What Ships: A Practitioner's Catalogue of Applied AI Product Patterns in Startups, 2024–2026

Yash Singh
IIIT Una

19 July 2026

Abstract

Between 2024 and 2026, applied AI product development went through a period of convergent evolution: startups working largely independently kept arriving at the same product shapes, the same orchestration architectures, and the same pricing structures. The underlying constraints (probabilistic outputs, context-window limits, per-token cost, and user trust) are the same everywhere, and they admit only so many good answers. This paper names those shapes. We synthesize public engineering writing, framework documentation, vendor pricing pages, practitioner surveys, and press-reported traction into a catalogue of twenty patterns organized in four layers: *interaction* (how users meet the model), *orchestration* (how model calls compose into systems), *data and context* (what the model is given to work with), and *trust and economics* (how the system is made dependable and paid for). For each pattern we state the problem it solves, the startups shipping it, and the failure mode of skipping it. We pair the catalogue with six recurring anti-patterns, short case vignettes of companies whose public traction validates particular stacks, and a decision guide mapping product requirements to pattern combinations. Our contribution is synthesis and naming rather than new empirical results: the thesis is that applied AI has quietly accumulated a reusable pattern language, and that most of the difference between AI products that ship and AI demos that stall is disciplined selection from this catalogue rather than novel research.

1 Introduction

The distance between a working model demo and a working AI product is now well documented: practitioner surveys consistently rank response quality, cost, and safety as the top blockers standing between agent prototypes and production deployment [23]. Yet a growing set of startups has crossed that distance repeatedly and at speed. Cursor, Harvey, Sierra, Decagon, and the app-builder generation (v0, Lovable, Bolt, Replit) ship continuously on top of the same handful of frontier models available to everyone else. Whatever separates them from the stalled majority is not model access.

We argue it is largely *pattern selection*. When many teams converge on the same solution shape under the same constraints, that shape is worth naming, cataloguing, and teaching, which is precisely the premise of the pattern-language tradition in architecture [1] and software design [16]. Applied AI in mid-2026 has reached the point where such a catalogue can be written down: the interaction idioms have stabilized (inline copilots, artifact generation, background agents with review gates), the orchestration playbook has been articulated by the model vendors themselves [2, 31], context

engineering has displaced prompt engineering as the operative discipline [4, 37], and pricing has visibly pivoted from seats toward work performed [7, 36].

This paper makes four contributions:

1. **A four-layer map** (Section 4) that organizes applied AI product decisions into interaction, orchestration, data and context, and trust and economics.
2. **A catalogue of twenty patterns** (Sections 5–8), each with the problem it solves, exemplar products, and the cost of skipping it.
3. **Six anti-patterns** (Section 9) that recur in practitioner accounts of stalled AI products.
4. **Case vignettes and a decision guide** (Sections 10–11) connecting the catalogue to companies whose press-reported traction suggests the patterns compose profitably.

Like its predecessors in the patterns tradition, this paper reports practice rather than inventing it. Every pattern here is observable in shipping practice under the inclusion test of Section 3; none is speculative.

2 Related work

The form of this paper borrows deliberately from the pattern-language tradition. Alexander et al. showed that recurring solution shapes in a design discipline can be mined from practice, named, and composed [1], and Gamma et al. carried the method into software [16]. We apply the same recurrence test to a much younger discipline; the price of that youth is the drift limitation discussed in Section 3.1.

The academic literature approaches the same terrain from the research side. Surveys of LLM-based agents organize systems by cognitive component: planning, memory, and tool use [40]; the retrieval-augmented generation line runs from the original formulation [24] through comprehensive surveys of its variants [17]; and Kapoor et al. critique how agent research is evaluated, arguing that benchmark-maximizing agents diverge from agents that are useful in deployment [21]. This catalogue is complementary to that body of work: our unit of analysis is the shipping product rather than the published system, and our layers follow the order in which product teams make decisions rather than the internal architecture of an agent.

Closest to our aim are the vendor playbooks: Anthropic's guidance on building effective agents [2] and OpenAI's practical guide [31] are prescriptive and practice-derived, and both are widely read. They concentrate on the orchestration layer, however. The catalogue here spans interaction, data, and economics as well, imposes an explicit three-product inclusion test, and pairs the patterns with the anti-patterns teams report when they stall.

3 Scope and method

Sources. We draw on four kinds of public evidence: (i) engineering writing by model vendors and infrastructure companies [2, 5, 6, 8, 29, 31]; (ii) framework and protocol documentation, particularly around tool integration [3]; (iii) practitioner surveys and investor field reports [7, 23, 28]; and (iv) product surfaces themselves: pricing pages, changelogs, and press-reported traction [20, 30, 36].

Inclusion criterion. A pattern enters the catalogue only if it is observable in at least three independent shipping products from different companies. "Shipping" means generally available and paid for, not demonstrated at a conference. For infrastructure- and data-layer patterns whose deployments are mostly internal (workflow substrates, gateways, eval harnesses, retrieval pipelines, fine-tuning programs), the three-product test was applied against public engineering writing, and Table 1 sometimes names the practice rather than three logos. Although the catalogue is aimed at startups, we cite incumbent and frontier-lab products (GitHub Copilot, Notion AI, Deep Research) where they are the clearest instance of a pattern; the unit of the catalogue is the pattern, not the company. We also scope out modality-specific interaction (voice and real-time agents, which several of our exemplar companies ship) and several near-universal implementation techniques that sit below the product-pattern grain: streaming and partial-result UX, cache-aware prompt layout, input-side prompt-injection defense, and first-run onboarding for blank-canvas products. Each deserves treatment of its own.

3.1 Limitations

Four biases are unavoidable. *Survivorship*: we see the patterns of companies that succeeded enough to write about them. *Disclosure*: revenue and traction figures are press-reported, not audited; we cite them as directional evidence only. *Drift*: this field re-tools quickly, and the catalogue is a snapshot dated mid-2026. *Diffusion*: not all convergence is independent; the vendor playbooks [2, 31] are widely read, so we cannot always distinguish teams rediscovering a shape under shared constraints from teams adopting a published one. We treat both as evidence that the shape survives production use, which is the property the catalogue certifies, while noting that it weakens any inference of inevitability. We mitigate the first bias by including anti-patterns sourced from practitioner writing and survey-reported failure modes, the second by labeling every traction figure as press-reported, and the third by preferring patterns with at least eighteen months of visible use.

4 A four-layer map

Product decisions in applied AI cluster into four layers (Figure 1). The layers are ordered by proximity to the user, and in practice they are also ordered by how early the decision gets made: teams choose an interaction shape first, discover the orchestration it requires, then discover the data machinery that orchestration needs, and confront trust and economics last, usually later than they should.

Table 1 summarizes the full catalogue; the next four sections walk through each layer.

5 Interaction patterns

5.1 I1: Copilot-in-Context

Problem. Users will not leave their working surface to consult a model, and copy-pasting context into a chat window loses the state that makes suggestions relevant.

Pattern. Embed generation *inside* the existing artifact at the point of attention: inline code completion, inline rewrite, cell-level spreadsheet fills. The model sees the surrounding document state; the user accepts with a keystroke. Cursor's Tab completion is the canonical example, and

#	Pattern	What it buys you	Exemplars
I1	Copilot-in-Context	Zero-friction help inside the existing workflow	Cursor Tab, GitHub Copilot, Notion AI
I2	Command Surface	Natural language as an accelerator over structured UI	Linear, Superhuman, Raycast AI
I3	Artifact, Not Transcript	Output is an editable object, not chat prose	v0, Lovable, Bolt, Claude Artifacts
I4	Background Agent	Minutes-long work moved off the interaction path	Devin, Claude Code, Deep Research
I5	Approval Queue	Delegation without giving up the final say	Fin drafts, Sierra handoffs, Clay outreach
O1	Workflow First	Predictability where the task shape is known	Support triage, document pipelines
O2	Tool Loop	Open-ended tasks via model-directed tool calls	Coding agents, computer use
O3	Sub-Agent Fan-Out	Parallelism plus context isolation	Anthropic research system, OpenAI and Gemini Deep Research
O4	Durable Execution	Agents that survive restarts and retries	Temporal/Inngest-backed agent products
O5	Model Cascade	Most traffic served below frontier-model cost	Routing layers in support and search
D1	Hybrid Retrieval	Recall from lexical + vector + reranking	Glean, Perplexity, enterprise RAG
D2	Agentic Search	Retrieval as an iterated tool, not one shot	Deep Research (OpenAI, Gemini), Cursor codebase QA
D3	Context Engineering	Deliberate budget for what the model sees	Claude Code, ChatGPT memory, Letta
D4	Structured Contract	Model output as a typed API, not prose	Extraction products, generative UI
D5	Customization Ladder	Prompt $\rightarrow$ RAG $\rightarrow$ fine-tune, in that order	Vertical AI (Harvey), high-volume tasks
T1	Evals as CI	Regressions caught before users see them	Golden sets + LLM-judge in CI
T2	Least-Privilege Tools	Bounded blast radius for agent actions	Sandboxes, allowlists, spend caps
T3	Trace Everything	Debuggable multi-step failures	LangSmith, Arize, MLflow tracing
T4	Price the Work	Margin aligned with value delivered	Fin \$0.99/resolution, Sierra outcomes
T5	Model-Agnostic Gateway	Model swaps in days, not quarters	Gateway layers, provider abstraction

Table 1: The pattern catalogue at a glance.

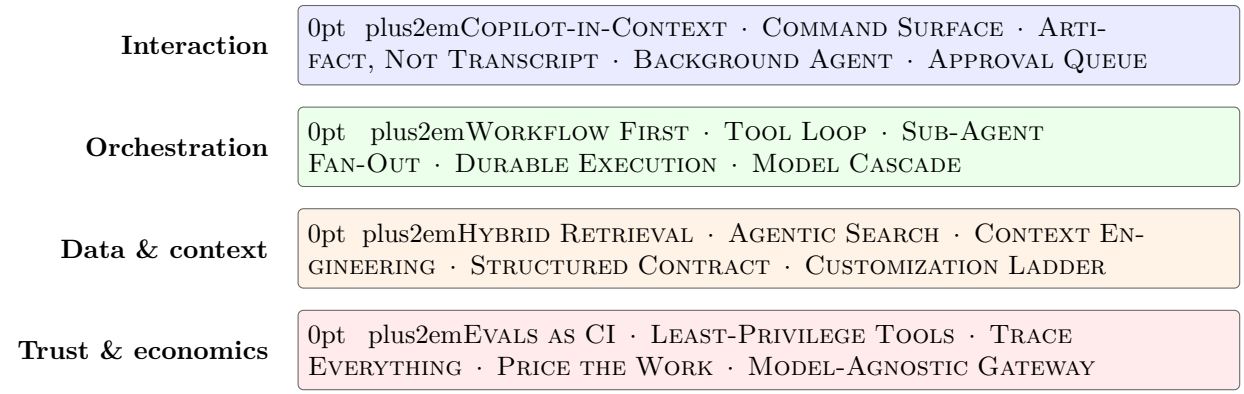

Figure 1: The four-layer map with the twenty catalogued patterns. Upper layers are visible to users; lower layers are visible in the margin structure and the incident log.

its press-reported multi-billion-dollar revenue run rate [30] is consistent with low-friction, high-frequency invocation being where individual willingness to pay concentrates. Latency budgets here are hundreds of milliseconds, which forces small or distilled models and aggressive caching, connecting this pattern directly to Model Cascade (O5).

Skip it and: your AI feature lives in a side panel nobody opens.

5.2 I2: Command Surface

Problem. Structured products accumulate hundreds of operations that users cannot remember how to reach.

Pattern. A natural-language command layer over the product's existing actions: "move everything assigned to me due this week to the next cycle." The model's job is intent parsing and parameter filling against a typed action schema (see Structured Contract, D4), not free generation. Linear, Superhuman, and Raycast all ship this shape. It is the safest first AI feature for an established SaaS product because the action set is already audited: the model chooses among known operations rather than inventing new ones.

Skip it and: you build an open-ended chatbot that can discuss the product but not operate it, which users try once.

5.3 I3: Artifact, Not Transcript

Problem. Chat transcripts are a poor container for work products. Users need to edit, version, share, and deploy what the model makes.

Pattern. The unit of output is a live artifact (an app, a document, a deck, a diagram) rendered beside the conversation and directly editable. Vercel's v0, Lovable, Bolt, and Claude's Artifacts all converged on the same split-pane shape within roughly a year of each other. The conversation becomes the *editing instrument*; the artifact is the product. Karpathy's framing of natural language as the new programming interface [22] is this pattern generalized: the prompt is source, the artifact is the build.

Skip it and: users screenshot your chat output into their real tools, and the value accrues to those tools.

5.4 I4: Background Agent

Problem. Real tasks (a research report, a pull request, a data migration) take minutes to hours, and no user will watch a spinner for that long.

Pattern. Move the agent off the interaction path: the user files a task, the agent works asynchronously with Durable Execution (O4), and returns a reviewable result such as a diff, a draft, or a cited report. Devin, Claude Code's cloud sessions, OpenAI's Deep Research, and the current generation of coding and research agents all share this shape. The critical design choice is the *return artifact*: it must be reviewable in minutes even though it took the agent an hour, which is why diffs and cited reports dominate.

Skip it and: you either cap task ambition at chat latency, or ship long-running work with no checkpoint, no cancellation, and no review surface.

5.5 I5: Approval Queue

Problem. Organizations want delegation without surrendering accountability, and regulators are moving toward human-oversight requirements for consequential automated actions (e.g., the EU AI Act's Article 14 [15]).

Pattern. The agent produces *proposed* actions (a support reply, an outbound email, a journal entry) that accumulate in a queue where humans approve, edit, or reject; approval rates are tracked and gate progressive autonomy. Support products (Fin, Sierra, Decagon) run this pattern for escalations; GTM products like Clay run it for outreach. The queue doubles as a labeling pipeline: every human edit is training and eval data (Evals as CI, T1).

Skip it and: the first bad autonomous action becomes a churn event or a headline.

6 Orchestration patterns

6.1 O1: Workflow First

Problem. Teams reach for autonomous agents when the task has a known, decomposable shape, and inherit non-determinism they did not need.

Pattern. When the steps are knowable in advance, hard-code them: prompt chaining (fixed task decompositions of the kind chain-of-thought prompting made visible [41]), routing, parallelization, orchestrator-workers, and evaluator-optimizer loops, in Anthropic's articulation [2], echoed by OpenAI's guidance to start with the simplest composition that works [31]. A workflow's control flow lives in code, so it is testable, traceable, and cheap. The practitioner heuristic that survives contact with production is: *use workflows for predictability, agents for flexibility, and workflows until proven otherwise.*

Skip it and: you debug an agent's planning on tasks that never needed planning.

6.2 O2: Tool Loop

Problem. Some tasks genuinely cannot be decomposed in advance: debugging, open-ended research, multi-system operations.

Pattern. The reason-act loop [42], now internalized as native tool calling: the model chooses tools,

observes results, and iterates until a stop condition, with self-correction along the way [27, 35]; see Wang et al. [40] for the academic lineage. What changed between 2023 and 2026 is not the loop but its harness: production tool loops now run inside budget caps, step limits, and Least-Privilege Tools (T2), and their traces feed Evals as CI (T1). Tool-augmented modeling was demonstrated early [34]; the surprise is how much of the engineering lives outside the model. The integration surface has meanwhile standardized: the Model Context Protocol, introduced in late 2024 [3] and since adopted by the other major providers, has become a de facto standard for exposing tools to models, letting one tool loop reach thousands of external systems without bespoke adapters.

Skip it and: you lose nothing if your tasks are decomposable, and everything if they are not.

6.3 O3: Sub-Agent Fan-Out

Problem. A single context window cannot hold a large task, and long contexts degrade attention over their middle [25].

Pattern. An orchestrator decomposes the task and spawns parallel sub-agents, each with a clean, scoped context; results are synthesized. Anthropic's public account of its research system reports substantial quality gains from exactly this isolation-plus-parallelism structure [5]. The pattern's tax is coordination: sub-agents cannot see each other's discoveries unless the orchestrator routes them, so it fits tasks that decompose cleanly (research, review, migration sweeps) better than tightly coupled edits. Cognition's engineering guidance argues against multi-agent designs for exactly the coupled case [12]; that is the boundary of this pattern, not a refutation of it.

Skip it and: large tasks either overflow the window or degrade quietly as the context fills.

6.4 O4: Durable Execution

Problem. A thirty-minute agent run *will* encounter a rate limit, a deploy, or a crash, and users will not accept losing twenty-nine minutes of work.

Pattern. Treat agent steps as checkpointed, replayable workflow activities: persist state after every tool call, make tools idempotent, and resume from the journal rather than restarting. Startups increasingly run agent loops on durable-execution substrates (Temporal, Inngest, or homegrown queues) so that retries, timeouts, and human-approval pauses (Approval Queue, I5) are first-class. The substrate vendors' own guidance frames durable execution as the foundation for any agent that outlives a request cycle [14, 33].

Skip it and: your agent's reliability is the product of every dependency's uptime, and that product rounds to zero over long runs.

6.5 O5: Model Cascade

Problem. Frontier-model pricing on every call destroys gross margin, but small models alone cap quality.

Pattern. Route by difficulty: a cheap model (or a cache hit) handles the easy majority, escalating to larger models only on low confidence, following the cascade logic formalized in FrugalGPT [10]. Production deployments layer semantic caching in front, then distilled or small open models, then a frontier model, then (for some products) a human, with each tier absorbing most of what reaches it (Figure 2). The economics compress to one expression: with per-call cost c_i at tier i and absorption

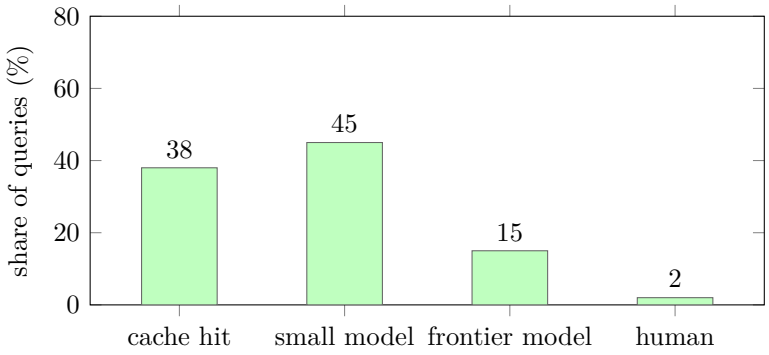

Figure 2: An *illustrative* model cascade for a support-style workload (bar values are notional, not measured): the cheap tiers together absorb the large majority of traffic, so frontier-model spend applies to a small residual. Real mixes vary by product; the shape recurs across them.

rate h_i (the fraction of traffic reaching tier i that stops there), expected cost per query over n tiers is

$$\mathbb{E}[c] \;=\; \sum_{i=1}^{n} c_i \prod_{j=1}^{i-1} (1 - h_j), \tag{1}$$

so the cheap tiers' absorption rates, not the frontier model's list price, dominate unit cost. Inline copilots (I1) are the extreme case: their latency budget alone forces the cascade. Margin pressure across the category is well documented [7].

Skip it and: you discover at scale that your unit economics require charging more than users will pay.

7 Data and context patterns

7.1 D1: Hybrid Retrieval

Problem. Pure vector search misses exact identifiers and rare terms; pure lexical search misses paraphrase; either alone caps answer quality.

Pattern. Retrieval-augmented generation [24] in its matured production form [17]: lexical and vector retrieval run together, a reranker orders the union, and metadata filters scope the corpus. The standard fusion step is reciprocal rank fusion [13]: a document d ranked by retrievers R scores

$$s(d) \;=\; \sum_{r \in R} \frac{1}{k + \mathrm{rank}_r(d)}, \tag{2}$$

with $k \approx 60$ in common practice, cheap enough to run on every query. Enterprise search (Glean), answer engines (Perplexity), and virtually every serious internal RAG deployment run this shape. A recurring practitioner report of 2024–2025 was that retrieval quality, not model choice, dominated answer quality for knowledge products [23, 28].

Skip it and: your model fluently summarizes the wrong documents.

7.2 D2: Agentic Search

Problem. One-shot retrieval cannot answer questions that require following leads: multi-hop questions, codebase questions, market research.

Pattern. Make retrieval a *tool inside a loop* (O2): the model searches, reads, reformulates, and searches again, accumulating evidence before answering. Deep-research products and coding assistants' codebase question-answering both replaced single-shot RAG with iterated search between 2024 and 2026. The cost is latency and tokens, which is why shipping products pair it with Background Agent (I4) rather than chat.

Skip it and: multi-hop questions get confident answers assembled from the first page of results.

7.3 D3: Context Engineering

Problem. The context window is the binding resource, attention degrades over long contexts [25], and naive accumulation of history, tool output, and retrieved text fills it with noise.

Pattern. Budget and curate everything the model sees on every call: system state, retrieved evidence, tool results, and memory. In practice this means compaction (rewriting older turns into summaries), tool-output truncation with keep-or-drop rules, tiered memory with explicit recall (the MemGPT lineage [32]), and per-call context budgets [4, 37]. By 2026 this discipline, not prompt wording, is what determines agent reliability; vendor engineering guidance is explicit that context curation is the highest-leverage lever for long-running agents.

Skip it and: your agent gets measurably dumber as the session gets longer, and nobody can say why.

7.4 D4: Structured Contract

Problem. Downstream code cannot consume prose, and parsing free text is a bug factory.

Pattern. Constrain output to a schema (JSON schema, function signatures, typed DSLs) and treat the model as a typed API. This is the load-bearing pattern beneath Command Surface (I2), generative UI, extraction products, and most B2B AI: the schema is simultaneously the interface contract, the validation gate, and the eval target (exact-match scoring against fields is cheap and objective, easing the evaluator-trust problem of judging prose [44]).

Skip it and: you maintain a regex museum between the model and your database.

7.5 D5: Customization Ladder

Problem. Teams jump to fine-tuning for problems that prompting or retrieval solve in a day, acquiring MLOps burden and a stale model.

Pattern. Climb in order: prompt engineering with few-shot examples first; retrieval (D1) when the knowledge is the gap; fine-tuning only when format, style, or latency at volume demand it, typically distilling a frontier model's behavior on a narrow task into a small model that slots into a cascade (O5). Vertical leaders like Harvey combine frontier models with domain workflow and data rather than betting on bespoke base models; the high-volume, narrow-task tier (classification, extraction, routing) is where fine-tuned small models earn their keep.

Skip it and: you spend a quarter fine-tuning your way to what a better retrieval pipeline would have delivered in a sprint.

8 Trust and economics patterns

8.1 T1: Evals as CI

Problem. Probabilistic systems regress silently: a prompt tweak, model upgrade, or retrieval change shifts behavior with no compiler error.

Pattern. A golden dataset of real cases, scored on every change and gating deploys exactly like tests: exact-match where outputs are structured (D4), LLM-as-judge with periodic human calibration where they are prose [44], and downstream task metrics where possible [21]. Production guides treat systematic evaluation as a first-class engineering discipline for agent teams [29]; survey data confirm quality is the top adoption blocker [23]. The approval queue (I5) and trace store (T3) feed the dataset continuously.

Skip it and: every deploy is a bet, and you find regressions via churn.

8.2 T2: Least-Privilege Tools

Problem. An agent with broad credentials turns a model mistake or a prompt injection into a production incident.

Pattern. Scope every tool to the minimum: allowlisted actions, sandboxed execution, read-only defaults, spend and rate caps, and human approval (I5) for irreversible operations. Vendor guidance formalizes this as guardrails around both inputs and actions [31], and coding agents ship sandboxes by default. The tool schema becomes a security boundary, reviewed like one.

Skip it and: your agent's first confidently wrong action writes to production.

8.3 T3: Trace Everything

Problem. A five-step agent failure is invisible in request logs; the error is distributed across model calls, tool results, and context state.

Pattern. Capture the full tree (prompts, tool calls, outputs, tokens, latency, cost) for every run, as spans, and make it queryable. The tracing substrate (LangSmith, Arize, MLflow, and homegrown equivalents) is what makes agent debugging, cost accounting per feature, and eval-set mining (T1) possible at all [6, 29].

Skip it and: "the agent did something weird yesterday" is a bug report you cannot act on.

8.4 T4: Price the Work

Problem. Seat-based pricing breaks twice for AI products: marginal cost per use is nonzero (heavy users destroy margin), and the product's value is the work performed, not the humans logged in; an agent that replaces labor has no seat to price.

Pattern. Charge in units of work or outcome: per resolution (Intercom's Fin at $0.99 per resolved conversation [20]), outcome-based contracts (Sierra [36]), usage-metered credits (the app builders), or hybrid base-plus-usage, which investor field data show displacing seat-only pricing, with category

Anti-pattern	Symptom	Remedy
Thin wrapper	Product is a prompt in front of an API; churn tracks the base model's improvements	Own workflow, data, and distribution; climb the layers (Fig. 1)
Chat for everything	A text box replaces designed UX; users must invent their own prompts	I1/I2/I3: meet users inside their artifact
Agent maximalism	Autonomous agent deployed on decomposable tasks; nondeterminism with no payoff	O1: workflow until proven otherwise [2]
Demo-driven development	Impressive happy path, no eval set, quality regressions on every change	T1 before scale; measure the boring cases [21]
Premature fine-tuning	Months on a bespoke model that retrieval or prompting matches	D5: climb the ladder in order
Below-cost pricing	Flat pricing meets power users; negative gross margin at exactly the moment of product-market fit	T4 + O5: meter the work, cascade the cost [7]

Table 2: Recurring anti-patterns and their catalogue remedies.

gross margins already well below classic SaaS [7]. T4 and O5 are two sides of one inequality: charging price p per resolution at resolution rate q is viable only while

$$q\,p \;>\; \mathbb{E}[c], \tag{3}$$

with $\mathbb{E}[c]$ the per-attempt serving cost of Equation 1; outcome pricing is what makes the cascade a requirement rather than an optimization. Two consequences follow: billing infrastructure becomes product infrastructure (metering, attribution, dispute handling), and the eval suite (T1) becomes a revenue system, because you only get paid for resolutions you can verifiably count.

Skip it and: either customers subsidize your heavy users, or your gross margin does.

8.5 T5: Model-Agnostic Gateway

Problem. Model rankings shuffle every few months, and a hard dependency on one provider converts each release cycle into a migration.

Pattern. All model calls go through one internal gateway that owns provider adapters, fallbacks, caching, routing (O5), and per-feature cost attribution (T3). Prompts and evals (T1) are versioned per model, so adopting a newly released model is an eval run plus a config change shipped in days, not a migration project; this is what lets startups ride the frontier rather than be disrupted by it.

Skip it and: you are one deprecation notice away from an unplanned quarter of migration work.

9 Anti-patterns

The catalogue's negative space is as consistent as its positive space. Table 2 lists the six failures that recur in practitioner writing and survey-reported failure modes [2, 21, 23, 28].

Two of these deserve a sentence more. *Agent maximalism* is the most expensive because it is the most seductive: the demo is genuinely better, and the production behavior is genuinely worse, a

gap that only an eval suite (T1) makes visible before customers do [21]. *Thin wrapper* is the most misdiagnosed: wrappers fail not because wrapping is bad but because the wrapper owns no layer of Figure 1 below the first; the companies dismissed as wrappers that thrived (Perplexity, Cursor) own retrieval, context, evals, and distribution.

10 Case vignettes

Traction figures below are press- or company-reported (cited per claim), offered as directional evidence that these pattern stacks compose into businesses, not as audited financials.

Cursor (AI coding). Pattern stack: I1 as the wedge (Tab completions), I4 layered later (background agents), O5 forced by completion latency, D3 for repository context, T5 to ride successive frontier models. Press reports place its annualized revenue at $2B in March 2026 and $4B by June 2026 [30, 39]; that growth spans Tab and the agent modes, though the aggregate figure cannot be attributed between them.

Harvey (legal). Pattern stack: D1/D5 (domain retrieval over authorities and work product), O1 workflows shaped to legal deliverables, I5 review surfaces fitting professional accountability, T2 for privilege boundaries. A March 2026 raise at an $11B valuation, with a company-reported 100,000-plus lawyers across 1,300 organizations and a majority of the AmLaw 100 as customers [18], supports the vertical thesis: the moat is workflow depth and professional trust.

Sierra (customer service agents). Pattern stack: O1 routing with O2 escalation, I5 human handoffs, T1 evals tied to resolution counting, and T4 as strategy, outcome-based pricing per resolution [36]. Press reports of $100M ARR within two years [38], $150M by mid-2026 [11], and a May 2026 raise at a $15B+ valuation [26] suggest that outcome pricing has not constrained enterprise adoption.

Decagon (support automation) runs the adjacent stack (I5, O1, T1, usage-metered T4) and reached a press-reported $4.5B valuation in February 2026, triple its prior round [43]. A valuation reflects investor expectation rather than delivered work, so we weight this vignette below the revenue-based ones; it still suggests the stack, not any single company's execution, is the repeated element.

The app builders (v0, Lovable, Bolt, Replit). Pattern stack: I3 as identity (the artifact is a deployed app), D4 underneath (generation targets component and config schemas), credit-metered T4, T5 across models. Lovable's climb to $100M ARR within eight months of launch and Replit's to $150M annualized within a year [9, 19] track the maturation of I3 from demo to default.

11 A decision guide

Figure 3 compresses the orchestration choice; Table 3 maps common product requirements onto pattern stacks. Both encode the same bias the vendor playbooks state outright [2, 31]: take the simplest composition that meets the requirement, and add autonomy only when the task shape is genuinely unknown.

12 Discussion and open problems

The catalogue also exposes what practice has *not* solved.

Evaluator trust. T1 increasingly leans on model judges [44] whose failure modes correlate with the systems they judge; agent benchmarks remain easy to overfit and expensive to make realistic

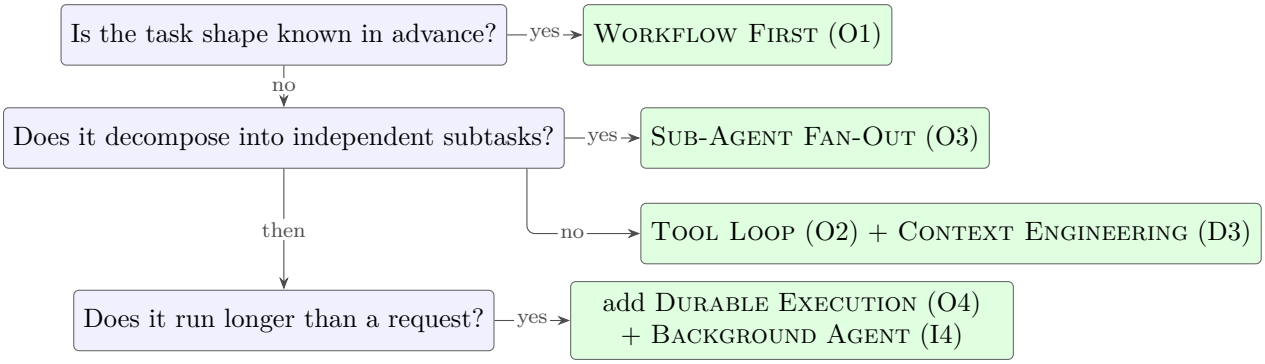

Figure 3: The orchestration decision. The first two questions choose a base pattern; the third applies to whichever base was chosen (the edge is drawn from the open-ended branch only to keep the figure compact) and overlays O4 and I4. Every path also gets T1 and T3; any path with side effects gets T2 and usually I5.

Product requirement	Minimal pattern stack
AI feature inside an existing SaaS	I2 + D4 + T1; add I1 if the product has an editing surface
Knowledge answers over a corpus	D1 + T1 + O5; add D2 when questions are multi-hop
Automate a back-office process	O1 + I5 + T1–T3; agents (O2) only for the exception path
Autonomous work product (code, research)	I4 + O2/O3 + O4 + D3 + T2
High-volume narrow task (extract, classify, route)	D4 + D5 + O5; eval by exact match
Sell the outcome, not the software	everything above, plus T4 with metering you can defend in a billing dispute

Table 3: Requirements to pattern stacks. Every stack implicitly includes T3 (you cannot run what you cannot see) and T5 (you will change models).

[21]. Outcome pricing (T4) raises the stakes: when the eval counts revenue, adversarial pressure on the eval is guaranteed.

Attribution for outcome pricing. "Per resolution" works when a conversation closes; it is unresolved for agents whose work is one input among many (marketing, research, code that ships weeks later). Until attribution matures, hybrid metering remains the honest default [7].

Trust UX for delegation. I5 does not scale to agents proposing hundreds of actions per day; nobody has shipped a convincing answer to *graduated* autonomy, where trust earned on one task class transfers legibly to another.

Margin structure under falling token prices. Cascades (O5) and falling frontier prices improve unit economics, but capability expectations rise in lockstep, keeping spend per task roughly constant; whether AI-native gross margins converge to SaaS norms or settle permanently lower [7] will shape which patterns remain affordable.

Defensibility. The catalogue is public and the patterns are imitable. The durable moats visible in the vignettes (workflow depth, proprietary eval data, distribution) predate AI; whether pattern execution alone compounds into defensibility is the category's open bet.

13 Conclusion

The applied AI products that ship in 2026 are not built from secret techniques. They are built from a small, nameable catalogue of patterns that teams keep arriving at, partly by independent rediscovery and partly by adopting published playbooks, because the constraints (probabilistic outputs, bounded context, per-token cost, human trust) admit only so many good answers. We have named twenty of them, organized them into four layers, paired them with the six failures teams repeat, and grounded them in the public traction of companies whose stacks we can read from the outside. The practical claim is modest and testable: for most AI product decisions in a startup today, the right move already has a name in this catalogue, and the expensive mistakes come from skipping a layer, not from missing research. Pattern languages end with an invitation rather than a proof, and so does this one: the catalogue is a snapshot of mid-2026, and it will be revised by the next capability jump, most likely first at the interaction layer, where each jump has rewritten what users meet.

References

[1] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977. ISBN 978-0195019193.

[2] Anthropic. Building effective agents. https://www.anthropic.com/engineering/building-effective-agents, 2024. Accessed July 2026.

[3] Anthropic. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol, 2024. Accessed July 2026.

[4] Anthropic. Effective context engineering for AI agents. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents, 2025. Accessed July 2026.

[5] Anthropic. How we built our multi-agent research system. https://www.anthropic.com/engineering/multi-agent-research-system, 2025. Accessed July 2026.

[6] Arize AI. Three production patterns for AI agents and how to evaluate each one. https://arize.com/blog/3-production-patterns-ai-agents-how-to-evaluate-each-one/, 2026. Accessed July 2026.

[7] Bessemer Venture Partners. The AI pricing and monetization playbook. https://www.bvp.com/atlas/the-ai-pricing-and-monetization-playbook, 2026. Accessed July 2026.

[8] Matt Bornstein and Rajko Radovanovic. Emerging architectures for LLM applications. https://a16z.com/emerging-architectures-for-llm-applications/, 2023. Andreessen Horowitz. Accessed July 2026.

[9] Julie Bort. Replit hits $3B valuation on $150M annualized revenue. https://techcrunch.com/2025/09/10/replit-hits-3b-valuation-on-150m-annualized-revenue/, 2025. TechCrunch, September 2025. Accessed July 2026.

[10] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[11] CNBC. Bret Taylor's Sierra raises nearly $1 billion months after last capital push. https://www.cnbc.com/2026/05/04/bret-taylor-sierra-fundraise-openai.html, 2026. CNBC, May 2026. Reports Sierra topping $150M ARR. Accessed July 2026.

[12] Cognition AI. Don't build multi-agents. https://cognition.com/blog/dont-build-multi-agents, 2025. Accessed July 2026.

[13] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759, 2009. doi: 10.1145/1571941.1572114.

[14] Cornelia Davis. Durable execution meets AI: Why Temporal is the perfect foundation for AI agent applications. https://temporal.io/blog/durable-execution-meets-ai-why-temporal-is-the-perfect-foundation-for-ai, 2025. Temporal blog, July 2025. Accessed July 2026.

[15] European Union. Regulation (EU) 2024/1689 (artificial intelligence act), article 14: Human oversight. https://eur-lex.europa.eu/eli/reg/2024/1689/oj, 2024. Accessed July 2026.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN 0-201-63361-2.

[17] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[18] Harvey. Harvey raises at $11 billion valuation to scale agents across law firms and enterprises. https://www.harvey.ai/blog/harvey-raises-at-dollar11-billion-valuation-to-scale-agents-across-law-firms-and-enterprises, 2026. Company announcement, March 2026. Accessed July 2026.

[19] Anna Heim. Eight months in, swedish unicorn Lovable crosses the $100M ARR milestone. https://techcrunch.com/2025/07/23/eight-months-in-swedish-unicorn-lovable-crosses-the-100m-arr-milestone/, 2025. TechCrunch, July 2025. Accessed July 2026.

[20] Intercom. Fin by Intercom: Pricing. https://fin.ai/pricing, 2026. Outcome-based pricing at $0.99 per resolution. Accessed July 2026.

[21] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

[22] Andrej Karpathy. Software is changing (again). https://www.youtube.com/watch?v=LCEmiRjPEtQ, 2025. Keynote, Y Combinator AI Startup School, June 2025. Accessed July 2026.

[23] LangChain. State of AI agents. https://www.langchain.com/stateofaiagents, 2024. Survey of 1,300+ practitioners. Accessed July 2026.

[24] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2005.11401.

[25] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 2024. doi: 10.1162/tacl_a_00638.

[26] Connie Loizos. Sierra raises $950M as the race to own enterprise AI gets serious. https://techcrunch.com/2026/05/04/sierra-raises-950m-as-the-race-to-own-enterprise-ai-gets-serious/, 2026. TechCrunch, May 2026. Accessed July 2026.

[27] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.17651.

[28] Menlo Ventures. The state of generative AI in the enterprise. https://menlovc.com/perspective/2025-the-state-of-generative-ai-in-the-enterprise/, 2025. Accessed July 2026.

[29] MLflow. Building production-ready AI agents in 2026. https://mlflow.org/articles/building-production-ready-ai-agents-in-2026/, 2026. Accessed July 2026.

[30] Richard Nieva and Anna Tong. Cursor hits $4 billion annualized revenue. https://www.forbes.com/sites/richardnieva/2026/06/08/cursor-4-billion-annualized-revenue/, 2026. Forbes, June 2026. Accessed July 2026.

[31] OpenAI. A practical guide to building agents. https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf, 2025. Accessed July 2026.

[32] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[33] Charly Poly. Durable execution: The key to harnessing AI agents in production. https://www.inngest.com/blog/durable-execution-key-to-harnessing-ai-agents, 2026. Inngest blog, February 2026. Accessed July 2026.

[34] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2302.04761.

[35] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.11366.

[36] Sierra. Outcome-based pricing for AI agents. https://sierra.ai/blog/outcome-based-pricing-for-ai-agents, 2024. Accessed July 2026.

[37] Sourcegraph. Context engineering: A practical guide for AI agents. https://sourcegraph.com/blog/context-engineering, 2026. Accessed July 2026.

[38] Marina Temkin. Bret Taylor's Sierra reaches $100M ARR in under two years. https://techcrunch.com/2025/11/21/bret-taylors-sierra-reaches-100m-arr-in-under-two-years/, 2025. TechCrunch, November 2025. Accessed July 2026.

[39] Marina Temkin. Cursor has reportedly surpassed $2B in annualized revenue. https://techcrunch.com/2026/03/02/cursor-has-reportedly-surpassed-2b-in-annualized-revenue/, 2026. TechCrunch, March 2026. Accessed July 2026.

[40] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18 (6), 2024. arXiv:2308.11432.

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2201.11903.

[42] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

[43] Alexandra York and Zoya Hasan. AI agent startup Decagon triples valuation to $4.5 billion. https://www.forbes.com/sites/alexyork/2026/02/06/ai-agent-startup-decagon-triples-valuation-to-45-billion/, 2026. Forbes, February 2026. Accessed July 2026.

[44] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. arXiv:2306.05685.