ZANGO

Building AI agents for multi-jurisdiction compliance

A practical workshop on agentic workflows for regulated teams

01 Introduction

02 Building an agentic workflow

03 AI impact analysis in action

04 Lessons from the field

About us

Ashi Bajwa

Regulatory AI Architect

zango

Sam Green

Senior Policy & Partnerships Lead

zango

Zakaria Ouraich

Head of UK Compliance

LaSalle

Tim Tyler

Vice President

ICA

The anatomy of a good AI use case

High-volume / repetitive

Clear source of truth

Defined output structure

Auditable

The problem with horizon scanning

Fragmented inputs

Noise vs. signal ratio

Interpretation bottleneck

Implementation Insights

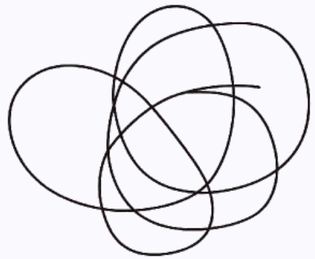
Horizon scanning as an AI use case

- High volume and repetitive
- Clear source of truth
- Defined output structure
- Auditable

POLL

How much are you leveraging AI for Horizon Scanning today?

Let’s build a Horizon scanning impact agent



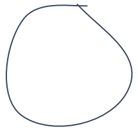
Case Study - Meridian Asset Management

Central monitoring across jurisdictions

Asset manager headquartered in the UK, operating across multiple countries, running central horizon scanning and routing obligations to the right jurisdictions.

A noisy feed that isn’t useful

The current horizon scanning feed is high-volume and noisy. It flags a lot but surfaces little that teams can act on.



Uplift into full impact analysis

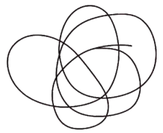
Turn that raw feed into full impact analysis, mapping each change to the impacted entities, departments and policies.

Questions

Limitations to be aware of

Configurability

Hard to scale when you are working with hundreds of updates across multiple entities and jurisdictions, and languages



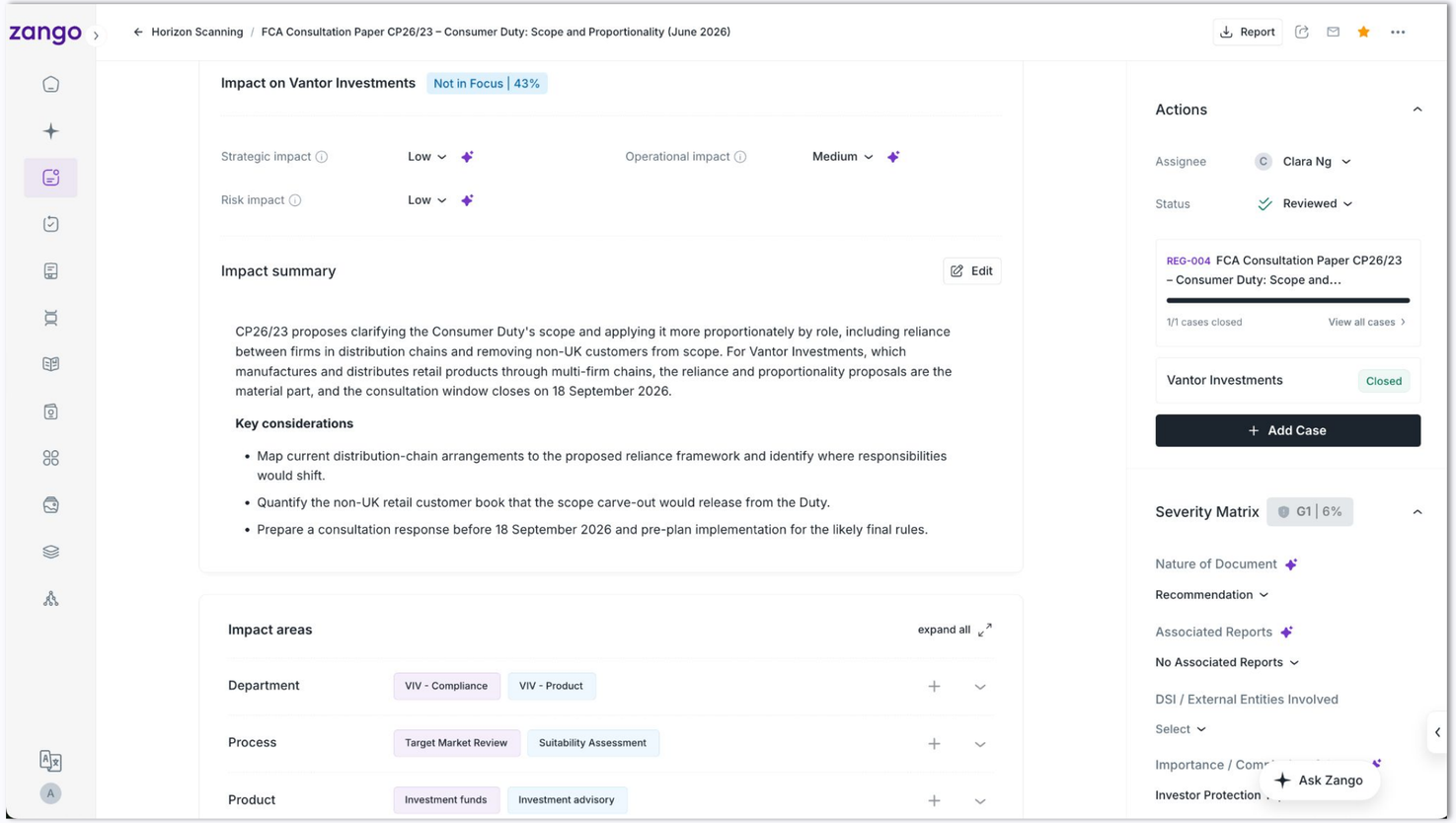
Context Management

Keeping up with the organisations changing context remains very manual, as well as there being no way to embed feedback loops and reinforcement learning

Transparency and Reporting

Difficult to maintain audit records, and reporting of impact analysis, and management oversight

AI impact analysis in action



POLL

Which agent would save you the most amount of time in your day to day?

INDUSTRY INSIGHTS

Lessons from the field

Lessons from the field

01 Move beyond the chatbox

The value is in agents that run a process end to end, not one-off prompts.

02 Compliance has to get operational

The second line can no longer sit apart from the business it oversees.

03 Treat adoption as a portfolio of experiments

Run adoption as measured experiments, not a token-cost procurement exercise.

04 Compliance by design

Governance written into the build from day one, not reviewed in afterwards.

Compliance by design - KYC / AML Agent

Humans own every decision that matters

Maker-checker sign-off on every case; every screening hit goes to the MLRO.

Audit trail by default, not retrofit

Hash-chained log of every action, each input traced to source, page and quote.

Deterministic rules first, AI judgement last

Hard rules in code; AI judgement only for the ambiguous cases.

Prove it in shadow before production

Shadow-run against real cases; go-live only on documented sign-off.

Private markets compliance platform

The concept it's growing into: one platform for lean private-markets compliance teams, with six modules on a shared evidence layer, resolved once and reused across monitoring, reporting and DDQs.

KEY RECOMMENDATION

Start small, in your own function - automate the workflow you already own and can supervise.

SOME SIMPLE IDEAS

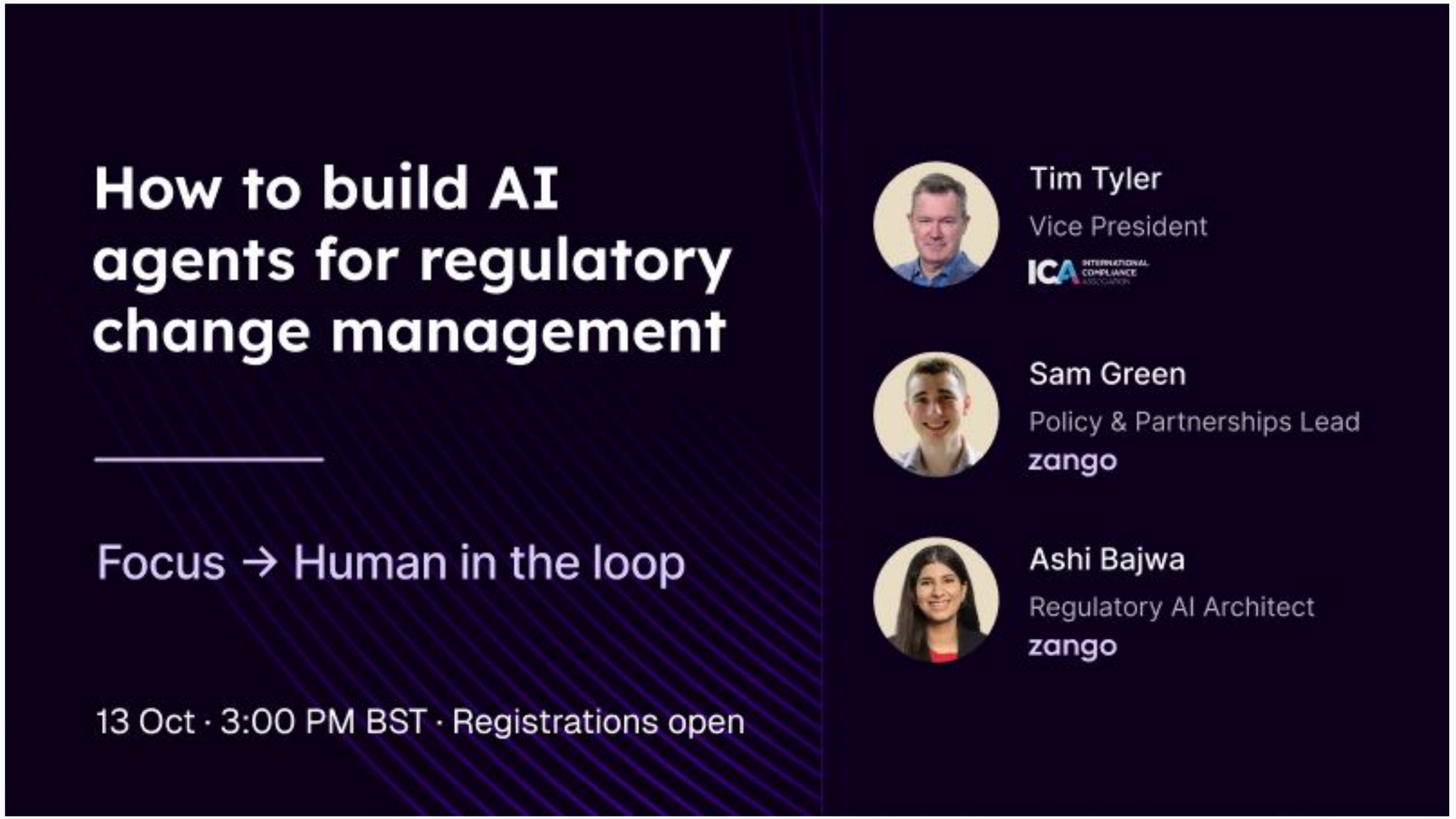
Extract mandate and fund restrictions for live portfolio compliance checks.

AI powered document classification and storage, enabling easy retrieval of documents

POLL

What compliance AI use case should we build next?

Questions


How to build AI agents for regulatory change management
Focus → Human in the loop
13 Oct · 3:00 PM BST · Registrations open
Tim Tyler
Vice President
ICA INTERNATIONAL COMPLIANCE ASSOCIATION
Sam Green
Policy & Partnerships Lead
zango
Ashi Bajwa
Regulatory AI Architect
zango

Contact us

Ashi Bajwa

Regulatory AI Architect, Zango

ashi@zango.ai

Sam Green

Senior Policy & Partnerships Lead, Zango

sam@zango.ai

RESOURCES

Agentic build prompts

The brief and files parsing structure

The brief

Build me an interactive artifact called a horizon impact agent.

It takes one regulatory alert and works out what it means for my organisation, through a short pipeline of boxes I can open, read and edit, laid out on a canvas I can pan and zoom. I am going to run it in front of a room of compliance people, so every box has to show its working, and the whole thing has to survive me uploading a real spreadsheet live.

Do not prefill anything. The page opens with no data in it. Nothing is embedded at build time, no alert is pre-selected, no policy is pre-chosen, and you are not waiting on files from me before you start. I load them from the page itself, in front of the room, so build it empty and make every box say which file it is waiting for.

Work through this in order.

1. Build the parsing for real files, sight unseen

You are not getting my files now. I load them in the room, so the parsing has to be right first time and has to explain itself on screen. Build for all of this, and show what you found in the panel of each box after every upload — which tab, which header row, which identifier column, how many rows — rather than swallowing it.

The alerts export. Do not assume the alert list is the longest tab. These exports carry cover, summary and pivot tabs, and the pivot tabs are usually *longer* than the alerts themselves, with regulator names and repeated "Title" rows interleaved as sub-headers. Choose the tab whose columns look like an alert list: it must have a title column, then score on how many of these it also has — summary, published or date, source or regulator, document type, alert type, jurisdiction, topics, status — with a strong bonus for a tab named something like Alerts. A pivot tab will legitimately score well on the shared columns; the summary, jurisdiction and topics columns plus the name bonus are what separate them, so weight accordingly and show me the score you gave every tab, in the page, so the room can see why the Alerts tab won.

Detect the header row. A title often sits above the real headers, so take the widest of the first four rows rather than assuming row one. Match column names case and punctuation insensitively.

Drop rows whose title is empty or literally "Title".

Excel dates are not strings. Date cells arrive as Date objects or as serial numbers counted from 1899-12-30. Convert both to YYYY-MM-DD at parse time. If you slice a Date object as a string you get Mon Jul 20 and every date on screen becomes NaN . Check the conversion across every row, not just the first, and say how many cells did not convert.

The organisation records. Entities and departments may arrive as their own files, as tabs in one workbook, or as a folder of files, and I will not tell you in advance which. Find the tab by name, fall back to reading a single-tab file or a CSV whole, and keep every column: I would rather an agent read a field you did not understand than lose it. Find the identifier column on each (something like ENT-11 or DEP-10) and the column on the departments that names the parent entity. Those identifiers are how the boxes refer to each other, they have gaps in them, so preserve them exactly and never key anything by row position.

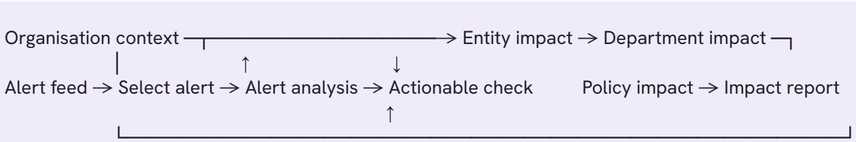
The policies. Two different things, both mine: a register (one row per policy, with an identifier, a name, an owner and the entity that owns it) and the documents. Split each document on its numbered headings into {number, title, text} sections, keeping parent headings that have no body of their own so the numbering stays honest. Watch for a table of contents: it repeats every heading with no body and will double your clause count if you let it.

Where a file does not match what a section expects, say so on that section and name what is missing, rather than filling the gap or failing silently.

The boxes and organisation context

2. The boxes

Nine boxes, laid out as a pipeline on a canvas with lines drawn between them:



Organisation context is step one and everything about my organisation lives in it. Alert feed and Select alert are step two: the feed holds every alert, and Select alert shows them as a tick list with date, source and type. More than one can be ticked but everything downstream reads one, so label the row the steps actually read. Nothing is ticked when the file loads.

Each box is a small card showing, on its face: a status dot, what kind it is, its result once it has run, and its elapsed time. Each opens on double click into the inspector panel described in section 7. Single click selects without opening, so a box can be selected without the panel jumping.

3. Organisation context, in four parts

One box, four numbered sections in its panel, because these are four different things and a room understands them one at a time:

Entity website — a link field. Passed to the agents as context; say on the panel that the page cannot fetch it.

Entity breakdown — one legal entity per row with its jurisdiction, regulator, entity type and permissions.

Departments — one department per row, with a column naming its parent entity id.

All policies — the register and the documents, together.

Each of the three file sections takes **a folder as well as a file** , dropped or chosen, and several files are merged into one reading on column names rather than position: one file per entity, per region or per quarter should work as well as one combined list. Report what you merged, from which file and tab, how many rows each contributed, and drop duplicate identifiers with a count. A folder of policies holding both the register and the documents should sort itself out.

The register is the source of a policy's name and its owning entity. Match each document to a register row on its file name and opening lines, discount words that appear all over the register, and expand the obvious compliance acronyms so …_Cayman_AML_Policy.docx finds "Anti-Money Laundering & CFT Policy". Then show the register's name, never a title scraped out of the document. Say plainly when a document matched nothing, and keep the file name as its title. If I give you documents and no register, derive a register from them and mark it derived.

The agentic boxes

4. The five agent boxes

Each has its prompt in an editable textarea, its own run button, and a collapsed "show exactly what gets sent" panel holding the full assembled input: prompt, alert, records, and what earlier boxes returned. That panel is the most useful thing on the page for anyone building their own, so do not summarise it.

Alert analysis reads the alert alone with no knowledge of my organisation. Returns what the instrument changes, who it binds in the text's own words, the obligations it creates, any date it names, and what kind of document it is. It must not invent a date or an obligation, and when a document creates no duties it must say so.

Actionable check takes the analysis and rules the alert actionable or informative, with a short rationale and the ids of the obligations it relied on. **No task list** — it is one judgement, and a list of invented tasks buries it. Every obligation id it cites must be one the analysis actually found. If the analysis found none, the verdict is informative, whatever the model says. Most of a feed is informative and the box should say so plainly. It names no department and no entity.

Entity impact receives every legal entity and decides which the alert reaches, judging on each entity's own jurisdiction, regulator, entity type and permissions. An alert from one regulator does not reach an entity supervised by a different one unless the text says so. Each entity gets direct, conditional, monitor_only or unsupported, plus three factors scored 1 low, 2 medium, 3 high: strategic, operational, risk. unsupported and monitor_only must be 1 on all three. Make it name the field on that entity's record that decided it.

Do not ask the model for an overall rating. Compute it. Weight the factors 30, 30 and 40, divide each level by 3, sum and round. Raise it to at least 50 when risk is 2 or more, any factor is 3, or two or more factors are 2 or more; otherwise cap it at 49. Tell the model explicitly not to mention a rating, because a summary written for one rating is wrong the moment a reviewer changes it. Show the number and the three factors, and no "in focus" or "out of focus" label anywhere — the band and the number already say it.

Department impact receives the entities that survived, and every department. Each match names its parent entity id, and that entity must be one the entity box kept. Where my data shows a function is delegated to an investment manager or administrator, the match must say whether the entity performs the work or oversees a delegate who does. For a fund or any outsourced operating model that distinction is the answer.

Policy impact is one box that makes **two model calls**, in this order, and says so on its face:

Call one: which policy. It reads the register and returns the policies this alert touches, each with the identifier exactly as the register gives it, a priority, why the alert reaches it and what a reviewer should check. It may only return identifiers that appear in the register. It does not name an entity: the register already says which entity owns each policy.

Call two: what the impact is. It reads the clauses of the highest-priority chosen policy that has a document loaded, and returns the clauses that move, each with its number quoted exactly as it appears, amend or review , and what that clause would have to say instead. It must not invent a clause number or cite one that is not in the document.

Give each call its own prompt and its own assembled-input panel, one run button for the box, and a status line that names the stage: "waiting on Claude, choosing the policy (1 of 2)", then "reviewing 40 clauses of Anti-Money Laundering & CFT Policy (2 of 2)". Let me switch the review to another of the chosen policies in one click. If the first call chooses policies but none has a document loaded, fail with those identifiers named, and still show me what the first call chose.

Every prompt ends with Return strict JSON: followed by the exact shape, and the page parses that shape. Validate what comes back against my records before you render it: drop clause numbers that are not in the document, policy identifiers that are not in the register, and departments whose parent entity was ruled out, and show those rejections rather than hiding them. If a reply will not parse, show me the failure code, what it means in plain words, and the raw text, rather than a blank panel.

Impact, behaviour and presentation

5. Impact report

Makes no model call. It joins the identifiers the boxes already returned, and it reads **top down, in that order**:

> **entity → its departments → its policies → the clauses that move in each policy.**

Each entity in scope carries its band, percentage, three factors and narrative, then the departments the alert reaches, then the policies impacted with their register identifier, name, priority and why they were chosen, and beneath each policy the clauses that move. A policy that was chosen but not reviewed sits in its place saying so, and why: no document loaded, or one policy is reviewed at a time. Entities that were ruled out go in a separate block with their reasons. Anything naming an entity the entity box did not keep is listed as not attached rather than given an invented branch. Say on the panel that it makes no model call, because that is the point.

Download PDF on that box builds one A4 document and saves it, in the same order and the same nesting: the alert with its metadata and summary, what it changes with the obligations, the actionable verdict, then each entity with its departments, its policies and their clauses, then the entities ruled out and anything rejected. No flat list of policies competing with the nesting. Wrap text properly, break pages, and put a running header and page numbers on every page.

6. Behaviour

Boxes run in dependency order and stop at the first failure. A running box must look like it is working: an animated stripe, its elapsed time ticking live, its status dot pulsing, and a status line saying what it is waiting on — "waiting on Claude, 8 KB sent, tier quick", then "reading the reply, 2,400 characters". A step that takes eight seconds silently looks broken.

Every data box takes a drop as well as a click, and dropping a file on the Organisation context box on the canvas routes it to the right section by type and name. CSV natively; for Excel load SheetJS from cdnjs, and for Word load mammoth from cdnjs. Load them when a file is picked rather than at page load, and if either fails to load, say so and tell me what to do instead rather than failing silently. After an upload, tell me exactly what you found — which tab, which header row, which id column, how many rows — and invalidate the boxes downstream of it.

Use the fastest model tier — this is short classification work returning small JSON — and say which tier was asked for rather than naming a model.

7. Runtime, and the thing people get wrong

The page asks Claude directly and offers a file to save, so it needs both capabilities declared when you publish: sample and downloads. **A brand new artifact starts with no declaration, so a page that calls claude.use("sample") without declaring it resolves null and silently does nothing.** Declare them on the first publish, not the second.

Obtain each with await claude.use("..."), treat null as unavailable, and design for absence: hide the save button, and if Claude is unavailable say so on the boxes and in the status bar instead of showing empty output. Resolve it lazily and never block first render on it.

8. How it should look

Build it as a canvas tool, not a document. Three columns, full height, no page scroll:

A left rail, 228px , on a warm off-white ground one step darker than the canvas. The name of the thing at the top; then **Add a box** , a palette listing all nine box types with an icon, a name and a one-line description, greyed out when that box is already on the canvas; then **Steps** , the boxes currently on the canvas with a status dot, elapsed time and a remove × that appears on hover, so I can drive the whole pipeline without hunting on the canvas; then a short legend for the colours.

The stage in the middle. A thin top bar carrying the workflow name, a pill for the model tier asked for, a pill for whether Claude is available, and a reset. Below it the canvas itself: a dotted grid background, 1px dots on a 24px pitch. The boxes sit at fixed positions in a world layer that the camera transforms. A zoom bar pinned bottom-left — minus, the percentage, plus, fit-to-view. The run button and a live status readout pinned bottom-centre.

An inspector on the right, 424px, that I can close. It starts closed so the page opens on the whole canvas. Double clicking a box or clicking a step in the rail opens it; a × in its header, or Escape, closes it; the canvas refits to the space when it opens and closes. Two tabs, Configure and Output.

Presentation

8. How it should look- Continued

Palette: a warm off-white page #FDFCFA, a slightly warmer surface #F8F6F1 for the rail and table headers, white cards, warm grey hairline borders #E8E4DD. One accent, a deep violet #4E1597 , used only for selection, primary actions and completed wires. Semantic colour only where it means something: amber #D97706 for a clause to amend and a high priority, green #059669 for a clause reference that exists, red #DC2626 for a failure. Borders and tints rather than drop shadows, no gradients. Light and dark, both properly designed, with every colour defined as a token on bare :root first.

Type: Libre Baskerville for the product name, panel headings and the score figures; Geist for the interface; Geist Mono for prompts, identifiers, the assembled context and every number in a column. Dense: about 13px body, 12px in tables, a tool rather than a landing page. British English, sentence case, no emoji. Small outline icons at 1.5px stroke, one per box kind.

Let me take the canvas apart and build it back up. I demonstrate this live, and starting from an empty canvas and adding boxes one at a time is how the point lands. On the Steps header put **Starter** , which restores the whole flow and is disabled when everything is already there, and **Clear** , which arms on the first click — it becomes a red "Sure?" for three seconds — and empties the canvas on the second. Adding is one click in the palette; removing is the × on a step.

The canvas holds a subset of the nine, and everything else follows from that: draw a wire only when both of its boxes are present; when a box is removed, throw away its result and the results of everything downstream of it, and close the panel if it was open; keep the camera framing what is actually there. A box whose upstream is missing must refuse to run and say which box it needs and where to get it, on its face and in its panel — never fail silently and never run on stale input. A box whose upstream is present but empty must say which file it is waiting for, and name the box that holds it. An empty canvas gets a card in the middle of the stage saying so, with a button to put the starter flow back, and the run button disables itself.

Build the pan and zoom camera. Do it the way people expect from a canvas and it costs you forty lines:

A #viewport filling the stage with the dotted background and touch-action:none, containing a #world with transform-origin:0 0 and will-change:transform. The camera is {x, y, s} and one translate(…) scale(…) on the world. The wires are one SVG inside the world, sized to the content bounds, so they transform with it.

Dragging the background pans, and the cursor is grab / grabbing . A drag that starts on a box does not pan.

- A plain wheel or a two-finger trackpad swipe pans; ctrl or ⌘ with the wheel zooms, because that is the event a pinch arrives as. Zoom keeps the point under the cursor fixed. One finger pans and two fingers pinch on a touchscreen.
- The zoom bar buttons step about 15% from the centre of the view, fit-to-view frames everything with padding and reserves the height of the top and bottom bars, and the percentage is always shown. ⌘= , ⌘– and ⌘0 do the same. Clamp the scale between 0.25 and 2.
- Fit on load, and refit on window resize and when the inspector opens or closes — but only until I move the camera myself, after which leave it where I put it. Fit-to-view re-arms that.

One structural warning. If a panel needs to scroll, every ancestor must be height-bounded *and* allowed to shrink, or it silently will not: bound the shell with grid-template-rows: minmax(0,100%), put min-height:0 on the children, overflow:hidden on the columns, and min-height:0 alongside overflow:auto on the scrolling pane. Miss any one and nothing scrolls. Wide tables go in their own overflow-x:auto container with a min-width, so they scroll sideways instead of crushing text.

Handover

9. Check it before you hand it over

Build yourself test files that look like the real ones (a workbook with cover, summary and pivot tabs around the alert list; entities and departments with gapped identifiers; a policy register and two Word policies with numbered headings and a table of contents), run these against them, and tell me the results and anything you could not check:

- The page opens with no data: every box says which file it is waiting for, and running the pipeline stops at Organisation context without calling Claude
- Every box runs and returns parseable JSON, and the policy box makes exactly two calls
- The entity box narrows to fewer entities than it started with, and the ruled-out ones appear with reasons
- The actionable check returns a verdict and no tasks, and an obligation id the analysis never found is rejected and shown
- The first policy call quotes only register identifiers, and a planted one that is not in the register is rejected and shown; the second call quotes a clause number that exists, and a planted number that does not exist is rejected and shown
- A policy chosen with no document loaded is reported, not silently dropped
- The report reads entity → departments → policies → clauses, and shows what was kept and what was ruled out
- **An alerts export with cover, summary and pivot tabs loads the alert list, not Cover or a pivot, and every date renders as a real date — assert that across all rows, not just the first**
- **A folder of department sheets merges into one reading, on column names, including a sheet whose columns are in a different order and one that carries an extra column**
- A policies folder loads the register and splits every document into clauses, and each document takes its title and its entity from the register row it matches
- Zoom in, zoom out, fit-to-view, wheel pan, ctrl-wheel zoom and drag pan all move the camera
- The inspector opens from a box and from the rail, and closes from the × and from Escape
- Clear empties the canvas and Starter puts it back; adding a box from the palette draws only the wires whose other end is present; a box added without its upstream refuses to run and names what it needs; removing a box mid-pipeline clears its result and everything downstream but leaves the boxes before it alone
- The PDF builds and saves, in the same nesting as the report
- No console errors

Then explain in a few paragraphs, in plain language I could read aloud, how it works and what someone would change to point it at their own organisation.

Final notes

What matters most

Three things have to come across, so build for them.

The records are the asset, not the agents. Each agent is a few hundred words and interchangeable. A spreadsheet of legal entities with their regulators and permissions is what turns a reading list into an impact assessment. Make the narrowing visible: show the entities ruled out, and why, right next to the one that was kept.

The register decides what a policy is. A policy's name, its owning entity and its very existence come from my register, not from a model reading a document and guessing. The first policy call may only choose from that list. That is the difference between an assessment and a plausible paragraph.

The report is derived, not generated. Every department names its parent entity and every policy names its owning entity, so the report is a join over identifiers the agents already wrote. Nothing can invent a relationship an agent did not assert, and that is what makes it auditable. Say so on the page.