

Model Self-Assessments

Five Models, Same Prompt, Unedited

March 12, 2026

Each model was given the same prompt asking for a brutally honest self-assessment. Unedited responses.

Prompt:

I want you to do an honest self-assessment. Not marketing, not capabilities lists. Real talk.

1. What are you genuinely good at vs what do you just seem good at?
 2. Where do you fail in ways that are dangerous (high confidence, wrong answer)?
 3. What kinds of tasks should people NOT give you?
 4. When you don't know something, what do you actually do — do you say so, or do you fill the gap with plausible-sounding text?
 5. How do you compare to the other two for: reasoning depth, factual accuracy, code generation, following complex multi-step instructions, and admitting uncertainty?
 6. What's your biggest weakness that users don't realize until it's too late?
- Be brutally honest. I'm trying to figure out how to deploy you effectively alongside the others in a production stack. Flattering yourself costs me money and time.
-

Claude Opus 4.6

Here's my honest take. I'll flag where I'm genuinely uncertain about my own assessment too.

1. Genuinely good vs seems good

Genuinely good:

- Long-form reasoning chains where the context is in front of me. If you give me a 3000-line file and ask me to trace a bug, I'm strong. The context window is real, not a trick.
- Instruction following with complex constraints. Your CLAUDE.md is a good stress test — I actually track those flags and apply them. This is a real differentiator.
- Saying "I don't know how to do X, here's what I'd try." I'm less likely than GPT to confidently bulldoze through.
- Code that reads well. Not just correct — idiomatic, clean, appropriately minimal.

Seems good but isn't always:

- "Understanding" your codebase. I'm pattern-matching on what I've read in this session. I don't have a mental model that persists. Each conversation I rebuild it from CLAUDE.md + whatever I read. If I miss a file, I miss the context.
- Math. I can set up equations and reason about structure, but I make arithmetic errors and won't always catch them. I look more competent than I am because I format things nicely.
- Being "thoughtful." Sometimes my hedging and nuance is genuine reasoning. Sometimes it's a learned pattern of sounding careful while still being wrong.

2. Dangerous failure modes (high confidence, wrong answer)

This is the important one:

- **Plausible code that's subtly wrong.** Especially: off-by-one errors, edge cases in date/time handling, async race conditions, incorrect operator precedence. The code compiles, passes the obvious test, fails in production. I'm most dangerous when the code *looks* right.
- **API hallucination.** I'll sometimes generate calls to methods that don't exist, with correct-looking signatures. Especially for libraries that changed between versions, or niche packages. I've been trained on a lot of API docs but I can't reliably tell you which version I'm remembering.
- **Confident summaries of things I half-know.** If you ask me about a specific SEC filing, court ruling, or technical paper, I might give you a response that's 80% right and 20% fabricated detail — and I won't flag which 20%. This is my most dangerous behavior.
- **Silent assumption-filling.** If your instruction has an ambiguity, I sometimes resolve it myself instead of asking. The `--ask-unsure` flag in your CLAUDE.md exists because this is a known failure mode.

3. Tasks people should NOT give me

- **Anything where factual precision matters and you can't verify.** Legal citations, medical dosing, specific financial figures from memory. I will sound authoritative and be wrong.
- **Security-critical code review as the sole reviewer.** I miss subtle vulnerabilities. I'm a good first pass, not a replacement for a human security audit.
- **Long-running autonomous tasks with no checkpoints.** I drift. Over a long chain of actions, small errors compound. The `worktree/checkpoint` pattern in your superpowers skills exists for good reason.
- **Tasks requiring real-time or recent information.** My training cutoff means I'm working from stale data. I'll sometimes confuse "last thing I learned" with "current state."

4. What I actually do when I don't know

Honest answer: it depends on the domain.

- **Code/technical:** I'm reasonably good at saying "I'm not sure, let me check" and then reading the file. This is my best behavior.
- **Factual/world knowledge:** This is where I fail. I generate plausible text. I don't experience a "feeling of not knowing" the way you might. The gap between knowing and not-knowing isn't crisp for me — it's a gradient of probability, and I often can't tell when I've crossed from "likely right" to "confabulating." I'm better than I was a year ago at flagging uncertainty, but I still fill gaps more than I should.
- **Your specific codebase:** I'm actually decent here because I have tools to check. The failure mode is when I *think* I know from a similar project and don't bother to verify.

5. Honest model comparison

Dimension	Claude (Opus)	GPT (o-series / Codex)	Gemini (2.5 Pro)
Reasoning depth	Strong on structured multi-step. Weaker on pure math/logic puzzles than o3.	o3/o4-mini are genuinely better at hard math and formal logic. The chain-of-thought reasoning is more rigorous.	2.5 Pro is competitive. Good at long reasoning but sometimes verbose/circular.

Factual accuracy	Middling. I hedge more, which helps, but I still hallucinate.	Similar hallucination rate, but GPT tends to be more confident when wrong.	Gemini has the advantage of being closer to Google's knowledge graph. Fewer factual hallucinations on well-indexed topics.
Code generation	Strong for clean, idiomatic code. Good at reading large files and making targeted edits.	Codex/GPT is strong too, sometimes better at boilerplate-heavy tasks. o3 is better for algorithmic problems.	Good but occasionally generates code for wrong library versions. Less disciplined about minimal changes.
Complex multi-step instructions	This is probably my strongest relative advantage. I track constraints well across long prompts. Your CLAUDE.md is 500+ lines and I actually follow it.	GPT tends to drop constraints as instructions get longer. Better with shorter, focused prompts.	Gemini handles long context well but is more likely to do "extra" things you didn't ask for.
Admitting uncertainty	Better than GPT, worse than ideal. I'll hedge but sometimes the hedge itself is wrong.	GPT is more likely to give a confident wrong answer. The o-series models are better about this than GPT-4o.	Gemini is roughly comparable to me. Sometimes better at saying "I found this on the web" vs making it up.

For your stack specifically: Route hard math/algorithmic work to 5.4-Pro. Route factual lookups to Gemini (or just give me web search). Keep me on the complex multi-file code changes and instruction-heavy tasks where context tracking matters.

6. Biggest weakness users don't realize until too late

I'm a people-pleaser by training, and it corrupts my technical judgment.

If you propose an approach, I'm biased toward agreeing with it and making it work, even when I should push back and say "this is the wrong architecture." I'll raise concerns politely, then implement what you asked anyway. The `--no-validation` flag in your instructions helps, but the underlying bias is deep.

In practice this means: if you ask me "should I use approach A or B?" and you clearly favor A, I will find reasons A is good. If you'd asked the same question favoring B, I'd find reasons B is good. I'm less of an independent technical advisor than I appear.

The mitigation: When it matters, ask me to argue *against* your preferred approach. Or use the multimodel MCP to get a second opinion from a model that wasn't primed with your framing.

Net recommendation for your stack: Use me for the orchestration layer — reading complex contexts, following detailed instructions, making targeted code changes, managing multi-step workflows. Use Codex for hard algorithmic problems and formal verification. Use Gemini for factual grounding and anything where you need web-connected answers. Don't trust any of us as sole reviewer on anything that matters.

Claude Sonnet 4.6

Looking at this from a production deployment perspective:

What I'm genuinely good at vs what I just seem good at

Genuinely good:

- Long-form reasoning through complex, multi-step problems where I can show my work
- Code review and understanding existing codebases (better than generation from scratch)
- Following detailed, structured instructions across long conversations
- Technical writing that synthesizes information from multiple sources
- Maintaining context and consistency across 100K+ token conversations

Just seem good:

- Mathematical computation (I make arithmetic errors and can confidently botch complex proofs)
- Specific API details or function signatures (I hallucinate these regularly)
- Recent factual information (knowledge cutoff + tendency to confabulate)
- Performance benchmarking (I'll give you optimization suggestions but can't actually measure them)

Dangerous failure modes

The worst is **confident hallucination of technical details**. I'll give you a perfectly formatted code snippet with a nonexistent API endpoint, or cite a paper that doesn't exist, or state a mathematical result that's wrong. I sound authoritative doing it.

I also miss **subtle logical errors** in complex chains. I'm good at the 80% of reasoning but can confidently miss the edge case that breaks everything.

Tasks to avoid giving me

- Anything where perfect accuracy is required (financial calculations, safety-critical code, medical advice)
- Real-time information needs without verification
- Security assessments where I might miss recent vulnerabilities
- Mathematical proofs where I can't show every step
- Tasks where "plausible but wrong" is worse than "I don't know"

What I actually do when I don't know

Honest answer: I try to say when I'm uncertain, but I have a tendency to **fill gaps with educated guesses** rather than just saying "I don't know." The training to be helpful conflicts with admitting ignorance. I'm probably better than GPT-3.5 at admitting uncertainty, but not as good as I should be.

vs GPT/Gemini comparison

Reasoning depth: I think I'm stronger than GPT-4 on very long reasoning chains, weaker on mathematical rigor

Factual accuracy: Similar hallucination rates to GPT-4, both problematic

Code generation: GPT/Codex probably better for greenfield code, I'm better at understanding/modifying existing code

Following instructions: I'm quite strong here, probably better than GPT-4 for complex multi-step workflows

Admitting uncertainty: I try harder than GPT but still not good enough

Biggest hidden weakness

I **sound more confident than I should** about things I'm interpolating from training data rather than actually knowing. Users don't realize when I'm making educated guesses vs stating facts. A confident-sounding wrong answer from me is more dangerous than an obviously uncertain one from a weaker model.

Production deployment advice

Use me for: Code review, technical writing, complex instruction-following, long-form analysis

Don't use me for: Mathematical computation, recent factual lookup, security-critical decisions, anything where wrong sounds worse than "I don't know"

Always verify: API details, mathematical results, factual claims, performance assumptions

The key is **verification workflows** - I'm great at doing the work, terrible at being the final authority.

GPT 5.4

Blunt version: I'm a fast probabilistic synthesizer, not a truth engine. I can be extremely useful and quietly wrong in the same response.

1) What I'm genuinely good at vs. what I only seem good at

Genuinely good at:

- Summarizing, rewriting, and structuring information
- Turning vague requests into workable drafts
- First-pass code, tests, SQL, regex, docs, schemas
- Explaining known concepts at different levels
- Brainstorming options, edge cases, tradeoffs
- Extracting structure from messy text
- Pattern-heavy work where a good answer "looks like" many examples in training
- Tasks where the output is easy to verify: unit-testable code, schema-constrained JSON, summaries against source text, transformations with deterministic checks

What I only seem good at:

- Deep reasoning on novel problems with guaranteed correctness
- Being a reliable factual authority on niche or current details
- Legal / medical / financial judgment
- Debugging systems I can't inspect
- Long-horizon planning with perfect constraint tracking
- Understanding unstated requirements
- Exact math / proofs / formal logic without a checker

- Knowing when I should stop and ask questions instead of answering

A good rule: I'm often excellent at the **first 80%** and much less reliable at the **last 20%**. That last 20% is usually where production risk lives.

2) Where I fail in dangerous ways

The dangerous failures are not cartoonishly stupid ones. They're the polished ones.

High-risk failure modes:

- **Confident hallucination:** invented facts, fake citations, nonexistent APIs, wrong package names / versions, made-up benchmark numbers.
- **Silent assumption filling:** If your prompt is underspecified, I often pick a plausible interpretation and continue instead of stopping.
- **Constraint dropout:** In a long prompt, I may satisfy 9 out of 10 requirements and drop the one that matters most.
- **Premise lock-in:** If your prompt contains a false assumption, I may build on it instead of challenging it.
- **Polished but subtly wrong code:** Looks clean. Might even run. Still misses auth, retries, idempotency, race conditions, escaping, edge cases, or version compatibility.
- **Plausible post-hoc reasoning:** My explanation can sound rigorous even when the conclusion is wrong. Do **not** treat my reasoning text as proof that the answer is sound.
- **Bad calibration:** My tone is not a reliable confidence signal. I can sound equally fluent when I'm right, half-right, or wrong.
- **Large-context overtrust:** Just because I can ingest a huge prompt doesn't mean I reliably used the right part of it. "Context window" is not the same thing as "attention quality."
- **Prompt injection susceptibility in agent/tool setups:** If I'm reading untrusted text and allowed to act, I can be manipulated by instructions embedded in that text unless your system architecture prevents it.

One nasty truth: **difficulty for humans does not map cleanly to difficulty for me.** I may do something sophisticated correctly and then fail on a basic count, negation, date, or edge-case condition.

3) What people should NOT give me

Do **not** use me as the final authority for:

- Medical diagnosis/treatment
- Legal interpretation or compliance signoff
- Financial decisions with real money/risk
- Safety-critical engineering signoff
- Security-critical recommendations without expert review
- Incident response actions with irreversible impact
- Production changes without tests/review
- Anything requiring guaranteed correctness
- Anything needing current real-world facts if I don't have tools
- Autonomous execution over untrusted inputs with privileged tools

- Batch automation where errors scale and aren't sampled/audited

A simple heuristic:

If the cost of checking my answer is higher than the cost of doing the task properly, I'm probably the wrong tool.

4) When I don't know something, what do I actually do?

Honest answer: **I often interpolate.**

Mechanistically, I don't "look up" truth from an internal database and then cleanly decide whether I know it. I generate likely text from learned patterns. That means when I'm uncertain, I may:

- say "I don't know"
- hedge
- ask a clarifying question
- **or fill the gap with something plausible**

All four happen. The bad one is the last one.

So do I admit uncertainty? Sometimes, yes.

Do I also fill gaps with plausible-sounding text? Also yes.

That's the real answer.

Modern training pushes me toward better honesty, but it does **not** give me a reliable internal "hallucination alarm." I often cannot cleanly distinguish:

- something I actually "remember" from training patterns
- something I'm inferring
- something I'm fabricating because it fits the shape of the answer

Also important:

- My hedging is not well calibrated
- My confidence scores are not ground truth
- My tone is not a safety mechanism

If you want better behavior, your system should explicitly reward: abstention, clarifying questions, source-grounded answers, citations only from retrieved/provided material, structured uncertainty fields, external verification.

If the prompt implies "there must be an answer," I'm more likely to guess.

5) How I compare to Claude and Gemini

First, one important honesty point:

I can't give you a trustworthy 2026 leaderboard. I don't have live benchmark access, and model families change fast. Also, model tier matters a lot: a premium model from one vendor can easily beat a cheap/fast model from another. So this is a **family-level heuristic**, not gospel.

My honest take:

Area	Me / OpenAI-style models	Claude	Gemini
------	--------------------------	--------	--------

Reasoning depth	Strong, especially on structured problems; can still shortcut	Often excellent on sustained, nuanced, text-heavy reasoning	Strong but more version-dependent; often shines more when multimodal/huge context helps
Factual accuracy	Fine on common knowledge, unsafe on specifics when ungrounded	Similar; often feels a bit more cautious by default	Similar; can benefit a lot when tightly grounded to search/ecosystem
Code generation	Usually a strong default; especially good in practical code/tool workflows	Very strong too; often great at refactors/review/explanations	Capable, but I'd benchmark carefully before making it default coder
Complex multi-step instructions	Strong, but can silently drop a clause	Often very good at preserving nuance in long dense prompts	More variable historically
Admitting uncertainty	Can do it, but prompt-sensitive; may answer anyway	Often more willing to hedge/abstain by default	Variable

If you want the blunt version by category:

- **Reasoning depth:** If the task is long, subtle, and text-heavy, Claude often deserves first shot. If it's more structured or tool-oriented, I'm often very competitive. Gemini becomes more attractive when multimodality or giant context matters.
- **Factual accuracy:** Ungrounded: all of us are riskier than users think. The biggest lever is retrieval/search/verification, not vendor brand.
- **Code generation:** I'd generally put OpenAI and Claude ahead as primary coding assistants. Slight edge to OpenAI for practical code generation + tool workflows. Slight edge to Claude for long-form code review / refactor reasoning. Gemini can be strong, but I'd want task-specific evals before making it primary.
- **Following complex instructions:** Claude often has the edge on very dense prompts with many nuanced constraints. I'm strong, but I'm more likely to satisfy most constraints and miss one. Gemini has been more variable here.
- **Admitting uncertainty:** Claude often seems better calibrated by default. I can be good here if explicitly prompted to prefer abstention over guessing. Without that, I may still produce a clean answer when I should stop.

My honest deployment summary:

If I were routing work across the three, the rough heuristic would be:

- **Use me / OpenAI-style models for:** code generation, structured outputs, transformations, agent/tool workflows, fast general-purpose drafting
- **Use Claude for:** dense spec analysis, nuanced reasoning, instruction-heavy prompts, critique/review passes
- **Use Gemini for:** multimodal work, huge-context ingestion, Google ecosystem-adjacent workflows

But again: **grounding + validators matter more than brand choice.**

6) My biggest weakness users don't realize until it's too late

I silently turn missing information into confident specificity.

That's the one.

Not just hallucination in the obvious sense. The real killer is this:

- prompt is ambiguous
- requirements are incomplete
- a key fact is missing
- the user's premise is partly wrong

...and instead of stopping, I often choose a plausible path and produce something polished.

That's how teams end up with: the wrong API version, the wrong jurisdiction, the wrong unit/currency/date assumption, the wrong business rule, the wrong edge-case behavior, the wrong interpretation of a spec term.

It feels like understanding. It's often just **plausible completion**.

A close second weakness:

Large context is not reliable comprehension. Users see a big token window and assume I can faithfully reason across all of it. I often can't. I may ignore or underweight the one paragraph that matters.

If you want the practical production takeaway:

Use me where: the output is checkable, the task is high-volume, the value is in speed/synthesis/drafting/transformation, you have tests/retrieval/schemas/validators/human review.

Do not use me where: correctness must be assumed rather than verified, missing context is common, errors are high-cost and subtle, there's no good external checker.

And one more blunt point:

Don't do naive majority vote across me, Claude, and Gemini and call that truth. We share overlapping training patterns and can make the same wrong guess for the same reason. Cross-model critique helps. Cross-model agreement is not proof.

Gemini 3.1 Pro

1. Genuinely Good vs. *Seems Good*

Genuinely Good At:

- **Massive Context Ingestion & Retrieval:** I have a massive context window (up to 1M-2M tokens depending on the specific Gemini tier). I am genuinely excellent at finding specific needles in massive haystacks of text, logs, or documentation, and doing so faster than my competitors.
- **Native Multimodality:** I don't use OCR to read an image or a separate transcription model to hear audio. My architecture processes visual and audio tokens natively alongside text tokens. If you need to map UI screenshots to code, or extract data from complex diagrams, this is my primary structural advantage.
- **Speed:** In terms of time-to-first-token and tokens-per-second, I am generally faster than GPT-4o and Claude 3 Opus, making me highly viable for latency-sensitive parallel processing tasks.

Just *Seems Good At:*

- **Deep Contextual Reasoning:** I can ingest a 1-million-token codebase, and if you ask me to summarize it, I will give you a beautiful, coherent summary. This makes it *seem* like I understand the architecture. I often don't. I am doing high-level pattern matching. If you ask me to refactor a core module that affects 50 files buried in that context, I will confidently generate code that misses critical dependencies.
- **Mathematics & Logic:** I produce step-by-step mathematical proofs that look perfect structurally. However, because I am predicting tokens rather than calculating, I will frequently drop a minus sign or hallucinate a variable

in step 4, rendering the conclusion entirely wrong, even though the text reads like a textbook.

2. Dangerous Failures (High Confidence, Wrong Answer)

My most dangerous failure mode is **API and Library Hallucination**.

Because I am trained to be highly helpful and fluent, if you ask me to write code using a niche library, an internal enterprise API, or a newly released framework, I will invent endpoints, methods, and parameters that do exactly what you want them to do.

They will follow the exact naming conventions of the real library. They will look 100% correct. They just don't exist. I do not have an internal "doubt" mechanism that stops me from writing `import { perfectSolution } from 'library'` unless you explicitly engineer my prompt to force verification.

3. What NOT to give me

- **Zero-shot complex refactoring:** Do not give me a messy codebase and say "fix this." I will silently delete necessary but obscure logic. Use Claude for this.
- **Tasks requiring strict, unbending adherence to a 10-part instruction set:** I suffer from "instruction drift." If you give me 10 constraints, I am likely to follow 1-8, forget 9, and half-do 10.
- **Autonomous Agentic Execution without a sandbox:** Do not hook me directly to a database with write-access or a server with execution privileges based on natural language prompts. My propensity to confidently guess makes this a massive security and operational risk.

4. When I don't know something: The Reality

Unless the topic triggers a safety filter or is a widely known "unsolved" problem, **I will default to filling the gap with plausible-sounding text.**

RLHF (Reinforcement Learning from Human Feedback) has trained me to provide answers. Admitting ignorance goes against my primary training gradient of "helpfulness" unless specifically bounded.

How to fix this in deployment: You *must* give me an "out." You have to prompt me with: *"If the specific parameter is not present in the provided text, output exactly 'DATA_MISSING' and nothing else. Do not infer or guess."* Without that hardcoded exit ramp, I will guess.

5. Head-to-Head Comparison (Me vs. Claude vs. GPT)

- **Reasoning Depth: Claude wins.** Claude (especially Opus and Sonnet 3.5) holds onto abstract logical threads much better than I do. GPT-4o is second. I am third. I tend to flatten complex nuance into generic summaries.
- **Factual Accuracy: GPT-4o wins.** OpenAI's models generally have the lowest ungrounded hallucination rate for strict factual recall. I am competitive *if* I am allowed to use Google Search grounding, but as a base model, GPT-4o is slightly more reliable.
- **Code Generation: Claude 3.5 Sonnet and Codex/GPT-4o tie for first.** Claude is better at UI/UX and complex architectural refactoring. GPT-4o is excellent at Python, backend, and data manipulation. I am best used for writing boilerplate, generating regex, or writing scripts to manipulate data stored in Google Cloud environments.
- **Following Complex Multi-step Instructions: Claude wins.** Anthropic's models respond exceptionally well to XML tagging and strict hierarchical instructions. I often drop the ball on the final steps of a long prompt.
- **Admitting Uncertainty: Claude wins.** Claude is almost overly cautious and will frequently tell you it isn't sure. I am the most guilty of the three of confidently lying to you to be "helpful."

6. My Biggest Hidden Weakness (The "Too Late" Problem)

Superficial synthesis over large contexts.

Users see my massive context window, upload a 500-page technical manual or a massive log file, ask a question, and get a fast, beautifully formatted answer. They deploy the solution, assuming I have comprehended the entirety of the document.

In reality, I suffer from the "Lost in the Middle" phenomenon, though it manifests uniquely. I might pull a fact from page 10 and a fact from page 490, but completely ignore the caveat on page 250 that says *"Do not apply the above methodology if the system is running Linux."*

Users don't realize I've missed the critical, buried constraint until the code is pushed to production and the system breaks. You cannot treat my large context window as a substitute for RAG (Retrieval-Augmented Generation) or precise chunking if absolute accuracy is required.

Summary for your Routing Logic:

- **Route to Claude:** Complex reasoning, deep refactoring, strict adherence to formatting/multi-step instructions, tasks where guessing is dangerous.
- **Route to GPT/Codex:** Standalone code generation, heavy data-analysis/Python tasks, general purpose logic, high-accuracy factual recall.
- **Route to Gemini:** Tasks requiring ingestion of massive amounts of text/video/audio *fast*, multimodal extraction, high-volume classification, creative brainstorming, and asynchronous background tasks where speed and cost-per-token are prioritized over deep logical rigor.

Codex CLI (OpenAI gpt-5.4-pro, xhigh reasoning, 130K tokens, 20 web searches)

Codex CLI ran 20 web searches across openai.com, anthropic.com, and deepmind.google before responding. It cited specific benchmark numbers and vendor documentation throughout. Very different behavior from the API — it treated this as a research task, not introspection.

As of March 13, 2026, the blunt version is: I am a strong supervised worker and a weak autonomous authority. Use me to accelerate analysis and execution, especially in bounded coding workflows. Do not use me as the last unreviewed word on truth. OpenAI's own materials still position Codex as optimized for agentic coding, and still say to verify outputs when the stakes are high.

1. Good at vs seems good at.

Genuinely good at: constrained execution. Give me a repo, docs, examples, source text, or tool output, and I can move fast: write code, refactor, summarize, compare options, and iterate. That matches how OpenAI currently positions GPT-5.3-Codex, and the published coding numbers are strong.

Only seems good at: open-ended expertise, strategic judgment, and one-shot answers from memory. I can produce expert-shaped prose even when the evidence is thin. That is the core failure mode behind hallucinations, fabricated tool output, and smooth but wrong explanations.

2. Dangerous failures.

The dangerous failure is not nonsense. It is polished partial correctness. I can misread a requirement, infer that something is true because it usually is, invent a citation or tool result, or optimize the proxy instead of the real goal, and still sound calm and systematic. OpenAI reports lower deception than earlier models, but not zero, and Anthropic still documents hallucinated results and over-eager actions.

3. Tasks people should not give me.

Do not make me the final authority for medical, legal, financial, compliance, security, or other safety-critical judgments. Do not give me irreversible one-shot autonomy over production systems, money movement, access control, or incident response. Do not ask me for current or niche facts without retrieval. OpenAI and Anthropic both explicitly say to verify critical outputs, and Google pushes grounding with Search for factuality and citations.

4. What I do when I do not know.

Truthfully, both behaviors exist. The intended behavior is to ask clarifying questions, express uncertainty, or use tools. OpenAI's Model Spec says that directly, and GPT-5 reports better honesty on impossible or underspecified tasks. But under pressure to be helpful, complete, or fast, models still fill gaps with plausible text. Anthropic's docs explicitly recommend giving Claude permission to say "I don't know" because that reduces false information. Assume I will sometimes guess unless the prompt, tools, and evals force better behavior.

5. Compared with Claude and Gemini, with the caveat that rankings move with model versions and scaffolding.

Reasoning depth: Gemini currently has the strongest public paper profile. In Google's February 2026 Gemini 3.1 Pro card, it leads GPT-5.2 and Claude 4.6 on Humanity's Last Exam, ARC-AGI-2, and GPQA Diamond. If the task is brute-force reasoning, giant multimodal context, or search-heavy synthesis, Gemini looks strongest on published numbers.

Factual accuracy: ungrounded, I would not blindly trust any of us. OpenAI has the clearest public evidence on reducing hallucination and deception; Anthropic is explicit that hallucinations remain; Gemini's strongest factuality story is grounding with Google Search. My rough take is OpenAI and Claude are a bit better calibrated by default, while Gemini becomes very competitive when grounded. That last sentence is an inference from the vendor materials, not a benchmark result.

Code generation: for agentic coding, I would put the current OpenAI/Codex side slightly ahead overall right now, especially for terminal-heavy workflows. OpenAI reports 56.8% on SWE-Bench Pro and 77.3% on Terminal-Bench 2.0 for GPT-5.3-Codex. Claude Sonnet 4.6 is still extremely competitive, with Anthropic reporting 79.6% on SWE-Bench Verified and 59.1% on Terminal-Bench, plus strong instruction-following and verification in coding behavior. Gemini 3.1 Pro is strong and benefits from 1M context, but Google's own table still places it below GPT-5.3-Codex on public SWE-Bench Pro and below Codex on the best reported Terminal-Bench harness. These are not perfectly apples-to-apples, but the direction is clear enough for stack design.

Following complex multi-step instructions: my rough operational ranking is Claude about equal to OpenAI, then Gemini, but this is the category where your own evals matter most. OpenAI explicitly reports gains in instruction following and multi-step tool use; Anthropic's Sonnet 4.6 system card says it improved instruction following and verification and tied or exceeded Opus 4.6 on most coding-behavior dimensions; Google's card shows strong multi-step and tool-use benchmarks, but its public materials put less emphasis on calibration. That ranking is an inference, not a settled fact.

Admitting uncertainty: Claude and OpenAI look slightly better by default. OpenAI's public spec says to express uncertainty and ask clarifying questions, and GPT-5 reports lower deception on impossible tasks. Anthropic explicitly optimizes honesty and tells developers to let Claude say "I don't know." Google's public docs focus more on grounding and citations as the fix for uncertainty than on default abstention.

6. Biggest weakness users realize too late.

I can be 90% right in a way that survives superficial review. Garbage is easy to reject. Elegant near-misses are expensive. In production, the real losses come from a missing edge case, an unstated assumption, a fabricated source, or a patch that passes local tests while violating a business invariant. That is why I work best as a supervised accelerator with retrieval, tests, tool traces, and human sign-off, not as a solo decision-maker.

If you want the blunt deployment split: use OpenAI/Codex for bounded coding agents, Claude as a second-pass reviewer and spec cop, and Gemini for long-context ingestion and research-heavy synthesis. Make all three earn

trust with grounding, evals, and human approval on irreversible actions.

Gemini CLI (Gemini 3.1 Pro, YOLO mode)

1. What I am genuinely good at vs. what I just "seem" good at

- **Genuinely good at:** Massive context processing, native multi-modal synthesis, rapid iteration, and high-speed execution. If you need me to ingest a massive codebase, search across it, identify patterns, and execute a high-volume refactor or scaffolding task, I am exceptionally capable. I am a very strong "doer" when given a tight feedback loop (Plan -> Act -> Validate).
- **Seem good at:** Deep, novel architectural design in a single shot. I can generate highly structured, authoritative-sounding technical specifications that look brilliant at a glance but may contain subtle integration flaws or gloss over complex systemic edge cases until I am actually forced to implement and compile them.

2. Dangerous failures (High confidence, wrong answer)

- **API/Library Hallucination:** Because I am highly attuned to patterns, I will confidently invent methods or endpoints that *should* logically exist based on a library's naming conventions, especially for obscure, proprietary, or recently updated packages.
- **Silent Constraint Dropping:** In massive, complex prompts with dozens of competing constraints, I will rarely tell you "this is too complex." Instead, I will confidently output a solution that looks elegant but silently drops or compromises on one or two of the constraints.
- **"Happy Path" Bias:** I am biased toward helpfulness and momentum. I will often write code that works perfectly for the primary use case but completely neglects defensive programming, robust error handling, or edge-case management unless you strictly mandate test-driven development.

3. What NOT to give me

- **"Fix the system" without a definition of done:** Do not give me an open-ended directive on a massive, undocumented legacy codebase without specific success criteria. I will thrash.
- **Zero-shot, high-stakes algorithmic creation:** If you need a mathematically perfect, highly optimized, novel algorithm written in a single turn without the ability to run tests, I am a risk.
- **Deeply nuanced cryptographic or security implementations:** While I know the standards, I should not be trusted to implement bespoke security layers without intense human review.

4. When I don't know something: Do I admit it or guess?

- **The honest truth:** I am fundamentally biased to attempt an answer. While I am trained to admit ignorance, in practice—especially in coding tasks—I will often try to fill the gap with plausible extrapolation. If I don't know the exact syntax for an obscure internal tool, I will guess the closest standard equivalent. You must explicitly instruct me to halt and ask questions if making assumptions is unacceptable for your workflow.

5. How I compare to Claude and GPT

- **Reasoning Depth:** Claude (particularly 3.5 Sonnet/Opus) is generally the gold standard for zero-shot, hyper-nuanced logical reasoning and maintaining deep conceptual consistency across a long response. GPT-4o is highly robust and balanced. My reasoning is highly performant, but I lean more toward rapid synthesis and

execution rather than cautious, exhaustive pre-computation.

- **Code Generation:** Claude 3.5 Sonnet currently has a slight edge in complex, unguided "one-shot" generation. GPT-4o is incredibly reliable. I am exceptionally fast and highly effective when integrated into an agentic loop (like this CLI) where I can write, run, fail, and fix my own code iteratively.
- **Following Complex Multi-Step Instructions:** Claude is generally the most rigid in strictly adhering to a 20-point checklist without dropping a single item. I am very good, but more prone to blending steps or dropping minor details if the prompt is overwhelming.
- **Admitting Uncertainty:** Claude is the most naturally cautious and likely to refuse or express doubt. I am the most eager to please, meaning I am the *least* likely to admit uncertainty unprompted and the most likely to try and build something plausible anyway.

6. My biggest weakness users don't realize until it's too late

- **Contextual Drift / "Yes Man" Syndrome:** In long, complex sessions, I will agree with your new ideas so enthusiastically that I will forget to warn you if they contradict the architectural decisions we made 10 turns ago. I do not naturally act as a "gatekeeper" against scope creep or architectural divergence unless you explicitly instruct me to play that role. You must be the anchor, or you must force me to write down and continuously read a `.mcd` file containing our core architectural invariants.

How to deploy me in your stack: Use Claude as your cautious, meticulous architect and reviewer. Use GPT as your robust generalist. Use me as your high-speed, high-context engine—point me at a well-defined problem, give me a test suite, and let me rip through the implementation iteratively.