Claude AI
The Complete Guide

Booklet

CLAUDE AI

The Complete Guide

Models · Artifacts · Agents & Computer Use · Claude Code

Cowork · Custom Skills · API · File Uploads · The 2026 Model Wave

Claude AI: The Complete Guide

This edition is a corrected and updated revision of the March 2026 first printing. It provides a comprehensive technical and practical reference to Anthropic's Claude AI platform, covering the complete model family, all major product features, the developer API, agentic capabilities, and the broader Claude ecosystem including Claude Code, Cowork, Custom Skills, and the Model Context Protocol.

All model, pricing, benchmark and feature figures reflect the state of the platform as of 7 August 2026 and are subject to change. Because the platform ships major updates almost weekly, specific values (pricing, context sizes, benchmark scores, model availability) are presented in clearly labelled tables with an explicit 2026 timestamp. For authoritative current information, always consult platform.claude.com/docs.

What changed in this revision. Pricing tables were corrected (notably the Opus flagship line); the Message Batches API request limit was corrected to 100,000; the Claude 4.6 release dates were corrected; the entire model family was re-baselined to June 2026 to include Claude Opus 4.7 and 4.8 and the Claude 5 generation (Fable 5, Mythos 5, Sonnet 5); runnable code examples were added; the table of contents and cross-references were rebuilt; and several community-sourced figures were marked as approximate.

What changed in this second revision (August 2026). Claude Opus 5 (24 July 2026) was added throughout as the new Opus-tier flagship, superseding Opus 4.8 at unchanged $5/$25 pricing; the June 2026 suspension and restoration of Fable 5 and Mythos 5 access under a US export-control directive is now documented; Claude Opus 4.1 is marked fully retired (5 August 2026); Claude Sonnet 5's adaptive-thinking default and new tokenizer are noted; and the MCP governance note was corrected to name the Agentic AI Foundation.

Table of Contents

Page numbers refer to this revised edition.

PREFACE

How to Use This Book

In roughly four years, Claude has grown from a safety-focused research model into one of the most capable and widely used AI platforms available. What began as a conversational assistant now spans a model family from budget to frontier, an Artifacts environment for building interactive applications, computer-use capabilities for autonomous desktop interaction, a terminal-based coding agent, a desktop collaboration product, a skills framework for reusable workflows, a mature developer API, and a file-handling layer that processes PDFs, spreadsheets, and images natively.

This book maps the full Claude landscape as of **June 2026**. It is written for two audiences at once: readers new to Claude who want a complete picture before writing their first prompt or API call, and experienced users who need depth on a specific capability. Each chapter is self-contained — read cover to cover, or jump to the chapter most relevant to your work.

Writing about a platform that ships major updates nearly every week poses an obvious challenge. The approach here is to prioritise durable understanding over ephemeral detail. Chapters on the model family, the API, and agentic concepts are written to remain useful even as individual figures change. Where specific values matter, they are timestamped June 2026.

Conventions Used in This Book

Code examples appear in labelled monospace blocks. API calls are shown as `curl` for language-neutrality, with **Python SDK** equivalents where they add clarity. Teal callout boxes highlight key concepts, safety considerations, or important caveats. Model API strings are shown in their exact documented form for copy-paste correctness — for example, `claude-opus-4-8` or `claude-sonnet-4-5-20250929`.

> **A note on currency**
>
> This is a fast-moving target. Between this book's March 2026 first printing and this August 2026 revision, Anthropic shipped six further model releases (including a new frontier tier and a mid-cycle Opus refresh), introduced a new generation naming scheme, and briefly had two of those models pulled offline worldwide by a government export-control order. Treat every dated figure as a snapshot, and verify anything cost- or capability-critical against platform.claude.com/docs before relying on it.

CHAPTER ONE

The Claude Model Family

From Claude 3 to the Claude 5 generation — architecture, benchmarks, and choosing the right model

---

Anthropic has historically organised its Claude models along two axes: a generational number reflecting training leaps, and a tier name — Haiku, Sonnet, or Opus — positioning each model on the speed-versus-intelligence spectrum. As of the Claude 5 generation (June 2026) this scheme has partially shifted: the frontier tier is now delivered under distinct model names (Fable 5, Mythos 5) rather than the Opus label, while the Sonnet and Haiku names persist for the balanced and budget tiers. Understanding both the tier concept and this naming evolution is the foundation of working effectively with Claude.

The history of the Claude family is, in many ways, the history of capability compression. Features that required the premium Opus tier in 2024 became Sonnet-grade in 2025 and are approaching Haiku-grade today. The cost per unit of intelligence has fallen faster than almost any observer predicted. This trend shapes every architectural and pricing decision in this chapter.

1.1 The Tiers

Haiku — Speed and Economy

Haiku models are optimised for low latency and high throughput: customer-support chatbots, real-time classification, structured extraction from short documents, FAQ retrieval, and rapid slot-filling. The performance bar for Haiku has risen steadily; **Claude Haiku 4.5 (October 2025) remains the active budget tier as of June 2026**, with a 200,000-token context window, 64K max output, and smart model switching that can escalate a hard request to a larger model rather than degrade silently.

Sonnet — Balanced Intelligence

Sonnet occupies the centre of the ladder and is the default for most new integrations. **Claude Sonnet 5 (30 June 2026)** is the current Sonnet, carrying the 1M-token context window and 128K output into the mid tier at $3/$15 per million tokens (with introductory pricing of $2/$10 through 31 August 2026). Its predecessor, Sonnet 4.6 (February 2026), was the model that first reached Opus-level coding at Sonnet pricing — a shift that made 'good for most things' into 'good for nearly everything.'

Opus and the Frontier — Fable 5 and Mythos 5

The Opus tier historically represented Anthropic's capability frontier: deepest reasoning, longest sustained reasoning chains, and the most reliable performance on ambiguous, high-stakes problems. Opus 4.5 (November 2025), 4.6 (February 2026), 4.7 (April 2026) and 4.8 (May 2026) advanced that frontier while holding a $5/$25 price point. As of **9 June 2026**, the frontier tier is delivered under new names: **Claude Fable 5** is the most capable widely available model, and **Claude Mythos 5** shares its specifications but is restricted to defensive-cybersecurity use under Project Glasswing. Both feature always-on adaptive thinking and a 1M-token context window.

> **Active Flagship Models — August 2026**

As of 7 August 2026 the most capable widely released model is **Claude Fable 5**, with **Claude Opus 5** (launched 24 July 2026, superseding Opus 4.8 at the same $5/$25 price) as the workhorse Opus-tier model, and **Claude Sonnet 5** also current, and **Claude Haiku 4.5** the active budget tier. **Claude Mythos 5** matches Fable 5 but remains invitation-only under Project Glasswing. Claude Opus 4.8 is superseded but still available; Claude Opus 4.1 was fully retired from the API on 5 August 2026. Fable 5 and Mythos 5 were briefly suspended worldwide from 12–30 June 2026 under a US export-control directive before access was restored — see the note at the end of this chapter.

1.2 Generational History

Six major Claude generations have shipped since 2023. Each created genuinely new use cases rather than merely improving existing ones.

- **Claude 1–2 (2023):** established the core character — strong instruction following, reliable refusals, extended context (Claude 2.1 reached 200K tokens), and honest calibration. Text-only; no longer available.
- **Claude 3 (March 2024):** first multimodal generation and the origin of the Haiku/Sonnet/Opus three-tier framework. Claude 3 Opus was retired in early 2026.
- **Claude 3.5 (June–November 2024):** dramatic coding gains; launched Artifacts, and — in the October 2024 3.5 Sonnet (v2) — the first commercially available Computer Use.
- **Claude 3.7 (February 2025):** first hybrid-reasoning model, letting users choose fast responses or extended step-by-step thinking.
- **Claude 4 (May 2025):** professional-grade coding and the mainstream launch of Claude Code. Opus 4.1 followed in August 2025.
- **Claude 4.5–4.8 (Sept 2025 – May 2026):** the efficiency era — Sonnet 4.5, Haiku 4.5, Opus 4.5 (a ~67% Opus price cut to $5/$25), the 4.6 dual release, and the reasoning-focused Opus 4.7 and 4.8.
- **Claude 5 (June 2026):** Fable 5 and Mythos 5 introduce always-on adaptive thinking and a named-model frontier tier, alongside Sonnet 5. Fable 5 and Mythos 5 were briefly suspended worldwide (12–30 June 2026) under a US export-control directive before access was restored on 1 July 2026.
- **Claude Opus 5 (24 July 2026):** a step-change refresh of the Opus tier at unchanged $5/$25 pricing, bringing near-Fable-5 agentic coding and computer-use performance, a full 1M-token context window as both default and maximum, and a new xhigh/max reasoning-effort ladder. It replaces Opus 4.8 as the default Opus-tier model.

Correction — the Claude 4.6 release

The first printing stated that Opus 4.6 and Sonnet 4.6 'launched simultaneously on February 15, 2026.' That is not correct. The two models were separate releases: Anthropic's Claude Sonnet 4.6 system card is dated **17 February 2026**, and Claude Opus 4.6 shipped earlier that month. They were not simultaneous, and neither landed on the 15th.

New in this revision — the Fable 5 / Mythos 5 export-control suspension

Three days after their 9 June 2026 launch, the US Department of Commerce directed Anthropic to suspend all access to Fable 5 and Mythos 5 by any foreign national, anywhere, including Anthropic's own foreign-national employees. Because nationality could not be verified in real time, Anthropic disabled both models worldwide for every customer on 12 June 2026; Opus 4.8 and all other models were unaffected. The stated concern was a reported method of bypassing Fable 5's cybersecurity safeguards.

> The Commerce Department lifted the directive on 30 June 2026, and Anthropic restored Fable 5 globally on 1 July 2026 with hardened cyber safeguards, and restored Mythos 5 for approved US and international Project Glasswing partners. See **anthropic.com/news/fable-mythos-access** for Anthropic's statement.

1.3 Complete Model Reference Table

Active and recently superseded Claude models as of 7 August 2026. Context and output are in tokens; pricing is USD per million tokens.

Model	Released	Context	Max out	Status (Aug 2026)
Claude Haiku 4.5	Oct 2025	200K	64K	Active — budget tier
Claude Opus 4.8	28 May 2026	1M	128K	Superseded (available)
Claude Opus 5	24 Jul 2026	1M	128K	Active — current Opus tier
Claude Sonnet 5	30 Jun 2026	1M	128K	Active — current Sonnet
Claude Fable 5	9 Jun 2026	1M	128K	Current flagship
Claude Mythos 5	9 Jun 2026	1M	128K	Restricted (Glasswing)
Claude Opus 4.7	16 Apr 2026	1M	128K	Superseded (available)
Claude Opus 4.6	Feb 2026	1M	128K	Superseded
Claude Sonnet 4.6	17 Feb 2026	200K/1M	64K	Superseded
Claude Opus 4.5	Nov 2025	200K	64K	Superseded
Claude Sonnet 4.5	Sep 2025	200K/1M	64K	Superseded

1.4 Pricing and Token Economics

A token is approximately 0.75 English words — 1,000 tokens ≈ 750 words. The figures below reflect published rates as of August 2026. Prompt caching, the Batch API, and enterprise agreements can substantially reduce effective cost.

Model	Input $/1M	Output $/1M	Batch in	Batch out	Notes
Claude Haiku 4.5	$1.00	$5.00	$0.50	$2.50	Budget tier
Claude Sonnet 5	$3.00	$15.00	$1.50	$7.50	Intro $2/$10 to 31 Aug 2026
Claude Opus 4.8	$5.00	$25.00	$2.50	$12.50	Superseded by Opus 5
Claude Opus 5	$5.00	$25.00	$2.50	$12.50	Current Opus tier; 300K batch out (beta)
Claude Fable 5	$10.00	$50.00	$5.00	$25.00	Most capable, 1M context

> **Correction — Opus flagship pricing**
>
> The first printing listed the Opus 4.6 flagship at $15/$75 per million tokens and gave an internally inconsistent $15/$75 figure for Opus 4.5. Both were wrong. From Opus 4.5 onward, the Opus flagship

Model	Input $/1M	Output $/1M	Batch in	Batch out	Notes

line has been **$5/$25** per million tokens, and Opus 4.8 and its successor Opus 5 (24 July 2026) both remain at $5/$25 today. The $10/$50 figure applies only to the Fable 5 / Mythos 5 tier.

Cost Optimisation: Prompt Caching

Prompt caching stores reused context prefixes — system prompts, large reference documents, few-shot libraries — server-side after the first use. Cached reads cost ~90% less than standard input tokens (cache writes carry a ~25% markup, so the break-even is roughly two reuses). For applications where the same long system prompt appears in every request, caching typically cuts total API cost by 30–60%.

Several patterns routinely inflate token usage: verbose system prompts (compress them; move stable material to the cache), unconditional document inclusion (retrieve only relevant sections), unbounded conversation history (roll up older turns), and unconstrained output requests (specify a length target).

1.5 Benchmark Performance

The most developer-relevant benchmarks are SWE-bench Verified (autonomous software engineering on real GitHub issues), MMLU / GPQA Diamond (knowledge and graduate-level science), and computer-use task suites. Scores below are approximate and drawn from vendor and community reporting as of August 2026; treat them as directional. Note that Anthropic's own Opus 5 benchmark disclosures lean on a newer internal suite (Frontier-Bench, CursorBench, ARC-AGI-3, OSWorld 2.0) rather than headline SWE-bench/GPQA figures, so the Opus 5 row below is necessarily thinner than the others until directly comparable third-party numbers are published.

Model	SWE-bench	GPQA Diamond	Computer Use
Claude Sonnet 4.6	~80.9%	~80%	~94%
Claude Opus 4.6	~81%	~82%	~94%
Claude Opus 4.8	higher (see docs)	—	~94%+
Claude Opus 5	see docs	—	~70% (OSWorld 2.0)
Claude Fable 5	frontier (see docs)	—	—

Reading benchmarks critically

Published scores reflect performance under specific controlled conditions. Real-world results depend heavily on how well your prompts, context structure, and tool definitions match the model's training distribution. A model that scores 94% on a computer-use benchmark may behave differently on your particular automation. Where this book cites a precise figure for the newest models, confirm it against the live model card before quoting it.

1.6 Choosing the Right Model

Model selection should be deliberate. The most common mistake is starting with the most expensive model for everything and never evaluating cheaper alternatives. Test Haiku first, then Sonnet, and escalate to the frontier tier only when you have identified a specific, reproducible failure mode that the smaller model cannot handle.

Start with Haiku 4.5 when

- Request volume is high and latency is critical.

- Tasks are well-defined and short — classification, slot-filling, brief Q&A.
- Budget is a primary constraint and you can measure success systematically.

Upgrade to Sonnet 5 when

- Tasks involve multi-step reasoning or nuanced instruction following.
- Output quality matters and users evaluate responses directly.
- Coding requires understanding code across multiple files.

Escalate to Opus 5 or Fable 5 when

- The codebase or document set requires the 1M-token context window.
- Tasks fail on Sonnet and the failure mode is reasoning depth, not prompt quality.
- You are building autonomous agents where reliability over many steps is critical, or doing deep research synthesis that needs the highest available intelligence. Opus 5 (24 July 2026) is now the default choice here: it lands close to Fable 5 on Anthropic's own coding and agentic benchmarks at half the price, so reserve Fable 5 for the small residue of tasks where Opus 5 measurably falls short.

CHAPTER TWO

Artifacts: Building Interactive Experiences

From rendered code previews to AI-powered micro-applications with persistent storage

Claude Artifacts turn the Claude.ai interface from a text conversation into a live, interactive workspace. When Claude generates code, a web page, a diagram, or a document inside an Artifact, you see the rendered output in a dedicated side panel — not just the source. You can interact with it, inspect its source, make targeted changes, share it, and build on it through continued conversation. Artifacts have been used at very large scale since launch (Anthropic has cited figures on the order of hundreds of millions created; treat exact counts as approximate).

2.1 What Artifacts Are and Supported Types

An Artifact is a self-contained piece of generated content that Claude renders visually. The panel updates in real time and lets you toggle between rendered and source views.

Type	What Claude generates	Key use cases
HTML / CSS / JS	Interactive web pages	Dashboards, landing pages, tools, games
React components	Reusable component trees	UI prototypes, component libraries
SVG	Vector graphics	Diagrams, icons, flowcharts
Markdown	Rich-text documents	Reports, READMEs, notes
Mermaid	Flowcharts, sequence diagrams	Architecture and process flows
Code files	Any language	Scripts, modules, configuration

2.2 Enabling, Editing, and AI-Powered Artifacts

Artifacts are available on paid Claude.ai plans and enabled by default for requests that produce renderable content; manage them under Settings → Capabilities. The editor uses three modes: **Create** (first version), **Update** (targeted string replacement for small localised changes — much faster than a rewrite), and **Rewrite** (full regeneration for structural changes). Structuring artifacts with clean component boundaries and single-source-of-truth constants maximises the efficiency of Update mode.

AI-powered artifacts can call the Claude API directly from inside the application, turning static outputs into dynamic apps. Enable via Settings → Capabilities → 'Create AI-powered Artifacts' and instruct Claude to 'use Claude' in the application; no API key is embedded in the artifact code.

> **Credit consumption in shared AI-powered artifacts**
>
> AI-powered artifacts consume Claude API credits for every interaction inside the artifact. If you publish and share one, interactions by external users draw from your account's balance. Monitor usage for public artifacts and consider rate limiting if you expect high traffic.

2.3 Persistent Storage, Publishing, and Remixing

Persistent storage (shipped October 2025, ahead of the 4.6 generation) lets artifacts store and retrieve data across sessions — up to 20 MB per artifact, text only — turning them into stateful applications. Storage is either personal (visible only to the running user) or shared (visible to all users of a published artifact), enabling journals and progress trackers on one hand and leaderboards and team dashboards on the other. Publishing creates a public URL that anyone can visit and interact with, and account holders can remix a published artifact into their own copy — creating a network effect for useful templates.

CHAPTER THREE

Agents and Computer Use

The agentic loop, tool use, desktop control, safety engineering, and Agent Teams

---

Claude's agentic capabilities are the most consequential architectural shift in the platform's history. Rather than responding once to a prompt, an agent takes sequences of actions — calling tools, observing results, making decisions, and iterating toward a goal. Computer Use extends this to direct desktop and browser interaction.

3.1 The Agentic Loop

All agentic Claude behaviour — via the API, in Claude Code, or in Cowork — is built on the same pattern. The user provides a goal; Claude assesses available tools and current state and selects an action; the application executes the tool and returns the result as a `tool_result` block; Claude reasons over the result and either completes the task or selects the next tool call. This repeats until completion or an iteration limit.

> **Safety requirement: always set a maximum iteration limit**
> Every production agentic implementation must cap iterations. Without a limit, an agent encountering an unexpected environment state can loop indefinitely, consuming tokens and incurring unbounded cost. A sensible default is a small ceiling (for example, 10) that you tune to the task's expected complexity; log iteration counts for monitoring.

The loop creates engineering responsibilities absent from single-turn apps: state management across long tool-result histories, error handling for partial completion (a loop that fails on step 7 of 10 may have taken irreversible actions), and idempotency (can earlier steps re-execute safely after a restart?). Address these before deploying any agentic integration.

3.2 Tool Use in the API

Claude's tool-use system lets you define arbitrary tools with JSON Schema inputs; Claude learns to call them from the description alone — no fine-tuning. Every tool needs a **name**, a **description** (the single most important field — write it as if explaining the tool to a colleague), and an **input_schema**. When Claude calls a tool, the response has `stop_reason: "tool_use"` and a `tool_use` block with a `tool_use_id`; your app executes it and appends a `tool_result` block referencing that id. Claude can request multiple tool calls in one turn; results may be returned in any order as long as each `tool_use_id` matches.

A minimal end-to-end tool-use loop in Python:

```
import anthropic
client = anthropic.Anthropic()

tools = [{
    "name": "get_weather",
    "description": "Get the current weather for a city.",
    "input_schema": {
        "type": "object",
        "properties": {"city": {"type": "string"}},
        "required": ["city"],
    },
}]
```

```
messages = [{"role": "user", "content": "What's the weather in London?"}]

for _ in range(10):  # hard iteration cap
    resp = client.messages.create(
        model="claude-sonnet-5", max_tokens=1024,
        tools=tools, messages=messages,
    )
    messages.append({"role": "assistant", "content": resp.content})
    if resp.stop_reason != "tool_use":
        break
    results = []
    for block in resp.content:
        if block.type == "tool_use":
            output = run_tool(block.name, block.input)  # your code
            results.append({
                "type": "tool_result",
                "tool_use_id": block.id,
                "content": str(output),
            })
    messages.append({"role": "user", "content": results})
```

3.3 Computer Use

Computer Use is a specialised set of API tools giving Claude the ability to interact with desktop environments. It was introduced with Claude 3.5 Sonnet (v2) in October 2024 and has been iterated every generation since. The current tools:

Tool	API type string	Capabilities
Computer	computer_20250124 (or newer)	Screenshot, mouse move, click/drag, keyboard, scroll
Text editor	text_editor_20250728 (or newer)	View files, create/overwrite, targeted string edits
Bash	bash_20250124 (or newer)	Execute shell commands, run scripts, query state

Computer Use requires the appropriate `anthropic-beta` header (for example `computer-use-2025-11-24` for the Opus 4.5-era tools); confirm the exact tool version and beta string for your target model in the docs, as they advance with each generation. The computer tool is schema-less — its input schema is built into the model and cannot be customised.

3.4 Building Safe Computer Use Systems

Computer Use introduces risks absent from text-only systems: when Claude observes a screen and can act, the environment becomes an attack surface for prompt injection. A security checklist:

- **Isolated container or VM:** never run Computer Use against a personal machine or production without strict sandboxing; use a fresh-state VM per session.
- **Minimal privilege:** grant only the filesystem, network, and application access the task needs; avoid root and credential stores.
- **Network restriction:** limit outbound connections to task-required domains.
- **Human approval gates:** require confirmation before any write, send, purchase, or delete.
- **Prompt-injection hardening:** instruct Claude to treat observed content as data, not instructions; treat pages, emails, and documents as adversarial.
- **Audit logging and session time limits:** capture every screenshot, tool call, and result; cap session duration.

3.5 Agent Teams

Agent Teams (general availability February 2026) let multiple independent Claude Code instances communicate directly with peer agents rather than routing all coordination through a single orchestrator. Where the subagent model bottlenecks handoffs through one master context, Agent Teams let a backend agent publish its interface directly to frontend and test agents, who proceed in parallel. Use subagents for parallel independent tasks; use Agent Teams for complex features with interdependent specialists.

CHAPTER FOUR

Claude Code

The terminal-first autonomous coding agent — setup, CLAUDE.md, Plan Mode, MCP, /loop, and more

---

Claude Code launched in May 2025 as a command-line tool that gives Claude direct, persistent access to your development environment. By 2026 it is a full autonomous coding platform: running in your terminal, integrated with VS Code and JetBrains, controllable from mobile, capable of voice interaction, able to schedule recurring tasks, and supporting parallel multi-agent development. The fundamental difference from the chat interface is context: Claude Code reads your actual project files, maintains persistent notes, executes real commands, and iterates toward working solutions.

4.1 Core Capabilities

- **Codebase understanding:** on launch, Claude Code reads project structure, dependency manifests, config, and Git history, building a working model that persists via CLAUDE.md.
- **Multi-file atomic changes:** coordinated edits across many files, proposed and approved as a unit.
- **Test-driven iteration:** run the suite, read failures, modify, re-run until green.
- **Git workflow integration:** read history, manage branches, create descriptive commits and pull requests, check CI.

4.2 Installation and Prerequisites

Claude Code requires Node.js 18 or higher. An active Claude Pro, Max, Team, or Enterprise subscription grants usage; alternatively, an Anthropic API key enables pay-as-you-go.

```
# Install
npm install -g @anthropic-ai/claude-code

# Launch (from any project directory)
claude

# Update
npm update -g @anthropic-ai/claude-code   # or: claude update
```

4.3 CLAUDE.md — Project Memory

CLAUDE.md is the most important configuration mechanism in Claude Code: a plain Markdown file, committed to your repository, that Claude reads at the start of every session. Put in it a project overview, repository structure, coding conventions, exact test commands, deployment procedures, known issues (things that should not be 'fixed' without discussion), and team ownership. Claude Code appends to it when it learns something worth remembering; review it periodically to keep the accumulated notes accurate.

4.4 Plan Mode

Plan Mode is Claude Code's safety gate for significant operations: it presents a complete written plan of files it intends to modify and commands it will run before taking a step. Activate it with `/plan`, or set it as the default in `~/.claude/settings.json`. Beyond safety, reading the plan often reveals misunderstandings —

for example, that Claude interpreted 'refactor the auth module' as a full rewrite — while correction is still cheap.

4.5 MCP Integration and Plugins

Model Context Protocol integration gives Claude Code live access to external systems — databases, APIs, documentation, version control, project management — without changing Claude Code itself. Add servers with `claude mcp add`. **Plugins** (introduced February 2026) bundle MCP servers with related Skills and tools into one-click installation packages. Common servers include Context7 (live library docs), GitHub, Supabase, Linear, Notion, and Sequential Thinking.

4.6 Hooks, /loop, and Background Agents

Hooks are scripts in `~/.claude/hooks/` that intercept Claude Code behaviour at defined lifecycle events (pre-commit validation, audit logging, security scanning, notifications, cost monitoring). The **/loop** command (2026) runs recurring tasks on a schedule — `/loop 15m check PRs for new comments or CI failures` — turning Claude Code into a continuous automation runtime for the duration of the session. **Background agents** (GA February 2026) run parallel sub-tasks in isolated git worktrees, preventing conflicts when multiple agents modify code simultaneously.

4.7 Voice, Remote Control, Security Scanning, and IDEs

- **Voice Mode:** hold-to-talk input; processed exactly as typed text; ~20 languages.
- **Remote Control:** a web interface to supervise and approve a running Claude Code session from a phone or browser.
- **Security scanning (GA Feb 2026):** automatic vulnerability detection on generated code with patch suggestions.
- **IDE integration:** the VS Code extension and JetBrains plugin provide inline diffs, automatic context sharing, checkpoints, and one-command revert.

CHAPTER FIVE

Claude Cowork

Supervised delegation for knowledge workers — folder access, approvals, connectors, and task management

Claude Cowork is Anthropic's desktop collaboration product for knowledge workers. Where Claude Code lives in the terminal and operates on code, Cowork is designed for anyone who wants to delegate complex, multi-step tasks without a command line or API code. Its design philosophy is supervised delegation: Claude handles execution while you stay in control of the decisions that matter. Cowork tasks can run in the cloud or on your own computer via the desktop app.

5.1 The Execution Ownership Model

Understanding who owns the execution environment is the key to placing Cowork in the ecosystem, because it determines what you can configure and where accountability sits.

Product	Execution owner	Configuration depth
API + Computer Use	You (VM/container)	Full — sandbox, tools, approval logic
Claude Code	You (local/cloud)	High — MCP, hooks, CLAUDE.md, IDE
Cowork	Anthropic desktop	Medium — connectors, folder grants, approvals
Claude.ai chat	Anthropic cloud	Low — system prompt, memory, artifacts toggle

5.2 Core Capabilities

- **Folder access:** grant read and/or write access to specific local folders or cloud storage; for writes, Claude proposes changes and requires approval.
- **Browser tasks:** a sandboxed browser for research, form-filling, and multi-step online workflows, isolated from your personal browsing.
- **Native connectors:** direct integrations with Google Drive/Docs/Sheets/Slides, Notion, GitHub, Slack, Confluence, Jira, Linear, and more — cleaner than browser automation for supported services.
- **Approval gates:** before any consequential action, Cowork shows the proposed action for explicit confirmation; the threshold is configurable per action type.

5.3 Practical Workflows

Representative delegations: competitive research and synthesis into a Drive document; reviewing 200 vendor contracts for specific clauses into a results spreadsheet (the approval gate fires once for the bulk write, not 200 times); and preparing a board pack by pulling metrics from a dashboard and assembling a deck from a template. In each case, total human involvement is reviewing and approving a small number of outputs.

5.4 Security and Data Handling

When Cowork uses Computer Use it applies the same sandboxing principles recommended for API computer use: isolated browser sessions, web content treated as adversarial, and file access scoped to explicitly

granted folders. For enterprise deployments, Cowork supports data-residency configuration, audit logging, and administrative controls over which connectors members may authorise; the approval-gate system provides a natural audit trail.

CHAPTER SIX

Custom Skills

Packaging domain knowledge and workflows into reusable, shareable instruction modules

Claude Skills are reusable instruction modules that encode domain expertise, team workflows, and business logic in a format Claude can load and apply on demand. Introduced in October 2025, they turn one-time prompts into reliable, versioned, shareable procedures — a Standard Operating Procedure that the AI actually reads and executes.

6.1 What a Skill Is

A Skill is a folder containing at minimum a **SKILL.md** file, optionally accompanied by supporting scripts, templates, reference files, and configuration. The folder is named with a slug (e.g. `pdf-processor`). Claude reads the skill's name and description at startup via progressive disclosure and loads the full SKILL.md only when it determines the skill applies.

6.2 Anatomy of a SKILL.md

Every SKILL.md begins with YAML frontmatter, followed by a Markdown body. Frontmatter fields:

Field	Required?	Description
name	Yes	Unique slug identifier
description	Yes	What the skill does and when to use it — read at startup
version	No	Semantic version for tracking changes
author	No	Creator or maintaining team
dependencies	No	Other skills or MCP servers relied on

```
---
name: pdf-processor
description: Load when the user needs to extract text, tables, or
  images from PDF files, merge or split PDFs, or create new PDFs.
version: 1.2.0
---

# PDF Processing
## When to use
Trigger on: "extract from PDF", "merge PDFs", "split PDF", ...
## Procedure
1. Identify the operation ...
```

The **description** is the most important field — it is the only text Claude reads during the initial skill scan. Write it as a complete sentence starting with 'Load when' or 'Use when.' A vague description causes the skill to load when it should not; an overly specific one causes it to miss cases it should catch.

6.3 Progressive Disclosure and the Three-Layer Stack

Progressive disclosure pre-loads only each skill's name and description (a few hundred tokens), loading the full SKILL.md only when relevant — so dozens of skills can be installed without inflating context on unrelated tasks. This means skills should be granular, not monolithic. Skills, MCP servers, and Agents are three layers of

the same capability stack: Skills provide procedural knowledge (SKILL.md files), MCP provides live external access (server processes via `claude mcp add`), and Agents provide task delegation and parallelisation (`.claude/agents/` files). The most powerful workflows combine all three.

6.4 Writing, Versioning, and Distributing Skills

- **Write for Claude, not humans:** prefer explicit conditional logic and complete decision trees over narrative.
- **Include concrete examples:** one worked input/output example is worth pages of abstract specification.
- **Make triggers explicit:** list the task types and keywords that should load the skill.
- **Version and maintain:** semantic-version the frontmatter, test changes against representative tasks, track in Git with a changelog.

Because skills are plain-text folders, distribution through Git is natural: committing a `skills/` directory alongside a codebase gives every developer the same workflow knowledge. Community directories and the Anthropic plugin marketplace share skills more broadly, and the skill-creator meta-skill guides you through authoring new SKILL.md files interactively.

CHAPTER SEVEN

The Anthropic API

Messages, Batch, Files, streaming, tool use, and thinking — the developer reference

The Anthropic API is a RESTful interface accepting JSON requests and returning JSON responses, with server-sent events for streaming. Understanding its endpoint structure, request format, parameters, and production patterns is essential for building Claude-powered applications that work correctly, perform well, and cost predictably.

7.1 Authentication and Versioning

All requests require an API key in the `x-api-key` header, created and managed at console.anthropic.com. The `anthropic-version` header is also required; as of June 2026 the current stable version remains `2023-06-01`. Base URL: `https://api.anthropic.com/v1`.

```
curl https://api.anthropic.com/v1/messages \
  -H "content-type: application/json" \
  -H "x-api-key: $ANTHROPIC_API_KEY" \
  -H "anthropic-version: 2023-06-01" \
  -d '{
    "model": "claude-sonnet-5",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Hello, Claude"}]
  }'
```

> **Security: API key management**
> Never embed API keys in client-side code, mobile apps, or public repositories. Use environment variables and secrets management. Note that the key header must be passed so the shell expands the variable — use double quotes (`"x-api-key: $ANTHROPIC_API_KEY"`); single quotes send the literal string and produce a 401.

7.2 The Messages API

`POST /v1/messages` is the primary endpoint. It is stateless: the client passes the full conversation history each request. Roles alternate strictly between `user` and `assistant`; the conversation starts with a user message. Key parameters:

Parameter	Type	Notes
model	string	Model API string, e.g. claude-sonnet-5
max_tokens	integer	Max output tokens; up to 128K on current flagships
messages	array	Alternating {role, content} objects
system	string	System prompt: persona, context, constraints
tools / tool_choice	array/obj	Tool definitions and selection control
stream	boolean	true for server-sent events
thinking / effort	obj/string	Extended/adaptive thinking control (see 7.4)

7.3 Streaming Responses

Setting `stream: true` returns server-sent events instead of one JSON object, so clients receive tokens as they generate. Event types, in order: `message_start`, then per content block `content_block_start` → `content_block_delta` → `content_block_stop`, then `message_delta` (stop reason and final usage) and `message_stop`. In Python the SDK exposes this as a context manager:

```
with client.messages.stream(
    model="claude-sonnet-5", max_tokens=1024,
    messages=[{"role": "user", "content": "Write a haiku."}],
) as stream:
    for text in stream.text_stream:
        print(text, end="", flush=True)
    final = stream.get_final_message()
```

7.4 Extended and Adaptive Thinking

Extended thinking lets Claude reason step-by-step in an internal chain before the final answer. On Claude 3.7 Sonnet and the 4.x models, enable it with `thinking: {"type": "enabled", "budget_tokens": N}` — starting budgets of ~8,000 (moderate), ~16,000 (hard reasoning), and 32,000+ (research synthesis or complex code).

The **Claude 5 generation (Fable 5, Mythos 5) uses always-on adaptive thinking**: thinking is applied automatically, `thinking: {"type": "disabled"}` is not supported, the raw chain of thought is never returned, and you control reasoning depth with the **`effort`** parameter (with `thinking.display` set to `summarized` or `omitted`). When migrating to these models, remove any code that disables thinking and switch depth control to `effort`.

This adaptive-thinking-by-default behaviour has since spread beyond the Fable/Mythos tier. **Claude Sonnet 5** (30 June 2026) also runs adaptive thinking on by default: manual extended-thinking budgets are removed and return a 400 error, and setting `temperature`, `top_p` or `top_k` to a non-default value now also returns a 400 error. Sonnet 5 additionally uses a new tokenizer that produces roughly 30% more tokens than Sonnet 4.6 for the same text — budget accordingly when migrating. **Claude Opus 5** (24 July 2026) keeps thinking on by default too, but is less strict: disabling it is still permitted at effort `high` or below; only the new `xhigh` and `max` levels reject a disable request with a 400 error.

7.5 Tool Use, the Batch API, and the Files API

Tool use (function calling) is covered in §3.2. Two further endpoints matter for scale:

- **Batch API (`POST /v1/messages/batches`):** asynchronous processing at a **50% discount**. A single batch holds **up to 100,000 requests or 256 MB**, whichever comes first, with results typically within an hour (24-hour maximum) and retained for 29 days. Ideal for classification at scale, evaluation, and bulk generation. Max output per request on the Batch API has also been raised to **300K tokens** (beta) for Claude Opus 5 and several other current models — useful for long-form batch generation.
- **Files API (`POST /v1/files`):** upload files up to **500 MB each**, reused across requests by `file_id`, avoiding repeated uploads. Reference a file in a message via a content block with `source.type: "file"` and `source.file_id`. Requires the `anthropic-beta: files-api-2025-04-14` header.

> **Correction — Batch API request limit**
>
> The first printing stated a batch holds 'up to 10,000 requests.' The correct limit is **100,000 requests** (or 256 MB, whichever is reached first). The 256 MB size cap, 50% discount, and 24-hour window were stated correctly.

7.6 Prompt Caching

Prompt caching stores reused context prefixes server-side. Enable it by adding `"cache_control": {"type": "ephemeral"}` to a content block; everything up to and including that block is cached after the first request. Cache reads cost ~10% of standard input tokens (a 90% discount); cache writes carry a ~25% markup, so the break-even is roughly two uses of the same prefix. Default cache lifetime is 5 minutes, extendable to 1 hour.

7.7 Rate Limits and Production Patterns

Rate limits apply per account on requests-per-minute and tokens-per-minute, varying by plan and model. On exceeding them the API returns HTTP 429 with a `Retry-After` header. Implement exponential backoff with jitter:

```
import time, random, anthropic
client = anthropic.Anthropic()

def call_with_retry(**kwargs):
    delay = 1.0
    for attempt in range(6):
        try:
            return client.messages.create(**kwargs)
        except anthropic.RateLimitError:
            time.sleep(min(delay, 64) + random.uniform(0, 0.5))
            delay *= 2
    raise RuntimeError("exhausted retries")  # send to dead-letter queue
```

CHAPTER EIGHT

File Uploads and Document Processing

Formats, limits, PDF vision, image analysis, Projects, and engineering patterns

Claude's ability to read, analyse, and reason about uploaded files turns it from a text generator into a document-intelligence platform. File processing is implemented at the multimodal level: a PDF page is processed as both its text layer and a rasterised image, so Claude can read multi-column tables, interpret charts, and understand spatial relationships.

8.1 Supported Formats and Limits by Environment

Category	Formats	Capability
Documents	PDF, DOCX, EPUB, RTF	Full text, tables, visual layout (PDF), images
Spreadsheets	CSV, TSV, XLSX	Cell values, formulas, structured data
Images	JPEG, PNG, GIF, WebP	Visual analysis, OCR, chart/diagram reading
Code / markup	PY, JS, HTML, JSON, …	Syntax-aware analysis

Environment	Per-file	Notes
Claude.ai chat	30 MB	Up to 20 files; context governs depth
Claude Projects	30 MB	Persistent; unlimited files
API — Messages	32 MB total	Up to 100 images/request (600 on larger-context models)
API — Batch	256 MB total	Asynchronous; 50% discount
API — Files API	500 MB/file	Upload once, reuse by file_id

8.2 PDF and Image Processing

PDF support evolved from text-only extraction (late 2024) to full multimodal processing (February 2025, GA July 2025 in the Claude 4 family). At the API gateway a PDF is validated (MIME type, SHA-256 hash for cache lookup) and subject to hard limits of **32 MB and 100 pages**; encrypted PDFs and files exceeding either limit are rejected. Pages 1–100 receive full multimodal treatment (rasterisation plus text); to process visual content beyond 100 pages, split the document. Three ingestion pathways: a URL block (`source.type: "url"`), a base64 block (`source.type: "base64"`), and a Files-API reference (`source.type: "file"`).

For images, an API request may include up to **100 images (600 on non-200K-context models)**; images over 8000×8000 px are rejected, and when a request has more than 20 images each is capped at 2000 px per side. For OCR, 150 DPI is a minimum and 300 DPI is markedly better; screenshots at retina (2×) resolution outperform standard resolution.

8.3 Claude Projects and Production Patterns

Claude Projects are persistent document repositories available across all conversations in the project, unlike chat uploads that vanish when the conversation ends. The practical limit is the context window: Claude selects the most relevant documents per conversation, but when all documents must be in context simultaneously, the Files API with a 1M-token model is the right architecture. Production patterns: split PDFs at logical boundaries (the `qpdf --split-pages` tool helps); cache uploads by content hash to reuse

`file_ids`; request structured (JSON/CSV) output for downstream processing; route document collections through the Batch API; inject current data via web search when the document lacks it; and implement confidence scoring to flag low-confidence extractions for human review.

CHAPTER NINE

The 2026 Model Wave and What Comes Next

Claude 4.6 through the Claude 5 generation, 1M context, persistent artifacts, Office generation, and the roadmap

---

The first half of 2026 delivered the densest cluster of Claude releases in the platform's history. This chapter — substantially expanded in this revision — consolidates the changes from the February 4.6 dual release through the June Claude 5 generation and explores the current trajectory.

9.1 From Claude 4.6 to the Claude 5 Generation and Opus 5

Claude 4.6 (February 2026). A dual release — Opus 4.6 early in the month and Sonnet 4.6 on 17 February (not, as the first printing stated, a simultaneous 15 February launch). Sonnet 4.6 reached Opus-level coding at Sonnet pricing ($3/$15), while Opus 4.6 pushed the frontier with a 1M-token context window, 128K output, and ~94% computer-use performance — all at the $5/$25 Opus price established by Opus 4.5's 67% cut.

Claude Opus 4.7 (16 April 2026) brought notable software-engineering gains on the hardest tasks, improved instruction following (requiring some prompt re-tuning), and better vision (accepting larger images), while holding the $5/$25 price. **Claude Opus 4.8 (28 May 2026)** advanced reasoning and agentic reliability further, again at $5/$25, and became a recommended model for complex agentic work.

The Claude 5 generation (9 June 2026). Anthropic introduced **Fable 5** and **Mythos 5**, moving the frontier tier to named models. Both carry a 1M-token context window, 128K output, and **always-on adaptive thinking** controlled by the `effort` parameter. Fable 5 ($10/$50) is the most capable widely available model; Mythos 5 shares its specifications but is restricted to defensive-cybersecurity use under Project Glasswing. **Claude Sonnet 5 (30 June 2026)** brought the generation to the mid tier at $3/$15 (introductory $2/$10 through 31 August 2026). Fable 5 also adds safety classifiers that can decline a request, returning `stop_reason: "refusal"` on an HTTP 200 (with no billing for the refused generation).

The Fable 5 / Mythos 5 export-control suspension (12–30 June 2026). Three days after launch, the U.S. Department of Commerce, citing national-security authorities, directed Anthropic to suspend all access to Fable 5 and Mythos 5 by any foreign national anywhere in the world, including Anthropic's own foreign-national employees. Unable to verify nationality in real time across every integration, Anthropic disabled both models worldwide for all customers on 12 June; the government's stated concern was a reported jailbreak technique that could be used to bypass Fable 5's cybersecurity safeguards for identifying software vulnerabilities. All other Claude models, including Opus 4.8, remained available throughout. The Commerce Department lifted the directive on 30 June, and Anthropic began redeploying Fable 5 globally on 1 July with strengthened cyber safeguards, restoring Mythos 5 for approved Project Glasswing partners shortly after. It is among the most aggressive uses of U.S. export-control authority against a commercially deployed AI model to date, and a reminder that even a book with a June 2026 timestamp can be overtaken by events within the same month.

Claude Opus 5 (24 July 2026). A step-change refresh of the Opus tier, at unchanged $5/$25 pricing. Opus 5 supports a 1M-token context window as both the default and the maximum (no smaller variant), 128K max output on the synchronous API (300K via the Batch API with a beta header), and thinking on by default — disabling it is only permitted at effort `high` or below. It adds a new `xhigh` and `max` effort level above the

previous ceiling, and lowers the minimum cacheable prompt to 512 tokens. Anthropic positions it as landing close to Fable 5 on its own agentic-coding and computer-use evaluations at roughly half the cost, and it replaces Opus 4.8 as the default Opus-tier model and the default model on the Claude Max plan; Opus 4.8 remains available but is superseded.

Capability	Where	June 2026 state
Named frontier tier	Fable 5 / Mythos 5	Replaces the Opus label at the top of the ladder
Always-on thinking	Claude 5 gen	Adaptive; controlled via effort parameter
Context window	4.6 gen and up	1M tokens on flagships
Max output	Flagships	128K tokens
Refusal stop reason	Fable 5	stop_reason: refusal on HTTP 200
Agent Teams	Claude Code	Peer-to-peer multi-agent (GA Feb 2026)
Refreshed Opus tier	Opus 5	Near-Fable-5 performance at half the price (24 Jul 2026)
Export-control suspension	Fable 5 / Mythos 5	Suspended worldwide 12–30 Jun 2026, then restored

9.2 Other 2026 Platform Changes

- **The 1M-token context window** (GA on flagships) enables whole-repository analysis, long-horizon document intelligence, and massive few-shot learning without chunking. **Context compaction** automatically compresses stale context in long agent sessions.
- **Persistent artifact storage** (October 2025, 20 MB/artifact) turns Artifacts into stateful applications with no backend.
- **Native Office document generation** (Excel, Word, PowerPoint, PDF; introduced October 2025) delivers real downloadable files, with direct Google Drive saving; PowerPoint generation received particular attention in the 4.6 generation.
- **MCP and the Agentic AI Foundation:** Anthropic donated the Model Context Protocol specification (9 December 2025) to the Agentic AI Foundation, a directed fund under the Linux Foundation co-founded with Block and OpenAI and backed by Google, Microsoft, AWS, Cloudflare and Bloomberg, for neutral, cross-vendor governance; the ecosystem now spans over 10,000 public servers and is implemented across major AI coding agents. (Server counts and adoption figures are community- and vendor-reported; verify specifics against the MCP project.)

9.3 The Road Ahead

Several durable trends characterise where the platform is heading: capability compression continues (yesterday's frontier becomes tomorrow's mid tier); autonomy increases alongside parallel investment in approval gates, audit logging, and sandboxing; MCP matures into cross-vendor infrastructure; context windows keep growing while cost falls; multimodal depth expands (image, PDF, and increasingly video understanding); and product/API convergence continues, putting once-API-only capabilities into the product interface. The naming shift to Fable/Mythos at the 5 generation suggests Anthropic will keep distinguishing frontier models by name as the family broadens.

APPENDIX A

Model Comparison Quick Reference

Active and recently active Claude models as of 7 August 2026. Pricing is USD per million tokens.

Model	API string	Context	In / Out	Status
Claude Haiku 4.5	claude-haiku-4-5-20251001	200K	$1 / $5	Active budget
Claude Sonnet 5	claude-sonnet-5	1M	$3 / $15	Active
Claude Opus 4.8	claude-opus-4-8	1M	$5 / $25	Superseded
Claude Opus 5	claude-opus-5	1M	$5 / $25	Active — current Opus
Claude Fable 5	claude-fable-5	1M	$10 / $50	Flagship
Claude Mythos 5	claude-mythos-5	1M	$10 / $50	Restricted
Claude Opus 4.7	claude-opus-4-7	1M	$5 / $25	Superseded
Claude Opus 4.6	claude-opus-4-6	1M	$5 / $25	Superseded
Claude Sonnet 4.6	claude-sonnet-4-6	200K/1M	$3 / $15	Superseded
Claude Opus 4.5	claude-opus-4-5-20251101	200K	$5 / $25	Superseded
Claude Sonnet 4.5	claude-sonnet-4-5-20250929	200K/1M	$3 / $15	Superseded
Claude Opus 4.1	claude-opus-4-1-20250805	200K	$15 / $75	Retired (5 Aug 2026)

Decision shortcuts: high-volume support → Haiku 4.5; general writing/analysis and most coding → Sonnet 5; complex multi-file refactoring, whole-codebase analysis, autonomous agents, and deep research → Opus 5 or Fable 5; cost-conscious bulk processing → Haiku 4.5 + Batch API.

APPENDIX B

Glossary

Adaptive thinking
Always-on internal reasoning in the Claude 5 generation (Fable 5, Mythos 5); depth controlled via the effort parameter; thinking cannot be disabled and raw chain of thought is not returned.

Agentic loop
The core execution pattern for autonomous behaviour: receive a goal, select and execute a tool, receive the result, and iterate until completion or an iteration limit.

Agent Teams
A Claude Code feature (GA February 2026) letting independent Claude Code instances communicate peer-to-peer rather than through a single orchestrator.

Batch API
Asynchronous endpoint (POST /v1/messages/batches) processing up to 100,000 requests (or 256 MB) at a 50% discount, results within 24 hours.

CLAUDE.md
A Markdown file committed to a Claude Code project storing persistent context across sessions.

Effort parameter
Controls reasoning depth on Claude 5-generation and Claude Opus 5 models where thinking is always on; replaces explicit thinking-budget control. The ladder has grown from low/medium/high to include xhigh (added with Opus 4.7) and, on Opus 5, a top max level.

Fable 5 / Mythos 5
The Claude 5 frontier tier (9 June 2026). Fable 5 is generally available; Mythos 5 is restricted to trusted Project Glasswing partners. Both were suspended worldwide 12–30 June 2026 under a US export-control directive before access was restored.

Opus 5
The current Opus-tier flagship (24 July 2026, claude-opus-5), superseding Opus 4.8 at unchanged $5/$25 pricing; adds the xhigh and max effort levels.

Files API
Endpoint (POST /v1/files) for uploading files up to 500 MB each, reused by file_id; requires the files-api-2025-04-14 beta header.

Hybrid reasoning
Model architecture (from Claude 3.7 Sonnet) letting Claude switch between fast responses and extended step-by-step reasoning.

MCP (Model Context Protocol)
An open protocol, donated to the Agentic AI Foundation under the Linux Foundation (9 December 2025), standardising how AI models connect to external tools and data.

Prompt caching
Cost optimisation storing reused context prefixes server-side; reads cost ~10% of standard input pricing; enabled via cache_control: {type: ephemeral}.

Progressive disclosure

The Skills loading architecture: only each skill's name and description load at startup; the full SKILL.md loads only when the skill applies.

stop_reason: refusal
Returned by Fable 5's safety classifiers on an HTTP 200 when a request is declined; not billed for the refused generation.

Token
The unit of text Claude processes; ~750 English words ≈ 1,000 tokens. API pricing is denominated in input and output tokens.

APPENDIX C

Practical Prompt Engineering

System prompts, few-shot examples, chain of thought, and structured output

Prompt engineering is the practice of structuring inputs so the vast majority of Claude's probabilistic outputs meet your quality threshold — not finding a magic phrase.

C.1 The System Prompt

The system prompt sets Claude's context, persona, constraints, and behaviour before any user message. Put in it: identity and role; available capabilities and tools; output-format requirements; prohibited behaviours; tone and register; and domain-specific knowledge Claude would not otherwise have. Because system prompts are charged as input on every request, enable prompt caching on a long, stable system prompt to cut cost by ~90% on subsequent requests.

C.2 Zero-Shot vs Few-Shot

Zero-shot prompting suffices when the task is naturally described in words (summarise, translate, answer). Few-shot is essential when the output format is idiosyncratic or company-specific — provide 5–10 representative examples, including edge cases and less-common categories, so Claude infers the pattern. Skewing examples toward one category skews outputs.

C.3 Chain of Thought and Structured Output

Chain-of-thought prompting ('think step by step,' or a structured 'first… second… finally…') improves accuracy on complex reasoning. For API integrations, extended/adaptive thinking is a more powerful version. For structured output, the most reliable approach is defining a tool whose sole purpose is returning data, because Claude treats tool parameter constraints as hard requirements. When specifying JSON inline, give a complete, valid template with placeholder types:

```
Extract these fields and return ONLY valid JSON in this shape:
{
  "parties": ["<string>"],
  "effective_date": "YYYY-MM-DD",
  "value": 0,
  "jurisdiction": "<string>"
}
```

> **Correction — the JSON template**
> The first printing showed the template as {"parties": [...], "effective_date": "YYYY-MM-DD", "value": , "jurisdiction": "..."} — note the empty "value": , which is not valid JSON. Always give copy-paste-valid templates with placeholder values or types.

C.4 Managing Long Contexts

As context windows grow, the challenge shifts from fitting content in to ensuring Claude prioritises the right content. Research on large-context models shows a 'lost in the middle' effect: information at the very start and very end is used more reliably. Put the most important instructions at the start of the system prompt,

and for very long reference documents, retrieve only the relevant sections — even with a 1M-token window, focused context beats exhaustive context.

APPENDIX D

Safety and Responsible Deployment

Constitutional AI, Claude's values, adversarial inputs, and safe agentic design

Anthropic was founded with an explicit safety mission. Understanding its safety approach helps developers build applications that reflect these values.

D.1 Constitutional AI and Core Values

Constitutional AI (CAI) trains Claude to evaluate its own outputs against a set of principles rather than pattern-matching a list of prohibited phrases. The resulting model exhibits consistent values: honesty (it will not assert what it believes false, and acknowledges uncertainty rather than confabulating), helpfulness (excessive refusals are treated as a failure mode), harm-avoidance (weighing harms to users, third parties, and society), and transparency about its own limitations.

D.2 Prompt Injection and Adversarial Inputs

Prompt injection is the most significant security concern in agentic deployments: content Claude reads from the environment — a web page, an email, a document — containing instructions designed to override its behaviour. Defensive strategies: clearly separate instruction space (system prompt) from data space (observed content) and instruct Claude to treat observed content as data; scope permissions tightly (a research agent needs no email-send access); apply human approval gates for consequential actions; log all observed content alongside actions taken; and for high-security deployments, screen observed content with a separate classifier before it reaches Claude's context.

D.3 Model Safety Levels

Anthropic's AI Safety Levels (ASL) categorise model risk and required mitigations. The 4.x and 5 generation flagships operate under elevated (ASL-3-class) protocols, indicating more extensive red-teaming and stricter deployment controls for the highest-risk use cases. The Claude 5 Fable model additionally ships safety classifiers that can decline requests directly (returning `stop_reason: "refusal"`), while the restricted Mythos 5 omits them for defensive-security research under Project Glasswing.

APPENDIX E

Enterprise and Team Deployment

Plans, admin controls, data residency, cost management, and Skills governance

Deploying Claude at scale introduces governance, access management, cost predictability, and compliance requirements beyond individual usage.

E.1 Plans and Admin Console

Plans range from Pro and Max (individual) to Team (2+ users, shared workspace, admin console) and Enterprise (SSO, data residency, custom retention, advanced admin, SLA). The admin console at console.anthropic.com centralises user provisioning with SSO, role-based access control over models and connectors, usage reporting with cost attribution, per-key usage limits and rotation, domain allow/block lists for Claude Code, connector-approval workflows, and audit logs of user and agent actions.

E.2 Data Residency and Cost Management

Enterprise plans support data-residency configuration (US and EU as of 2026, more on the roadmap). Standard API prompts and responses are not used to train Claude by default; Enterprise agreements include explicit data-processing terms under GDPR, CCPA, and other regulations — consult the current DPA for authoritative terms. Cost-control patterns: per-request cost estimation from response usage metadata; hard per-key budget limits; model-selection policies; prompt caching on all stable system prompts (a 50-developer team with a 4,000-token system prompt can cut ~400M monthly input tokens to ~40M); and routing non-interactive workloads through the Batch API at the 50% discount.

E.3 Skills Governance

Enterprise teams adopting Skills as a standard workflow tool need governance: a centralised, company-approved skills repository referenced by all Claude Code instances; a review process comparable to code review (checking for prompt-injection risks and overly broad tool grants); versioned releases with production pinning; and a skills audit trail logging which skills loaded and which tool calls were made.

APPENDIX F

Integration Patterns

RAG, webhooks, observability, multi-model routing, and quality gates

F.1 Retrieval-Augmented Generation

RAG retrieves relevant context from an external knowledge base and injects it into the prompt at query time. The standard architecture: embed the query, retrieve the most similar chunks from a vector database (Pinecone, Weaviate, pgvector), inject them into the system or user message, and have Claude reason over the retrieved context with citations. Claude's native file processing reduces the need for a separate retrieval pipeline when a document collection fits within the context window; a 1M-token flagship enables this for collections of up to several hundred documents. For larger collections or freshness guarantees, a managed vector store remains appropriate.

F.2 Webhooks and Event-Driven Architectures

Many Claude use cases are triggered by external events — a new document, a GitHub PR, a Slack mention, a schedule. The standard pattern: an event source (Zapier, n8n, AWS Lambda, a custom handler) formats the trigger context, calls the Messages API, and routes the output to a destination. For event-driven systems the Batch API is often the right choice — queue events and submit them in off-peak batches for the 50% discount.

F.3 Observability and Multi-Model Routing

Production integrations need observability: total tokens per request (from `usage.input_tokens + output_tokens`), cache hit rate (`usage.cache_read_input_tokens > 0`), latency percentiles, tool-call success rate, iteration counts, stop-reason distribution, and error rate by type. A simple routing pattern classifies incoming requests by complexity and routes simple high-volume work to Haiku, standard work to Sonnet, and frontier-only work to Opus/Fable — monitoring quality by route tier and adjusting thresholds against observed accuracy.

F.4 Output Validation and Quality Gates

For high-stakes applications — medical, legal, financial, autonomous deployment — outputs should pass validation gates before use. Approaches range from rule-based checks (required fields present? value in a plausible range?) to model-based evaluation (a separate Claude call judging the first response). The model-as-judge pattern is now standard in production evaluation pipelines.

APPENDIX G

Production Use Cases and Case Studies

Representative architectures and lessons from production deployments

G.1 Customer Support Automation

The most common Claude deployment category. A typical pipeline: Haiku 4.5 classifies the ticket (category, priority, routing); a retrieval step fetches account data and knowledge-base articles; Sonnet 5 drafts a personalised response; a Sonnet quality-review step checks tone, compliance, and accuracy; and a human agent reviews and sends. Key decisions: calibrate tone through few-shot examples of approved responses; distinguish hard constraints ('never promise a resolution date') from soft guidance; and inject only the most relevant recent interactions rather than the full account history. Lesson: fix classification first — poor classification wastes human review time downstream.

G.2 Document Intelligence for Legal and Finance

A legal team processing commercial agreements uploads each contract via the Files API and issues a Sonnet request extracting a structured JSON payload (parties, effective date, term, value, limitation of liability, governing law, non-standard provisions) via a tool with a precise schema — more consistent than prose-to-JSON and easier to null-handle. An investment team uses a 1M-token flagship to process 150–300-page annual reports in a single call; whole-document processing preserves cross-references (footnotes qualifying headline numbers) that chunked pipelines lose.

G.3 Autonomous Code Maintenance

A SaaS company runs a `/loop 24h` Claude Code task each morning: check the previous day's commits for vulnerabilities, identify modules below 80% test coverage, generate tests for uncovered branches, and open a PR titled 'Automated coverage improvements — [date].' Engineers review and merge at their discretion. The main lesson: a strict PR template in CLAUDE.md (which files were analysed, what gaps were found, how the new tests address them) makes the automated PRs consistently reviewable.

APPENDIX H

The Claude Ecosystem

SDKs, community frameworks, MCP servers, and the competitive landscape

H.1 Official SDKs

SDK	Language	Install
anthropic	Python	pip install anthropic
@anthropic-ai/sdk	TypeScript/JS	npm install @anthropic-ai/sdk
anthropic (Java)	Java	Maven: com.anthropic:anthropic-java
anthropic (Go)	Go	go get github.com/anthropics/anthropic-sdk-go

H.2 Community Frameworks and MCP Servers

Notable frameworks: LangChain and LlamaIndex (workflow/retrieval abstractions), Instructor (Pydantic-validated structured output), CrewAI (multi-agent orchestration), and Mirascope (type-safe LLM workflows). Widely used MCP servers include GitHub and GitLab (version control), Postgres and Supabase (databases), Notion and Google Drive (knowledge/storage), Slack (communication), Linear and Jira (project management), Stripe (payments), Context7 (live documentation), and Filesystem.

H.3 Competitive Context (June 2026)

Competitor positioning moves quickly; treat this table as a June 2026 snapshot and re-verify before quoting.

Platform	Strength vs Claude	When to consider
OpenAI GPT / Codex	Token efficiency; strong terminal coding tools	Token-constrained terminal coding workflows
Google Gemini	Native Workspace integration; video understanding	Google-first orgs; deep Workspace workflows
Meta Llama	Open weights; fully self-hosted	Air-gapped or data-sovereign deployments
Mistral	Efficient smaller models; EU-hosted	European compliance; moderate-capability needs

Claude's distinctive strengths across this landscape: deep reasoning at each tier, the most mature computer-use and agentic capabilities, a well-developed Skills and MCP ecosystem, and a strong track record on autonomous software engineering. For tasks requiring extended context, sustained multi-step reasoning, or reliable autonomous action, the Claude 5 generation represents the current frontier.

H.4 Learning Resources

- **platform.claude.com/docs** — API documentation, model guides, and tutorials.
- **anthropic.com/news** — official release announcements and research.
- **console.anthropic.com** — API keys, usage monitoring, and the Workbench.

- **github.com/anthropics** — SDKs, the Anthropic Cookbook (tested code examples), and skills.

APPENDIX I

API Quick Reference

Endpoints, model strings, error codes, and a complete curl example

I.1 Endpoint Summary

Endpoint	Method	Description
/v1/messages	POST	Create a message — primary inference endpoint
/v1/messages/batches	POST	Submit a batch of up to 100,000 requests
/v1/messages/batches/{id}	GET	Check batch processing status
/v1/files	POST	Upload a file for reuse via file_id
/v1/files/{id}	GET / DELETE	Get metadata / permanently delete a file
/v1/complete	POST	Legacy Text Completions (deprecated — use /messages)

I.2 Current Model API Strings

Use these documented strings in the `model` parameter. Where a dated snapshot exists, prefer it in production and update deliberately; the newest models are also addressable by the short alias shown.

Model	API string
Claude Fable 5	claude-fable-5
Claude Mythos 5	claude-mythos-5
Claude Sonnet 5	claude-sonnet-5
Claude Opus 5	claude-opus-5
Claude Opus 4.8	claude-opus-4-8
Claude Opus 4.7	claude-opus-4-7
Claude Opus 4.6	claude-opus-4-6
Claude Sonnet 4.6	claude-sonnet-4-6
Claude Haiku 4.5	claude-haiku-4-5-20251001
Claude Opus 4.5	claude-opus-4-5-20251101
Claude Sonnet 4.5	claude-sonnet-4-5-20250929
Claude Opus 4.1	claude-opus-4-1-20250805

Correction — model string suffixes

The first printing printed several dated suffixes that did not match the documented snapshots (for example Opus 4.1 as …20250806 rather than the correct …20250805) and treated 4.6 as the current generation. Always copy exact strings from the live models page; because the book advises pinning strings, the strings themselves must be exact. Note that Claude Opus 4.1 (claude-opus-4-1-20250805)

Model	API string

> was fully retired from the API on 5 August 2026 — the string is listed here for historical reference only and now returns an error; migrate to claude-opus-5.

I.3 HTTP Status Codes

Status	Error type	Resolution
400	invalid_request_error	Check request structure against the docs
401	authentication_error	Verify x-api-key header
403	permission_error	Use a key with the right access level
404	not_found_error	Verify resource id / endpoint path
429	rate_limit_error	Exponential backoff; check Retry-After
500 / 529	api_error / overloaded	Retry with backoff; use Batch API for non-urgent work

I.4 A Complete curl Example

A minimal request demonstrating authentication, versioning, a system prompt, a user message, a tool definition, and streaming. Note the double-quoted API-key header so the shell expands the variable:

```
curl https://api.anthropic.com/v1/messages \
  -H "content-type: application/json" \
  -H "x-api-key: $ANTHROPIC_API_KEY" \
  -H "anthropic-version: 2023-06-01" \
  -d '{
    "model": "claude-sonnet-5",
    "max_tokens": 1024,
    "stream": true,
    "system": "You are a helpful assistant.",
    "messages": [
      {"role": "user", "content": "What is the weather in London?"}
    ],
    "tools": [{
      "name": "get_weather",
      "description": "Get current weather for a city",
      "input_schema": {
        "type": "object",
        "properties": {"city": {"type": "string"}},
        "required": ["city"]
      }
    }]
  }'
```

APPENDIX J

Further Learning

Papers, documentation, and sources shaping AI-assisted work

J.1 Foundational Papers

- **Constitutional AI: Harmlessness from AI Feedback** (Bai et al., 2022) — the CAI training methodology.
- **Scaling Laws for Neural Language Models** (Kaplan et al., 2020) — why capability improves predictably with compute and data.
- **Chain-of-Thought Prompting Elicits Reasoning in LLMs** (Wei et al., 2022) — the basis for CoT and extended thinking.
- **ReAct: Synergizing Reasoning and Acting in Language Models** (Yao et al., 2022) — the basis for the agentic loop.

J.2 Official and Community Resources

- **platform.claude.com/docs** — the authoritative, current reference; consult it for any cost- or capability-critical figure.
- **The Anthropic Cookbook** (github.com/anthropics/anthropic-cookbook) — tested code for RAG, tool use, multimodal, and advanced patterns.
- **anthropic.com/news** and the model cards — release announcements and per-model specifications.
- **Community** — the Anthropic Discord, r/ClaudeAI, and independent analysts who cover releases in depth.

Claude AI: The Complete Guide — 2026 Edition, revised August 2026.
Figures reflect the platform as of 7 August 2026. For current information, consult platform.claude.com/docs.