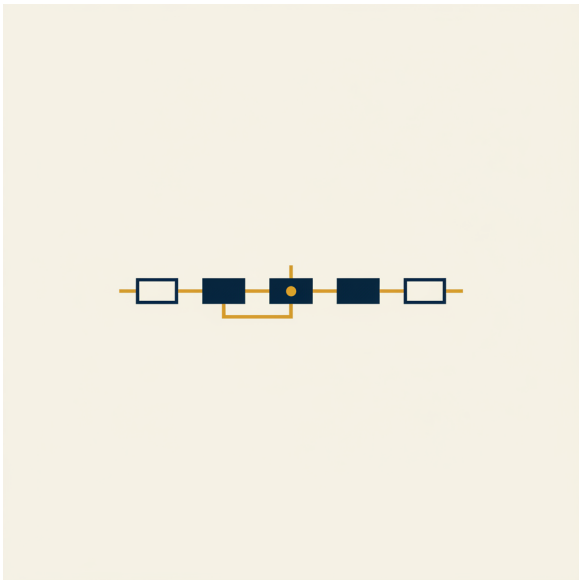
TIER 2 - PROFESSIONAL * V1.0 -- MAY 2026

MULTI-STEP TASK CHAINS

How to break real-world work into a sequence of prompts the model can actually handle. Threading context across steps, recovering from a step that goes sideways, and the pattern that turns a 5-page report into a clean back-and-forth.



BY Alex Jahn / Agent Logic

v1.0 -- May 2026 **EDITION**

Anyone hitting the ceiling of single-prompt AI -- proposals, reports, multi-section docs that come out generic and unusable **AUDIENCE**

15-20 minutes **READ TIME**

Free. Forever. **COST**

Prepared by Agent Logic / alexanderjahn79@icloud.com / theaiguywi.com

CONTENTS

What's in here

1 **Why "write me a 5-page proposal" always fails** 3

You hit a ceiling. You've gotten good at single-prompt work -- short emails, summaries, quick research. So you try the next thing up:...

2 **The chain pattern** 4

The default shape for any big task chain has four phases. Some chains have all four. Some skip one or two. None of them try to do...

3 **Threading context across steps** 5

Each step in the chain needs to know what happened in prior steps. The naive approach -- paste the entire prior conversation into every...

4 **Recovering when one step goes sideways** 6

Sometimes step 3 of a 7-step chain produces garbage. The model misunderstood the input. Got the customer mixed up with another job....

5 **Worked example -- a multi-step proposal chain** 7

A real chain, prompt by prompt. Imagine you're me drafting a proposal for the Schmidt kitchen remodel. Eight steps. Each is a small,...

6 **The pricing-cascade pattern (real production example)** 9

To make this less abstract, here's a chain in production at my own carpentry business -- running every time I generate a proposal.

7 **When NOT to chain** 10

The chain pattern is the right tool for big tasks. It's the wrong tool in three categories:

8 **Where to go from here** 11

You're now four modules into Tier 2. Two left:

SECTION 1

Why "write me a 5-page proposal" always fails

The wrong shape of task

You hit a ceiling. You've gotten good at single-prompt work -- short emails, summaries, quick research. So you try the next thing up: *"Write me*
including scope, timeline, pricing, and terms."

You hit return. The model produces 5 pages of professionally-shaped proposal. It's also, somehow, useless. The pricing is generic. The scope misses three things that matter to this customer specifically. The terms section sounds like it was written by a lawyer who's never met a customer. By the time you've edited it into something you'd actually send, you've spent the time you would have spent writing it from scratch.

People conclude: *AI doesn't*

The model isn't the problem. The shape of the request is. *"Write me*
shape of task for one prompt -- for the same reason *"build me*
conversation for one back-and-forth with a contractor. Big tasks aren't bigger versions of small tasks. They're sequences of small tasks, run with care.

This module is the operating procedure for big tasks.

What this module does

By the end of this primer:

- You'll have the **chain pat**
  small ones.
- You'll know how to **thread co**
- You'll know how to **recover** w
- You'll see a worked example: a real multi-step proposal chain, prompt by prompt.
- You'll see a real production example: my own three-tier pricing-resolution chain, running in my proposal pipeline right now.
- You'll know **when NO**

> ***An LLM is great at small tasks done well. Big tasks are a sequence of small tasks done well. The skill isn't a smarter prompt. It's better task decomposition.***

SECTION 2

The chain pattern

Outline -> draft -> refine -> integrate

The default shape for any big task chain has four phases. Some chains have all four. Some skip one or two. None of them try to do everything in one prompt.

The four phases of a task chain:

1. **Outline** -- skeleton, not content. You verify the skeleton is right before any drafting happens. *"Given the*
2. **Draft (per section)** -- gets its own prompt, conditioned on the outline plus any specific context that section needs. *"Given the*
3. **Refine** -- model is good at editing its own output if you ask it to. *"Read this*
4. **Integrate** -- The final pass that turns a stack of sections into a coherent document. *"Knit these*

That's it. That's the chain. Four prompts (or more, depending on draft count). Each one a focused, doable task instead of a monster ask.

Why each phase exists

Outline first because the model can produce a workable structure in 30 seconds, and it's cheap to fix the structure before any drafting happens. Skipping outline means catching structural problems on page 4 of a draft and re-running everything from scratch.

Draft per section because the model can produce 500 words of high-specificity output reliably, but 5,000 words of high-specificity output not at all. Sectional drafting plays to the model's strength.

Refine because the model's first draft is a draft, not a deliverable (Module 3 *Critically* covered this in detail). A targeted refinement pass on each section catches issues the *Reading A*

integration phase can't see.

Integrate because sectional drafting produces sections that can disagree with each other -- different tone, different terminology, conflicting claims. The integrate pass smooths it.

Skip any phase and you can feel the gap in the output. Run all four and the output is uncomfortably close to what you'd produce yourself, in a fraction of the time.

SECTION 3

Threading context across steps

The "here's what we have so far" pattern

Each step in the chain needs to know what happened in prior steps. The naive approach -- paste the entire prior conversation into every prompt -- works for short chains but breaks at scale (long context, attention dilution, expensive in tokens).

The better approach is to pass forward a **short sum** capturing what's been established and what each subsequent step needs to know. The model is good at producing these summaries on demand.

Worked example. You're three steps into a proposal chain:

> *"Summarize what we've established so far in this proposal chain in 4 bullet points: customer, scope, key decisions made. We'll use this summary to ground the next step."*

The model produces:

> *Customer: Schmidt residential remodel. Scope: kitchen + adjacent half-bath, ~180 sqft total. Decisions: cabinets customer-supplied; we install. Tile backsplash, two pendant lights, paint -- included. No flooring (customer hired separately). Key constraint: timing tied to their kid's birthday on July 12.*

Now your next prompt becomes:

> *"Given the summary above, draft the timeline section of the proposal. Length: 200 words. Account for the July 12 constraint and the customer-supplied cabinets."*

The model has just enough context to do the next step well, without you having to paste the entire conversation. Pattern: **explicit s**
context into every prompt.

Why this works better than "remember everything"

Two reasons. First, attention dilution -- Module 1 (*What an LLM Actually Is*) covered this. The longer the context, the thinner the model's attention spreads. A focused summary lets the model concentrate on the current step.

Second, you control what's load-bearing. The summary forces you to identify what *actually m* for the next step. The model isn't great at deciding which prior facts are still relevant -- you are. By writing the summary, you do the relevance filtering once and the next step inherits clarity.

4

Bullets in your summary, max.

If your between-steps summary is longer than 4 short bullets, you're either keeping things you don't need or trying to do too much in the next step. Compress harder.

SECTION 4

Recovering when one step goes sideways

The git-checkpoint mental model

Sometimes step 3 of a 7-step chain produces garbage. The model misunderstood the input. Got the customer mixed up with another job. Hallucinated a constraint. Whatever -- the output isn't usable.

Bad recovery: re-run the whole chain from step 1. Good recovery: catch it locally, fix the prompt or input for that one step, re-run *just* that st
3. Re-run step 3 with a tweaked prompt or clearer context.

If you've kept the prior steps' outputs as separate, named artifacts (file, doc, copy-paste somewhere), you have **checkpoi**
and re-run forward from there. You almost never need to throw out the whole chain.

How to keep checkpoints -- the simple way

Three pragmatic options for checkpointing without any tools:

1. **One Doc, named sections.** Open a Google Doc or a text file at the start of the chain. After each step, paste the output under a section heading: *"Step 1 -- Summary,"* etc. If a later step misfires, you scroll up, edit your inputs, re-run that step, paste the new output below.
2. **Numbered messages in chat.** Most chat tools let you scroll back and reference specific prior messages. If you've labeled them clearly in your prompts (*"For step 3, use the outline from step 1 above..."*), recovery is just editing one message and re-running.
3. **Separate sessions per phase.** For really big chains, run different phases in different chat sessions. The "outline" lives in one chat, the "drafts" in another, the "refinement" in a third. Cleaner separation; harder to lose track.

Whichever you pick, the principle is the same: **don't lose** insurance against re-running everything every time something misfires.

The most common recovery -- re-prompting the same step with more context

Nine times out of ten, when a step misfires, the fix is *more cont* assumed the model knew something it didn't. Add the missing piece to the step's input and re-run.

Example: the model drafted a timeline section that ignored the July 12 birthday constraint. The fix isn't a stronger model. It's:

> *"Re-do the timeline. The customer needs the kitchen done by July 12, 2026 -- that's a hard constraint, not a preference. Build the timeline backwards from that date."*

Same model, same chain, one targeted re-prompt. Garbage step -> useful step in 30 seconds.

SECTION 5

Worked example -- a multi-step proposal chain

A real chain, prompt by prompt. Imagine you're me drafting a proposal for the Schmidt kitchen remodel. Eight steps. Each is a small, focused prompt. Together they produce a proposal you'd actually send.

Step 1 -- Outline the proposal

> *"Outline a residential remodel proposal for: Schmidt family, kitchen + adjacent half-bath, ~180 sqft total. Sections needed: cover summary, scope of work, timeline, pricing, terms. Output as a numbered section list with 1-line descriptions of what each section will cover. No drafting yet, just the structure."*

Model produces a clean 5-section outline. You tweak the descriptions.

Step 2 -- Customer-context summary

> *"Based on the walkthrough notes below, write a 3-bullet customer summary I can reuse across the next steps. Capture: their priorities (what matters most to them), their constraints (deadlines, budget concerns, sensitivities), and any context that should color how I write the proposal. Walkthrou*
> *notes: [paste your notes here]"*

This is the "short summary" pattern from Section 3. You'll feed this into every subsequent step.

Step 3 -- Draft cover summary

> *"Draft the cover summary section of the proposal. Length: 120 words. Tone: warm but professional, not marketing. Use the customer-context summary above. Lead with what we're going to build. Mention the July 12 deadline."*

200-word section drafted. You read, adjust if needed.

Steps 4-6 -- Draft scope, timeline, pricing sections

One prompt per section. Each conditioned on the outline and the customer-context summary. Lengths and tone constraints stated. Output reviewed.

Step 7 -- Draft terms

Slightly different -- the terms section has more standard boilerplate, less customer-specific work. Often this is a template you reuse, not a freshly-drafted section. The chain is flexible.

Step 8 -- Integrate and polish

> *"Read the draft below as one document. Check: (1) tone consistency across sections, (2) any contradictions between sections, (3) anywhere we sound apologetic or hedging that we shouldn't, (4) any place we said the same thing twice. Suggest specific edits, don't rewrite. Keep the structure as-is.*
> *Draft: [pas*

Final pass. Apply the edits you agree with. Send.

8

Steps in a real proposal chain.

That sounds like a lot. It's not -- each step is a 30-second to 2-minute interaction. The whole chain runs in 20-30 minutes including reviewing each step. Compare to manually drafting a proposal from scratch (90+ minutes for a careful one). Compare to single-prompt "write me a 5-page proposal" (45 minutes editing the unusable output it produced).

The chain isn't slower than one big prompt. It's faster. And the output is dramatically better.

SECTION 6

The pricing-cascade pattern (real production ex

A multi-step chain running in my own pipeline right now

To make this less abstract, here's a chain in production at my own carpentry business -- running every time I generate a proposal.

When my proposal pipeline needs to figure out a price for a line item (drywall, doors, flooring), it doesn't ask one model and trust the answer. It runs a three-tier resolution chain:

Three-tier pricing resolution -- a real chain:

1. **Tier 1 -- Master rates lookup.** First, check the master rates list (curated, hand-edited, the ground truth). If the (trade, unit) pair is in the list, use that price. Done.
2. **Tier 2 -- RAG fallback.** If not in master rates, query the RAG index of past job records. The RAG returns 3-5 historically similar jobs with their pricing. The model picks the best match and proposes a price grounded in real prior data.

3. **Tier 3 -- HITL escalation.** If RAG returns nothing useful, ping me on Telegram with a structured message: *"Need pric*
   proposal continues.

Three tiers. Each is a small, well-defined task. The chain is what makes the whole thing reliable. Without it, the model would hallucinate prices on novel items and underbid jobs (which, before this chain existed, it did -- there's a 16x-underbid drywall story I don't tell at parties).

Why this matters as a pattern

The pricing-cascade isn't unique to construction. The same shape appears everywhere:

- **Customer support escalation** -- Tier 1 (canned answer) -> Tier 2 (search the knowledge base) -> Tier 3 (escalate to a human).
- **Sales qualification** -- Tier 1 (BANT-style scoring) -> Tier 2 (deeper discovery prompt) -> Tier 3 (human SDR review).
- **Hiring screening** -- Tier 1 (resume parser) -> Tier 2 (assessment chain) -> Tier 3 (human recruiter).
- **Compliance check** -- Tier 1 (rule-based pass) -> Tier 2 (LLM fuzzy match against policy) -> Tier 3 (compliance officer review).

The pattern: **cheap, de**
the loop only when the first two can't resolve. Every layer adds confidence and reduces cost. The chain is what ties them together.

If you're trying to figure out where AI fits in a workflow that already has humans + rules, the pricing-cascade pattern is usually the right shape.

SECTION 7

When NOT to chain

The chain pattern is the right tool for big tasks. It's the wrong tool in three categories:

- **Tasks small enough for one prompt.** Most daily AI work fits one prompt. The chain pattern is overhead -- don't apply it to tasks the model can finish in 200 words. (Module 4 *When NO*
  Use AI covers the speed red line -- chains have the same problem at the small end.)
- **Truly novel work where the chain shape isn't obvious yet.** If you've never done this kind of task, you don't know what the steps should be. Don't try to design a chain on first contact -- do the task manually a few times, observe where it naturally breaks into stages, *then* build
  chain.

- **Tasks where every step needs real-time human judgment.** A chain dies in the wait time. If steps 2 and 4 and 6 all need a human to make a call before the next step can run, the chain isn't really automated -- it's a workflow with eight handoffs and three of them are you. That can be the right structure, but it's not what this module's chain pattern was designed for.

The right test: **chain wh**
the steps are stable, and automated enough that you only need to be in the loop for review.

> ***The chain pattern is for tasks where the structure is repeatable. New territory? Do it manually first. Patterns become chains. Chains don't become patterns.***

SECTION 8

Where to go from here

You're now four modules into Tier 2. Two left:

- **Picking the right model** -- ChatGPT vs Claude vs Gemini vs the open-source pack. Cost-per-task, switching costs, real per-task economics from a real operator.
- **Privacy and what not to paste** -- the workplace version of Tier 1's privacy module. Data classifications, consumer vs enterprise tiers, the 1-page policy.

After Tier 2 closes, Tier 3 (Employable) starts -- the operator-level skills for being the AI-fluent person on your team.

Get the next module the day it drops: theaiguywi.com/training

One email per release. No drip. No spam. Opt out anytime.

If you have a workflow at your business that *needs* a m
support, intake, scheduling, anything -- that's the consulting offer. I design and install chains in real businesses. Not whiteboard architecture; actual running chains in your actual stack, like the one in mine.

Reach out: alexanderjahn79@icloud.com

A short call. Honest scope. We figure out together if it's a fit.

Closing -- the lock-in line

The single mental shift that separates beginner AI use from professional AI use:

1

One prompt = one small task.

Big tasks are sequences of small tasks. The skill isn't a smarter prompt. It's better task decomposition. Internalize that and the ceiling on what AI can do for your business goes up by an order of magnitude.

You have the chain pattern. The next module is how to pick the right model to run it on.

Agent Logic -- *theaiguyw*

Fond du Lac, WI. This is module 4 of 6 in Tier 2 (Professional).