ZEN.SOLUTIONS

The Practitioner's *Field Manual*

Build with intention. Scale with clarity.

Hans Havlik
Founder, Zen Solutions
Writer, Developer, AI/ML Systems Architect

zen-solutions.dev

Version 1.0 / May 2026 / Free to share

Preface: Why This Exists

I have helped guide teams of developers and engineers building, deploying AI/ML automations and Agentic enterprise agentic AI systems while working as a Consultant at Capgemini. We ran systems and automations which processed hundreds of thousands of incidents per month for enterprise client accounts across numerous industries such as IT, healthcare, telecom, defense, and aerospace. I also do this at home: my own knowledge base, my own agents, my own pipelines running in the background while I do the rest of my life.

I learned this work the way most working practitioners learn anything: years of doing it, building it, getting things wrong, and writing down what worked. This guide is the reference I wish someone had handed me three years ago. The principles here are what I have actually tested in environments where being wrong has consequences.

What this guide is: a calm, grounded, honest walkthrough of the AI landscape in 2026, what is actually possible right now, what is overhyped, what is dangerous, and a 90-day path to becoming a real practitioner.

What this guide is not: a prompt dump. It is not a list of "10 ChatGPT hacks" or copy-paste templates that promise to change your business overnight. The shortcut economy has saturated this space. You are reading this because you suspect there is something deeper, and there is.

This is free. There is no upsell at the end. If it lands, the best thing you can do is go to zen-solutions.dev and subscribe to the work. The 26-week Field Manual series is where this guide goes deeper, episode by episode.

> **A note on the field.** The AI practitioner space is still small. Where another writer or builder has coined a useful term, I name them. Dan Martell uses the phrase "reverse prompting" in his AI Business Cheat Sheet and the term is worth keeping. Anthropic's engineering team writes publicly about how Claude Code is meant to be used, and that writing is worth reading. Most of what follows is from my own years in the work.

Part 1: Where We Actually Are

The 2026 Landscape

Most people are using a 2026 AI model like it is a 2023 AI model. They open a chat window, type one line, get a mediocre response, and conclude that "AI is overrated." They are not wrong about their experience. They are wrong about the cause.

The frontier has moved on three axes since 2023, and almost nobody has updated their mental model:

Capability. Today's frontier models can reason across long, complex contexts that would have broken a 2023 model in the first paragraph. They can hold multi-step problems in working memory, plan, revise, and verify their own work when prompted to do so. The ceiling for what is possible in a single response has lifted by a full order of magnitude.

Context. Context windows have gone from a few thousand tokens to hundreds of thousands. This is not a marketing number. It is a structural change. A model in 2026 can hold an entire codebase, a year of meeting notes, or a small book in working memory at once. The skill of "feeding the model the right material" has become more important than the skill of "writing the perfect prompt."

Agency. The model is no longer just a text generator. It can use tools. It can read and write files, run code, navigate the web, call APIs, and orchestrate other models. The same underlying intelligence that helps you draft an email can also spend four hours autonomously researching a topic, writing the report, and dropping it in your inbox while you sleep.

None of these advances help you if you are still using the chat box like a search engine. The chasm between average users and serious practitioners is wider in 2026 than it has ever been, and it is not closing. It is widening.

"Stop using AI. Start building your harness."

The model is the engine. The harness is everything around it: context, tools, structure, discipline, governance, and judgment. The harness is what determines whether your AI feels like a slot machine or a colleague. This guide is about the harness.

Part 2: The Four Practitioner Levels

A practitioner moves through four levels. Each one is a meaningful change in what you can do, what limits you, and what kind of leverage compounds. Most users never leave Level 1. Most enthusiastic builders plateau at Level 3. Level 4 is where the actual operating leverage lives, and it is the level most of the discourse pretends does not exist.

Level	Identity	What you do	What unlocks
1	**User**	Chat	Ideation, drafting, research
2	**Builder**	Harness	Context engineering, projects, skills
3	**Operator**	Systems	Agents, pipelines, automations
4	**Strategist**	Doctrine	Restraint, security, leadership

Level 1: The User

You use a chat interface (Claude, ChatGPT, Gemini, Grok) for ideation, drafting, summarizing, and basic research. This is where 95% of users live. Your output quality is directly proportional to the quality of your prompts, and you spend most of your time fighting generic results.

What limits you here: the assumption that AI is a thing you talk to, not a thing you build with. You are missing the structural moves that compound.

The single highest-leverage upgrade at this level: stop writing prompts. Start writing context. A 200-word context block ("here is who I am, here is what we are working on, here is the standard I want, here is the audience") at the start of a project beats any prompt hack you will ever learn.

Level 2: The Builder

You have stopped treating each conversation as fresh. You use Projects in the chat app. You write CLAUDE.md files for your serious work. You install Claude Code and start working from the terminal or your IDE. You connect your AI to your filesystem, your notes, your calendar. You are building a personal harness around the model.

What limits you here: you are still operating one session at a time. Your AI helps you when you are at the keyboard. When you stop, it stops.

The single highest-leverage upgrade at this level: treat your context as a versioned artifact, not as something you re-type each session. CLAUDE.md, system prompts, skills, project READMEs. Commit them to git. Improve them like you would improve any other piece of code.

Level 3: The Operator

You have moved from sessions to pipelines. You run agents that spend hours doing research while you do something else. You have a content pipeline, a research pipeline, a capture pipeline. You have a personal knowledge base your agents can read. You think in terms of inputs, processes, and outputs, not turns of conversation.

What limits you here: complexity. Every new agent and pipeline is another thing to maintain. You will eventually have an over-engineered, fragile system you do not trust. This is where most enthusiastic operators wash out.

The single highest-leverage upgrade at this level: delete things. The mature operator runs a small number of agents and pipelines extremely well, with strong observability and clear failure modes. Most of what you built in your first six months should be retired or simplified by month nine.

Level 4: The Strategist

At Level 4, the question is no longer "how do I use AI?" but "what does this organization or this life look like with AI integrated correctly, including the decision to not use AI in certain places?" Most of the public discourse on AI stops at Level 3. This is the level where I have spent the last several years and where most of the actual value gets made or lost.

You lead agents the way you would lead a team of junior staff. You hold a doctrine. You enforce security boundaries because you understand the threat surface. You know which problems deserve AI, which deserve traditional automation, and which deserve a human, full stop. You can argue against AI being deployed somewhere it does not belong, and that is more valuable than another agent.

> **The currency at Level 4 is judgment.** The market is flooded with operators. Strategists are rare.

Part 3: Two Techniques That Compound

Most users treat the model as an answer machine. They ask a question, get a response, and judge the result. This is the lowest-leverage use of frontier AI. The practitioner uses the model differently. The two techniques below are the ones I rely on every day. Both share a principle: do not push at the model, invite the model into the work.

Analogical Translation

When you are learning a new domain, the largest cognitive cost is not the concepts themselves. It is the wall of foreign vocabulary that arrives before the concepts do. Every unfamiliar term is one more thing your brain has to hold while it is still trying to grasp the underlying idea. By the third unfamiliar word in a paragraph, you have stopped learning and started surviving.

The model can dissolve this barrier. Tell it which domain you already know well, and ask it to translate the new domain into analogies from your existing expertise. Then, after the analogy lands, ask it to introduce the actual vocabulary tied to the analogy you now understand.

Consider a nurse learning to be a software developer. Networking, databases, APIs, system architecture, these all sound abstract. They are not. They are the same kinds of patterns the nurse already understands from human anatomy, hospital workflows, and clinical handoffs. A database is a chart. An API is the handoff at change of shift. A network outage is a code blue without staff. A microservice architecture is a hospital with specialized departments that have to communicate cleanly or patient care breaks down.

The concept lands before the jargon does. When the jargon then arrives, it attaches to a concept that is already there. The learning sticks because it has somewhere to live.

I have used this technique on every domain crossover I have made: combat medic to software developer, developer to AI solutions architect, individual contributor to team lead. It is also how I onboard new team members. The model is exceptionally good at producing these translations on demand. Most users never ask for them because they assume the model is for getting answers. The model is also a translator between domains, and that is one of its highest-leverage uses.

A working template

> "I am trying to understand [new concept]. I have deep expertise in [your domain]. Before you introduce any of the technical terms or jargon, translate the concept into analogies, vocabulary, and mental models from [your domain]. After the analogy lands, then introduce the real terminology, and tie each new term back to the analogy."

You can adjust this for a single concept or for an entire field you are onboarding into. The principle is the same: anchor the unfamiliar to the familiar, then let the unfamiliar earn its own ground.

Reverse Prompting

Pull, do not push. Most users push instructions at the model and get generic results. The practitioner inverts the flow: state your goal, then ask the model to ask you questions until it has the context to produce what you actually want.

This technique has been part of how I work for years. Dan Martell calls it "reverse prompting" in his AI Business Cheat Sheet, and the term is useful enough to keep. The shape, in its simplest form:

> "I need your help with [task]. Before you help me, ask me 3 to 5 questions to make sure you have all the context you need. Ask one at a time."

That is the floor. Here is the version I actually use:

> "I need your help with [task]. Here is what you should know about me and this work: [paste your stable context block, or reference your CLAUDE.md]. Read that first. Then ask me 3 to 5 questions to fill in only what is missing for this specific task. Ask one at a time. Once I have answered, give me your best work, then immediately critique it as if a senior practitioner in this field were reviewing it."

The structure does three things at once: it loads stable context, it pulls only the marginal context needed for this task, and it forces self-critique. Most practitioners stop after the first move. The second and third are where the leverage compounds.

Both techniques (analogical translation and reverse prompting) come from the same instinct: stop trying to extract output from the model, and start working with it. The model is not a vending machine. It is a collaborator. Treat it accordingly and the work changes.

Part 4: The Five Disciplines

How the practitioner operates at any level

If the four levels are where you are, the five disciplines are how you operate at any level. These are not skills you graduate from. They are practices you return to.

1. Context Discipline

Your AI is only as good as the context you give it. Most users skip this because it feels like overhead. The senior practitioner spends 80% of the lift on context and 20% on the prompt.

In practice: every project has a CLAUDE.md or equivalent. Every recurring workflow has a system prompt that has been refined and committed to git. You version your context like code.

> **Field-tested rule.** If you find yourself writing the same kind of instruction more than twice, stop. Move it into context. Make it permanent.

2. Restraint Discipline

Knowing when not to use AI is the rarest discipline and the most valuable. Most people in the industry are AI-washing every problem they see. Some problems do not want AI. They want a database query, a deterministic script, a phone call to the right person, or nothing at all.

In practice: before deploying AI anywhere, ask "what is the simplest thing that would work?" Often it is a regex. Sometimes it is a meeting. AI is the right tool when the problem requires reasoning across unstructured inputs, judgment, or generation. It is the wrong tool when the problem is solved by exact-match logic, when consequences of failure are catastrophic and reversibility is low, or when a human is already doing this in 30 seconds.

> **Field-tested rule.** If you can write the deterministic version in less than an hour, write the deterministic version. AI is for the problems where deterministic logic does not fit.

3. Verification Discipline

The model can be wrong. The model can also be confidently, fluently, hallucinated-citation wrong. The practitioner builds verification into the workflow, not as an afterthought.

In practice: the model produces, then critiques its own work in a separate pass. Important claims get a "what would a skeptical reviewer say?" check. Cited facts get a web fetch. Code gets executed. Outputs intended for external use cross a human review gate.

> **Field-tested rule.** Never let a model's output go to a high-stakes destination (a client, a public post, a deployed system) without at least one verification pass. Speed without verification is just a fast way to be wrong.

4. Security Discipline

The agentic threat surface is real and growing. If you give an agent access to your email, your filesystem, your credentials, or your bank, you have made it a target. Almost nobody covers this honestly because it is not what sells.

In practice: principle of least privilege for every agent. Read-only where possible. Sandboxed environments for autonomous work. Audit logging on every action. Prompt-injection awareness: an agent reading a webpage can be hijacked by content in that webpage. An agent reading your emails can be instructed by an email. Treat all untrusted content as potentially adversarial.

> **Field-tested rule.** For any new agent capability, ask "if this agent were fully compromised, what is the worst it could do?" Design the answer to be tolerable.

5. Leadership Discipline

You lead an AI agent the way you lead a junior team member. You set context, you set standards, you give feedback, you correct course. You do not assume the agent will figure it out. You write down what good looks like. You do an after-action review.

In practice: every recurring agent has a "job description" (a system prompt). Every output gets reviewed against the standard. When the output is wrong, you do not just rerun, you update the context so the next run does not make the same mistake. The agent improves the way a person would improve, through coaching written into context.

> **Field-tested rule.** If your agent makes the same mistake twice, it is your fault, not the agent's. Fix the context.

Part 5: When Not to Use AI

The AI-washing pattern is everywhere. Founders pitch "AI-powered" everything because the term raises funding. Teams add an LLM to a workflow that was working fine, increase the operating cost, decrease the reliability, and call it innovation.

A practitioner respects the problem more than the tool.

Use this approach	When the problem has...
AI / LLM	Unstructured inputs that need parsing; generation requirements; reasoning across ambiguous context; high variation across instances; a human reviewing the output before consequence
Traditional automation	Structured inputs; deterministic logic ("if X then Y"); high volume with low variation; low tolerance for non-determinism
Classical ML (not LLMs)	Large labeled datasets; well-defined prediction target; fast inference at scale; latency or cost constraints that rule out LLMs
A human	High consequence and low reversibility; trust or relationship dynamics; genuine novelty (no analog in training data); ethical, legal, or fiduciary weight

These categories overlap. Most real systems blend several. The job of the strategist is to draw the boundaries with care, not to default to AI because AI is the trend.

A short list of places I have removed AI from a workflow and replaced with something simpler:

- Email triage where a sender-and-subject rule covered 90% of the cases.
- Document classification where the documents had unique IDs you could match on.
- "Smart" routing where a lookup table did the job in a millisecond.
- A research summarizer where the user was reading the summary and then reading the full document anyway.

Restraint is a senior practice. The market does not reward it. The work does.

Part 6: Security and Governance

The part nobody covers

Every cheat sheet on the internet tells you how to start using AI. Almost none tell you how to not get hurt using AI. This section is short because the topic is large, but the few points below will protect you from most of the damage I have seen in production.

The threat model in plain English

When you give an agent access to your data and tools, three things can go wrong:

1. Prompt injection. Hostile content inside something the agent reads (a webpage, an email, a PDF) contains instructions the agent treats as commands. The agent does something you never asked for.
2. Data exfiltration. The agent is tricked into sending sensitive information to an attacker, often by including it in a URL or an API call to a malicious endpoint.
3. Unintended action. The agent acts on a misunderstood instruction and does something irreversible: deletes files, sends emails, makes purchases.

The practitioner's defaults

- **Least privilege.** Every agent gets only the access it needs for the job, nothing more. A research agent does not need your email. A draft-writing agent does not need your bank.
- **Read-only by default.** Write access is granted explicitly and minimally. Destructive actions (delete, send, purchase) require human confirmation.
- **Sandboxed execution.** Code-running agents work in isolated environments. They cannot reach your real filesystem or network without explicit permission.
- **Logging and observability.** Every agent action is logged. You can replay what an agent did, in order, after the fact. If you cannot audit it, you cannot trust it.
- **Treat all external content as untrusted.** Webpages, emails, PDFs, even shared documents from colleagues, can carry prompt injection. The agent's job is to use the content, not to obey it.
- **Human-in-the-loop where consequences matter.** Approval gates before any high-stakes action. The agent prepares, the human approves.

A reasonable governance baseline

For a small team or solo operator:

1. A written list of what each of your agents is allowed to do and what they are not.
2. Audit logs you actually look at, weekly.

3. A documented incident response: "if this agent does something it should not, here is what I do."
4. A periodic review (monthly is fine) of which agents are still in use and which can be retired.

This is not the exciting part. It is the part that lets the exciting part survive contact with reality.

Part 7: The 90-Day Practitioner's Path

Most guides hand you frameworks and leave you to figure out the path. Here is the path I would walk if I were starting again.

Days 1 to 30: Master the Chat Box

- Pick one model and commit for the month. Claude, ChatGPT, or Gemini. Not all three.
- Start using Projects (or Custom GPTs, or Gems). Treat every recurring kind of work as a project.
- Write your first CLAUDE.md (or equivalent). 200 to 500 words about who you are, what you do, how you write, what you value, what your standards are. Use it in every Project.
- Practice reverse prompting on every task for two weeks. Force yourself to let the model pull context from you.
- Pick one workflow you do weekly. Use AI on it every week for the month. Refine. Notice what context made the biggest difference.

Outcome at Day 30. You have stopped using AI as a search engine. You have a stable context artifact. You have one workflow that is meaningfully better than it was a month ago.

Days 31 to 60: Build the Harness

- Install Claude Code (or Cursor, or your preferred IDE-integrated AI). Use it weekly, not daily. Get past the awkward phase.
- Set up Obsidian as your personal knowledge base. Keep it simple: daily notes, project notes, reference notes.
- Initialize a private GitHub repo for your CLAUDE.md, system prompts, and any skills or templates you build. This is the start of your harness.
- Connect your AI to your notes. Let it read your Obsidian vault when relevant.
- Pick one place where you do too much repetitive thinking. Build a starter pattern (a template, a custom command, a project) to remove the friction.

Outcome at Day 60. The AI knows what you have written, what you have decided, and what you are building. Your context is versioned. You are working from an IDE or terminal at least once a week.

Days 61 to 90: First Agent, First Pipeline

- Identify one autonomous task: something you would happily hand off to a junior employee if you had one. Research a topic. Draft weekly content. Process incoming messages.
- Write a "job description" for that agent: what it does, what good output looks like, what it must never do.
- Set up the agent with the smallest privilege footprint that works. Read-only if at all possible.
- Run it for two weeks. Review every output. Refine the job description after each session.
- After two weeks, decide: keep, simplify, or retire.

> **Outcome at Day 90.** You have a working agent you trust to do one thing well. You have learned, in your own hands, the difference between "AI helped me" and "AI did the work for me." That difference is the leverage everything else is built on.

Beyond Day 90: The 26-Week Field Manual

This Field Guide is the foundation. The 26-week Field Manual series at zen-solutions.dev takes you from this 90-day base through the full builder, operator, and strategist arcs. Each week is a long-form video plus a Substack companion plus a contribution to the open-source Starter Kit on GitHub. It is free.

Closing: The Operator's Mindset

There is a Sanskrit phrase from the Bhagavad Gita, *karmany evadhikaras te*, that translates roughly as "you have a right to the work, never to its fruits." It is one of the oldest pieces of operating doctrine in human history, and it is shockingly relevant to working with AI.

The work is yours. Showing up, choosing well, being honest about what is yours to do and what is not. The fruits are not. Whether the model works today, whether the project lands, whether the audience comes, whether the business pays off, all of that is downstream of forces you do not control.

The operator who internalizes this gets quieter and more effective. They do not chase the latest tool. They do not panic when a model release breaks something. They do not over-engineer in pursuit of the perfect setup. They show up to the work, they build their harness, they lead their agents, they exercise restraint where it is called for, and they ship.

Methods are many. Principles are few. The one who grasps the principles selects their own methods.

That is the line I keep coming back to in this work, and it is the thread that runs through every section of this guide.

This is the work I do every day, and this is the work I want to share. If it lands, come find the rest of it at zen-solutions.dev. The 26-week Field Manual series is where the real journey begins.

I am glad you read this far. That alone tells me something about you. Welcome.

— Hans

About the Author

Hans Havlik (Hamsa) is a Senior Developer and AI Solutions Architect at Capgemini, leading a 20-person team building enterprise agentic AI solutions for clients across healthcare, telecom, aerospace, and other sectors. He is a US Army veteran (combat medic, 2010 to 2020), a father, a writer in the Gaudiya Vaishnava tradition, and the technical co-founder of MindRev. He writes at Zen Solutions about the intersection of AI engineering, contemplative practice, and the discipline of building well.

Follow the work

- **Website:** zen-solutions.dev
- **Substack:** substack.com/@zensolutions
- **YouTube:** youtube.com/@zen-solutions-dev
- **GitHub:** the Zen Solutions Starter Kit (link on the website)

Writers and practitioners worth your time

- **Dan Martell,** whose AI Business Cheat Sheet names the reverse prompting move and gives it a clean operational form.
- **Dan Koe,** for the steady work he is doing in public on what one person can build with modern tools.
- **The Anthropic engineering team,** for writing publicly about how Claude Code is meant to be used.

This is version 1.0 of the Practitioner's Field Manual.
It will be revised quarterly. The current version is always available at zen-solutions.dev.