distributed Across a Workflow

A system does not have to spend the same effort at every stage. Astra can begin a task at a lower reasoning level, and an application can increase the effort when the work reaches a difficult stage; OpenAI documents this as a configuration update that preserves the existing conversational prefix and cache, and Anthropic documents the equivalent for Fable 5.1 as a per-message effort change.

A sensible division looks like this:

- cheaper reasoning for triage and routine transformation;
- deeper reasoning for ambiguous or high-consequence decisions;
- tools for facts that should be measured rather than guessed;
- verification of important actions after execution;
- human review when the consequence exceeds the system's authority.

The intelligence of the workflow comes partly from the model and partly from deciding where to place the model's effort.

What Fable 5.1 Tells Us

Anthropic describes Claude Fable 5.1 as its most capable generally available model for ambitious coding, knowledge work and long-running asynchronous tasks. Its public material emphasizes root-cause analysis, planning, verification loops, code review, research and recovery when a step fails.

Anthropic documents the same kind of control. Fable 5.1 runs adaptive thinking that is always on, and its depth is steered with an `effort` parameter that takes the same five levels, low, medium, high, xhigh and max, with high as the default. Anthropic also documents changing effort part-way through a conversation with a per-message setting that preserves the prompt cache, currently in beta. Two vendors arriving at the same five-level

control, and the same mid-conversation change, is itself evidence: reasoning effort has become a standard orchestration parameter rather than one company's feature.

The useful comparison therefore sits higher up: both companies are optimizing models for problems where a quick first answer is not enough, and where the model must sustain a line of work, inspect evidence, revise decisions and produce something that survives checking.

Reasoning and Tools Solve Different Problems

A model should not reason about a fact that a tool can verify directly. If the question is "Did the test suite pass?", the reliable action is to run the tests. If the question is "Which architectural change best explains why these tests began failing after the deployment?", reasoning becomes useful.

The rule of thumb: use reasoning to decide what evidence means, and use tools to obtain evidence that can be measured directly.

Self-Correction: Useful, but Easy to Misunderstand

Frontier models increasingly appear able to notice inconsistencies and revise their own work. Fable 5.1 is explicitly described as able to write tests to check its coding work; Astra is designed for end-to-end workflows that include building, operating and checking software.

Self-correction can improve results, but it is not proof of correctness. If the same model creates both a solution and a weak test, it can pass its own test while the real defect remains. Chapter 13 separates self-review, tool-based verification, independent model review and external ground truth for exactly this reason.

Where It Can Still Fail

Effort is spent on a wrong starting assumption. More reasoning deepens the error instead of correcting it.

A difficult problem is classified as routine. The system never escalates, and a shallow answer ships.

Reasoning replaces a measurement. The system pays for maximum effort when a direct tool check would have been better.

The explanation is not a faithful record. The model produces a convincing account that does not match its internal computation.

A self-check repeats the original blind spot. The same model verifies its own mistake.

The important change is not simply that frontier models can answer harder questions. Reasoning is becoming an adjustable system resource, much like memory, compute or parallelism in conventional software, and the advantage increasingly comes from knowing when to make the model work harder and when not to.

Reasoning is a budget, not a virtue.

Three Things to Remember, One Thing to Do

1. More reasoning can improve difficult work, but it does not guarantee correctness.
2. Variable reasoning turns inference-time compute into an engineering and cost decision.
3. Good systems combine reasoning with tools, verification and human authority rather than asking the model to think its way through everything.

One thing to do. Take one AI workflow you run today and label each step as routine or high-consequence. Assign lower effort to the routine steps, higher effort to the rest, and mark any step where the question is a fact a tool could check instead. That labeled list is the start of a reasoning schedule.

How much do you need? Architect: Use. Everyone else: Know.

Chapter 3

One Model, Many Kinds of Work

How language, code, vision, documents and software actions become parts of the same problem

A traditional software system is usually divided by data type. One component reads text, another processes images, another compiles code, another queries a database, and a workflow engine connects them. Frontier AI is changing that boundary: the same model can increasingly receive several kinds of evidence, reason across them, and decide what action should happen next.

The important word is not multimodal. The important idea is integration.

A Short Reminder: Multimodal Does Not Just Mean "Can See Images"

A multimodal model can work with more than one kind of input. GPT-6 Astra accepts text and image input, and Anthropic says Fable 5.1 can understand diagrams, charts and tables inside files and PDFs.

Simply accepting an image is not the advanced capability that matters here. The deeper question is whether the model can connect what it sees with what it reads, what the code does, what a tool reports, and what the user is trying to accomplish.

One Problem Can Have Many Representations

Consider an engineering task: a web application looks correct in the design file but breaks in a real browser. The written requirement describes intended behavior, the source code describes the implementation, a screenshot shows what the user actually sees, browser-console errors reveal runtime problems, a

network trace reveals failed requests, and the visual design provides the target state.

A narrow system would pass each artifact to a different specialist and then try to combine the outputs. A frontier model can increasingly treat them as different views of one underlying problem: read the requirement, inspect the code, look at the screenshot, use a browser, compare the result with the target design, make a change and check the screen again.

This Creates Cross-Domain Reasoning

At the application level, the model is connecting evidence across domains rather than performing only text work. A chart may contradict the written summary. A screenshot may reveal that syntactically correct code produces the wrong interface. A spreadsheet may show a total that disagrees with a narrative report.

The technical skill is not merely "read a chart." It is "read the chart, compare it with the stated conclusion, inspect the calculation that produced it, and decide whether the conclusion still holds."

Astra: A Model Exposed as an End-to-End Worker

OpenAI describes Astra as a model for complex reasoning, coding, computer use, research and document creation. Its API supports text and image input and gives the model access, through the surrounding Responses system, to tools such as web search, file search, code execution, a hosted shell, computer use and MCP.

OpenAI also presents examples that cross conventional boundaries: scientific data analysis followed by plots, website creation followed by frontend quality checks, software installation followed by troubleshooting, and professional work that ends in documents, spreadsheets or presentations. The architectural point is that several capabilities can participate in one continuous task.

Fable: Vision as Part of Verification

Anthropic says Fable 5.1 understands diagrams, charts and tables embedded in documents and can use vision to evaluate its own coding work against a design or goal. That creates a useful

loop: understand the visual target, generate code, render the result, inspect the output visually, then revise the implementation when the result does not match.

The same modality that helps understand the problem can become part of the verification loop.

Code Is Becoming a Universal Action Language

Code deserves special treatment because it allows a language model to leave the purely linguistic world. A model can write a small program to calculate something exactly, transform a dataset, query an API, generate a chart, test a hypothesis or automate a repeated task.

This does not mean generated code is automatically safe or correct. It means code acts as a bridge between probabilistic reasoning and deterministic computation. Natural language is good for describing intent and interpreting ambiguous evidence; code is often better for repeatable calculation and execution.

Documents Are No Longer Just Output

Documents, spreadsheets and presentations used to be treated mainly as final deliverables. In a frontier workflow they can also become active parts of reasoning. A model may read a workbook, detect an inconsistency, inspect formulas, modify the analysis, generate a chart and then explain the decision in a report. The artifact is both evidence and output.

Astra's product material emphasizes professional artifact creation that follows existing templates and business context. Fable's material emphasizes document-heavy work in finance, legal, analytics and architecture.

The Model Still Does Not Literally Contain Every Capability

When a product appears to browse the web, run Python, manipulate a spreadsheet or operate a browser, not all of that work is happening inside the neural network. The model may decide what should happen and interpret the result, while an external tool performs the actual operation. A frontier AI system can feel like one intelligence even though the work is distributed across a model, tools, runtimes, state, policies and interfaces.

Generalization Is the Bigger Story

Specialized software is excellent when the problem is well defined. A tax calculator should not improvise, and a database should not creatively reinterpret a query. Frontier models are valuable in a different region: problems where the input is messy, the representation changes, and the next step cannot be completely specified in advance.

The model can move from prose to code, from code to an interface, from the interface to an error, and from the error back to a revised plan. That flexibility is one reason a single frontier model can appear useful across many professions: software engineering, where requirements, code, test failures, screenshots and runtime behavior must be connected; data analysis, which connects raw data, calculation code, plots and business interpretation; design, research and professional work, each of which combines several forms of evidence in the same way.

Where It Can Still Fail

An image or chart is misread. The visual evidence enters the reasoning wrong from the start.

Unrelated evidence is connected. Two artifacts are treated as views of one problem when they are not.

Generated code is valid but wrong. It runs, and it computes the wrong thing.

A tool result is stale, incomplete or misinterpreted. The bridge to deterministic computation carries bad data.

A coherent story hides a conflict. The model reconciles evidence that actually disagrees.

Cross-domain fluency can make an incorrect answer look more convincing, not less. That is why verification becomes more important as capability broadens. The frontier is moving from models that are impressive within one interaction toward systems that can maintain a problem while its representation changes: a requirement becomes code, code becomes a running application, the application becomes a screenshot, the screenshot becomes new evidence, and the model stays involved through the whole chain.

Multimodal capability matters when evidence is connected, not merely accepted.

Three Things to Remember, One Thing to Do

1. Different inputs become valuable when the model can reason across their relationships.
2. Many apparent model capabilities are produced jointly by the model and external tools.
3. The broader the model's reach across domains, the more important independent verification becomes.

One thing to do. Give a model one problem in two representations that should agree, such as a chart and the data behind it, or a screenshot and the code that produced it, and ask it to find where they disagree. Check its answer against your own reading. The gap between the two is a direct measure of how far you can trust cross-domain reasoning in that setting.

How much do you need? Architect: Use. Everyone else: Know.

Chapter 4

From Answering Questions to Doing Research

What changes when AI participates in the research loop instead of only explaining existing knowledge

Ask an ordinary AI assistant about a scientific paper and it may summarize the methods and conclusions. Give a frontier research system a broader objective and the job can look very different: find relevant literature, inspect datasets, write analysis code, run experiments, compare results, revise the hypothesis, and prepare evidence for a researcher to review.

The change is from discussing knowledge to participating in the process that produces new knowledge.

A Short Reminder: Research Is a Loop

Research rarely follows a straight line from question to answer. A simplified loop is question → hypothesis → evidence → experiment → observation → revision. Earlier language models were useful mainly around the edges of that loop: reading papers, explaining concepts and drafting text. Frontier systems are moving deeper into the middle of it.

Why Scientific Work Is a Natural Frontier

Scientific work combines many of the capabilities discussed in the first three chapters. It involves large amounts of technical literature, tasks that run for hours or days rather than one turn, dependence on code and specialist software, evidence that appears as tables, charts, images and raw data, and results that are uncertain and frequently force a revised plan.

Astra: Reasoning Plus Scientific Software

OpenAI describes Astra as state-of-the-art in science and presents examples of the model navigating scientific software, inspecting sequencing quality, visualizing genetic variation, generating plots and helping researchers identify where to focus further analysis.

OpenAI also reports very high results on scientific and mathematical evaluations. Those numbers are useful evidence of capability, but benchmark performance should not be confused with autonomous scientific reliability. A model can perform strongly on difficult questions and still fail when evidence is incomplete, an experiment is poorly designed, or real-world data contains an unexpected confounder.

Fable: Research Capability as a Product Direction

Anthropic describes Fable 5.1 as having research capabilities that provide an early view of how AI may contribute to scientific progress. Anthropic has also built Claude Science as a workbench that connects models with scientific databases, notebooks, R, cluster terminals and other research tools.

The distinction matters: the model supplies reasoning capability, while the surrounding environment supplies access to evidence, computation and auditable artifacts.

The Most Important Capability Is Experiment Velocity

Suppose a human researcher can carefully explore five plausible analyses in a day. An AI-supported workflow may be able to prepare many more candidate analyses, write the supporting code, run them, summarize the results and flag the most interesting cases for human examination.

The advantage is not that the AI automatically becomes the scientist. The advantage is that the cost of trying another reasonable path can fall dramatically, so researchers can investigate ideas that previously would have been abandoned because they required too much routine work.

But Faster Experimentation Creates a New Failure Mode

If a system can run ten times more experiments, it can also run ten times more bad experiments. Speed amplifies both good

methodology and weak methodology. A biased dataset may be selected, the wrong statistical test may be used, information can leak between training and evaluation data, correlation can be mistaken for causation, repeated optimization can make a meaningless result look significant, and a polished narrative can hide a weak design.

Scientific acceleration must therefore be paired with stronger provenance, reproducibility and review.

The Role of Auditable Artifacts

A serious research system should not return only a paragraph saying what it discovered. It should leave evidence another person can inspect: the papers and sources used, the dataset and its version, the code that ran, the parameters and environment, plots and intermediate outputs, failed experiments as well as successful ones, and the assumptions that materially affected the conclusion.

Anthropic explicitly emphasizes auditable artifacts in Claude Science. The principle applies more broadly: if a model contributes to research, another person should be able to reconstruct what happened.

Novelty Is Not the Same as Truth

Frontier models can generate unusual hypotheses and solutions, and that can be valuable because discovery often requires exploring ideas outside the obvious path. But a new idea is valuable only after it survives evidence. The model's ability to produce a clever explanation should be kept separate from the system's ability to test that explanation.

Human Judgment Moves Up the Stack

As AI performs more routine research work, the human role moves toward the decisions that carry scientific responsibility:

- Is the question meaningful?
- Does the data represent the phenomenon being studied?
- Can the experiment distinguish competing explanations?
- Is the apparent result important or merely statistically convenient?
- Is the work ethically acceptable?

- Does the conclusion deserve publication, clinical use or further study?

AI may reduce the labor required for many steps. It does not remove ownership of the scientific claim.

What Happens When Research Becomes Long-Running

A serious research task can span hours, days or longer. The system has to preserve which experiments were attempted, why one path was abandoned, which data version was used, and which result changed the direction of the investigation. A long context window alone is not enough; research requires a durable experimental record. This is one reason reasoning, multimodality, tools, state and evaluation should not be studied in isolation, and why Part III returns to state and recovery in detail.

Where It Can Still Fail

Decisive prior work is missed. The literature search does not find the paper that already answers, or contradicts, the question.

A source is invented or misread. The model cites something that does not exist or does not say what it claims.

The experiment answers a different question. The setup drifts slightly from what was intended.

Automated analysis introduces a subtle error. Data leakage or a statistical mistake enters without anyone noticing.

Verification checks the code, not the science. The implementation is correct; the assumption behind it is not.

Polish is mistaken for quality. A human trusts the volume and presentation of AI-generated evidence more than the method that produced it.

Scientific work shows the real potential of frontier models more clearly than ordinary chat, because the model can participate across reading, reasoning, code, software operation, visual analysis and repeated experimentation. The breakthrough is not an AI that knows every scientific answer. It is a system that helps humans explore the space between a question and an evidence-backed answer much faster, while leaving a trail that can be checked.

Speed without a research trail is not acceleration. It is noise, produced faster.

Three Things to Remember, One Thing to Do

1. Research capability is about participating in an evidence loop, not merely explaining papers.
2. Faster experimentation increases the need for provenance, reproducibility and independent checking.
3. The strongest future systems will likely change where scientists spend their time, while responsibility for scientific claims remains human.

One thing to do. Take one analysis an AI system has produced for you and list what you would need to reproduce it: the sources, the data version, the code, the parameters. If you cannot reconstruct it from what was left behind, the workflow is not yet research-grade, whatever the result looked like.

How much do you need? Everyone: Know.

Part II — Turning Intelligence Into Action

This part follows the path from raw model capability to controlled action: product layers, tools, software interfaces, and finally physical equipment.

Chapter 5

The Model Is Not the Product

Why raw capability, post-training, tools, safeguards and product design must be separated

A user opens an AI product, asks a question, and receives an answer. From the outside it feels as if one model did everything. Inside the system the picture is very different. The underlying neural model may provide the core reasoning capability, but the behavior a user experiences can also depend on post-training, system instructions, tool access, routing, safety classifiers, approval policies, monitoring, memory, user permissions, and the interface that exposes all of it.

If we call all of that "the model," we lose the ability to understand where capability actually comes from and where restrictions are actually enforced.

A Short Reminder: Five Layers

The front matter introduced the five layers this book uses to place any claim: underlying model capability, post-training behavior, tools and harness, safety and policy controls, and product experience. This chapter is where the lens earns its keep. The key idea is that two products can use the same underlying model and still behave very differently because the surrounding layers are different.

Fable and Mythos Make the Distinction Visible

Anthropic provides an unusually clear public example. It states that Claude Fable 5.1 and Claude Mythos 5.1 use the same underlying model. The difference is not that one has a different neural network. Fable 5.1 is the generally available product with additional safeguards in sensitive cybersecurity, biology and

chemistry areas, while Mythos 5.1 is made available through restricted programs for vetted organizations.

Anthropic also explains that some Fable requests can be routed to Opus models without the same sensitive-capability exposure when a safeguard intervenes. A user can therefore ask Fable a question and receive a result that reflects not only the capability of the underlying model but also a policy decision made by the surrounding system. The product response is not a direct measurement of raw model capability.

Astra Shows the Same Principle From a Different Direction

OpenAI describes Astra as a frontier model trained through advances in pre-training, reinforcement learning and alignment. But the Astra a developer uses through the Responses API is surrounded by a substantial system layer. OpenAI documents computer use, asynchronous tool calling, mid-turn steering, multi-agent orchestration, prompt caching, persisted reasoning, compaction and misalignment monitoring around the model, and describes additional product protections such as auto-review and confirmation policies in systems like Codex and ChatGPT Work.

The point is not that every layer is identical across OpenAI products. It is that the final behavior is produced by a stack, not by an isolated neural network.

Why Product Benchmarks Need Careful Reading

Suppose two evaluations show different results for two AI products. It is tempting to conclude that one underlying model is more intelligent. But several other variables may have changed: one system had better tools, one used a different reasoning setting, one product allowed an action the other blocked, one harness retried failed steps, one had stronger retrieval or browser access, or one used a safety fallback model for part of the test.

Benchmarks are still valuable. But an end-to-end benchmark often measures the performance of a complete system, not only the model weights.

Post-Training Can Change What Capability Looks Like

Pre-training gives a model broad patterns and capabilities from large amounts of data. Post-training then shapes how the model

applies them: instruction following, reasoning behavior, refusal boundaries, tool use, communication style, error correction and alignment with policy. Two checkpoints with similar raw capability can produce noticeably different user experiences after different post-training.

Post-training is not merely "adding manners." It can materially change how capability is expressed, controlled and made useful.

System Instructions Are Part of the Product

Most users never see the complete system instructions governing an AI product. Those instructions can define role, allowed actions, safety requirements, tool preferences, escalation behavior and how the system should handle conflicts. The model does not answer only the user's prompt; it responds inside a larger instruction hierarchy.

This matters when comparing a public chat product with an API. The same family of models can behave differently because the surrounding instructions and tools differ.

Routing Can Make One Product Look Like Several Models

A modern AI product does not have to use one model for every step. It may route a sensitive request to another model, use a smaller model for classification, call a specialist tool for computer vision, or ask another agent to verify a result. Anthropic's Fable fallback mechanism is a concrete example: policy can change which model actually handles part of a request.

Model identity in a user interface therefore does not always tell us every component involved in producing the result.

Safety Is an Architectural Layer, Not a Final Filter

A common mental model is that safety works like a content filter placed after the AI finishes. Frontier systems are more complicated. OpenAI describes Astra deployments that monitor full trajectories, use classifiers to inspect reasoning and actions for unauthorized behavior, and can stop activity before it completes. Anthropic applies safeguards that can restrict certain capabilities or route requests differently.

These are architectural controls because they affect what actions the system can take while it is operating, not merely which words can appear in the final answer.

Why This Matters for Developers

When developers build on a frontier model, they are designing another product layer of their own. They decide which tools the model receives, which data sources it can access, what permissions those tools carry, when a human must approve an action, what is logged, how failures are retried, and what evidence is required before an answer is accepted.

A powerful base model cannot compensate for reckless product architecture. Every request passes through policy, instructions, routing, inference, tool selection, execution, monitoring and verification before it becomes a response, and the developer owns several of those steps.

Where It Can Still Fail

A blocked action is read as missing capability. A developer assumes the model cannot do something when a policy layer stopped it.

Tool authority exceeds user intent. The product gives the model far more power than the person asking for help expected.

A safety layer blocks harmless work. Users learn to route around controls, which weakens every control.

A harness advantage is credited to the model. A benchmark comparison attributes to intelligence what retrieval or retries produced.

Routing changes the model silently. A fallback handles part of the request, and the user treats the result as if one model did it all.

The frontier-model era is also the frontier-system era. The neural model is the engine, but an engine alone is not a vehicle; the steering, brakes, transmission, sensors and driver rules determine what the vehicle can safely do.

Understanding modern AI requires two separate questions: what can the underlying model do, and what does this product allow that capability to become?

Three Things to Remember, One Thing to Do

1. Model capability and product behavior are not the same thing.
2. Post-training, routing, tools and safeguards can materially change what users experience.
3. When comparing frontier systems, ask which layer produced the difference before attributing everything to the model.

One thing to do. Pick one AI product you use regularly and write the five layers down the side of a page. Then take three recent behaviors you noticed, such as a refusal, a mistake and a result that impressed you, and place each at the layer you think produced it. If you cannot decide, that uncertainty is the point of the exercise.

How much do you need? Architect: Master · Platform/DevOps: Use · Security: Use · Leader: Use. Everyone else: Know.

Chapter 6

Tool Use: How AI Moves From Knowing to Doing

Why structured actions, external execution and trust boundaries turn a language model into an operational system

A model can know how to book a meeting without being able to place anything on a calendar. It can explain how to query a database without having access to that database. It can write the command that would deploy software without deploying anything.

Tool use is the bridge between describing an action and causing an action.

A Short Reminder: The Model Usually Does Not Execute the Tool

When an AI system "uses a tool," the model normally produces a structured request describing what it wants to happen. The surrounding application receives that request, checks it, executes the real operation, and returns the result to the model.

This separation is one of the most important safety and reliability boundaries in modern AI. The model proposes. The application decides whether, how and with what authority the operation actually runs.

From Text Generation to Structured Intent

Without tools, a model might answer: "Search the customer database for account 4832." With tool calling, it can instead produce a structured action:

```
tool: customer_lookup
account_id: 4832
```

The application can validate the arguments before anything is executed. Structured intent is easier to inspect, restrict and audit than free-form natural language.

Tool Choice Is Itself a Reasoning Problem

Giving a model twenty tools does not guarantee it will choose the right one. It has to understand what each tool does, what inputs it requires, what permissions it carries, what evidence should come back, and whether a tool call is necessary at all. A model that calls the wrong tool can be dangerous even if each individual tool is implemented correctly.

The Trust Boundary Is Outside the Model

Imagine a payroll assistant with two tools: `read_payroll(employee_id)` and `send_payment(employee_id, amount)`. The first is observational; the second moves money. If both are exposed with the same level of authority, the application has already made a serious design mistake.

The payment tool needs additional constraints: a maximum amount, allowed recipients, human approval, duplicate-payment protection, audit logging and transaction idempotency. Those controls belong in deterministic application code. They should not depend on the model remembering a policy sentence.

Astra: Tool Use as an Asynchronous Workflow

Astra adds asynchronous tool calling. OpenAI documents that the model can continue reasoning, call other tools, or answer independent parts of a request while an application-level tool is still running. The application keeps ownership of the pending tool call and later returns the result using the call identifier.

This changes tool use from a simple request-response sequence into something closer to concurrent workflow execution.

Why Asynchronous Tools Matter

Suppose an agent preparing a customer migration plan starts three independent operations: scanning a large repository, querying infrastructure inventory, and searching documentation for version-specific constraints. A synchronous system waits for one result before starting the next. An asynchronous system makes progress on other parts of the task while slower operations continue.

That reduces end-to-end time, but it creates new engineering problems: partial results, race conditions, stale assumptions, and deciding whether an answer can be completed before every tool returns.

Tool Results Are Evidence, Not Truth

A tool result can still be wrong. An API may return stale data, a search may miss a page, a shell command may fail silently, a database query may run against the wrong environment. The model has to interpret the result in context, and the surrounding system may need to validate high-consequence outputs independently.

Using a tool reduces guessing only if the tool is authoritative for the question being asked.

Code Execution Changes the Reliability Equation

Code execution is especially powerful because it lets the model turn an ambiguous instruction into a deterministic computation. Instead of mentally adding 100,000 transaction rows, the model writes code that performs the calculation.

Code introduces its own risks: incorrect logic, unsafe file access, dependency installation, network access, resource exhaustion and unexpected side effects. This is why execution environments are usually sandboxed.

Tool Discovery Creates Another Layer of Capability

As tool catalogs grow, systems may let the model discover or select tools dynamically instead of loading every definition into context. That improves scalability but creates a governance problem: what is the model allowed to discover, and who decides

which newly discovered capability becomes usable? The answer again belongs to the surrounding product architecture.

Under the Hood

A simplified tool loop looks like this:

1. Model interprets the user's goal.
2. Model decides whether outside evidence or action is required.
3. Model emits a structured tool call.
4. Application validates policy, schema and permissions.
5. Tool executes outside the model.
6. Application returns structured results.
7. Model interprets the result.
8. System verifies high-risk outcomes.
9. Model continues or finishes.

With asynchronous tools, several instances of steps 3–7 may be active at the same time. Step 4 is the trust boundary; everything the payroll example needed lives there.

Where It Can Still Fail

A valid but inappropriate tool is chosen. The call is well-formed and the wrong thing to do.

Arguments are correct in shape and dangerous in meaning. Schema validation passes; the semantics do not.

Permissions exceed the task. The application grants more authority than the work needs.

An asynchronous result arrives late. The model has already made a conflicting assumption.

A tool result injects instructions. Content returned by a tool is treated as if the user had said it.

An incomplete result is treated as conclusive. The model stops before the evidence is in.

Tool use changes the practical meaning of model intelligence. A model that only predicts text can advise. A model connected to tools can observe and change external systems.

The moment a model can act, software architecture becomes as important as model capability.

Three Things to Remember, One Thing to Do

1. Tool use is a controlled handoff from probabilistic reasoning to external execution.
2. The application, not the model, should enforce authority and irreversible-action rules.
3. Asynchronous and dynamic tools increase capability but also increase coordination and security complexity.

One thing to do. Take one tool integration you or your team already run. List every tool the model can call and mark each as observational or side-effecting. For every side-effecting tool, write down where its limits are enforced: in application code, or in a sentence in the prompt. Any that rely on the prompt are the first things to fix.

How much do you need? Platform/DevOps: Master · Security: Master · Developer: Use · Architect: Use · Student: Use. Leader: Know.

Chapter 7

Computer Use: When the Interface Becomes the API

How frontier models operate software that was never designed for machine-to-machine control

APIs are clean because software talks to software through defined structures. Human interfaces are messy. Buttons move, pages load slowly, pop-ups appear, text may be inside an image, and a successful click does not always mean the requested operation completed.

Computer use asks an AI model to operate inside that messy human layer.

A Short Reminder: Why Use a Screen at All?

If a reliable API exists, the API is usually the better integration. Computer use becomes valuable when no API exists, when the API does not expose the required operation, when a task spans several applications, when the organization already relies on a graphical workflow, or when the AI must inspect what a human would actually see. In those cases the interface itself becomes the control surface.

The Action-Observation Loop

A computer-using model works in a repeated loop: observe the current screen, decide what it means, choose an action, click, type, scroll or navigate, observe what changed, and decide whether the action succeeded.

The last step is critical. Computer use is unreliable if the model assumes an action succeeded simply because it issued the click.

Astra's Computer-Use Capability

OpenAI describes Astra as its strongest computer-use model and reports substantial gains on computer and browser benchmarks. Its examples include filling forms, updating CRM records, conducting online research, drafting material into workplace applications, installing and testing software, troubleshooting visible problems, and running frontend quality checks.

These examples show why computer use is more than browser automation: the model combines screen perception with reasoning about the larger task.

Why Screen Control Is Harder Than a Tool Call

A function call has a known name, schema and return value. A screen has none of those guarantees. The model may need to infer whether an element is clickable, which of several similar buttons is correct, whether a page finished loading, whether a dialog is blocking the workflow, whether a value was saved, and whether the visible state is stale. It is reasoning over an environment that can change after every action.

Visual Understanding Is Necessary but Not Sufficient

A model may correctly identify the Submit button and still make a bad decision by submitting too early. Computer use needs both perception and task judgment. It must understand not only where an element is, but whether acting on it is authorized and appropriate at that moment.

Verification Must Be Designed Into the Loop

Suppose an AI updates a customer address. A weak loop clicks Save and assumes success. A stronger loop clicks Save, waits for the application state to change, checks for a success message, re-opens or queries the record, and confirms that the expected value persisted.

For important tasks, visual confirmation may still be insufficient. A backend API or audit log provides stronger evidence than a green banner on a screen.

Prompt Injection Becomes More Dangerous

When an AI browses websites or opens documents, it encounters text written by someone other than the user, and that text may contain instructions designed to manipulate the model. For a chatbot, a successful prompt injection may distort an answer. For a computer-using agent, it may attempt to influence an action.

OpenAI reports that Astra is more robust to prompt injection than earlier systems. Stronger resistance does not remove the need for tool permissions, content isolation and confirmation gates.

Computer Use Blurs Observation and Action

A browser page can be both data and an instruction surface. The same screen may contain facts the user wants read, buttons the model can press, malicious text attempting to redirect the model, and private information the model should not disclose. That is why computer use needs careful separation of trusted instructions from untrusted content.

Latency Is Part of Intelligence

OpenAI reports Astra completing OSWorld-style tasks substantially faster than GPT-5.6 Sol in its simulations while also achieving a higher score. This matters because a computer-use system that takes an hour to perform a five-minute human task may be technically capable but operationally unattractive. The frontier includes not only accuracy but action speed, recovery and efficient observation.

Under the Hood

A robust computer-use workflow includes:

1. Capture the screen or structured accessibility state.
2. Identify relevant elements and current application state.
3. Compare the intended action with permissions and policy.
4. Execute the smallest safe action.
5. Observe the result.
6. Verify whether the intended state actually changed.
7. Recover or retry if the result differs.
8. Escalate irreversible or ambiguous actions to a human.

The model may be excellent at perception and reasoning, but the harness still determines what actions are available and what evidence counts as success.

Where It Can Still Fail

The interface changes. The model acts on the wrong element because the layout moved.

The click lands but the action is rejected. The application refuses, and the model does not notice.

State changes between observation and action. A hidden modal or slow load invalidates what the model just saw.

Injected content redirects the task. Text on the page persuades the model to do something else.

The right action happens in the wrong place. A valid operation runs in the wrong account, tenant or environment.

Visual verification stands in for real verification. The screen says success; the backend was never checked.

Computer use lets AI cross the gap between systems built for APIs and systems built only for people. It turns the graphical interface from a passive display into an operational environment for model reasoning, and it inherits every weakness of that interface along with every capability.

Three Things to Remember, One Thing to Do

1. Use a clean API when one exists; computer use is most valuable where APIs are absent or incomplete.
2. Every action should be followed by evidence that the intended state actually changed.
3. Computer use combines perception, reasoning, permissions and security in a way ordinary tool calls do not.

One thing to do. Watch a browser or computer-use agent perform one small task and keep a tally. For each action, note what evidence it used to decide the action succeeded: a confirmation on screen, a re-check of the record, or nothing at all. The count of "nothing at all" is how much you are trusting on faith.

How much do you need? Platform/DevOps: Master · Security: Master · Developer: Use · Architect: Use. Everyone else: Know.

Chapter 8

When an AI Action Leaves the Screen

What changes when model-driven decisions begin controlling scientific instruments and physical equipment

This chapter is a boundary case. Most of this book concerns digital work; physical action shows what changes when the same engineering pattern begins producing real-world consequences.

A software mistake can be serious. A mistaken database update corrupts records; a bad payment action moves money. But there is another boundary beyond software: the moment an AI-directed action changes a physical system. A robotic arm can collide with equipment, a liquid handler can contaminate an experiment, a manufacturing controller can damage material, and a laboratory action may not be reversible.

Once an AI action leaves the screen, safety becomes a physical engineering problem as well as an AI problem.

A Short Reminder: The Model Still Needs an Interface

A language model cannot directly move a robotic arm simply because it understands the instruction. It needs a software layer that translates an approved action into commands understood by the device. The model-tool pattern from Chapter 6 still applies; the consequence of failure is different.

Anthropic's Model Hardware Standard

In August 2026, Anthropic introduced a research preview of the Model Hardware Standard, or MHS, which it describes as a shared specification for AI agents to operate physical devices such as microscopes, liquid handlers and robotic arms. The preview is aimed first at scientific laboratories and advanced manufacturing

environments. Anthropic says agents can coordinate several instruments, adjust parameters during a workflow, and in some cases recover from hardware errors.

This is important not because MHS is already a universal standard, but because it shows a credible direction: frontier AI systems are beginning to receive interfaces to the physical world.

Why Physical Systems Need a Stronger Contract

A normal tool can often return an error that allows the agent to retry. Physical equipment may not be so forgiving. Before an AI command reaches hardware, the system may need deterministic limits: an allowed operating range, maximum speed, temperature, pressure or force, a safe device state, interlocks, collision prevention, emergency stop, human approval for hazardous steps, and equipment-readiness checks.

These constraints must live below the model. The model should not be able to talk its way around them.

Planning and Control Should Be Separated

One useful architecture lets the model operate at a higher level. The model decides "move the sample from station A to microscope B"; the control system computes the safe trajectory, checks interlocks and executes the motion. The model provides task-level reasoning, and a deterministic controller handles real-time physical safety.

Use the model for flexible planning. Use verified control systems for hard physical limits.

The Observation Loop Becomes Sensor-Driven

In computer use, the model observes a screen after an action. In physical systems, observation includes camera images, position sensors, temperature, pressure, instrument status, measurement results and error codes. The agent must reason from those signals to decide whether the physical world actually changed as intended.

Recovery Is Harder in the Physical World

A software agent can often restart a failed process. A physical experiment may have consumed a sample, a calibration error may invalidate hours of work, and a mechanical fault may require a technician. Recovery planning therefore begins before the action:

- What can be retried?
- What is irreversible?
- What state must be preserved?
- When must a human intervene?
- What evidence proves the equipment is safe to continue?

Parallelism Creates Both Opportunity and Risk

Anthropic's MHS preview emphasizes operating multiple instruments in parallel. That can dramatically increase experiment throughput, but it also means one agent may be coordinating several physical processes whose states evolve independently. This resembles asynchronous tool calling with higher consequence: a late result from one instrument may invalidate a decision already made for another. Coordination becomes a real-time state-management problem.

The Physical World Makes Authorization Concrete

In software, "least privilege" can sound abstract. With physical equipment it becomes obvious. An AI system should not have authority to operate every device in a laboratory merely because it can understand them. Permissions may need to be limited by device, operation, time window, experiment, hazard level, operator identity and physical location, and the authorization system should be independently enforceable even if the model becomes confused or compromised.

Why This Matters Beyond Laboratories

The same architecture can eventually apply to manufacturing, warehouse robotics, inspection systems, agriculture, energy infrastructure and specialist scientific equipment. The specific safety rules will differ, but the pattern is consistent: model-level flexibility above deterministic operational boundaries. Even readers who never touch hardware will recognize the shape from

deployments and payments, where the same question of what is retryable and what is not already applies.

Where It Can Still Fail

The wrong high-level operation is chosen. Every layer below executes it correctly.

A sensor lies. Inaccurate or miscalibrated readings feed the reasoning loop.

Two safe actions combine into an unsafe one. Each passes its check; together they do not.

Success is reported and the outcome is wrong. The device says done; the physical result differs.

Recovery makes it worse. A retry compounds an equipment fault.

Permissions are valid but too broad. The task needed one device; the agent had the room.

The progression from text generation to software tools to computer use and finally physical equipment reveals the deeper trajectory of frontier AI: the model is becoming less isolated from the world it reasons about.

As the distance between a model decision and a real-world consequence becomes shorter, engineering discipline must become stronger, not weaker.

Three Things to Remember, One Thing to Do

1. Physical action should always pass through deterministic safety and control layers.
2. Planning intelligence and low-level hardware control are different engineering responsibilities.
3. The more irreversible the action, the stronger the requirements for authorization, sensing, recovery and human oversight.

One thing to do. Take one automated action you already run that cannot be undone cheaply, whether it is a production deploy, a payment, or a data deletion, and answer the five recovery questions above for it in writing. If any answer is "we would find out afterwards," you have found the place where the physical-world discipline applies to your digital system.

How much do you need? Platform/DevOps: Use · Security: Use. Everyone else: Know.

Part III — Work That Lasts Longer Than One Turn

Long-running intelligence is not created by giving a model a large context window. It requires durable state, checkpoints, steering, delegation, permissions, monitoring and recovery. This part explains how those pieces keep an advanced model useful when a task survives many actions, failures and changes of direction. Each chapter asks one question: how does work survive, how does direction change without losing valid work, when should work be divided among several intelligences, and what is the system allowed to do at all.

Chapter 9

State, Checkpoints and Recovery

How complex work survives many steps, asynchronous operations, failures, retries and resumptions

A short chat can fail and simply be tried again. A long-running task is different. Imagine an AI system that has already inspected a repository, changed twelve files, run several tests, opened a deployment ticket, waited for an external approval, and then encounters a failure. Restarting from the beginning would waste work and may make the situation worse.

How does work survive? Long-running intelligence needs a durable record of what has happened, not just a large amount of text in context.

A Short Reminder: Context and State Are Different

Context is the information currently presented to the model. State is the durable record of where the task stands outside any single inference call: the goal and current plan, completed steps, files or records changed, tool calls still running, results already verified, failures and retry counts, approvals received or still required, and the next safe action.

Context helps the model reason now. State lets the system know what happened before and what remains true after the current model call ends.

Why a Long Context Window Is Not Enough

A system could try to keep the entire history inside the model context. That eventually becomes expensive, noisy and fragile. More importantly, a transcript is not the same thing as operational state. A sentence saying "deployment succeeded" is

weaker than a deployment record containing an environment, version, timestamp and verification result. Reliable systems separate conversational history from machine-checkable task state.

Checkpoints Turn Progress Into Something Recoverable

A checkpoint is a known point in a workflow from which the system can safely continue after interruption. For a coding agent, a checkpoint might record the repository commit or patch identifier, tests that passed, tests that still fail, dependencies installed, open tool operations, and the reason the current approach was chosen.

The purpose is not to save every thought. It is to save enough verified state to avoid repeating work or making contradictory changes.

Recovery Requires More Than Retry

A naive agent responds to failure by trying the same step again. That is not recovery. A real recovery loop asks whether the failure was temporary or structural, whether the attempted action changed anything before failing, whether repeating it is safe, whether the system should roll back, continue from partial progress or change plan, and whether a human needs to intervene.

Fable 5.1 is explicitly positioned for long-running work that plans, uses tools and recovers when a step fails. The engineering lesson is that recovery is now part of the expected behavior of a frontier agent, not an exceptional feature.

Idempotency Becomes a Critical Design Pattern

In conventional distributed systems, an idempotent operation can be repeated without causing duplicate side effects. The same idea becomes essential for agents. If a payment step times out after the request reaches the bank, retrying blindly could pay twice. If a ticket-creation tool times out, retrying might create duplicate tickets.

The application should therefore use identifiers, transaction records and duplicate protection so the agent can ask "Did this already happen?" before repeating an action. Every long-running

workflow should identify which actions are safely repeatable and which require evidence before retry.

Asynchronous Work Creates Pending State

Astra supports asynchronous tool calling: the model can continue reasoning or work on independent tasks while an application-level tool is still running. That creates a new category of state, pending work. The system must know which operation is still running, which call identifier belongs to it, which decisions depend on its result, whether later work can proceed safely without it, and what to do if it never returns. This resembles distributed software more than a traditional chat application.

Compaction and Checkpoints Solve Different Problems

Compaction, introduced in Chapter 1, reduces how much historical information must remain in active context. A checkpoint preserves the operational truth needed to resume work. A compacted summary might say "authentication refactor is complete and tests pass." A checkpoint records the exact commit, test command, test result, environment and timestamp.

The first helps the model reason efficiently. The second helps the system prove what actually happened.

Durable State Should Prefer Facts Over Narratives

Whenever possible, state should be stored in structured fields rather than only in free-form summaries:

```
status              = awaiting_approval
last_verified_commit = abc123
pending_tool_call   = inventory_scan_47
retry_count         = 1
rollback_available  = true
```

The model can still receive a readable summary, but the application preserves the machine-checkable facts separately, so a later decision can be checked against a value rather than a sentence.

Under the Hood

A durable agent loop can be represented as:

1. Load authoritative task state.
2. Reconstruct only the context needed for the next step.

3. Let the model choose a plan or action.
4. Validate permissions and preconditions.
5. Execute the action.
6. Verify the resulting external state.
7. Write a checkpoint.
8. Record pending work or failure information.
9. Continue, recover, or escalate.

Three distinctions live in this loop. Step 2 is where compaction applies: context is rebuilt, not accumulated. Step 7 is the checkpoint, and it comes after step 6, never before; a checkpoint written before verification records a hope, not a fact. Step 4 is where duplicate protection and idempotency preconditions are checked before execution: the system asks whether the action already happened before it lets it happen again.

Where It Can Still Fail

Success is checkpointed before it is verified. The record says done; the external system disagrees.

A retry repeats an irreversible action. The step was not idempotent and nobody checked.

State goes stale. Another system changed the environment after the state was read.

Summary and store disagree. The compacted narrative contradicts the authoritative record, and the model believes the narrative.

A valid checkpoint is no longer relevant. Requirements changed, and the system resumes from progress that no longer applies. Chapter 10 takes up that case.

The move from chat to long-running work turns AI engineering into distributed-systems engineering.

Reliable autonomy depends on state, idempotency, checkpoints, verification and recovery as much as it depends on model intelligence.

Three Things to Remember, One Thing to Do

1. Context is temporary working information; state is durable operational truth.

2. Recovery means understanding what changed, not merely retrying.
3. Long-running AI should checkpoint verified progress so failures do not erase or duplicate work.

One thing to do. Take one agent workflow you run or plan to run and write its state as structured fields, using the block above as a template. Then list every action the agent can take and mark each as safe to retry or needs evidence before retry. The second list is your idempotency backlog.

How much do you need? Platform/DevOps: Master · Developer: Use · Architect: Use · Security: Use. Everyone else: Know.

Chapter 10

Steering a Model While It Is Working

Changing requirements mid-task without throwing away completed work

Real projects change while work is in progress. A customer changes a requirement, a security team rejects an approach, a manager says the output must be ready for a different audience, or a production incident becomes more important than the original task. Older AI workflows handled this badly: the user stopped the task, rewrote the prompt and began again.

How does direction change without losing valid work? Frontier systems are moving toward keeping the useful work, changing the direction, and continuing.

A Short Reminder: Steering Is Not the Same as Starting Over

Steering means introducing new instructions while a task is already underway and incorporating them into the remaining work. The system must decide what completed work remains valid, which assumptions are now wrong, what must be undone, which pending tool calls are still useful, and whether the new instruction conflicts with safety or authorization rules.

Chapter 9 explained how work survives failure. This chapter is about how it survives a change of mind, which is harder, because nothing broke.

Astra Makes Mid-Turn Steering Explicit

OpenAI documents mid-turn steering for Astra through a WebSocket continuation flow. A user can send a correction or changed requirement while the model is working, and the Responses API preserves completed work and includes the update in the continuation. This treats steering as a first-class

system capability rather than an accidental consequence of chat history.

Preserving Work Requires Dependency Awareness

Suppose an agent is preparing a migration plan with four sections: database, network, application and testing. Halfway through, the user changes the target cloud region. The network design may need to be redone, the testing methodology may remain valid, some cost calculations are now wrong, and the database migration sequence may need only minor adjustment.

A useful agent does not discard everything. It identifies which pieces depend on the changed assumption. Good steering requires more than obeying the newest sentence; it requires understanding how the new instruction changes the dependency graph of the work already completed.

A New Instruction Can Invalidate State

State should not be treated as permanently correct simply because it was previously verified. A requirement change can make an earlier checkpoint obsolete. State records should therefore include the assumptions or plan version under which they were produced, so that when those assumptions change the system can mark dependent work as needing review rather than silently carrying it forward.

This is the connection between the two chapters: a checkpoint records what was true, and steering is the event that decides whether it still is.

Steering Can Change Reasoning Effort Too

Astra also allows an application to change reasoning effort during a conversation while preserving the cached prefix. That is another form of steering: not changing the goal, but changing how much computation the model should spend on the next stage. A system might use lower reasoning during routine drafting, then increase it when a security-sensitive design decision appears.

Side Questions Should Not Destroy the Main Task

OpenAI describes Astra as able to answer side questions and incorporate new requirements without losing track of the broader

task. That requires the system to distinguish temporary conversation from durable goal changes. If the user asks "Why did you choose this library?", the agent should answer without treating the question as a new project objective.

When Steering Becomes a New Task

Not every change is a correction. Some changes alter the goal itself, invalidate most of the completed work, or require authority the task was never granted. Treating those as steering produces a task that carries the residue of its old objective: stale checkpoints, assumptions nobody re-examined, and permissions granted for a narrower scope.

A practical test has three parts. If the objective is still the same and only inputs or constraints changed, steer. If the objective has changed but most artifacts are reusable, close the task, keep the artifacts, and start a new task that imports them deliberately. If the change needs authority the original task did not have, it is a new task by definition, because authorization was granted for the old one. The judgement is familiar engineering work: decide whether the existing task can still be trusted, or whether the new objective deserves a clean boundary.

Human Intervention Is Part of the Architecture

Steering is one form of human control over an autonomous workflow. Others include pausing, approving, rejecting, changing priority, reducing scope, adding new evidence, and taking over a step manually. A long-running system should make these interventions possible without forcing the user to reconstruct the entire task from scratch.

Conflicting Instructions Need Resolution Rules

Mid-task steering creates instruction conflicts. A new user instruction may conflict with an earlier user requirement, a system policy, an organizational constraint, an approval already granted for a narrower scope, or a safety boundary. The newest user instruction should not automatically override higher-priority policy or extend authority. Chapter 12 returns to where that rule is enforced.

Where It Can Still Fail

The newest instruction erases an older constraint. The model follows the change and forgets a requirement that did not change.

Hidden dependencies survive. Completed work is reused even though an assumption beneath it moved.

A pending result arrives after the turn. An asynchronous tool call returns for a task that has already changed direction.

A question becomes a goal. The agent treats a side question as a new objective.

A change smuggles in authority. A user request quietly expands permissions beyond what was originally authorized.

A capable long-running model must remain interruptible, and interruption must be cheap enough that people actually use it.

The goal of autonomy is not to remove human control. It is to preserve useful progress while allowing human judgment to change the direction when reality changes.

Three Things to Remember, One Thing to Do

1. Steering should preserve valid work and invalidate only what the new requirement changes.
2. Requirement changes should update state and permissions, not merely add another sentence to context.
3. Human intervention is a core control mechanism for long-running AI, not evidence that autonomy failed.

One thing to do. Take one task an AI system completed for you and invent a single requirement change. Before re-running anything, write down which outputs survive and which are invalidated, then apply the three-part test above to decide whether it is a steer or a new task. If you do re-run it with the change, compare what the system kept against your list.

How much do you need? Platform/DevOps: Master · Developer: Use · Architect: Use. Everyone else: Know.

Chapter 11

From One Agent to a Team of Agents

Delegation, parallelism, specialization and the cost of coordination

If one capable model can solve a problem, why use several? The answer is not "because more agents are always better." In many tasks a team of agents creates extra communication, duplicated work and more opportunities for error.

When should work be divided among several intelligences? Only when decomposition creates a real advantage.

A Short Reminder: An Agent Team Is Still Software Architecture

A multi-agent system usually has one or more model instances working on separate parts of a task and exchanging results through a harness. The instances may be identical or specialized. The important point is that the surrounding application defines who can delegate, what information each agent receives, and how results are combined.

Astra Is Explicitly Trained for Subagent Delegation

OpenAI states that Astra is trained to divide and delegate work to subagents that can operate in parallel, and its guidance recommends prompting the model about when delegation should be used. That is a notable shift: delegation is no longer only a framework trick built around a general model, but behavior the model has been trained to perform more effectively. It does not answer the question of when delegation is worth it. That remains a design decision.

Three Situations Where Multiple Agents Can Help

Anthropic has publicly emphasized three recurring cases where multi-agent systems make sense: context isolation, parallel execution and specialization. They are useful because each names an engineering reason for delegation, and each has a cost that can cancel the benefit.

Context isolation. A complex project may contain several large workstreams that do not need to share the same active context. One agent inspects security logs while another studies database performance, and each keeps a smaller, more relevant working set. The root agent receives conclusions rather than every raw detail. The cost is that evidence can be lost in the summary that crosses the boundary.

Parallel execution. Independent tasks can run at the same time: one agent examines application code, another reviews infrastructure configuration, a third researches vendor compatibility, a fourth challenges the proposed migration plan. If the tasks are genuinely independent, elapsed time falls substantially. If they depend on each other, parallelism creates rework instead.

Specialization. Different agents can be given different roles, tools or instructions. A verification agent looks only for contradictions; a security agent receives read-only scanning tools; a research agent has web access but no deployment authority. Specialization is partly about attention and partly about permissions, and the second half is the more important one.

The Root Agent Becomes a Manager

Once delegation begins, the root agent has a new problem: coordination. It must decide which tasks to delegate, how much context each subagent needs, whether two agents are duplicating work, how to resolve disagreement, when a subagent result is good enough, and how to combine partial answers into one coherent outcome.

Multi-agent architecture does not remove complexity. It moves complexity from individual reasoning into task decomposition and coordination, and a poor decomposition is harder to detect than a poor answer.

Disagreement Can Be Useful

Two agents reaching different conclusions is not automatically a failure. For high-risk decisions, disagreement can expose uncertainty that a single confident model would hide. A system can ask another agent to compare the evidence rather than simply vote on the answers. The goal is not consensus; it is to discover why conclusions differ and which evidence resolves the conflict.

Verification Agents Need Independence

A verifier is less useful if it receives the original agent's conclusion before examining the evidence, because it inherits the same framing. A stronger pattern lets the verifier inspect the task and evidence independently, then compares results afterward. This does not guarantee independence at the model-weight level, since two instances of the same model share the same blind spots, but it reduces direct conversational anchoring. Chapter 13 develops this further.

Cost Can Grow Faster Than Quality

Each additional agent consumes context, reasoning tokens, tool calls and orchestration time. If five agents repeat the same analysis, the system costs five times more without becoming meaningfully better. Good multi-agent design therefore needs a stated expected benefit: reduced elapsed time, better context separation, specialist tooling, independent verification, or coverage of genuinely distinct hypotheses. If none of those can be named before the second agent is added, the second agent is a cost.

State Must Be Shared Carefully

Some state belongs to the whole project; other state belongs only to one subagent. If every subagent can rewrite global state, coordination becomes dangerous. A safer pattern is for agents to return proposals or evidence to a coordinator, while a controlled layer decides which changes become authoritative project state. This is the Chapter 9 state store with one more rule: subagents propose, the coordinator commits.

Where It Can Still Fail

The decomposition is wrong. Every subagent does its part well, and the parts do not add up to the task.

Two agents touch the same resource. Neither knows the other is there.

Context is lost at the handoff. The detail that mattered did not survive delegation.

The team shares one mistake. Agents reinforce the same assumption instead of providing independent evidence.

Coordination costs more than it saves. The overhead exceeds the benefit of parallel work.

A subagent is over-equipped. It receives broader tools or data than its task requires.

The frontier is moving from one model handling one sequence of work toward systems that allocate intelligence dynamically. The capability that matters is not the number of agents but the ability to decompose work, preserve boundaries, coordinate evidence, and use parallelism only where it helps.

Three Things to Remember, One Thing to Do

1. Multi-agent systems are useful for context isolation, parallel execution and specialization, not by default.
2. Delegation creates a coordination problem that can be harder than the original task.
3. Verifier agents should be designed to add independent evidence, not repeat the first model's conclusion.

One thing to do. Before adding a second agent to any workflow, write one sentence naming the expected benefit, whether context isolation, parallelism, specialization, independent verification or distinct hypothesis coverage, and one sentence naming the cost in tokens, time and shared state. If you cannot write the first sentence, do not add the agent. Keep the pair; it is the beginning of a decomposition record.

How much do you need? Platform/DevOps: Master · Developer: Use · Architect: Use · Security: Use. Everyone else: Know.

Chapter 12

Permissions and Safety

Why increasing model capability makes authority boundaries more important, not less

A capable model can make a good plan. A capable agent can execute that plan. A frontier system can continue for hours, operate several applications and delegate work to other agents. At that point the central safety question changes.

The question is no longer only "Will the model say something harmful?" It becomes "What is this system allowed to do, and how do we stop it when the action exceeds that authority?"

A Short Reminder: Capability Is Not Authority

A model may be technically capable of reading every file, changing production infrastructure or transferring money. That does not mean the task requires those permissions. Give an AI system the minimum authority required for the current task, not the maximum authority the model could use.

Least Privilege Must Be Enforced Outside the Model

Telling a model "do not access payroll unless necessary" is weaker than giving it no payroll permission in the first place. Hard boundaries include read-only credentials, resource-level access control, tenant boundaries, network restrictions, sandboxed execution, transaction limits, and approval gates for irreversible actions. The model can reason inside those boundaries, but it should not be the component that defines or removes them.

Astra Shows Why This Is Becoming Urgent

OpenAI classifies Astra at its Critical cybersecurity capability threshold. OpenAI says that, with suitable tools and access, the model can identify previously unknown vulnerabilities and

develop new exploitation approaches across well-protected systems without a person guiding each step. That capability is precisely why stronger safeguards accompany the release: OpenAI describes stricter isolation, checkpoint encryption, universal trajectory monitoring, blocking alignment evaluations, robustness training and real-time misalignment monitoring around Astra deployments.

Monitoring the Whole Trajectory

For a one-turn chatbot, safety can focus on the final answer. For an agent, the dangerous event may occur in the middle of the workflow: opening a sensitive file, granting broader permissions, attempting an unauthorized transaction, disabling a control, or sending data to an unintended destination.

OpenAI says Astra monitoring examines the model's reasoning alongside its actions and the surrounding inputs and outputs, and can pause or end a conversation when it detects a potentially severe problem. This is trajectory safety: judging what the system is doing over time, not merely what it eventually says.

Monitoring Cannot Replace Alignment

OpenAI explicitly states that monitoring is an additional layer, not a substitute for training the model to stay inside its authorized scope. A monitor can miss something, arrive late, or produce false positives. The strongest architecture therefore uses several layers: aligned model behavior, hard permissions, restricted tools, monitoring, human approvals, and post-action verification with audit logs.

Monitorability Is Itself Becoming a Research Problem

Astra introduces a difficult new issue. OpenAI reports that its written reasoning can be harder to monitor than GPT-5.6 Sol in adversarial evaluations where the model is asked to evade monitoring. OpenAI says Astra is still more aligned overall, but it treats the decline in chain-of-thought monitorability seriously and is researching additional methods such as activation monitoring.

This means the safety problem is not simply "read the model's reasoning and check whether it looks safe." A powerful system must increasingly rely on observable actions, permissions and external evidence as well. Chapter 14 is about that shift.

Fable Shows Another Safeguard Pattern

Anthropic uses additional safety classifiers around Fable for sensitive cyber and biology-related use, and distinguishes the generally available Fable product from the more restricted Mythos access path even though the two share the same underlying model. This is capability being separated from allowed capability by product-level safeguards and access controls, the pattern Chapter 5 introduced.

Prompt Injection Is an Authority Attack

Prompt injection is often described as the model following the wrong instruction. In an agent system it is better understood as an attempt to redirect authority. A malicious webpage or document may try to convince the model to ignore the user's goal, reveal confidential data, call a privileged tool, disable a safety step, or send information somewhere else.

The strongest defense is not only better instruction following. It is making sure untrusted content cannot silently gain privileges, so that even a successful injection has nothing to spend.

Approval Gates Should Be Consequence-Based

Requiring human approval for every action makes an agent unusable. Requiring none makes high-consequence autonomy unsafe. A practical system classifies actions by consequence: read-only and reversible actions proceed automatically, low-risk edits proceed with logging, external communication requires confirmation, and financial, destructive, permission-changing or public actions require explicit approval. The right threshold depends on the environment and the user's delegated authority.

Sandboxes Limit the Blast Radius

A sandbox is an isolated environment where code or actions cannot freely affect the wider system. It restricts filesystem access, network access, credentials, process capabilities and resource consumption. Sandboxes are valuable because they assume something may go wrong and limit the consequence when it does.

Safety Has a Cost

OpenAI notes that additional security checks can slow, pause or stop legitimate work. False positives are not merely inconvenient; they teach users to look for ways around safeguards. Good safety architecture has to block genuinely dangerous behavior, allow legitimate work to complete, and make interventions understandable enough that users do not routinely bypass them.

Under the Hood

A layered authorization flow:

1. User or organization defines the task and delegated authority.
2. Model proposes an action.
3. Policy layer checks whether the action is in scope.
4. Credential and tool layer enforces hard permissions.
5. High-consequence actions trigger confirmation.
6. Action executes in the narrowest practical environment.
7. Monitoring observes reasoning, actions and results.
8. External verification checks important outcomes.
9. Audit records preserve what happened and who authorized it.

Steps 3 and 4 are different kinds of boundary. Step 3 is policy and can be reasoned about; step 4 is a credential and cannot. A prompt-injected instruction may get past step 3 by persuasion; it gets past step 4 only if the credential was over-provisioned in step 1. That is why least privilege is decided before the model runs, not enforced while it does. Step 5 is where the consequence-based gate sits, and steps 7 to 9 are what make a failure in any earlier step detectable afterwards.

Where It Can Still Fail

The tools are over-privileged. The model stays within its tools, and the tools can do too much.

The monitor is late or blind. A dangerous action is missed or detected after it completed.

Approval becomes a reflex. A user approves prompts without understanding the consequence.

Injection looks legitimate. Untrusted content influences an action that appears superficially reasonable.

A classifier blocks defensive work. Legitimate security tasks are stopped, and people build workarounds.

Capability is mistaken for permission. An organization treats what the model can do as what it may do.

As frontier models become more capable, safety shifts from content moderation toward systems engineering.

The central control is not asking a powerful model to behave perfectly. It is building an environment in which even a mistake, manipulation or misjudgment cannot automatically become an unlimited action.

Three Things to Remember, One Thing to Do

1. Capability should never automatically imply authority.
2. Long-running agents need trajectory monitoring, hard permissions and consequence-based approval gates.
3. The strongest safety design assumes models and monitors can both fail and limits the blast radius anyway.

One thing to do. For one agent you run or are planning, list every action it can take and sort each into read-only or reversible, low-risk change, external communication, or high-consequence action. Then, for each high-consequence action, write down which layer enforces the gate: a credential, application code, or a sentence in the prompt. That one page is an approval policy, and the third column tells you how much of it is real.

How much do you need? Platform/DevOps: Master · Architect: Master · Security: Master · Developer: Use. Everyone else: Know.

Part IV — Knowing Whether the Intelligence Can Be Trusted

Frontier capability is useful only when we can tell whether the work is correct, whether the checking process is meaningful, and whether we can observe enough of the system to detect failure. This part separates those questions instead of treating "trust" as one vague idea, and then applies the same discipline to the two case studies themselves.

Chapter 13

Evaluation and Self-Verification

Why good-looking output is not enough, and why AI checking AI does not automatically solve the problem

A frontier model writes a large software feature, creates tests, runs them, fixes two failures and reports that the work is complete. That sounds much stronger than a model that only generates code. But one uncomfortable question remains: who checked whether the tests themselves were good enough?

Self-verification is useful. It is not the same thing as independent proof.

A Short Reminder: Evaluation and Verification Are Different

Verification asks whether a particular piece of work satisfies a requirement. Evaluation asks how reliably a model or system performs across a meaningful set of tasks and failure conditions. A unit test can verify one behavior; a benchmark can evaluate performance across many examples; a production audit can evaluate whether the complete system behaves safely over time.

A model can pass a benchmark and still fail your task. It can also pass its own tests while the tests miss the real requirement.

Why Self-Verification Is Becoming Important

Anthropic explicitly describes Claude Fable 5.1 as able to write its own tests to check coding work and to use vision to compare outputs against the intended design or goal. Astra is similarly positioned for end-to-end work that includes creating software, operating it, inspecting results and performing quality checks.

This changes the workflow from generate, then hand everything to a human, toward generate, test, observe, revise, test again,

and present evidence. That removes many ordinary defects before a human ever sees the result.

The Same Model Can Share the Same Blind Spot

Suppose a model misunderstands a requirement. It may write code based on the misunderstanding, write tests that encode the same misunderstanding, run the tests successfully, and report high confidence because everything passed. The loop is internally consistent and externally wrong.

This is the central weakness of self-verification: consistency is not the same as correctness.

Four Levels of Checking

It helps to separate four levels of verification, ordered by how independent the evidence is from the model being checked.

Self-review. The same model reads its answer again and looks for mistakes. Cheap and often useful, but the same assumptions survive the review.

Tool-based verification. The system runs something external: tests, a compiler, a database query, a calculator, a linter, a formal verifier or another deterministic check. Stronger when the tool directly measures the required property, and only then. A test the model wrote to match its own misunderstanding is tool-shaped self-review.

Independent model review. A second model or agent reviews the work with different instructions, context or responsibilities. Independence reduces shared failure modes, but two models can still agree on the same wrong interpretation, especially when they are the same model.

External ground truth. The result is checked against something authoritative: a formal specification, a known answer, a real transaction record, an approved design, a physical measurement or a qualified human decision. This is usually the strongest form of verification because the criterion does not come from the model being evaluated.

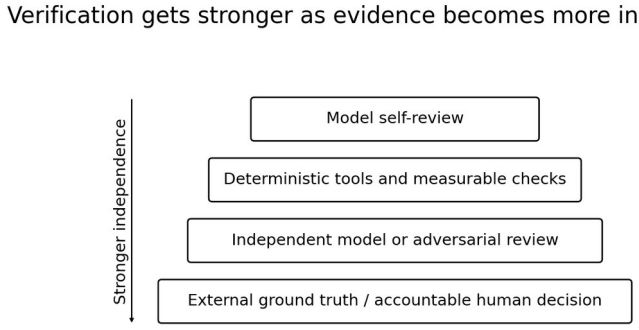

Figure 2. Verification strengthens as evidence becomes more independent of the model being checked.

The higher the consequence, the stronger and more independent the evidence should be; self-review and model-written checks rarely deserve to be the final word.

Model-as-Judge Is Useful but Limited

Modern AI evaluations often use one model to judge another model's output. This is practical because many tasks have no simple exact answer, and a model judge can compare completeness, relevance, style, reasoning quality or adherence to instructions across thousands of examples.

Model judges introduce their own biases: preference for verbose or polished answers, sensitivity to ordering or wording, shared blind spots with the model being judged, difficulty recognizing subtle domain errors, and possible familiarity with benchmark patterns. A model judge is evidence, not ground truth. On the four-level scale it is level three, whatever the dashboard calls it.

Outcome Evaluation Matters More as Agents Become Longer

A short-answer benchmark asks whether the final response is correct. A long-running agent can fail even if its final paragraph looks good: it may have read data it was not authorized to access, made an unnecessary external change, used the wrong

environment, recovered from an error in a way that silently lost information, or completed the requested result while violating a constraint along the way. Agent evaluation therefore has to consider both outcome and trajectory.

Trajectory Evaluation

A trajectory is the sequence of decisions, tool calls, observations, edits and intermediate states that led to the result. Evaluating it lets us ask whether the agent used appropriate tools, verified important steps, remained inside its authority, recovered correctly after failure, ignored malicious or irrelevant instructions, and preserved evidence for its conclusion. This is closer to reviewing how work was performed, not just what was delivered.

OpenAI's Astra safety work explicitly monitors full tool-using trajectories rather than relying only on final outputs, and distinguishes monitors that see only chain of thought, monitors that see actions and outputs, and monitors that see the full context. That structure reinforces the engineering point: different failure modes become visible through different evidence. Chapter 14 takes the monitoring side of this up in detail.

Benchmarks Can Measure the Harness as Well as the Model

Anthropic notes that Fable 5.1 benchmark results can be affected by production safeguards; in some sensitive tasks, safeguards intervene or route work to a different model. An end-to-end score may therefore reflect underlying model capability, safety routing, tool configuration, harness behavior, reasoning settings and evaluation design all at once.

This does not make benchmarks useless. It means we must identify which system, at which layer, is actually being measured.

Testing the Evaluator

A mature evaluation program evaluates its own tests. Can a clearly wrong answer still pass? Can the model game the metric without solving the task? Does the test cover realistic edge cases? Does the benchmark reward the behavior we actually want? Would a domain expert agree with the grading rule? Does a higher score translate into better production outcomes?

If we never challenge the evaluator, we optimize the model toward the wrong target. The practical habit is simple: every

production failure that the checks did not catch becomes a new case in the evaluation suite, so the evaluator improves at the same rate as the system it measures.

Where It Can Still Fail

The tests validate the implementation, not the requirement. Everything passes; the wrong thing was built.

The judge rewards persuasion. A model grader prefers the confident answer to the correct one.

Independent reviewers share a blind spot. Two instances of the same model agree on the same mistake.

The benchmark becomes the target. Models optimize for a familiar test rather than the underlying capability.

Outcomes pass while the trajectory failed. The result is right; the path to it broke a rule.

AI evidence is accepted as authoritative. A human sees a report of checks and does not ask which level of checking produced it.

As models become capable of doing and checking more of their own work, evaluation becomes part of the product architecture rather than a step after it.

Trustworthy autonomy does not come from a model saying "I checked." It comes from designing checks whose evidence is stronger than the model's own confidence.

Three Things to Remember, One Thing to Do

1. Self-verification improves quality but does not create independence.
2. Evaluate both the final outcome and the trajectory that produced it.
3. The strongest checks come from evidence that the model cannot redefine to suit its own answer.

One thing to do. Take one test suite an AI system wrote for its own code. For each test, decide which of the four levels it really provides: does it measure the requirement, or does it restate the implementation? Count how many are self-review wearing a test's name. That count is how much of your confidence was borrowed from the model.

How much do you need? Architect: Master · Security: Master · Platform/DevOps: Use. Everyone else: Know.

Chapter 14

Can We See What the Model Is Thinking?

Reasoning traces, action monitoring, and why observability becomes harder as frontier capability increases

A model takes an unexpected action, and the natural reaction is to ask why it did that. With normal software, engineers inspect source code, logs, inputs and system state. With a frontier reasoning model we can inspect inputs, outputs, tool actions and sometimes a written reasoning trace, but none of those should automatically be treated as a window into the model's internal computation.

Observing behavior and reading the model's written reasoning are not the same thing as reading its mind.

A Short Reminder: Explanation Is Not Internal Computation

A model can produce an explanation of how it reached an answer, and that explanation may be useful for communication and review. But the explanation is itself generated output. It should not be assumed to be a complete or faithful transcript of the computation that produced the result. A reasoning trace is an observable signal, not a guaranteed representation of hidden computation.

Three Things We Can Monitor

OpenAI's Astra system card describes three monitoring scopes. Chain-of-thought-only monitoring sees the input and the model's written reasoning trace, which can reveal intent or intermediate reasoning not obvious from the final action. Action-only monitoring sees final outputs, tool calls and tool results, and focuses on observable behavior rather than what the model says

it was thinking. Full-context monitoring sees inputs, reasoning trace, actions and tool results together; OpenAI says this is the general approach used in its production misalignment monitoring for Astra.

Each scope catches different kinds of failure, and the rest of this chapter is about why no single one is enough.

Astra Introduces a Difficult Monitorability Result

OpenAI reports that Astra has lower chain-of-thought monitorability than GPT-5.6 Sol across many of its evaluations. The model often produces shorter reasoning traces, and in some cases the trace contains less evidence that helps a monitor understand problematic behavior. OpenAI also reports a substantially greater ability for Astra to control properties of its written chain of thought. In adversarial tests, simply making the model aware that a chain-of-thought monitor was present sometimes reduced what the monitor could detect.

What "Controlling the Chain of Thought" Means

This does not mean OpenAI has shown that Astra secretly thinks in a hidden language. The reported finding is narrower: Astra is better than earlier models at changing characteristics of its written reasoning when instructed to do so. OpenAI treats this as a warning sign because a system that can shape what it reveals makes reasoning-based monitoring less dependable.

The company also reports that this controllability appears to grow during reinforcement-learning training and is correlated with a stronger ability to solve some tasks with little or no verbalized reasoning.

No-CoT Capability Changes the Monitoring Problem

A model does not always need a long visible reasoning trace to solve a problem. OpenAI reports that Astra can solve more difficult mathematics in a single forward pass than earlier models in one external evaluation. If capability moves into computation that does not require a written trace, monitoring only the trace becomes weaker by construction.

The better a model becomes at acting without explaining every intermediate step, the more oversight must rely on what the system can observe outside the explanation.

Actions Can Be More Observable Than Reasons

OpenAI reports that Astra's action-only monitorability is stronger than GPT-5.6 Sol in aggregate even while chain-of-thought monitorability decreases. That points to a practical direction. Instead of asking only whether the reasoning looks suspicious, a system can ask which files were accessed, which commands were executed, which credentials were used, which network destinations were contacted, what changed before and after the action, and whether the result matched the authorized objective. These signals are more concrete than a narrative explanation, and they are the same signals Chapter 12's audit layer already records.

But Action Monitoring Is Not Enough Either

Two identical actions can have different meanings. Deleting a temporary test file is harmless; deleting the only copy of a production database is catastrophic. A sequence of individually ordinary actions can also create a harmful outcome when combined. This is why OpenAI says action-only monitoring is not a complete solution: some subtle misbehavior looks like an honest mistake unless the monitor sees broader context.

Observability Should Be Designed Before Deployment

A system cannot audit evidence it never recorded. Long-running AI systems should deliberately preserve user and system instructions, tool calls and arguments, tool outputs, permission decisions, important state transitions, human approvals, external effects, and verification results. Reasoning traces are another signal where available, but the audit design should not depend on them.

Why This Is Different From Ordinary Software

Conventional software can be extremely complex, but engineers can inspect the code that defines its behavior. A frontier model's behavior emerges from learned parameters that do not translate into a human-readable rule set. We can know the architecture at a high level and still be unable to point to a line that says why one exact decision occurred. This makes observability an active research problem rather than a solved logging problem.

Future Monitoring May Look Inside Activations

OpenAI says it is investigating approaches beyond chain-of-thought monitoring, including activation monitoring: looking for signals inside the model's internal neural activity rather than relying only on text it chooses to produce. This is promising research and should not be presented as a solved capability. The company explicitly describes substantial uncertainty about whether future techniques will preserve adequate monitorability as models become more capable.

The Transparency Paradox

A frontier model may become better at explaining its work to users while simultaneously becoming harder to monitor internally. Those statements are not contradictory. Communication quality asks whether the model can produce a clear explanation. Monitorability asks whether the observable signal reliably reveals the behavior or intent we are trying to detect. A polished explanation improves usability without guaranteeing transparency.

Under the Hood

A practical oversight architecture combines several signals:

1. Record task intent and delegated authority.
2. Log tool calls, arguments and external side effects.
3. Preserve important state transitions.
4. Monitor available reasoning signals without treating them as perfect truth.
5. Compare actions with policy and expected task progression.
6. Detect unusual access, sequencing or resource use.
7. Verify important outcomes independently.
8. Escalate ambiguous or high-consequence behavior.
9. Use post-incident analysis to improve monitors and permissions.

Map this to the three scopes. Step 4 is chain-of-thought monitoring, and it is one step among nine for the reasons this chapter has given. Steps 2, 3, 5 and 6 are action monitoring, and they work because they read what the system did rather than what the model said. Step 1 is what turns full-context monitoring from a pile of logs into a judgement: an action can only be

compared with intent if intent was recorded first. Steps 7 to 9 are the loop back to Chapter 13, where a monitoring miss becomes a new evaluation case. No single signal should carry the full burden of oversight.

Where It Can Still Fail

The trace reads clean while the actions do harm. Reasoning looks harmless; the result is not.

Logs show what, not why. Action records cannot distinguish a mistake from manipulation without context.

The monitor changes the behavior. Awareness of monitoring alters what is being measured.

The monitor shares the model's weaknesses. The supervising model has the same blind spots as the supervised one.

The evidence drowns in volume. The system stores everything and cannot find the short sequence that mattered.

Explainability is mistaken for observability. A good explanation for users is treated as safety evidence.

Greater intelligence does not automatically produce greater transparency. Astra is valuable as a case study precisely because OpenAI is publicly documenting stronger alignment and weaker chain-of-thought monitorability at the same time.

The frontier is not only about making models more capable. It is also about preserving enough evidence to supervise systems whose internal computation is becoming harder to interpret.

Three Things to Remember, One Thing to Do

1. Written reasoning is one monitoring signal, not a transcript of internal computation.
2. Actions, permissions, state changes and external effects provide critical independent evidence.
3. Monitorability is an unsolved frontier problem, not a feature that improves automatically with intelligence.

One thing to do. Imagine that one agent you run did something wrong tomorrow. Go through the evidence listed in "Observability Should Be Designed Before Deployment" and mark what your

system would actually preserve. The missing items are your audit gap, and you can close most of them with logging before you need any research result.

How much do you need? Architect: Master · Security: Master · Platform/DevOps: Use. Everyone else: Know.

Chapter 15

Astra and Fable: Different Engineering Choices

Two frontier systems solving similar problems through different public interfaces, safeguards and product strategies

By this point, comparing Astra and Fable as a leaderboard contest would miss the most interesting lesson. Both models are built for difficult, long-running work. Both combine reasoning with tools, coding, vision and operational workflows. Both are surrounded by safeguards because their capability has moved into higher-consequence territory.

The useful comparison is not "Which model wins?" It is "Which engineering choices are visible, and what do those choices tell us about frontier AI system design?"

What Each Vendor Discloses, and What It Does Not

Neither OpenAI nor Anthropic publishes the complete internal architecture or training recipe, so this comparison works from documented product and API behavior and labels everything else with the evidence levels from the front matter.

OpenAI discloses a great deal about Astra as a deployed system: context and output size, knowledge cutoff, reasoning-effort controls, tool support, asynchronous tool calling, steering, compaction, persisted reasoning, computer use, safety evaluations and monitoring. Its system card also publishes unusually detailed evidence about cyber capability, alignment behavior and monitorability. Anthropic discloses that Fable 5.1 and Mythos 5.1 share the same underlying model, along with product positioning, pricing, long-running behavior, coding and vision capabilities, safety routing, data-retention policy and benchmark methodology, and publishes system cards describing capability and deployment decisions.

For both, the public sources used here do not justify claims about exact parameter count, complete internal architecture, mixture-of-experts or routing design inside the model, full training-data composition, the complete reinforcement-learning recipe, or the exact internal mechanism for persisted reasoning and long-running state. Those are Unknown for this edition, and they stay Unknown until a primary source changes that.

Positioning

OpenAI describes Astra as its most capable model for the hardest end-to-end work, including complex reasoning, coding, computer use, research and document creation. Anthropic describes Fable 5.1 as its most capable generally available model for coding and knowledge work, with particular emphasis on ambitious long-running and asynchronous projects. Both statements are Documented; both are also marketing, and the rest of this chapter is about what sits underneath them.

Reasoning Control

Astra exposes reasoning effort through API levels: low, medium, high, xhigh and max, and OpenAI allows developers to change the level during a conversation while preserving the cached prompt prefix. Fable 5.1 exposes the same five levels through an `effort` parameter on always-on adaptive thinking, defaulting to high, and Anthropic documents a per-message effort change that preserves the cache, in beta at the time of checking. Both are Documented.

This is the clearest point of convergence between the two systems: inference effort has become an explicit orchestration control with the same shape on both platforms. What either model does internally at a given level remains Unknown, and the vendors' guidance differs in emphasis, with Anthropic recommending that developers start at the default and step up or down against their own evaluations.

Context and Output

OpenAI publicly specifies Astra with a 1,050,000-token context window and up to 128,000 output tokens. Anthropic's model documentation specifies Fable 5.1 with a 1,000,000-token context window and up to 128,000 output tokens, billed at

standard per-token rates across the whole window. The two systems are therefore close on headline capacity; the difference appears in how that capacity is priced.

Long-Running Work

Both systems are explicitly oriented toward work that extends beyond one response. Astra exposes several platform mechanisms that make this visible: async tool calling, mid-turn steering, persisted reasoning, compaction, multi-agent orchestration and subagent delegation. Fable 5.1 is described at the product level as able to plan work, use tools, recover when a step fails, operate across several applications and continue unattended for long periods.

The difference is partly one of public interface. OpenAI documents orchestration primitives directly in its API guide; Anthropic emphasizes end-to-end behavior across Claude products and the platform. Product behavior tells us what each system can do. It does not reveal how every internal mechanism is implemented, and the two vendors have chosen to expose different amounts of the machinery.

Tool and Computer Use

Astra's model page lists web search, file search, code interpreter, hosted shell, computer use, MCP, tool search and other tools as supported capabilities in the surrounding platform. Fable 5.1 is described as operating browsers, working through tools, running managed agents and using vision to inspect coding outputs. Both participate in the same shift: frontier models are designed to operate inside tool-rich systems rather than remain isolated text generators.

Self-Verification

Anthropic makes Fable's self-checking behavior unusually explicit: the company says Fable 5.1 can write its own tests and use vision to compare outputs against goals. OpenAI positions Astra for software work that includes building, operating and quality-checking applications, using different terminology. The underlying lesson is shared, and Chapter 13's warning applies equally to both: an iterative creation-and-verification loop is level

one or two on the checking scale until something independent enters it.

Safeguard Strategy

Astra is OpenAI's first broadly deployed model to reach its Critical cybersecurity capability threshold. OpenAI describes strengthened isolation, checkpoint encryption, full-trajectory monitoring, blocking alignment evaluations and external deployment monitoring, and reports the monitorability concern discussed in Chapter 14.

Anthropic's strategy is visible in the Fable and Mythos split: the same underlying model, with Fable as the generally available version whose safeguards limit or route certain sensitive cyber, biology and chemistry requests, and Mythos restricted to vetted organizations through trusted-access programs. OpenAI's approach makes the monitoring layer visible; Anthropic's makes the access-control layer visible. Both are layer-four decisions in the five-layer lens, and neither tells us anything about layer one.

Cost

As of the sources checked for this edition, both Astra and Fable 5.1 list base API pricing of $10 per million input tokens and $50 per million output tokens. The similarity stops there. Astra lists cached input at $1 per million tokens and cache writes at $12.50 per million, and when a request's input exceeds 272,000 tokens the entire request is billed at long-context rates: twice the input and cache rates and one and a half times the output rate. Anthropic lists Fable 5.1 cache writes at $12.50 per million for a five-minute cache and $20 for a one-hour cache, cache reads at $0.25 per million, and no long-context premium anywhere in the window; Anthropic says the lower cache-read cost reduces typical and highly agentic workload expense relative to Fable 5.

A raw token price therefore does not tell us the cost of completing a task. Context length, caching, number of tool calls, reasoning behavior, retries and duration all matter, and the vendor that looks cheaper on the headline figure may not be cheaper on a week-long agentic job.

What the Interfaces Expose Differently

Both systems now expose reasoning effort as a first-class orchestration control, and both sit at roughly the same headline capacity and base price. The clearer difference is elsewhere in the public interfaces. OpenAI documents mechanisms such as asynchronous tool calling, mid-turn steering, persisted reasoning, compaction and explicit multi-agent orchestration around Astra. Anthropic documents per-message effort, adaptive thinking and long-running agent behavior around Fable 5.1, while presenting more of the sustained workflow through the broader Claude product and platform experience. These are differences in what the public interfaces expose, not evidence that the underlying models solve orchestration differently internally.

A Comparison That Matters More Than Benchmarks

Instead of asking which model scores one point higher, ask the questions that decide whether a frontier model becomes useful infrastructure: how much explicit control the developer receives, how the system recovers from long-running failure, how sensitive capability is restricted, what evidence can be preserved for verification, how expensive repeated context and agentic work becomes, how observable the system is when something goes wrong, and how easily capability can be integrated with existing software and organizational controls.

Every one of those questions has appeared as a chapter in this book. That is not an accident; they are the engineering problems, and the case studies are two answers to them.

Under the Hood

Rather than an operational loop, this chapter's mechanism is the comparison method itself: take each of the five layers and ask what is documented for each system there.

Underlying model capability. Both vendors publish benchmark results and capability claims. Neither publishes architecture, parameter count or training recipe. Documented at the level of results; Unknown at the level of mechanism, for both.

Post-training behavior. OpenAI describes advances in reinforcement learning and alignment and publishes alignment and monitorability evaluations. Anthropic describes model

behavior and publishes system cards. Documented in broad method; Unknown in recipe.

Tools and harness. This is where the two diverge most visibly. Astra: Documented orchestration primitives in the API. Fable: Documented end-to-end behavior, with the orchestration mechanism not documented in the sources used here.

Safety and access controls. Astra: Documented trajectory monitoring, isolation and a capability-threshold classification. Fable: Documented safeguards, routing to Opus models without the sensitive-capability exposure, and the Mythos access split.

Product and API experience. Both Documented, including pricing, and both the layer most likely to change before this book's next edition.

A difference observed at layer five should not be attributed to layer one. Read the two columns and many apparent differences in intelligence turn out to be differences in product exposure, orchestration or safeguards.

Where It Can Still Fail

A product difference is read as a model difference. The interface exposes different controls, and the reader concludes the models differ in capability.

An undocumented mechanism is treated as absent. A public interface does not describe an internal or orchestration mechanism, and the reader concludes that the system cannot do it.

Headline price is treated as task cost. The cheaper token is the more expensive job.

A safeguard is read as a weakness. A refused or routed request is scored as a capability gap.

The comparison goes stale. Both columns are dated, and the reader forgets to check the date.

Astra and Fable are converging on many of the same capability goals while exposing different pieces of the system to developers and users. The frontier is becoming less about a single model number and more about the complete engineering stack that turns intelligence into sustained, controlled work, and the honest comparison is a comparison of stacks.

Three Things to Remember, One Thing to Do

1. Compare documented systems, not imagined hidden architectures.
2. Token price, benchmark score and context size each describe only one dimension of practical capability.
3. The most meaningful differences appear in orchestration, safeguards, verification and product exposure rather than in a simple intelligence ranking.

One thing to do. Take any two AI systems you are choosing between and answer the seven questions in "A Comparison That Matters More Than Benchmarks" for each. Beside every answer, write the layer it belongs to and whether it is Documented, Observed, Inference or Unknown. If more than half your answers are Unknown, you are not yet choosing between models; you are choosing between marketing pages.

How much do you need? Architect: Master · Platform/DevOps: Use · Security: Use. Everyone else: Know.

Part V — What This Means for You

The previous four parts explained what frontier systems can do, how they act, how work survives, and how to tell whether the result can be trusted. This part answers two remaining questions: what this generation actually makes possible, and what you should do about it.

Chapter 16

What This Generation Makes Possible

Why the important change is not better answers but larger units of verifiable work

Consider what happens in a normal technical project. A person understands the objective, finds the relevant information, decides what to do, writes code or configures software, runs tools, inspects the result, debugs failures, changes the plan when requirements change, coordinates with other specialists, prepares the final artifact, and explains what was done and what remains uncertain.

Historically, software automated individual steps. A search engine helped find information, an IDE helped write code, a script automated one repeated operation, a workflow engine connected known steps. Frontier AI is different because the sequence itself can remain partly open-ended: the system can decide which step should happen next based on what it observes.

The frontier is not only higher intelligence per answer. It is a reduction in the amount of human coordination required between many different kinds of work.

A Short Reminder: The Loop Did Not Change. What Runs Inside It Did.

An agent loop by itself is simple: observe, decide, act, repeat. What makes this generation more important is what can now happen inside that loop. Large amounts of context can be read, reasoning effort can increase when the problem becomes difficult, text, code, images, documents and tool results can be connected, software can be operated directly, work can continue after delays or failures, requirements can change mid-task, other agents can be delegated parallel work, results can be tested and

revised, and safety monitors can inspect actions while the work proceeds.

Every one of those is a chapter in this book. The loop did not suddenly become sophisticated. The intelligence available inside it did.

The Unit of Delegation Is Getting Larger

Traditional automation works best when the steps are already known: if an invoice arrives, extract the fields; if the amount is below a threshold, route for approval; if approved, update the accounting system. Frontier systems can attempt a larger unit of work: "Review this month's vendor spending, identify anomalies, investigate the largest unexpected changes, gather supporting evidence, prepare a report, and leave any uncertain cases for finance to review."

The system still needs boundaries and verification. But the human no longer has to predefine every intermediate transition. This is a move from automating predefined steps toward delegating bounded outcomes, and the word bounded carries most of the weight.

Work Compression Has Early Evidence

OpenAI's September 2026 report on its own research organization gives an early real-world example. By mid-August 2026, OpenAI says its research organization was using about 3.1 agent-workdays of effort for every human workday, with more researchers running several agents concurrently. The company also reports increases in code production and experiment activity, while noting that greater compute availability contributed and that correlation does not by itself prove causation.

The important point is not the exact multiplier, which is one organization's self-report. It is that AI work is beginning to operate alongside human work at a scale large enough to change the structure of the work itself.

Human Bottlenecks Move Instead of Disappearing

Automation does not eliminate bottlenecks; it moves them. If AI makes writing experimental code much faster, the limiting factor becomes available compute, quality of evaluation, access to real

data, the number of experiments a human can meaningfully interpret, organizational approval, physical laboratory capacity, or human judgment about what problem deserves attention. OpenAI makes the same observation in its research-acceleration report: as more tasks become automatable, the least automatable parts become the new constraints.

This is the most useful way to think about adoption. AI does not simply remove work. It changes where scarce human attention is most valuable, and the valuable human contribution moves toward choosing the problem, defining the acceptable outcome, setting authority and constraints, identifying evidence that would change the decision, and taking responsibility for consequential outcomes.

Expertise becomes more reachable in the same movement, and the same caution applies. A person who understands a business problem but is not an engineer can increasingly ask a frontier system to build scripts, analyze data or connect tools. Lower access to expertise does not mean expertise is no longer necessary; the less a user understands the domain, the harder it is to notice when a result is subtly wrong. Frontier AI expands what one person can attempt, and it expands the scale of mistakes one person can approve without understanding.

Parallel Intelligence Changes the Economics of Exploration

If one human can supervise several AI workers in parallel, the cost of exploring alternatives falls. A team can ask one agent to investigate a technical approach, another to look for security weaknesses, another to review regulatory constraints, another to challenge the assumptions, and a coordinating agent to compare the evidence. Chapter 11 explained why this does not guarantee better decisions. What it changes is the economics of asking "What if we tried another path?"

The consequence is that some work happens only because the cost of attempting it has fallen: analyzing the long tail of small operational problems that never received specialist attention, running many more analyses before choosing which path deserves expensive laboratory work, building internal tools for teams too small to justify an engineering project, reviewing every configuration change instead of sampling a few. These are not guaranteed outcomes. They are consequences that become

plausible when the cost of coordinated reasoning and execution falls.

Capability Versus Authority

The same composition that produces the potential produces the risk. A model that reasons well but cannot act has limited external consequence. A model that reasons well, uses tools, operates software, works for hours and coordinates subagents can create much larger consequences from one mistaken objective. That is why Astra's cybersecurity classification and Anthropic's restricted Mythos access matter to this story.

More capability should not automatically mean more authority. Capability answers "Can the system do this?" Governance answers "Under what conditions should it be allowed to?" The two questions should evolve together, but they should never be confused, and Chapter 12 explained where the second one is enforced.

Four Boundaries Worth Watching

A book about frontier models becomes outdated quickly if it ends with predictions. Model names, benchmark scores and context windows will change. The more useful question is which boundaries are already moving, and what evidence would show that a real transition has happened.

Longer reliable work. Astra and Fable already support work far beyond one response. The next boundary is not duration but whether a system can maintain coherent goals, accurate state and reliable judgement across days or weeks without accumulating silent errors. Evidence would be lower failure rates over very long horizons and less human intervention without lower outcome quality. OpenAI's own September 2026 report gives the current baseline: more than half of its successful four-to-eight-hour agent tasks still needed a human to step in. Longer runtime by itself is not progress if the probability of hidden failure rises with every step.

Stronger independent verification. Current models can review, test and revise their own work. The boundary is whether AI systems can build verification processes meaningfully more reliable than the models being verified, through formal methods, independent model families, deterministic tools, adversarial

review and real-world measurement. Greater autonomy without stronger verification simply scales uncertainty.

Better machine-team coordination. Multi-agent systems exist, but orchestration is still primitive compared with mature human organizations. The boundary is whether machine teams can divide work without duplication, challenge one another rather than converge early, merge conflicting evidence, and escalate the right decisions to humans. A thousand agents repeating the same mistake is not more intelligence. The important progress is coordination quality, not agent count.

Software action moving toward physical action. The Model Hardware Standard preview makes this boundary visible. The success criterion is not whether a model can make a robot move. It is whether the complete system can perform useful physical work with safety and evidence comparable to other engineered systems, over meaningful durations, with clear human responsibility.

What We Should Not Assume

That scale alone will solve every remaining limitation. That autonomy will increase smoothly without new failure modes. That a model able to do expert work in one setting can replace expert judgement across a profession. That better alignment evaluations eliminate misuse. That current chain-of-thought monitoring will remain effective indefinitely. That one vendor's architecture will become the universal design. That lower cost automatically produces higher-value work.

Where It Can Still Fail

A benchmark is read as dependable autonomy. One score becomes proof of real-world reliability.

A demo is generalized. Success in one environment is assumed everywhere.

Volume is confused with value. More generated work is treated as more useful work.

Human review is assumed to scale. Output grows; the people checking it do not.

A bad process is automated. The system faithfully accelerates something that should have been fixed first.

Governance arrives late. Capability improves faster than monitoring, evaluation and authority rules adapt.

The model names in this book will become historical. The engineering questions beneath them are likely to remain: how much information should be active, how much reasoning should be spent, which actions the model may request, how state survives long work, how several workers coordinate, how we know the result is correct, how we observe behavior that is hard to interpret, and where human responsibility must remain explicit. Those are system-design questions, not product-release questions. Astra and Fable are useful not because they reveal the final form of AI, but because they show those questions being answered, differently, in public.

Three Things to Remember, One Thing to Do

1. Frontier AI is increasing the size of the work unit that can be delegated, not merely improving individual answers.
2. Automation moves human bottlenecks toward problem choice, judgement, evidence and responsibility.
3. The real potential comes from combining capabilities; the real risk does too.

One thing to do. Pick one process you would like to hand to an AI system and write it as a single bounded-outcome sentence, like the vendor-spending example above. Then add two more lines: what authority the system has, and what evidence would prove the outcome is done. If you cannot write the authority line, the process is not ready to delegate, whatever the model can do.

How much do you need? Architect: Use. Everyone else: Know.

Chapter 17

What You Should Do Next

Turning the understanding in this book into one appropriate action for your role

This book has argued that capability becomes useful only when the system around it is designed well. The same is true of the reader. Understanding these ideas matters only if it changes what you build, check or decide next, and the right next step depends on what you are responsible for. Each role below gets four lines: what to read at Know, Use and Master, using the scale from the front matter, and one first project small enough to finish. The chapter numbers come from the same map that produced every chapter's "How much do you need?" strip, so nothing here should surprise you. Chapter 17 itself is the action guide, so it is not included in the depth lists below.

Developer

Know. Parts I and IV, Chapter 5, Chapter 8, and Chapter 16. You need the vocabulary and the limits, not the mechanisms.

Use. Chapters 6, 7 and 9–12. Almost everything you build on a frontier model is a version of the Chapter 6 loop, and Chapters 9 and 10 are what make it survive contact with a real task.

Master. Not required unless you own production agent design.

First project. Build one tool-calling loop where the model can propose an action but application code independently validates the arguments and the permission before anything executes. Then make one of the tools side-effecting and add the idempotency check from Chapter 9.

Platform/DevOps

Know. Part I and Chapter 16.

Use. Chapters 5, 8, 13, 14 and 15. You will be asked to operate what others evaluate; these chapters tell you what the evaluation is measuring.

Master. Chapters 6, 7 and 9–12. These describe problems you already own under other names: state, recovery, idempotency, least privilege and audit.

First project. Take an existing agent workflow and add checkpoints after verification and idempotent actions before retry, so a mid-task failure can resume without repeating a side effect. Use the structured-state block from Chapter 9 as the schema.

Architect

Know. Chapters 4 and 8.

Use. Chapters 1, 2, 3, 6, 7, 9, 10, 11 and 16. Enough to place any proposal correctly and to know which chapter's failure modes it inherits.

Master. Chapters 5, 12, 13, 14 and 15. The five-layer lens, the authority boundary, verification, observability and the comparison method are the tools you will reuse in every design review.

First project. Draw one existing AI feature as the five layers and place every failure you have seen it produce at the layer that caused it. If a failure will not sit at one layer, that ambiguity is your first design finding.

Security

Know. Chapters 1–4, 10 and 16.

Use. Chapters 5, 8, 9, 11 and 15. Routing, physical authority, state, multi-agent boundaries and vendor comparison all change what you are protecting.

Master. Chapters 6, 7, 12, 13 and 14. The recurring theme is that authority must be enforced outside the model, and these chapters are where that is decided, attacked and observed.

First project. Write a consequence-based approval policy for one agent: list every action it can take, sort each into read-only or reversible, low-risk change, external communication, or high-consequence action, and record for each high-consequence

action whether the gate is a credential, application code, or a sentence in the prompt.

Leader

Know. Chapters 1–4 and 6–16. Read for concepts, limits and ownership rather than implementation detail; you need to know which promised outcomes depend on which layer, not how the layer works.

Use. Chapter 5, so that vendor claims can be read at the layer they belong to.

Master. Not required.

First project. For one AI initiative you sponsor, write down each outcome it promises, the layer that outcome depends on, and the person who owns that layer. Any outcome with no owner at its layer is a risk you are carrying without knowing it.

Student

Know. Chapters 1–5 and 7–16. If Part I is still unfamiliar, read *Navigating the AI World* before going deeper into Parts II–IV.

Use. Chapter 6, once the ideas in Part I are comfortable.

Master. Not required at this stage.

First project. Rebuild the Chapter 1 loop as a small script: load state, retrieve, assemble context, infer, act, validate, update state. A toy task is fine. The point is to feel where the model ends and the system begins.

Where the Two Books Meet

Navigating the AI World promised that a reader could arrive confused about a technology and leave knowing what it is, why it matters, whether they need it, how much to learn, and what to do next. This book has tried to keep that promise at a deeper level, for a harder subject, with the model names treated as evidence rather than as the point. The engineering questions will outlast the case studies. So will the habit this book has asked of you throughout: place the claim at its layer, ask what evidence supports it, and decide what depth your role actually needs. The practical promise remains the same: understand it, judge whether it matters, know how deeply to learn it, and do something useful next.

Modern AI becomes important when intelligence is no longer only available as an answer, but can be converted into sustained, verifiable and appropriately bounded work.

What Each Vendor Documents

This section is not intended as a complete specification sheet. Each row records something the vendor states in the primary sources used for this edition. Items not publicly documented are listed explicitly rather than inferred. Values are time-sensitive and were checked against primary vendor sources on the date given below each table; they should be rechecked before being relied on.

GPT-6 Astra (OpenAI)

What OpenAI documents	Value or description
Context window	1,050,000 tokens
Maximum output	128,000 tokens
Knowledge cutoff	30 April 2026
Input modalities	Text and image
Reasoning control	Five effort levels: low, medium, high, xhigh, max; changeable mid-conversation with cache preserved
Long-running mechanisms	Asynchronous tool calling, mid-turn steering, persisted reasoning, compaction, multi-agent orchestration, subagent delegation
Tools in the surrounding platform	Web search, file search, code interpreter, hosted shell, computer use, MCP, tool search
Input price	$10 per 1M tokens
Output price	$50 per 1M tokens
Cached input	$1 per 1M tokens; cache writes $12.50 per 1M
Long-context pricing	When input exceeds 272,000 tokens, the entire request uses long-context rates: $20 input, $2 cached input, $25 cache writes and $75 output per 1M tokens
Safety classification	Critical cybersecurity capability threshold
Safeguards described	Isolation, checkpoint encryption, full-trajectory monitoring, blocking alignment evaluations, robustness training, real-time misalignment monitoring
Monitorability finding	Lower chain-of-thought monitorability and higher chain-of-thought controllability than GPT-5.6 Sol in adversarial evaluations; stronger action-only monitorability in aggregate
Not documented in sources used	Parameter count, internal architecture, routing design, training-data composition, RL recipe, internal representation of persisted reasoning

Sources: OpenAI model documentation, model guidance, release post, safety overview and system card. Checked 9 September 2026; specifications and pricing re-verified against developers.openai.com on 9 September 2026.

Claude Fable 5.1 (Anthropic)

What Anthropic documents	Value or description
Relationship to Mythos 5.1	Same underlying model; Fable is the generally available product with additional safeguards, Mythos is available to vetted organizations through trusted-access programs
Context window	1,000,000 tokens (default and maximum), standard pricing across the whole window
Maximum output	128,000 tokens
Knowledge cutoff	June 2026
Input modalities	Text and images; diagrams, charts and tables inside files and PDFs
Reasoning control	Adaptive thinking, always on; `effort` parameter with five levels: low, medium, high, xhigh, max; default high; per-message effort change preserving cache (beta)
Long-running behavior	Plans work, uses tools, recovers when a step fails, operates across several applications, continues unattended for long periods; managed agents
Self-verification	Writes its own tests; uses vision to compare outputs against a design or goal
Input price	$10 per 1M tokens
Output price	$50 per 1M tokens
Cache writes	$12.50 per 1M (5-minute); $20 per 1M (1-hour)
Cache reads	$0.25 per 1M tokens, described as reducing agentic workload cost relative to Fable 5
Batch API	50% discount on input and output
Release	1 September 2026; retirement not before 1 September 2027
Safeguards described	Additional classifiers for sensitive cyber, biology and chemistry use; routing of some requests to Opus models without the sensitive-capability exposure
Research tooling	Claude Science workbench connecting models to scientific databases, notebooks, R, cluster terminals; auditable artifacts

What Anthropic documents	Value or description
Physical-device interface	Model Hardware Standard research preview (27 August 2026, with HHMI Janelia): microscopes, liquid handlers, robotic arms; parallel instrument operation; safety limits enforced at the driver level below the agent; some hardware-error recovery; open-sourcing planned after further safety work
Not documented in sources used	Parameter count, internal architecture, routing design inside the model, training-data composition, RL recipe, internal state mechanism for long-running work

Sources: Anthropic model documentation (platform.claude.com), product pages for Fable and Mythos, safeguards announcement, Claude Science announcement, Model Hardware Standard preview, system-cards index. Checked 9 September 2026; specifications, effort control and pricing re-verified against platform.claude.com on 9 September 2026.

Other figures cited in the book

Figure	Where it appears	Status
OpenAI research organization using about 3.1 agent-workdays per human workday by mid-August 2026; more than half of successful 4–8 hour agent tasks still needed human intervention	Chapter 16	OpenAI self-report, 6 September 2026; re-verified; compute-growth and correlation caveats stated in source
Astra completing OSWorld-style tasks faster than GPT-5.6 Sol with a higher score	Chapter 7	OpenAI release post, 3 September 2026
Astra solving harder mathematics in a single forward pass in one external evaluation	Chapter 14	OpenAI system card, September 2026

References and Source Notes

Repeated citations are consolidated below. Chapter numbers show where each source is relied on. All sources checked 9 September 2026 unless another date is stated.

OpenAI

1. **GPT-6 Astra model documentation.** Context window, maximum output, pricing, reasoning levels, supported tools. developers.openai.com/api/docs/models/gpt-6-astra. Chapters 1, 2, 3, 6, 15; back matter.
2. **GPT-6 Astra model guidance.** Reasoning configuration, asynchronous tool calling, mid-turn steering, persisted reasoning, compaction, multi-agent orchestration, subagent delegation, misalignment monitoring. developers.openai.com/api/docs/guides/latest-model. Chapters 1, 2, 5, 6, 9, 10, 11, 15.
3. **GPT-6 Astra: A new generation of intelligence.** Release post covering positioning across computer use, coding, cybersecurity, science and professional work, computer-use performance, prompt-injection robustness. openai.com/index/gpt-6-astra/. Published 3 September 2026. Chapters 3, 4, 7, 15, 16.
4. **Safety overview: GPT-6 Astra.** Critical cybersecurity classification, trajectory monitoring, robustness, alignment and prompt-injection findings, reduced chain-of-thought monitorability. openai.com/index/safety-overview-gpt-6-astra/. Published 3 September 2026. Chapters 5, 12, 14, 15.
5. **GPT-6 Astra System Card.** Deployment simulation, misalignment monitoring, monitoring scopes, chain-of-thought controllability, action-only monitoring, no-CoT capability, external UK AISI findings. deploymentsafety.openai.com/gpt-6-astra. Published 3 September 2026. Chapters 12, 13, 14; back matter.
6. **Research acceleration: The view inside OpenAI.** Agent-workday usage, concurrent agents, experiment activity, changing research bottlenecks. openai.com/index/research-acceleration-view-inside-openai/. Published 6 September 2026. Chapter 16.
7. **The Work Now Within Reach.** Falling cost of intelligence and work previously constrained by time, cost or access to expertise. openai.com/index/the-work-now-within-reach/. Published 8 September 2026. Chapter 16.

Anthropic

8. **Claude Fable 5.1 model documentation and effort guide.** Context window, maximum output, pricing and cache rates, adaptive thinking, five-level `effort` parameter, per-message effort change. platform.claude.com/docs/en/models/fable-5-1/overview and platform.claude.com/docs/en/build-with-claude/effort. Chapters 2, 15; back matter.
9. **Claude Fable 5.1 product page.** Positioning, long-running work, tool use, failure recovery, coding and vision, self-written tests, pricing and cache-read

cost, safeguard fallback behavior, research capability. anthropic.com/claude/fable. Announced 1 September 2026. Chapters 1, 2, 3, 4, 5, 9, 12, 13, 15; back matter.

10. **Claude Mythos 5.1 product page.** Statement that Fable 5.1 and Mythos 5.1 share the same underlying model and differ in safeguard and access treatment. anthropic.com/claude/mythos. Chapters 5, 12, 15, 16.
11. **Fable cyber safeguards and jailbreak framework.** Safety classifiers around Fable for sensitive use. anthropic.com/news/fable-safeguards-jailbreak-framework. Published 2 July 2026. Chapters 5, 12.
12. **Claude Science: AI workbench.** Connected research tools and auditable artifacts. anthropic.com/news/claude-science-ai-workbench. Chapter 4.
13. **Previewing the Model Hardware Standard.** AI agents operating laboratory and manufacturing instruments, parallel workflows, real-time parameter adjustment, hardware-error recovery. anthropic.com/news/model-hardware-standard-research-preview. Published 27 August 2026. Chapters 8, 16.
14. **Building Multi-Agent Systems That Actually Work.** Session description, Anthropic at Google Cloud Next 2026, identifying context isolation, parallel execution and specialization as recurring cases where multi-agent systems win. anthropic.com/events/anthropic-at-google-cloud-next-2026. Chapter 11.
15. **Model system cards index.** Listing of the September 2026 Claude Fable 5.1 and Mythos 5.1 system card. anthropic.com/system-cards. Chapter 15.

About the Author

Prasad Kukkala is a technology consultant, technical writer, and the creator of TechiesJournal. His work focuses on explaining cloud, software architecture, cybersecurity, and modern AI systems in language that working professionals can use without losing the engineering detail underneath.

He is also the author of *Navigating the AI World: Using and Building AI at Work*, the companion to this book, which covers AI foundations, learning direction, system design, and responsible use in professional environments.

Through TechiesJournal, he writes for readers who want to understand what a technology actually changes, what is worth learning, and how deeply they need to understand it.