

PORTFOLIO CASE STUDY

Designing my personal website

1. The problem

I built my website to help people understand my work and judge how it relates to theirs. My experience spans research and product development, where similar methods can serve very different purposes. The connection between analysing microscopy and reading chemical structures is easier to explain through the work than through a list of job titles. I wanted someone considering me for a role or a project to be able to follow those connections for themselves.

A CV is like a movie trailer. It gives a quick idea of what I've done, while the website has room for the thinking behind it. Visitors can inspect a case study, watch the Conversico product film, explore research or ask how an approach in one project relates to another. Building the site also gives me a way to demonstrate the work directly: visitors use a product I have designed, built and released.

I wanted that product to feel personal. Photographs, a film made from old family photos and the cosmic background sit alongside the software. They give people something to discover beyond qualifications and job titles. Bringing those elements together created the design problem: make a distinctive, substantial site that remains easy to read and practical for me to maintain.

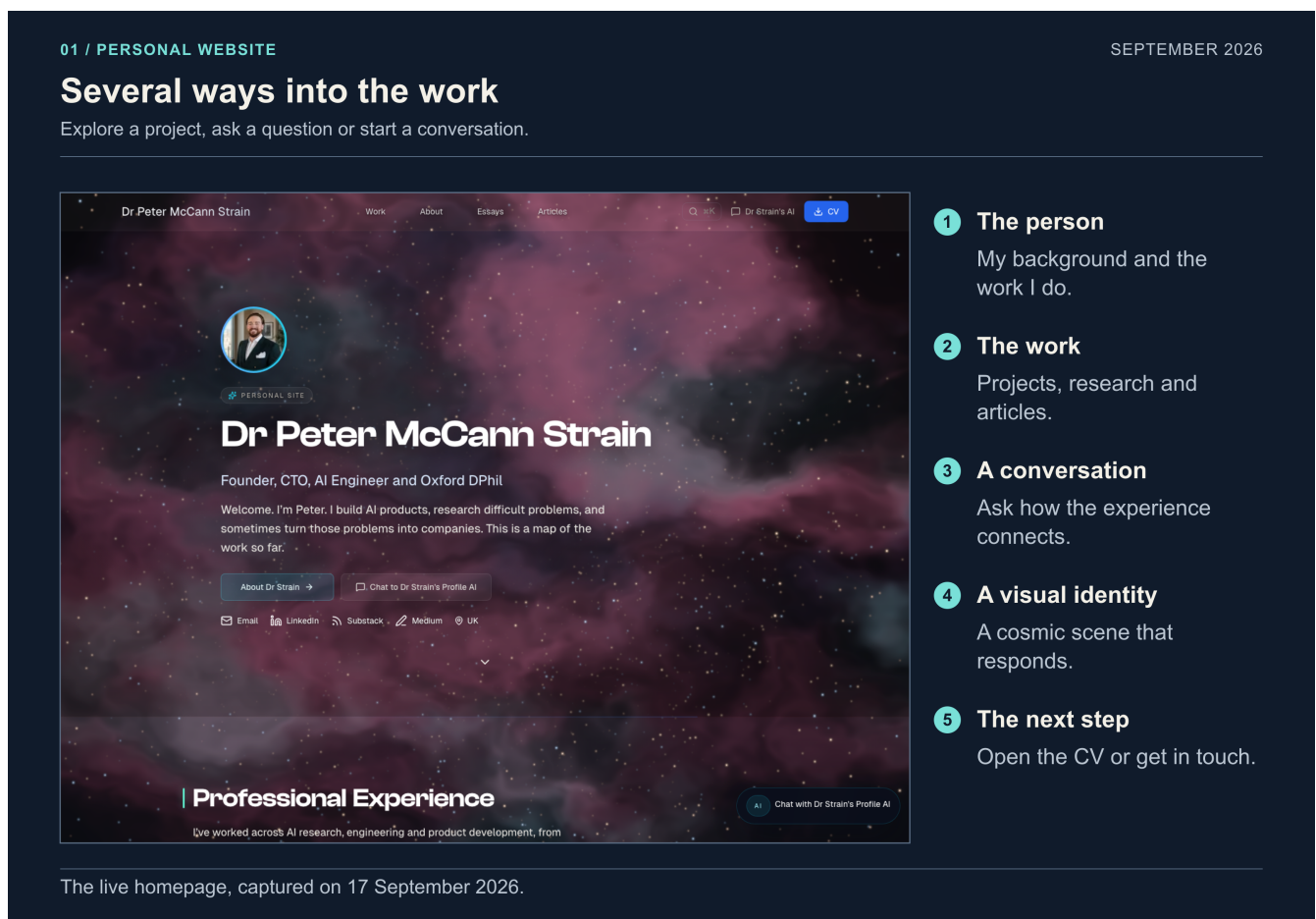


Figure 1. The homepage gives visitors several ways to explore the work, with the CV and contact details close at hand.

2. Functional requirements

Someone arriving to assess my background can scan the experience section, open a relevant project and download the CV or get in touch. The project details explain what I built and the decisions behind it. Films give visitors another way to inspect the work, and the command palette offers a quick route around the collection.

A reader following a shared article can use its contents navigation, expand an image or switch to focus mode. If a passage raises a question, they can open the assistant from that page. The conversation carries the article context, while retrieval can bring in related work from elsewhere on the site. Source links take the reader back to the material behind the answer.

Someone assessing me for a role can attach a job description and ask which parts of my experience are relevant. Follow-up questions let them compare projects or examine a particular claim without starting again. The chat keeps an expandable record of its searches and relationship exploration alongside the answer, so the visitor can see where the material came from.

Visitors can dictate into an editable draft, use read aloud, stop a response or choose a suggested follow-up. Keyboard navigation, visible focus and motion preferences support the same reading and conversation journeys.

3. Non-functional requirements

A visitor should be able to start reading promptly and have the page respond when they use it. My page targets are a Largest Contentful Paint of 2.5 seconds and an Interaction to Next Paint of 200 milliseconds at the 75th percentile. Browser checks help me investigate loading and interaction; field measurements show what visitors experience. Conversation has a different wait: retrieval happens before generation, so I track both the time to useful text and the time to finish an answer.

I separated failures according to what they interrupt. Articles and the CV remain accessible when the model is unavailable. A graphics failure can remove the background without removing the reading surface. If an answer is interrupted, the chat shows that state rather than presenting the partial response as complete.

Someone reading with a keyboard or with reduced motion should still be able to reach the content and use the controls. I use semantic elements, visible focus and a clear reading surface. Privacy has a similarly direct consequence: a question or job description belongs to that visitor's conversation and stays outside the public knowledge index. Conversation history has a bounded lifetime, and private server retention is off by default.

I operate the site myself. A shared monthly spending cap covers the assistant's model and Voyage calls, alongside rate and concurrency limits. Reaching that cap stops paid conversation while browsing continues. Routine updates also need to be affordable: changing one article should refresh that document's index without paying to process the whole collection again.

Visitor need	Design response	Practical check
Responsive reading	Deferred graphics and separate page delivery	Measure loading and input response against the page targets
Useful answers	Hybrid retrieval, adaptive tools and source links	Ask a project comparison and follow its sources
Accessible controls	Keyboard support, clear focus and motion settings	Navigate dialogs by keyboard and check reduced motion
Private conversation	Separate content and conversation lifecycles	Check uploads and expiry; confirm they stay outside the index
Continued access	Independent reading, graphics and chat paths	Exercise chat and graphics failures while reading

Figure 2. Reading, conversation and privacy each have a concrete design response and a practical check.

4. Expected scale

This is a personal portfolio with a small audience. One application, managed hosting and cacheable assets suit that scale. A shared article can still create a burst of traffic, so I separate page delivery from simultaneous conversations. Reading mostly costs content and asset delivery. Chat adds provider latency, concurrency and spending on each request.

Knowledge size is a separate concern. The first hybrid index covers 45 documents and 1,833 passages. It fits in a server-only application bundle, where exact vector search and adjacency lists are straightforward to operate. I do not need a separate vector or graph database to retrieve from this collection. Keeping it local also makes the relationship between a deployment and its index explicit.

The browser has its own workload. Every visitor's device must draw the scene and respond to input, regardless of how many other people are on the site. There's no group discount for pixels. I therefore treat rendering cost as a device problem and adapt it on the device, while the server controls shared conversation capacity.

5. High-level architecture

The live website is one Next.js 16 application hosted on Vercel, with React 19 handling the interface. Page requests resolve authored content into a reading surface. The browser runs its controls and the cosmic scene, built with Three.js and React Three Fiber. The assistant's search index ships with the application as a server-only bundle of published passages, vectors and graph records.

When a visitor asks a question, server code searches that bundle and supplies the selected passages to Claude Sonnet 5.5 at high effort through the Vercel AI SDK. Voyage supplies embeddings and reranking; keyword search and exact vector search run inside the application. Upstash Redis holds shared rate, concurrency and spending controls. PostHog and Vercel instrumentation show how visitors use the product and how the system behaves.

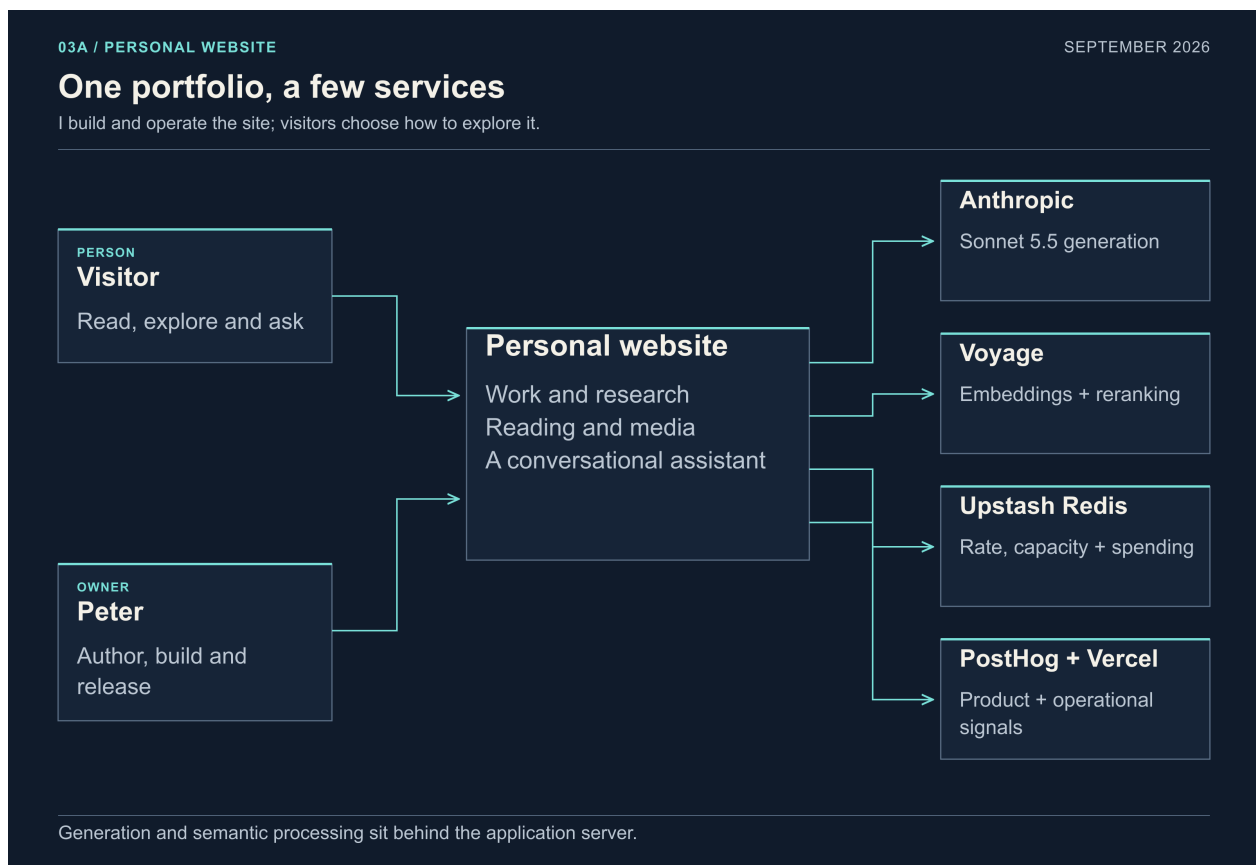


Figure 3A. I operate one portfolio; external services supply generation, semantic processing and shared controls.

The chat endpoint receives the question with its conversation state and page references. It validates those references and keeps a concise profile context alongside the retrieved evidence. Sonnet can then answer or ask the server to retrieve more. Reading a page does not run that conversation loop or call the model.

The publishing extension has a separate job: activating and rolling back page content independently of an application release. A publication selector in Neon points to immutable releases in Vercel Blob. The owner-only Payload CMS is a separate authoring application. The hybrid index is built from the local published-content collection. Publishing a page and updating that index are separate operations. A signed page receipt can select the edition being read; the server checks that its index matches that edition before using its vectors or graph.

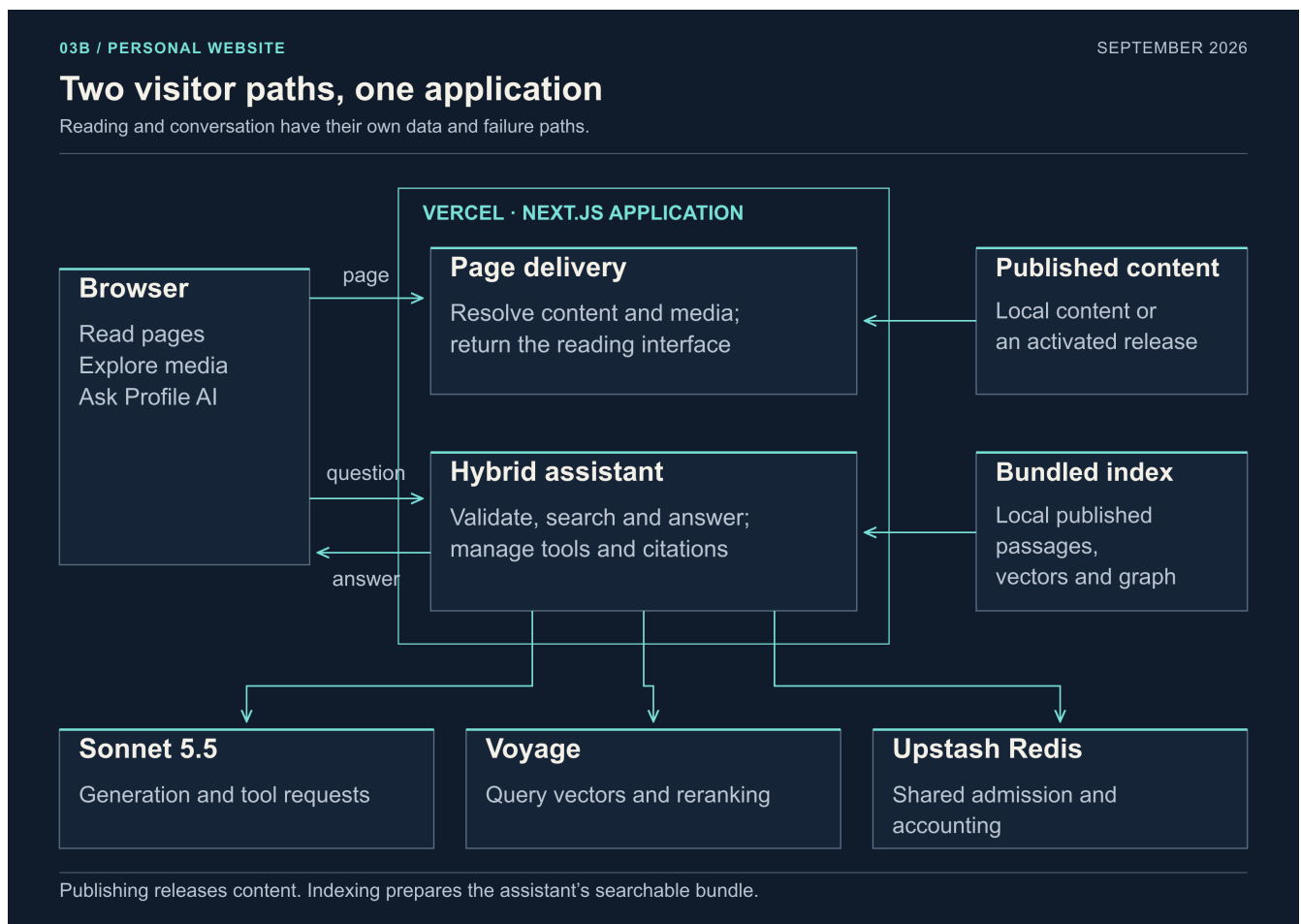


Figure 3B. Page delivery and conversation share an application while reading different representations of the content.

Data model and interfaces

Profile facts, the CV, articles, research and media descriptions form the authored record. Documents have stable identities, routes and metadata. Passages preserve their original text and citation locations. The index adds vectors, entities and source-linked relationships to make that material searchable in different ways.

I keep the distinction between source and interpretation in the data model. An extracted relationship records whether it describes an explicit statement or an inferred connection, and retains its supporting passages. Its description helps the assistant navigate; the original passage supplies the citation. Browser state has a different role again: it carries a visitor's question and conversation, while the server resolves what that state refers to.

Content, evidence and conversation

The index describes published work. A visitor brings the question.

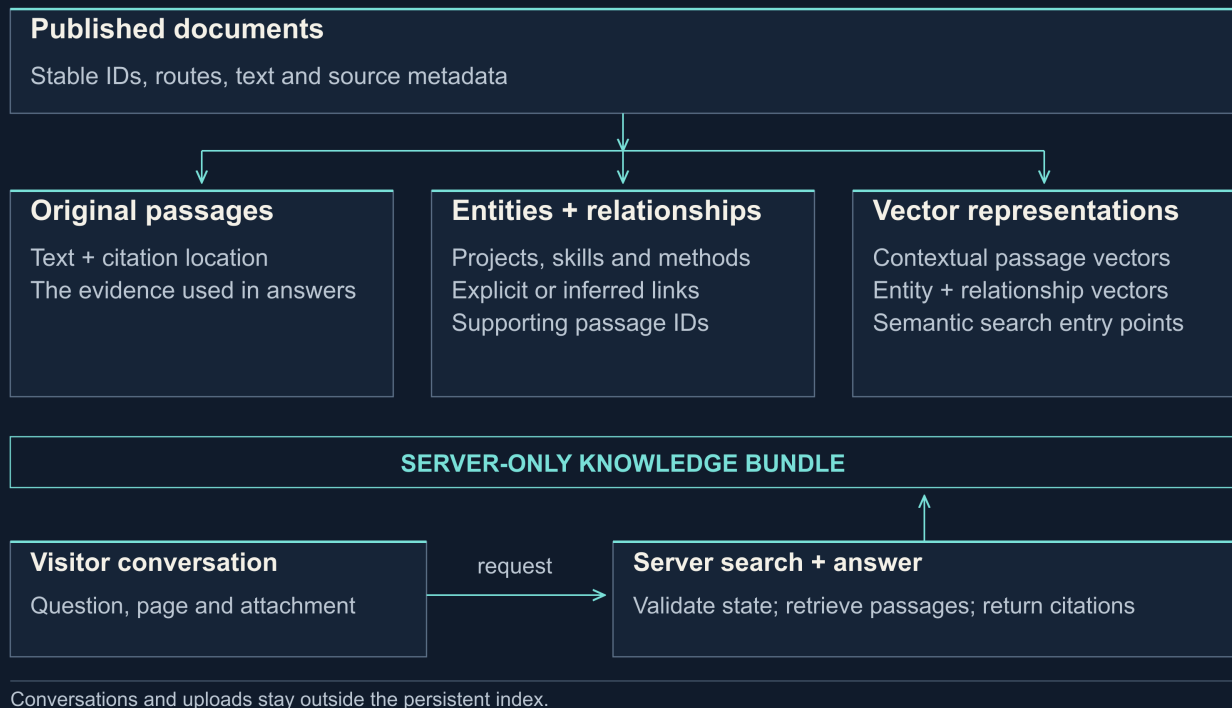


Figure 4. Published content supplies the knowledge index. A visitor's question searches it without becoming part of it.

Interface	Input → output	Responsibility
Public page read	Route → content and media	Resolve the published page
POST /api/chat	Question, history and page context → answer stream	Validate state and manage the conversation
searchKnowledge	Targeted query → original passages	Hybrid search and source registration
followRelationships	Graph starting points → linked passages	Explore directly connected relationships
Job-description attachment	Role material → attachment context	Keep visitor material separate from public content

6. Deep dives

Finding relevant experience

People describe the same work in different language. Someone asking about "reducing expert review" may need a passage about active learning. A recruiter may ask about leading a product from an early idea into use, while the case study discusses architecture, release decisions and operational problems. I wanted the assistant to find those connections without relying on the visitor to guess my vocabulary.

I combined keyword and semantic search. Keyword search uses BM25 to rank passages by matching terms, preserving exact names and technical language. Semantic search compares the question's vector

with vectors for the published passages, finding related meaning even when the wording differs. Voyage's `voyage-context-4` embeds passages in document groups, with 1,024 dimensions per vector, so their representation includes the surrounding document context.

The server combines the two result lists using reciprocal-rank fusion: passages gain weight from their position in either ranking. It then sends the strongest candidates to Voyage `rerank-2.5`, which orders them against the question. This gives exact matches and paraphrases a route into the same candidate set before the assistant reads the selected passages. Broad introductions also receive career summaries, so repeated matches on my name do not crowd out the work itself. Heading-only passages and duplicate text are filtered out.

Page context helps with shorter questions. If someone opens the Aurum article and asks, "Why did you build it?", the article gives that question its subject. The assistant carries that context into retrieval while keeping the rest of the collection available for comparisons.

A connection across two domains

A live comparison of my Oxford and Praviar work shows what this retrieval is for. The assistant brought together the project passages and the published Computer Vision & Active Learning skills record, then explained the connection with citations. It gave the reader a way to examine how experience in one setting could be useful in another.

In my [Oxford research](#), I built computer-vision methods for analysing placental microscopy, locating cells and turning image data into structured measurements. In [Praviar](#), computer vision reads compound and generic Markush structures from patent drawings. The objects and the intended use differ. The shared work is recovering useful structure from images and passing it into a system that can use it: biological analysis in one case, patent research in the other.

The citations let the reader examine that connection in the original projects.

Following relationships

A career question often moves from a project to a method, then to another setting where that method was useful. I built a lightweight graph to support that exploration. During indexing, Sonnet extracts entities such as projects, roles, skills and outcomes, together with their relationships and source passages. Entity and relationship descriptions also have embeddings, giving semantic search entry points into the graph.

At answer time, Sonnet decides whether the initial passages are enough. It can request a targeted search through `searchKnowledge` or use `followRelationships` to expand from known entities. One hop means moving from an entity to a directly connected one. A call can explore several such links; later calls can continue from the entities it finds. Each result brings back the original supporting passages.

The loop is adaptive. A simple factual question can finish from the initial retrieval; a comparison can gather more material. The server deduplicates passages, suppresses repeated searches and tracks visited edges. An empty search tells the model that it found no new supporting passages. It can try a different lead or finish with what it has. A turn can use up to five additional retrieval rounds, with two retrieval calls per round and eight model calls overall. It can gather at most 40 unique passages and 80,000 characters of evidence. These bounds leave the model room to follow a useful connection while limiting the cost of continued exploration.

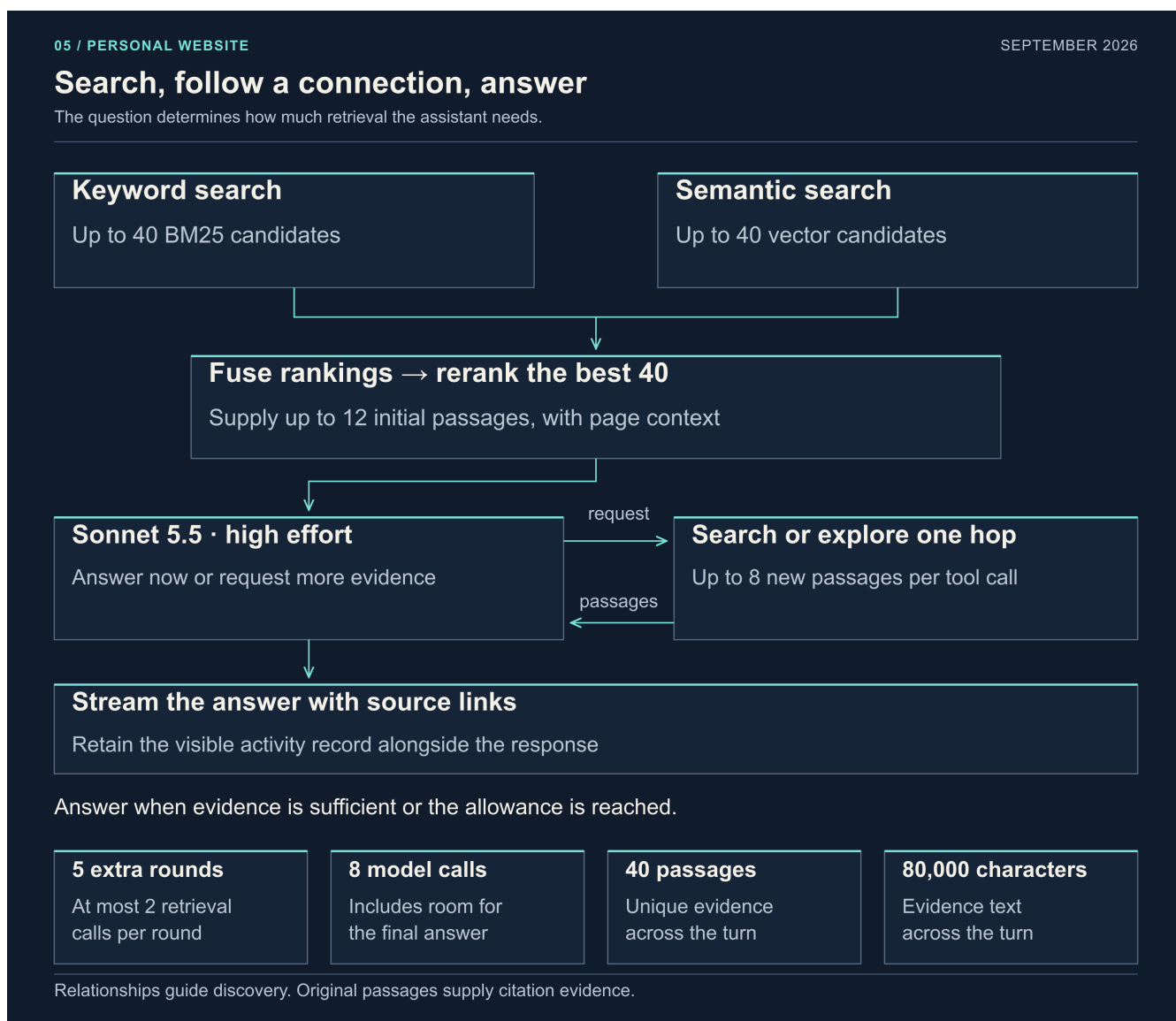


Figure 5. The assistant can answer from its first retrieval or gather more evidence through searches and successive relationship expansions.

Updating the knowledge

An unchanged rerun of the first index made no additional indexing calls. I built updates around documents so that this reuse is part of the ordinary workflow. A flat manifest records each document's stable identity, searchable-content hash and processing versions. The hash covers the text and relevant metadata. A CSS change or a new build timestamp does not cause another embedding request.

Unchanged documents reuse their vectors and extracted graph. An edit refreshes the affected document's contextual embeddings because its passages share context. Graph extraction has its own cache: changing the embedding model does not force relationship extraction, and changing the extraction configuration does not force passage embedding. Deleted documents lose their vectors and relationships. Shared entities remain while other sources reference them; their descriptions and embeddings update when their contributing information changes.

I chose a flat manifest over a Merkle tree because updates are already resolved at document level. It gives me one list to inspect and maintain. One incremental command assembles a replacement bundle after

indexing succeeds, retaining the previous usable bundle if the run fails. Ordinary application builds consume that bundle without paid indexing calls.

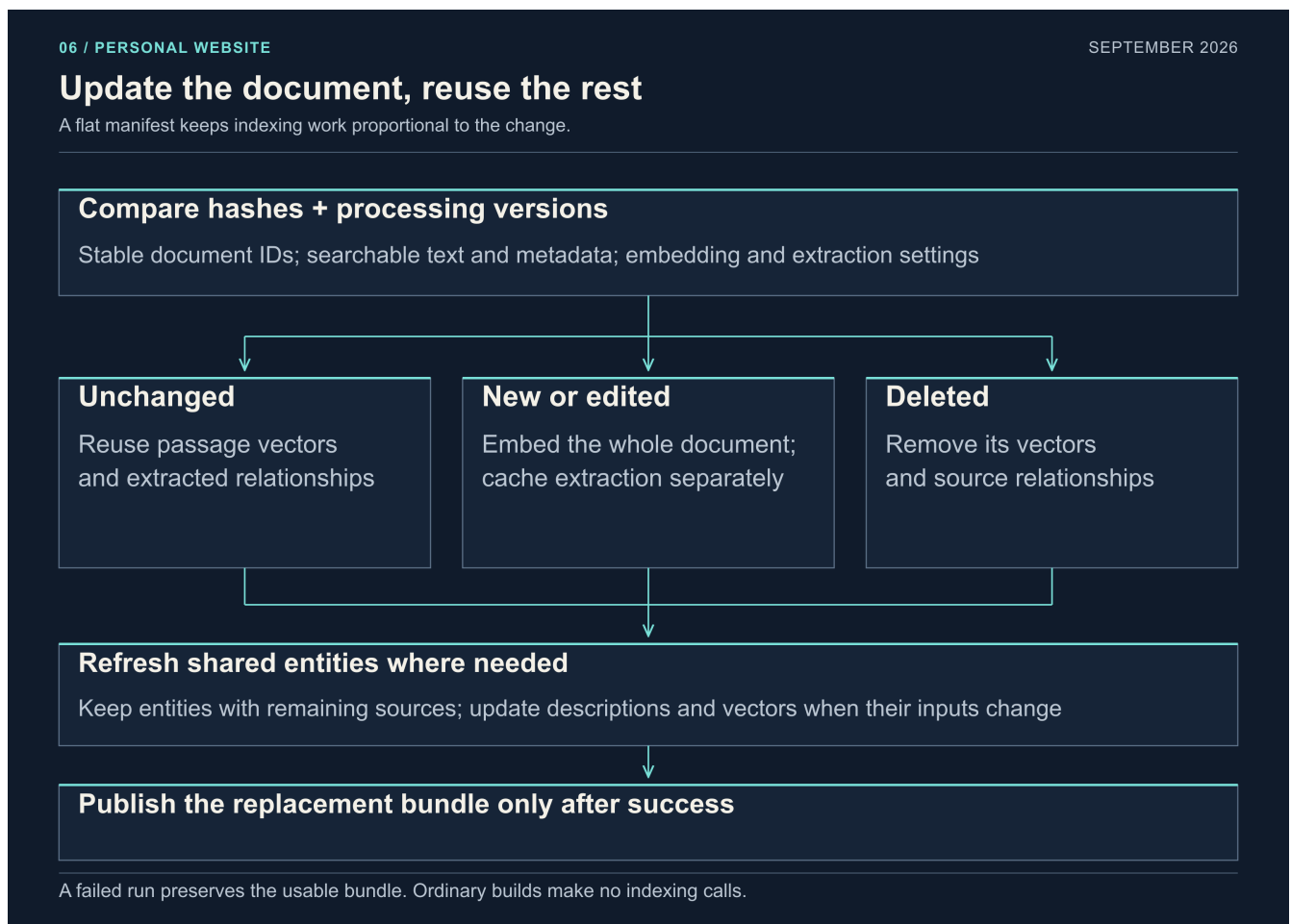


Figure 6. Document-level change detection keeps routine updates small while preserving contextual embeddings.

Delivering the answer

Progress and sources

The activity display gives the wait some character. "Same idea, different words" labels semantic search; "Following a useful thread" labels graph exploration. Initial retrieval happens before the answer stream opens: the visitor first sees a general retrieval indicator, then its completed search steps. Further searches and graph expansions stream their status as they happen. The expandable record keeps source previews, relationship details and any fallback states alongside the answer.

The server supplies a source registry before answer text and updates it as retrieval finds more passages. The interface uses that registry to display numbered links and a single source list, without relying on the model to call a citation tool. On a follow-up, the server reloads passages referenced by authenticated earlier answers from the selected collection. Signed assistant turns preserve their origin and position in the conversation; source display remains separate from that signed text.

The interface records whether an answer completed or was interrupted. Once an answer ends with suggested follow-ups, the server stops instead of making another model call for a second closing paragraph. A visitor can stop generation, scroll back without being pulled down by arriving text, or return to the latest response. When history reaches its limit, the interface offers export and a fresh conversation.

Privacy and spending

Job descriptions are reference material for the question. Provider credentials stay on the server, and visitor conversations remain outside the public index. The chat explains that Anthropic processes the conversation and Voyage processes retrieval queries and reranking passages. Generated Markdown images render as text, preventing an image URL from silently sending conversation content to an external host. Links to sources remain available for the visitor to open.

The server accounts for the whole conversation turn, including additional model calls and Voyage requests. It checks rate and concurrency limits, reserves an allowance before generation and budgets Voyage requests separately. Completion and cancellation use the same settlement path. Accounting and capacity release are attempted independently, so a failure in one does not prevent an attempt at the other. If usage is unknown, the reservation remains counted. Stopping the browser stream does not erase the cost of work already sent to a provider.

Rendering the nebula and cosmic scene

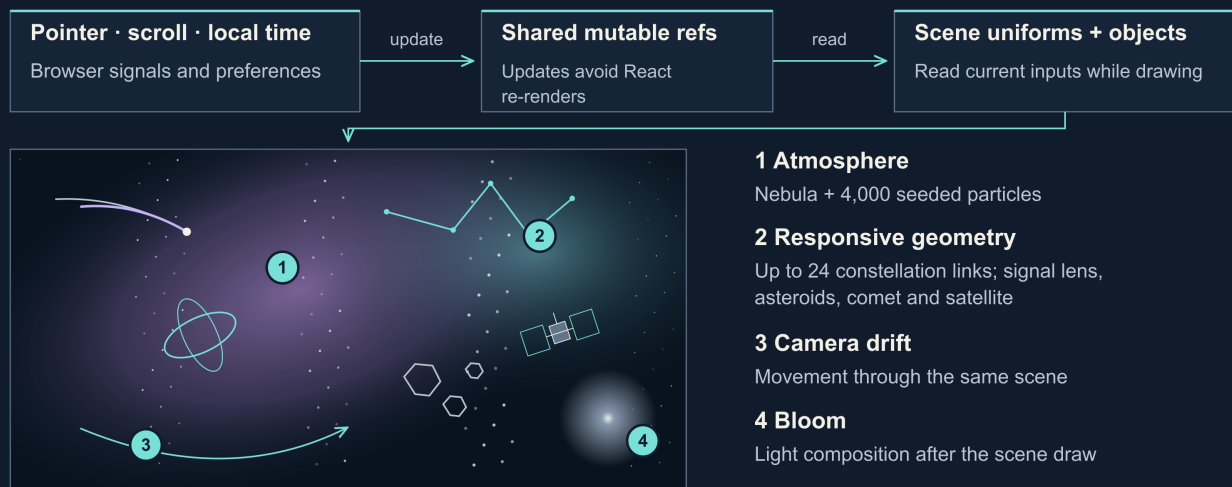
The background gives the site its visual character. I kept the full composition on phones and desktops and adapted its drawing cost to sustained frame delivery. The scene contains nine elements: nebula, particles, pointer constellations, a signal lens, asteroids, comet, satellite, camera drift and bloom. Objects appear at different scroll positions, the camera drifts through the scene and bloom softens bright areas.

The nebula is a full-screen plane whose GPU shader calculates cloud colour at each pixel. Three evolving noise fields each contribute five scales of detail. The first two distort the coordinates sampled by the third, producing the folded cloud shapes. Scrolling shifts the palette from rose and amber towards teal and violet. A vignette darkens the edges, and a brightness adjustment restrains the strongest glow. Portrait screens change the cloud proportions and drift; smaller objects take positions relative to the visible frame.

The 4,000 stars start from a seeded layout, with their shader supplying drift and scroll parallax. Up to 24 lines respond to the pointer. React manages structure and lifecycle, while shared references and shader uniforms carry frequent updates such as time and pointer position. Continuous animation stays outside React's component-render cycle.

Keep the scene; adapt the cost of drawing it

Inputs shape one complete scene. A feedback loop changes its resolution and pacing.



Schematic view of the scene and its contributing elements.

QUALITY FEEDBACK · ALL NINE ELEMENTS REMAIN

Start at native pixel ratio (minimum 1), or the saved orientation setting.

Draw → observe delivered frames → adjust one step → draw again

2 poor windows

Reduce pixels first

DPR -0.25, down to 1;
then 60 → 45 → 30 FPS

4 stable windows

Restore frames first

30 → 45 → 60 FPS, within cap;
then DPR +0.25, up to native

Otherwise

Keep current settings

Also hold during cooldown
or at the applicable limit.

Five-second cooldown between adjustments.

DPR = device pixel ratio. FPS = frames per second.

Hidden tab

No scheduled draw

Reduced motion

Slower; cap 30 FPS

Article focus

Renderer unmounted

Renderer failure

Reading stays available

Controller settings and transitions from the implementation.

Motion preferences and renderer failures have separate controls.

Figure 7. The controller keeps the scene's composition while adjusting the cost of drawing it.

The controller gives up sharpness before smoothness. Sustained poor delivery lowers resolution first, then the frame-rate target; recovery restores frame rate first. Warm-up and cooldown periods stop it reacting to every brief disturbance. A new session starts at native pixel ratio, with a floor of one, and later loads can restore the saved setting for that orientation. The scene stays intact as drawing cost changes.

Graphics load separately from the reading interface and have import retries, an error boundary and WebGL context-loss recovery. Reduced motion slows individual effects and caps the frame target at 30 fps. Article focus mode removes the renderer and surrounding navigation entirely, leaving the reading surface and an exit control.

There are smaller discoveries too: a console greeting with a Cmd+K hint, a tab-title flourish and an optional cinematic welcome. The photographs and films make the site personal in another way. I made

the family-photo film for my gran's 85th birthday and left out animations that changed the person's likeness. Visitors can watch it and read about those choices alongside the software case studies.

Keeping page delivery independent

In the publishing extension, I originally placed page content and assistant evidence in one immutable object. That kept a release together, but coupled page loading to knowledge size. I split it into a public-render pack and an assistant-knowledge pack, joined by a manifest identifying the same release. The page reader can load its content without fetching the knowledge object.

A local synthetic-growth check confirmed that pages loaded without reading the knowledge pack. Assistant preparation fetched that separate pack and exposed the cost of loading and validating it from cold. The split let me see which path carried that work, rather than treating knowledge growth as a general page-loading problem. The historical measurements live in the supporting notes.

Publication writes and verifies a new release before switching the active selector. Readers keep receiving the stable release during that work. A signed page receipt identifies the edition the visitor opened, so a question can use that edition even after a newer one is published. The assistant loads the matching knowledge pack and uses the bundled vectors and graph only if they match its content. Otherwise, it searches the selected edition lexically. An unavailable or incompatible source edition prompts a refresh; it is not silently replaced by the newer page.

7. System-design decisions and trade-offs

Keeping search inside the application means carrying the index in the deployment and paying its memory cost in the server process. A separate search service would move that work elsewhere and add another dependency to operate. For this collection, I prefer the local bundle: I can inspect the source records, regenerate it with one command and know which index ships with the application.

Hybrid retrieval adds provider calls before generation, and each further search or graph expansion can extend the wait. I accept that cost for questions that need connections across the work. A short factual answer should finish from its initial evidence. The model chooses whether another retrieval is useful within the allowance; the server limits how much work it can request.

The fallback choice depends on what failed. Losing semantic search reduces the assistant's retrieval options, so it continues with keywords. Losing the spending authority removes the basis for approving another paid call, so chat stops. I accept that loss of availability to keep a public assistant within its operating budget.

Failure	Visitor impact	Response
Embedding service unavailable	Search loses semantic matching	Continue with BM25
Reranking fails	Candidate order is less refined	Use the fused rankings
Model or answer stream fails	Failed or partial answer	Show the state and settle usage
Budget or capacity authority fails	Chat cannot start	Refuse paid generation; browsing continues
Graphics fail	Background is absent or recovering	Retry or recover the renderer
Remote content cannot be read	An earlier edition may be served	Try an earlier release or packaged content if eligible
A pinned edition is unavailable	Its question cannot proceed	Ask the visitor to refresh
Conversation grows too long	A new conversation is needed	Offer export before restarting

Figure 8. Search can fall back to keywords; a failed spending check stops paid generation.

The cosmic scene is a different compromise. I spend browser rendering capacity on a visual identity, then adapt resolution and pacing and give the reader a focus control. Independent publishing costs me storage, manifests and release handling in exchange for activating and rolling back content separately.

8. Live monitoring and improvement

PostHog helps me understand how people explore the site: which pages and sections they reach, whether they watch a film, and where they open the assistant or choose to get in touch. Assistant events include entry method, question length, citations and feedback without the question text. Browser analytics state stays in memory, with person profiles and autocapture disabled. Conversation retention follows its own policy.

Vercel instrumentation and server traces cover web vitals, failures, graphics recovery and aggregate model, token, tool and latency information. Deployment records identify the application version involved. These observations answer different questions from product analytics: whether an answer completed, where the time went, or which rendering settings a device reached.

A live visitor check exposed a problem at the end of a long graph answer. Retrieval had gathered its material, but the response was cut off during completion and the next turn lacked its signed history. An old provenance receipt still allowed only 20 source identifiers, while hybrid retrieval allowed 40 passages. The answer reached a boundary that had not been updated with the retrieval system.

I raised the receipt limit to match the retrieval allowance and checked complete output and signatures with different numbers of passages. The live retest returned a full answer using 33 passages, three adaptive graph calls and seven rendered citations. A follow-up in the same conversation also completed with signed history. That repair depended on finding the mismatch between retrieval and delivery; changing the search ranking would have left it in place.

I use the same distinction when investigating other problems. A missing passage points me towards retrieval; a misread passage points towards generation. Graphics recoveries lead me to the affected route and device conditions. In the chat, the activity record and source links let the visitor inspect the searches and material behind the answer too.

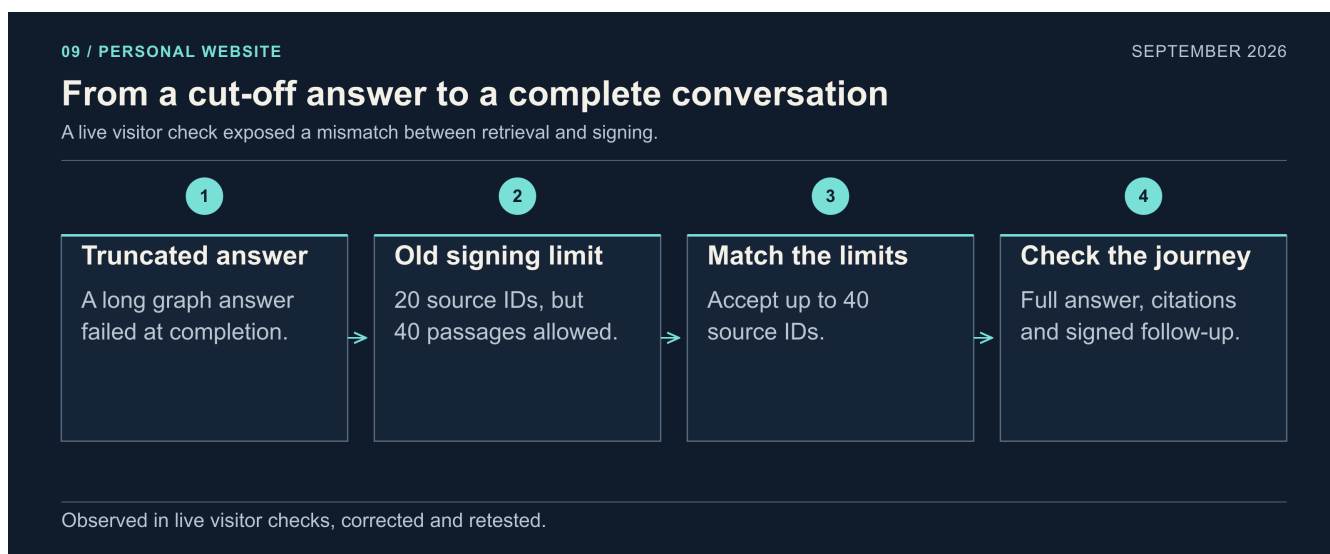


Figure 9. An outdated signing limit cut off a long answer. The corrected release completed the answer and its follow-up.

9. CI/CD and deployment

For the September assistant releases, I built a production candidate and checked its output and health at a fixed deployment URL. I then promoted that same build through Vercel's rolling-release controls. The previous deployment stayed available while I prepared and checked the candidate.

Live visitor checks exercised navigation, mobile interactions, citations and multi-step conversation. The code review added focused regressions for the defects it found, including the completion failure described above.

The repository also has CI workflows for code quality, architecture, tests and builds, with browser and graphics checks alongside them. An owner-triggered preview workflow can check an exact revision in a protected project. Those are available paths for checking changes; the releases described here used a checked production candidate and manual promotion.

Application versions and content editions have separate identities. Deployment IDs help an open page continue talking to its matching application version through [Skew Protection](#). Content receipts identify a source edition where release-pinned retrieval is used. Keeping those identities distinct makes deployment and publishing problems easier to diagnose.

I advance [rolling releases](#) manually and retain rollback to the previous deployment.

10. How the design would change

If loading the index or scanning its vectors became a substantial part of the wait for an answer, I would reconsider the local bundle. Splitting the index into relevant subsets would be one option; moving vector search into a dedicated service would be another. Either change would preserve document-level update tracking and the original passages used for citations.

Sustained conversation traffic would create a different pressure. If requests regularly waited for provider capacity, I would examine those waits separately from retrieval and generation time. Repeated provider outages would make an automatic circuit breaker useful: it would stop new calls after repeated failures and probe for recovery before resuming. A gateway would earn its place if several applications needed the same traffic and spending policies. This site keeps those controls in the application and uses one generation model throughout.

I want someone considering working with me to see how I make decisions: what I preserve, where I accept a cost and how I respond when something breaks. The full scene on a phone, the original passages behind an answer and the likenesses in my gran's film all involve choices I can explain. The website gives those explanations room, and lets the visitor use the result.