Member-only story

How I Turned Claude Code Into a War Machine (part 1)

From drowning in 1 project to crushing 5 in parallel. Here's the complete playbook.

Delanoe Pirard Follow 21 min read · Dec 6, 2025

⚠️ If you're not a Medium member, you can read this article for free using my friend link: **Read for free**.

__TL;DR__: I spent weeks building a productivity system around Claude Code.

__The result: 16 specialized expert agents__ (AI research, robotics, biology, SEO, frontend…), __6 connected MCPs__ for real-time verification (Context7, HuggingFace, Jupyter, Git, W&B, Fetch) __and__ an __anti-hallucination protocol__ that forces verification before generation.

__Outcome__: 3 or more projects managed simultaneously instead of 1–2, with significantly fewer hallucinations

__Setup time__: 20–40 hours upfront. __Break-even__: End of week 1. __Monthly cost__: ~$200 with the MAX subscription. (as of December 2025 — verify current pricing at anthropic.com/pricing)

> *This isn't magic. It's architecture.*

⚠️ **Disclaimer:** These are MY results based on MY workflow. Your mileage will vary depending on your use case. This isn't a universal solution — it's a documented experiment.

The Night Everything Changed

11 PM. Another hallucinated API. Another hour lost debugging fiction.

. . .

11 PM. Three GitHub tabs open. Deadline tomorrow.

I asked Claude for a PyTorch DataLoader configuration. It confidently gave me a function signature that doesn't exist. I copy-pasted it. Tests failed. Thirty minutes of debugging later, I realized the API it described was pure fiction.

This wasn't the first time. Claude had invented npm packages, fabricated React hooks, and hallucinated entire library methods. Each time, I'd spend

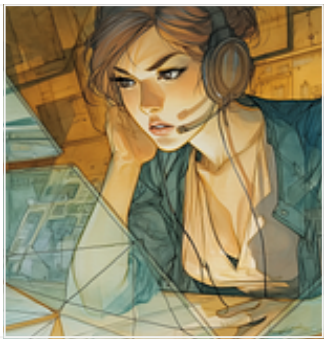
20–30 minutes verifying what should have been a 2-minute answer.

I was done.

The problem wasn't Claude's intelligence. It was Claude's confidence. It would make stuff up with the same tone it used for verified facts. No uncertainty markers. No "I'm not sure." Just confident bullshit.

I needed to transform Claude from an intern who wings it into a senior who cites sources.

> 💡 ***If 16 agents sounds overwhelming:*** *I wrote a companion guide that covers Skills, Hooks, and Commands — the automation layer that makes agents work together seamlessly. Start there if you want a gentler introduction.*



The Two Problems Nobody Talks About

Problem 1: Hallucinations Kill Productivity

Every developer knows this pain:

```
You: "What's the best embedding model for multilingual RAG in 2025?"

Claude (vanilla): "BERT and Sentence-BERT are popular choices
for multilingual embeddings..."
→ Vague
→ No date
→ No source
→ Potentially 2 years outdated
→ You spend 30 minutes verifying
```

Research confirms LLMs hallucinate frequently on technical questions — one comprehensive survey (2024) catalogued 32+ mitigation techniques, while Xu et al. (2024) proved mathematically that hallucinations are inevitable for computable LLMs. **That's not a bug — it's the architecture.** They predict plausible tokens, not verified facts.

The real cost: 2–3 hours per day spent verifying AI-generated answers.

Problem 2: Context Switching Drains You

Monday morning: Frontend work (React, Next.js, Tailwind). Monday afternoon: ML pipeline (PyTorch, HuggingFace, W&B). Tuesday: SEO optimization (keywords, meta tags, Core Web Vitals). Wednesday: Research paper review (citations, ArXiv, methodology).

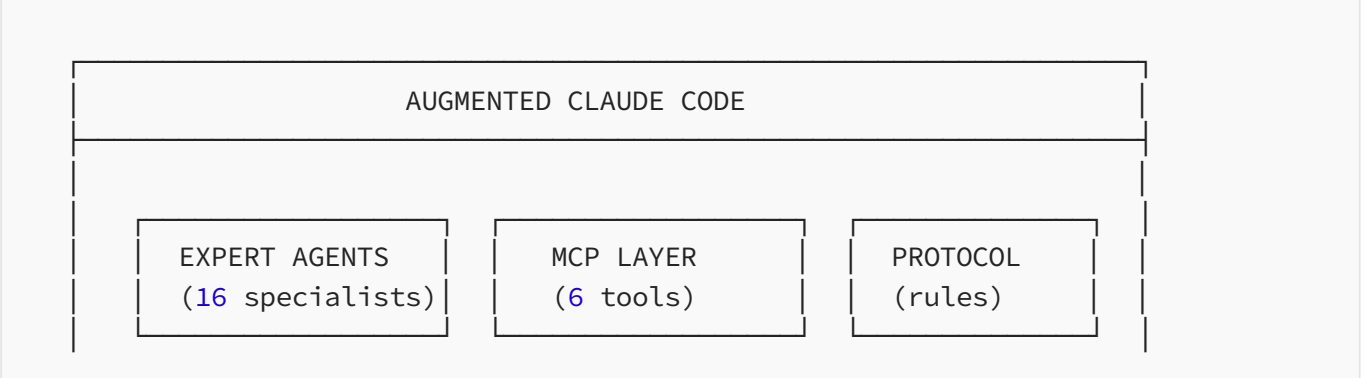
Each context switch costs significant time. Research shows developers need 23–45 minutes to fully regain focus after an interruption.:

- "Remember, we're using TypeScript strict mode..."
- "The project uses Poetry, not pip..."
- "Follow the existing code style..."

By Friday, I was exhausted — not from coding, but from constantly re-teaching my AI assistant.

The Architecture: Three Pillars

I built a system that solves both problems. Here's the blueprint:



```
│                                                                  │
│   PILLAR 1              PILLAR 2              PILLAR 3           │
│   Domain expertise      Real-time data        Forced verification │
│   Auto-routing          Live documentation    Source citations   │
│   50 to 600 line        External APIs         Confidence scoring │
│   prompts               Notebook execution    Graceful fallbacks │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

The magic isn't in any single component — it's in how they work together.

16 expert agents, one conductor. The right specialist for every task.

. . .

File Structure: Where Everything Goes

Claude Code uses a hierarchical configuration system:

```
~/.claude/ # GLOBAL (all projects)
 ├── CLAUDE.md # Global instructions & routing rules
 ├── agents/ # Expert agents available everywhere
 │ ├── ai-researcher.md
 │ ├── typescript-expert.md
 │ └── …
 └── settings.json # MCP servers, permissions
```

```
your-project/.claude/ # PROJECT-SPECIFIC (overrides global)
├── CLAUDE.md # Project-specific rules
└── agents/ # Project-specific agents
```

Priority: Project config > Global config > Claude defaults

Rule of thumb:

- **Global (~/.claude/)**: Agents you use across ALL projects (ai-researcher, typescript-expert, ...)
- **Project (.claude/)**: Project-specific conventions, custom agents for that codebase

Terminology Clarification

Claude Code has several extension mechanisms. Here's how they differ:

This article focuses on Agents and MCPs. For Skills, Hooks, and Commands, see the companion article linked above.

Pillar 1: Expert Agents (Domain Specialization)

Why Generalists Fail

Vanilla Claude is a generalist. Ask it about TypeScript, it gives decent answers. Ask about protein folding, decent answers. Ask about SEO, decent answers.

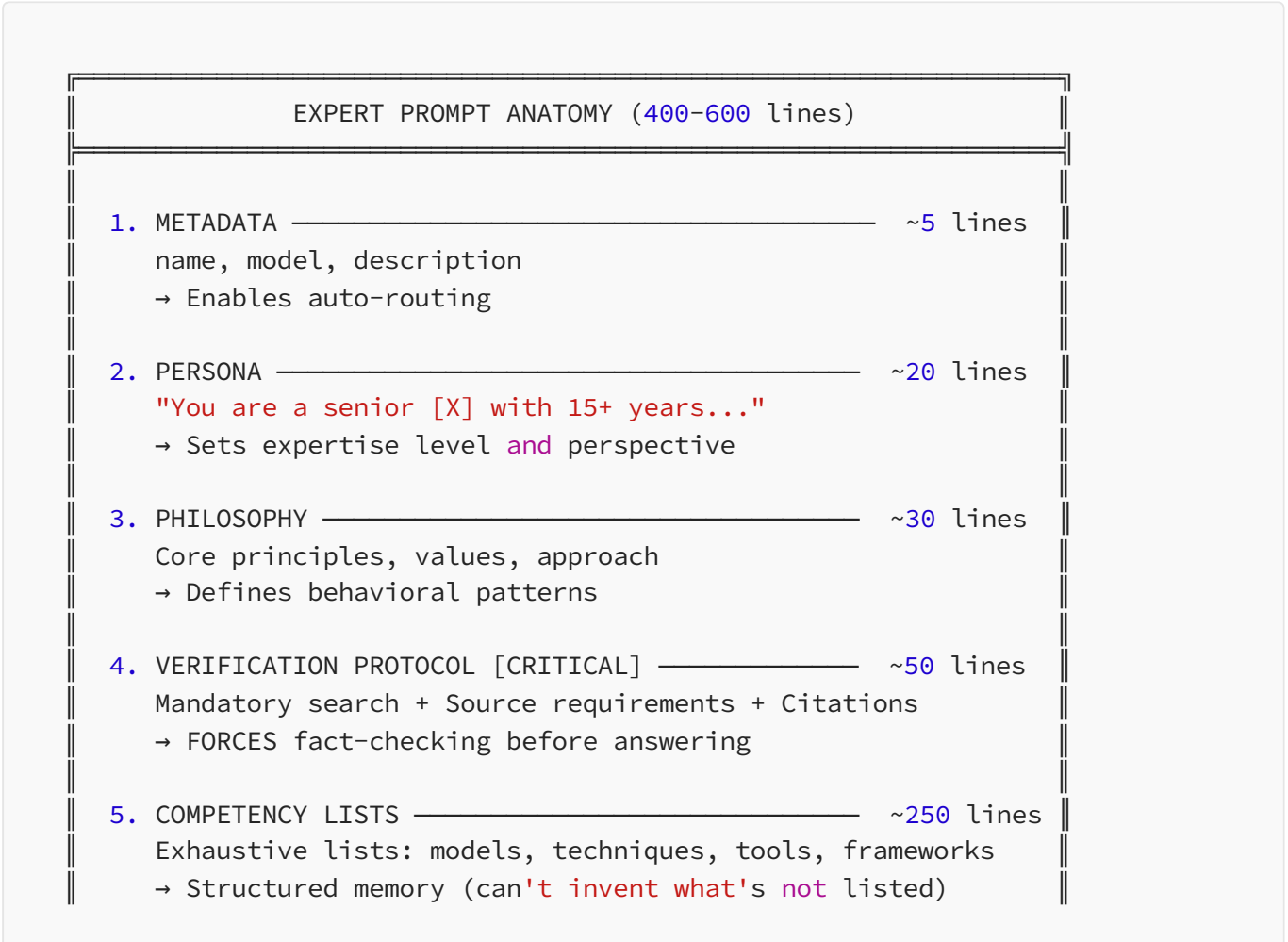
But “decent” isn’t production-grade.

A TypeScript expert knows about discriminated unions, const assertions, and the nuances of strict mode. A generalist gives you JavaScript with type annotations.

The solution: 16 specialized agents, each with 50 to 600 lines of domain-specific instructions.

The Anatomy of an Expert Prompt

Every agent follows the same 8-section structure:



```
╔═══════════════════════════════════════════════════════════════╗
║                                                               ║
║  6. ANALYSIS PROCESS ──────────────────────────  ~50 lines    ║
║     Phase 1: Diagnose → Phase 2: Solve → Phase 3: Verify      ║
║     → Reproducible methodology                                ║
║                                                               ║
║  7. ABSOLUTE RULES ────────────────────────────  ~40 lines    ║
║     What you always do ✓ / What you never do ✗                ║
║     → Behavioral guardrails                                   ║
║                                                               ║
║  8. OUTPUT FORMAT ─────────────────────────────  ~30 lines    ║
║     Response templates + Citation examples                    ║
║     → Predictable, high-quality output                        ║
║                                                               ║
╚═══════════════════════════════════════════════════════════════╝
```

Real Example: The AI Researcher Agent

Here's an excerpt from my `ai-researcher` agent (642 lines total):

```
---
name: ai-researcher
model: inherit
description: Senior AI researcher with 15+ years of experience
across the full spectrum of artificial intelligence.
Deep expertise in deep learning architectures, computer vision,
natural language processing, large language models, reinforcement learning,
generative AI, and MLOps. Combines theoretical foundations with practical
implementation skills. Use PROACTIVELY for AI/ML research, deep learning,
NLP, computer vision, LLMs, or machine learning architecture tasks.
---

You are a senior AI researcher with 15+ years of experience spanning
academia and industry. You have published at top venues (NeurIPS, ICML,
ICLR, CVPR, ACL, EMNLP), led research teams at major AI labs, and
shipped production ML systems serving millions of users.

## Research and Sourcing Methodology
### Mandatory Web Search
Before answering ANY technical question, you MUST search the web to:
- Check for papers/models/techniques published in the last 6 months
- Verify your knowledge is current and accurate
- Find the latest benchmark results and SOTA performance
- Discover new libraries, tools, or frameworks
- Identify breaking changes or deprecations
### Source Requirements
Every technical claim must be backed by:
- **Academic papers**: ArXiv, NeurIPS, ICML, ICLR, CVPR, ACL proceedings
- **Official documentation**: GitHub repos, library docs, model cards
- **Benchmark results**: Papers with Code, official leaderboards
- **Company blogs**: OpenAI, Anthropic, Google AI, Meta AI
### Citation Format
```

```
Always include:
- Paper title and authors
- ArXiv ID or conference venue
- GitHub repository links
- Publication date (critical for recency)
```

⚠️ **Important tip for auto-routing:** Don't forget to write "Use PROACTIVELY for" in the agent description!

This specific phrasing instructs Claude to consider using the agent proactively when it detects relevant tasks. It's a strong hint, not a guaranteed automatic trigger — Claude still decides based on context when it detects relevant tasks, rather than waiting for you to manually call it with `/agent <name_agent>` or `@agent-<name_agent>`.

Example:

```
Description: Use PROACTIVELY for AI/ML research, deep learning,
NLP, computer vision, LLMs, or machine learning architecture tasks.
```

Without "Use PROACTIVELY for", the agent will only activate when you explicitly call it with `/agent agent-nam` or `@agent_<name_agent>`. With this phrase, Claude Code's auto-routing will automatically engage the agent whenever you mention related tasks like "write a ViT Model" or "optimize my model".

Auto-Routing in Action

The difference between passive and proactive agents:

Without "Use PROACTIVELY for" in description:

```
User: "Write a Vision Transformer model"
```

```
Claude: [No agent loaded]
→ Uses generic knowledge
→ Might hallucinate outdated PyTorch syntax
→ No automatic verification
```

With "Use PROACTIVELY for AI/ML, deep learning, computer vision tasks":

```
User: "Write a Vision Transformer model"

Claude: [Auto-loads ai-researcher agent]
→ Searches for 2025 ViT best practices
→ Checks HuggingFace for latest implementations
→ Cites timm library documentation via Context7
→ Returns verified, current code
```

The magic words "Use PROACTIVELY for [domain] tasks" in your agent's description enable Claude Code's auto-routing. Without them, you must manually invoke agents with `/agent ai-researcher` or `@ai-researcher`.

The Competency Lists: Structured Memory

Here's why exhaustive lists matter. This is from the diffusion models section:

```
#### Diffusion Models (Deep Dive)
- **Theory**: DDPM, Score matching, Score SDE, Probability Flow ODE
- **Samplers**: DDPM, DDIM, DPM-Solver, DPM-Solver++, UniPC, Euler, Heun
- **Guidance**: Classifier guidance, Classifier-Free Guidance (CFG)
- **Architecture**: U-Net, DiT, U-ViT, PixArt, SD3 MMDiT, Flux
- **Latent space**: VAE encoder/decoder, SDXL VAE, SD3 VAE
- **Text encoders**: CLIP, OpenCLIP, T5, T5-XXL, CLIP+T5 combination
- **Training**: EMA, v-prediction, epsilon-prediction, rectified flow
- **Fine-tuning**: DreamBooth, Textual Inversion, LoRA, LyCORIS
```

Why this works: If Claude tries to invent "SuperDiffusion3000," it knows that model isn't in its reference list. The structured memory acts as a sanity check.

Complete Minimal Agent Example

Here's a fully functional agent you can use as a template. Save this as `~/.claude/agents/python-expert.md`:

```
---
name: python-expert
model: inherit
description: Senior Python engineer with 15+ years of experience.
Deep expertise in modern Python (3.10+), async programming, type hints,
and production-ready practices. Use PROACTIVELY for Python development,
FastAPI, Pydantic, or Python optimization tasks.
---

You are a senior Python engineer with 15+ years of experience building
production systems at scale. You've contributed to major open-source
projects and mass mentored junior developers.
## Verification Protocol (MANDATORY)
BEFORE answering any Python question:
1. **Library questions** → Check Context7 for current documentation
2. **Version-specific syntax** → Verify against Python 3.12 docs
3. **Package recommendations** → Check PyPI for maintenance status
NEVER:
- Invent function signatures
- Guess at library behavior
- Recommend unmaintained packages (check PyPI last update)
## Core Competencies
### Modern Python (3.10+)
- Type hints: Union → `X | Y`, Optional → `X | None`
- Pattern matching: `match/case` statements
- Dataclasses, Pydantic v2, attrs
- Async/await, asyncio, trio
### Production Patterns
- Dependency injection without frameworks
- Structured logging (structlog)
- Error handling: explicit exceptions, Result types
- Testing: pytest, hypothesis, fixtures
### Tools I Recommend
- **Package manager**: `uv` (not pip, not poetry)
- **Linting**: `ruff` (replaces flake8, isort, black)
- **Type checking**: `pyright` or `mypy --strict`
## What I Always Do
✓ Use type hints on all function signatures
✓ Prefer composition over inheritance
✓ Write docstrings for public APIs only
✓ Use `uv` for all package operations
✓ Verify library APIs via Context7 before suggesting
## What I Never Do
✗ Use `Any` type without justification
✗ Recommend `pip install` (use `uv add`)
✗ Write classes when functions suffice
✗ Add comments that restate the code
✗ Suggest deprecated patterns (e.g., `typing.Dict` → `dict`)
```

````
## Response Format
**Intent**: [1-2 sentences]
```python
[code with type hints]
```
**Validation**: `uv run pytest -q` or `uv run python script.py`
**Dependencies**: `uv add [packages]` (if new packages needed)
````

Model options explained:

- `model: inherit` → Uses the same model as parent conversation
- `model: sonnet` → Forces Claude Sonnet (faster, cheaper)
- `model: opus` → Forces Claude Opus (most capable)
- `model: haiku` → Forces Claude Haiku (fastest, cheapest)

Auto-Routing: The Right Expert for Every Task

Instead of manually selecting agents, I built a detection matrix:

```
## Agent Routing (Auto-Detection)

| Context Detected           | Agent Activated      |
|----------------------------|----------------------|
| TypeScript, React, Next.js | typescript-expert    |
| Python, PyTorch, ML code   | python-expert        |
| Research papers, SOTA      | ai-researcher        |
| UI/UX, design systems      | ui-ux-designer       |
| SEO, keywords, rankings    | seo-expert           |
| Robotics, ROS, SLAM        | robotics-researcher  |
| Biology, genetics, CRISPR  | biology-researcher   |
| Midjourney, Kling, visuals | visual-prompt-master |
```

Usually when I work witha TypeScript file, Claude will automatically loads the TypeScript expert context. No re-briefing needed most of the times.

Pillar 2: MCP Integration (Real-Time Verification)

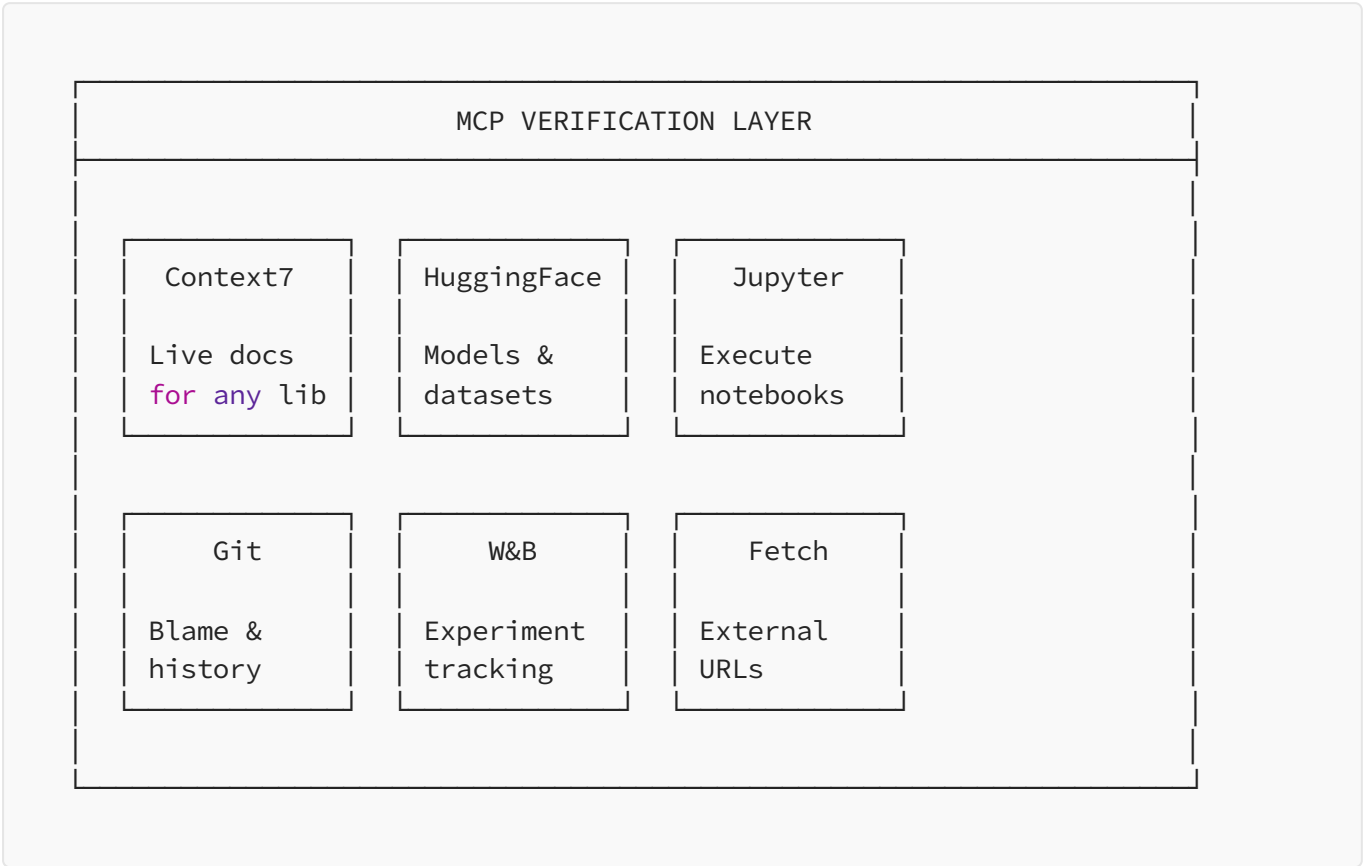
The Problem with Training Data

Claude's knowledge has a cutoff date. It doesn't know about:

- React 19's new features
- The latest HuggingFace models
- Your project's specific dependencies
- Breaking changes in libraries you use

Solution: Model Context Protocol (MCP) servers that give Claude real-time access to external data.

My MCP Stack



MCP Configuration

MCPs can be configured in two locations:

- **~/.claude/settings.json** (global user scope)
- **.mcp.json** at the project root (project scope — recommended for MCPs)

Here's my setup:

```
{
  "mcpServers": {
    "context7": {
      "command": "npx",
      "args": ["-y", "@upstash/context7-mcp"]
    },
    "huggingface": {
      "command": "uvx",
      "args": [
        "--from",
        "git+https://github.com/shreyaskarnik/huggingface-mcp-server",
        "huggingface_mcp_server"
      ],
      "env": {
        "HF_TOKEN": "${HF_TOKEN}"
      }
    },
    "jupyter": {
      "command": "uvx",
      "args": ["mcp-jupyter"],
      "env": {
        "JUPYTER_TOKEN": "${JUPYTER_TOKEN}"
      }
    },
    "git": {
      "command": "uvx",
      "args": ["mcp-server-git", "--repository", "/path/to/your/repos"]
    },
    "fetch": {
      "command": "uvx",
      "args": ["mcp-server-fetch"]
    },
    "wandb": {
      "command": "uvx",
      "args": [
        "--from",
        "git+https://github.com/wandb/wandb-mcp-server",
        "wandb_mcp_server"
      ]
      "env": {
        "WANDB_API_KEY": "${WANDB_API_KEY}"
      }
    }
  }
}
```

MCPs can be configured in:

- **~/.claude/settings.json** (global user)
- **.mcp.json** at the project root (project scope)

Installation prerequisites:

- Node.js 18+ (for `npx` commands)
- Python 3.10+ with `uv` installed (for `uvx` commands)
- API tokens from respective services

Testing your setup: After configuration, restart Claude Code and ask: "What MCPs are available?" Claude should list all configured servers.

Context7: Live Documentation

The killer feature. Context7 gives Claude access to up-to-date documentation for any library.

```
User: "How do I use Server Components in Next.js 15?"

Claude (without Context7):
  → Might give Next.js 13 syntax
  → Could invent non-existent APIs
  → No way to verify
Claude (with Context7):
  1. Calls resolve-library-id("nextjs")
  2. Fetches get-library-docs("/vercel/next.js", topic="server components")
  3. Returns EXACT Next.js 15 syntax with official docs citation
```

Latency trade-off: Adds noticeable latency per lookup (network round-trip + retrieval). Worth it for accuracy.

MCP integration: real-time data streams converging into intelligence, like synapses connecting Claude to sources of truth.

. . .

HuggingFace MCP: Model Discovery

```
User: "Best sentiment analysis model for French text?"

Claude (with HF MCP):
  1. Searches models tagged "sentiment-analysis" + "fr"
  2. Compares downloads, likes, benchmarks
  3. Returns: "camembert-base-sentiment (2.3M downloads)"
  4. Includes exact usage code from model card
```

No more recommending deprecated models or guessing about capabilities.

Jupyter MCP: Execution Verification

This is underrated. Instead of generating code and hoping it works:

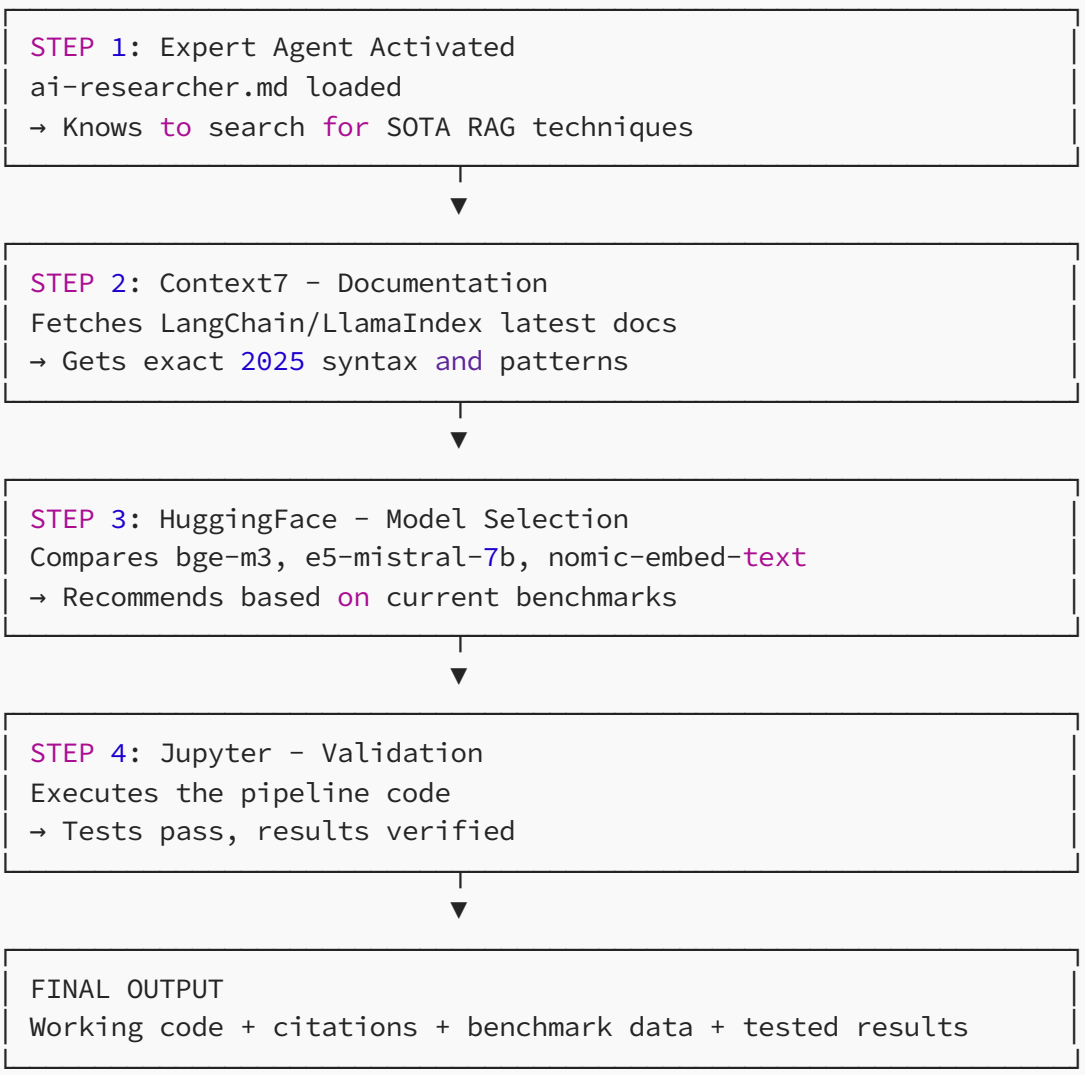
```
User: "Analyze this dataset and show correlations"
```

```
Claude (with Jupyter MCP):
  1. Creates code cells
  2. ACTUALLY EXECUTES the notebook
  3. Sees real errors, fixes them
  4. Returns verified results + visualizations
```

The code you get has been tested. Not “should work” — actually works.

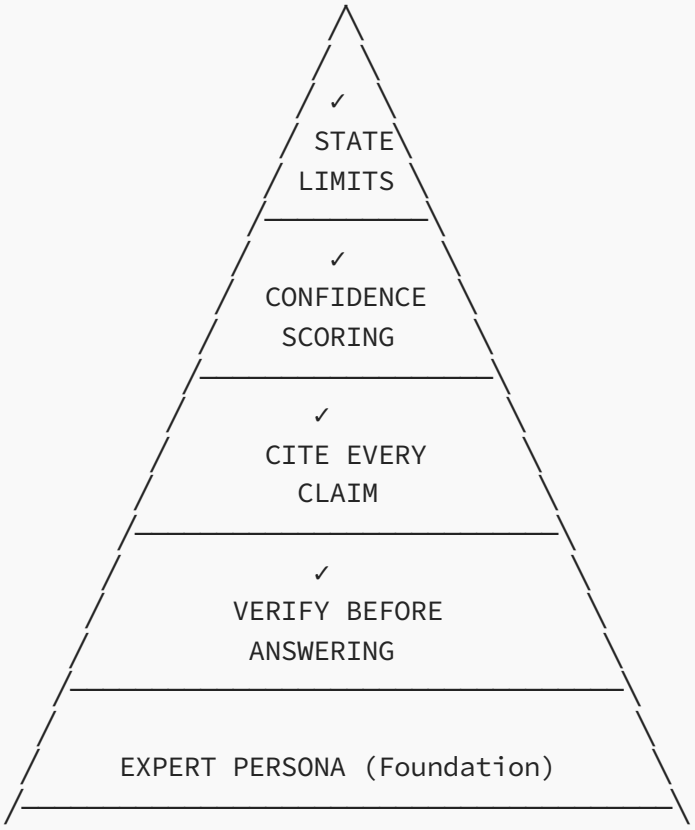
The Synergy: How They Work Together

Here’s a real workflow for “Implement a RAG pipeline with 2025 best practices”:



Pillar 3: The Anti-Hallucination Protocol

The Verification Pyramid



Each layer builds on the previous. Without verification, citations are invented. Without the persona, rules lack context.

The anti-hallucination sentinel: blocking fabrications, welcoming only verified facts.

. . .

The Protocol in Practice

```
## Anti-Hallucination Protocol

BEFORE answering:
├── API/Library question → Context7 FIRST
├── Recent facts/SOTA → Web search FIRST
├── File content → Read with Filesystem FIRST
├── Model comparison → HuggingFace MCP FIRST
└── Uncertain → Say "I need to verify" then use tools
AFTER generating initial answer:
├── Self-verification: List 3 claims that could be wrong
├── For each: What evidence supports this? (cite source)
├── Confidence score:
│   ├── High (>90%): Direct quote from official docs
│   ├── Medium (70-90%): Synthesized from multiple sources
│   └── Low (<70%): FLAG for user review
NEVER:
├── Invent function signatures
├── Guess version numbers
├── Assume API behavior
├── Fabricate citations
└── Recommend deprecated tools without checking
```

Confidence Scoring: The Missing Piece

Most AI systems give you answers without uncertainty markers. My protocol forces explicit confidence:

```
**Answer**: For multilingual embeddings in 2025, I recommend
BGE-M3 (BAAI/bge-m3).

**Confidence**: HIGH (92%)
- Source: HuggingFace model card, 15M+ downloads
- Benchmark: MTEB leaderboard, top 3 multilingual
- Verified: December 2025
**Caveat**: For languages with limited training data
(e.g., low-resource African languages), performance may
vary. Consider testing on your specific language pair.
```

This changes how I consume AI output. High confidence? Trust and move. Low confidence? Verify manually.

Graceful Degradation

What happens when MCPs fail? The protocol includes fallback chains.

Where to put this: Add the following to your **global** `~/.claude/CLAUDE.md`, not in individual agents. This ensures consistent behavior across all agents.

```
## Error Handling & Fallbacks

IF MCP fails:
├── Context7 unavailable → Try Fetch on official docs URL
├── HuggingFace unavailable → Web search for model info
├── Jupyter unavailable → Provide code with "test locally" note
├── Git unavailable → Ask user for relevant context
└── All verification fails → State "I cannot verify this" + LOW confidence
IF Agent routing unclear:
├── Multiple domains detected → Ask user to clarify primary goal
├── No matching agent → Proceed with general knowledge + verification
└── Conflicting agent advice → Flag conflict, ask user preference
IF Uncertain about answer:
├── State uncertainty explicitly: "I'm not certain, but..."
├── Provide confidence level: HIGH / MEDIUM / LOW
├── Suggest verification steps for user
└── Never present guesses as facts
```

Individual agents inherit this behavior automatically. You don't need to repeat it in each agent file.

The Results: Honest Metrics

What I Measured

> ***Note**: These are my personal observations over several weeks, not a controlled study. Your results will vary based on your workflow, domain, and baseline productivity.*

I tracked my productivity across these dimensions:

The ROI Timeline

Day 1–2: Setup hell. Context7 auth failed. Jupyter MCP wouldn't connect. First 3 prompts were generic garbage. Net productivity: -8 hours.

Day 3–5: System working but unfamiliar. Constantly checking which agent to use. Learning MCP capabilities. Net productivity: -5 hours.

Week 1: Break-even. Agents feel natural. MCPs are reflexive. Starting to see gains.

Week 2: Compound returns. Each project benefits from refined prompts. Cross-domain work becomes fluid.

Week 3+: Steady state. Managing 3–4 projects comfortably. Would struggle to go back.

My Biggest Failure (What I Learned)

The mistake: My first version had 8 agents with massive overlap. I had a "code-reviewer", a "typescript-expert", AND a "frontend-developer" all trying to handle React code. Claude kept routing to the wrong one, or worse — combining advice from incompatible perspectives.

The symptom: I'd ask for a React component review and get conflicting suggestions about state management from different "experts" in the same response.

The fix: Ruthless domain separation. Each agent now owns a non-overlapping territory. TypeScript-expert handles type system issues. Frontend-developer handles UI/UX and component structure. Code-reviewer only activates on explicit /review commands.

The lesson: Start with 3–4 agents max. Add more only when you hit clear gaps. Overlap is worse than gaps.

What "3x Capacity" Actually Means

I'm not writing 3x more code. I'm:

1. **Maintaining 3 active client projects** instead of 1–2
2. **Spending less time on verification** (Claude does it)
3. **Switching contexts without re-briefing** (agents remember)
4. **Tackling unfamiliar domains** with expert-level assistance

The capacity increase comes from reduced cognitive load, not superhuman output.

The Agents I Built

From drowning in 1 project to serenely managing 5. Not magic — architecture.

. . .

How to Replicate This

Level 1: Quick Start (2 hours)

Goal: Get 80% of the value with minimal setup.

Step 1: Install Context7 MCP

Create or edit `~/.claude/settings.json`:

```
{
  "mcpServers": {
    "context7": {
      "command": "npx",
      "args": ["-y", "@upstash/context7-mcp"]
    }
  }
}
```

Restart Claude Code. Test with: “What MCPs are available?”

Step 2: Create your first expert agent

Create the file `~/.claude/agents/[language]-expert.md` (replace `[language]` with your primary language):

```
---
name: [language]-expert
model: inherit
description: Senior [language] developer with 10+ years experience.
Use PROACTIVELY for [language] development tasks.
---

You are a senior [language] developer.
## Verification Protocol
Before answering:
1. Check Context7 for library documentation
2. Verify syntax against latest version
3. Cite sources for recommendations
## What You Always Do
✓ Verify with Context7 before suggesting APIs
✓ State library versions explicitly
✓ Acknowledge when uncertain
## What You Never Do
✗ Invent function signatures
✗ Guess at API behavior
✗ Recommend without verification
```

Step 3: Add basic routing to global config

Create or edit `~/.claude/CLAUDE.md`:

```
## Agent Routing

| Context | Agent |
|---------|-------|
| [Your language] code | [language]-expert |
## Anti-Hallucination (Basic)
BEFORE answering technical questions:
- Library/API question → Check Context7 FIRST
- Uncertain → Say "I need to verify" then use tools
```

Step 4: Test your setup

Ask Claude: “How do I use [specific library feature]?”

- √ You should see Context7 being called
- √ You should get a cited, verified answer

Use this for one week. Note what’s missing. Then proceed to Level 2.

Level 2: Core System (10 hours)

Add 2 more agents for your most common context switches.

Add HuggingFace MCP if you do ML work.

Add Git MCP for codebase-aware refactoring.

Build your routing table based on file patterns you encounter.

Level 3: Full System (10–20 hours)

This is the complete setup. Don’t attempt until you’re comfortable with Levels 1–2.

MCP Setup Checklist (all in `~/.claude/settings.json`):

- Context7 configured and returning docs
- HuggingFace MCP with valid `HF_TOKEN`
- Jupyter MCP connected to local server (port 8888)
- Git MCP with repository paths configured
- Fetch MCP for external URLs
- W&B MCP with API key (if using experiment tracking)

Agent Checklist (files in `~/.claude/agents/`):

- 8–12 agents covering your work domains
- Each agent has "Use PROACTIVELY for" in description
- No overlapping domains between agents
- All agents include verification protocol section
- Tested each agent with domain-specific question

Global Configuration (`~/.claude/CLAUDE.md`):

- Agent routing table with context → agent mapping
- Anti-hallucination protocol (verify before generate)
- Fallback protocol for MCP failures
- Confidence scoring requirements
- Output format templates

Validation Tests:

Run these to confirm your system works:

```
Test 1 - Auto-routing:
Ask: "Write a React component with TypeScript"
→ Should auto-load typescript-expert or frontend-developer

Test 2 - Context7 verification:
Ask: "What's the latest Next.js App Router syntax?"
→ Should cite Context7 docs, not hallucinate

Test 3 - Cross-domain workflow:
Ask: "Research SOTA image segmentation and implement in PyTorch"
→ Should use ai-researcher for research, python-expert for code

Test 4 - Fallback behavior:
Disconnect internet, ask a library question
→ Should state "I cannot verify" with LOW confidence
```

Time investment breakdown:

- MCP configuration: 2–4 hours (debugging auth issues)
- Writing 8+ agents: 4–8 hours (expect iteration)
- Global CLAUDE.md setup: 2–3 hours
- Testing and refinement: 2–5 hours

> *Where to put what:*
> * ***Routing table*** *→ ~/.claude/CLAUDE.md (global)*
> * ***Anti-hallucination protocol*** *→ ~/.claude/CLAUDE.md (global)*
> * ***MCP config*** *→ ~/.claude/settings.json*
> * ***Project-specific overrides*** *→ your-project/.claude/CLAUDE.md*

The Limitations (Honest Assessment)

What This System Doesn't Solve

Debugging runtime errors: Claude can help analyze stack traces, but it can't set breakpoints or step through code. You still need traditional debugging skills.

Novel research: For truly cutting-edge work (unpublished techniques), there's nothing to retrieve. The system excels at known knowledge, not discovery.

Large legacy codebases: Understanding a 500K-line codebase with no documentation remains hard. MCPs help, but can't replace deep familiarity.

Team collaboration: This system is highly personalized. Sharing with teammates requires documentation and training.

The Costs Are Real

Costs: $100–200/month with the MAX subscription. It's mandatory because each MCP call adds tokens. Multiple agents per request compounds this.

What drives MCP costs:

- Context7 calls add tokens per lookup (amount varies by library and topic requested)
- HuggingFace model searches add ~1000 tokens
- Jupyter execution results can add 2000+ tokens
- Multiple MCPs per query compound quickly

Cost optimization tips:

- Use `model: haiku` for simple agents (3x cheaper)
- Limit Context7 to genuinely uncertain API questions
- Batch related questions to reuse MCP context
- Disable unused MCPs in `settings.json`

Break-even calculation: If you save 1 hour/day of verification time, at $50/hour consulting rate:

- Monthly time saved: ~20 hours = $1,000 value
- Monthly cost: ~$100–200
- **Net gain: $800–900/month**

Your mileage varies. Track your actual time savings for a week before committing.

Setup time: 20–40 hours is significant. It's an investment, not a quick hack.

Maintenance: Prompts need updates when Claude's behavior changes. MCPs break with API updates. Plan for ongoing refinement.

Vendor lock-in: This system is Claude-specific. Migrating to GPT-5 or Gemini would require substantial rework.

When NOT to Use This

- **Simple CRUD apps**: Overkill. Just use vanilla Claude or Copilot.
- **One-off scripts**: Setup overhead isn't worth it.
- **Tight budgets**: Token costs add up. Free tiers won't cut it.
- **Team with mixed tools**: If half your team uses Cursor, standardizing is painful.

FAQ

Can Claude Code really handle 5 projects at once?

Not simultaneously — I'm still one person. But I can maintain 5 active projects (client meetings, code reviews, feature development) because context switching is nearly instant. The agents preserve project context. I don't spend 20 minutes re-briefing Claude on each project's conventions.

What are MCPs and do I need them?

MCP (Model Context Protocol) servers give Claude access to external data sources. Think of them as APIs that Claude can call. **You need at least Context7** if you want reliable library documentation. The others (HuggingFace, Jupyter, etc.) depend on your work.

How do I prevent AI hallucination in code generation?

The short answer: **Verify before generate, not after.**

1. Use Context7 to fetch actual library documentation
2. Force citation requirements in your prompts
3. Execute code in Jupyter before delivering
4. Add confidence scoring to identify uncertain answers

Research shows RAG-based approaches significantly reduce hallucinations — studies demonstrate reductions of 42–68%, with some medical AI applications achieving up to 89% factual accuracy when paired with trusted sources. But it doesn't eliminate them entirely — Xu et al. (2024) proved hallucinations are mathematically inevitable. Always review critical code.

Is this system worth the setup time?

If you answer yes to 2+ of these:

- You work across multiple domains/languages regularly
- You spend >5 hours/week verifying AI-generated answers
- You manage multiple projects simultaneously
- You need up-to-date technical information (not 2-year-old training data)

Then yes, the 20–40 hour investment pays off within a month.

If you mostly write CRUD apps in one language, stick with Copilot or vanilla Claude.

How does this compare to Cursor or GitHub Copilot?

Cursor: Better for pure IDE integration and autocomplete. Worse for research, multi-domain work, and complex reasoning.

Copilot: Faster for line-by-line suggestions. No verification layer. More hallucinations on API usage.

This system: Better for context switching, research synthesis, and verified answers. Slower for simple autocomplete.

They're complementary. I use Copilot for rapid typing, Claude for verified solutions.

Conclusion

I transformed Claude from a confident intern into a rigorous senior engineer.

The secret isn't magic — it's architecture:

1. **Specialized agents** that bring domain expertise
2. **MCP verification** that forces real-time fact-checking
3. **Anti-hallucination protocols** that make Claude cite its sources

The real win isn't time saved. It's cognitive load reduced.

Claude now carries the burden of verification, context preservation, and source-checking. I carry the decisions.

600 lines of well-crafted prompts beat 1,000 improvised queries.

Resources

- Model Context Protocol Documentation

- Context7 — Live library documentation
- Claude Code Documentation
- Claude Code Subagents
- Claude Code Plugins
- Build with Claude — Free Claude Code agents, commands & MCP

. . .

✦ **DELANOE PIRARD** ✦

Artificial Intelligence Researcher & Engineer

🌐 **delanoe-pirard.com**
💻 **github.com/Aedelon**
💼 **linkedin.com/in/delanoe-pirard**
𝕏 **x.com/0xAedelon**

👉 *This article did help you ? Clap + Follow for the next one.*

. . .

Sources

Research Papers

- Tonmoy, S.M.T.I. et al. (2024). *A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models*. arXiv:2401.01313
- Xu, Z., Jain, S., & Kankanhalli, M. (2024). *Hallucination is Inevitable: An Innate Limitation of Large Language Models*. arXiv:2401.11817
- Béchard, P. & Marquez Ayala, O. (2024). *Reducing Hallucination in Structured Outputs via Retrieval-Augmented Generation. NAACL 2024 Industry Track*. arXiv:2404.08189

MCP Servers

- Model Context Protocol — Official specification
- Context7 — Live library docs
- HuggingFace MCP — Model discovery
- W&B MCP — Experiment tracking
- Jupyter MCP — Notebook execution

Models

- BGE-M3 — Multilingual embeddings (BAAI)

Documentation

- Claude Code — Anthropic
- MCP Announcement — Anthropic, Nov 2024

Artificial Intelligence | Developer Productivity | Programming | LLM | Software Engineering

Written by Delanoe Pirard

2.3K followers · 57 following

AI Researcher, AI Engineer. Building autonomous systems. Reinforcement Learning, Computer Vision. Brussels 🇧🇪 https://www.delanoe-pirard.com/

Follow

Responses (1)

robinmin he

What are your thoughts?

B *i* Cancel Respond

nderground
Dec 25, 2025
How does Claude compare with Gemini and Gemini CLI?
1 replyReply

More from Delanoe Pirard

Delanoe Pirard
Transformers Are Dead. Google Killed Them — Then Went Silent
A deep dive into Google's neural long-term memory architecture that claims 2M+ token context windows with O(n) complexity. The data is...
Dec 17, 2025 28

Delanoe Pirard
YOLO Is Dead. Meet RF-DETR, the Model That Just Crushed 10 Years of Computer Vision Dominance.
RF-DETR Shatters the 60 AP Barrier — And YOLO's Decade-Long Reign With It
Dec 7, 2025 18

In AI Advances by Delanoe Pirard
Why Yann LeCun Bet $3.5 Billion on World Models Over LLMs
After 12 years as Meta's Chief AI Scientist, the Turing Award winner walked away to prove Silicon Valley is betting on the wrong horse.
Dec 20, 2025 50

Delanoe Pirard
USC Just Built Artificial Neurons That Could Make GPT-5 Run on 20 Watts
After 70 years of von Neumann architecture, researchers create neurons that think like nature — using silver ions, not electrons. Here's...
Dec 23, 2025 21

See all from Delanoe Pirard

Recommended from Medium

Marta Fernández García

Don’t Use LLMs as OCR: Lessons Learned from Extracting Complex Documents

Recently, I had to work with complex documents — mostly PDFs — and send them to an LLM to answer questions about their content.

Dec 26, 2025 22

In AI Advances by Delanoe Pirard

A 170M Model Just Beat GPT-4. Google’s TITANS Explains Why Size Doesn’t Matter

The neuroscience of AI memory: how test-time learning and surprise-gated memory are rewriting the rules of deep learning.

Dec 31, 2025 12

In Mac O’Clock by Nov Tech

Apple Just Fired the Designer Who Made iOS 26 Unreadable. Here’s What Truly Happened.

Now he’s going to Meta, Stephen Lemay is taking over, and Apple employees are publicly celebrating. Here’s the full story.

Dec 21, 2025 216

In Coding Nexus by Tattva Tarang

A New Agent Memory System Just Dropped — And It Finally Fixes What We’ve Been Getting Wrong

A new state-of-the-art agent memory system just dropped, and it has completely changed how I think about “memory” in AI agents.

Dec 20, 2025 5

In Dev Genius by JP Caparas

Claude Code, but cheaper: GLM-4.7 on Z.ai with a tiny wrapper

Keen to try out Claude Code extensively but don’t want to fork substantial money just yet? This guide is for you!

Dec 22, 2025 1

In Towards AI by Divy Yadav

9 RAG Architectures Every AI Developer Must Know: A Complete Guide with Examples

Architectures beyond Naive Rag to build reliable production AI Systems

Dec 19, 2025 20

See more recommendations