

Google Opal.

Build × Debug × Ship.

Opal is a directed graph of prompts, models and tools with a hosted front end. This workshop covers the actual object model, the 2026 agent step, the three build paths, the console, the governance gap — and Gemini Notebook, formerly NotebookLM.

A visual workflow engine that chains prompts, models and tools — with **hosting included**.

THE VERIFIED FACTS

Launched 24 July 2025 by Google Labs, as a US-only public beta. Described by Google as "a new experimental tool."

Expanded to 15 countries, then on 6 Nov 2025 to **more than 160 countries**.

What it does: "create multi-step flows that chain prompts, model calls, and tools together." You describe it in natural language; Opal builds an editable visual workflow and a mini-app.

Hosting: "Opal handles all the hosting for you, no web servers required." Apps are stored as files in your Google Drive.

Editor: optimised for desktop. Mobile can use or view an already-created app, not build one.

WHAT IT IS NOT — SET EXPECTATIONS NOW

Not a production platform. Google's own FAQ opens with: "Opal can make mistakes so it's important to check your prompts and test your app." It is a Labs experiment with no SLA.

Not a public API. There is no documented programmatic endpoint to invoke an Opal. Integration happens through the shared app link, not a REST call.

Not a replacement for n8n / Zapier / Cloud Run. No cron triggers, no webhook ingress, no queue semantics documented. Opal runs when a human opens the app and presses start.

Treat it as: the fastest path from idea to a shareable, working AI prototype.

An Opal is a **directed graph**. Nodes are steps, edges are references, and the prompt is the wiring.

THE CANONICAL SHAPE

```
// every Opal reduces to this

[User Input] collect from the human
|
▼
[Generate] model + prompt + tools
| ↳ may fan out / fan in
▼
[Output] webpage / Sheets / ...

// plus, attached anywhere:
Assets — static reference (files, YouTube links)
Tools — invoked with @ inside a prompt
```

Multiple output steps are allowed in one Opal. That is how you write to a webpage and a spreadsheet from the same run.

The critical asymmetry: edges are not the source of truth — the *prompt text* is. Dragging a connector edits the prompt for you. Delete the reference from the prompt and the data stops flowing, even if the line still looks connected.

FIVE PRIMITIVES • THE WHOLE VOCABULARY

01 • Step — a node. Has a name, a prompt, and a type. Runs when its upstream dependencies have run.

02 • Reference — an @ mention of another step inside a prompt. This is the actual data edge.

03 • Tool — built into Opal, exposed to the model when referenced. Web search, maps, weather.

04 • Asset — static context: an uploaded file, or a YouTube link. Best for things that don't change per run.

05 • Theme — generated, not hand-edited. You can prompt for one and switch between saved themes.

Three types. **Generate is the one that costs you** — everything else is plumbing.

TYPE 01 · USER INPUT

Collect from the human.

```
prompt shown to user:
"When is your birthday?"

advanced options:
input type → text | image
```

Key behaviour: the prompt on this step is *shown to the user* — it is UI copy, not model instruction. Write it as a question, not as a system prompt.

Advanced options let you constrain the input type (e.g. text or image), which is how you build image-in workflows.

Design rule: one concept per input step. Three separate inputs debug far better than one free-text box you then have to parse.

TYPE 02 · GENERATE ★

The primary building block.

```
select model, then prompt:
"The user's birthday is:
@[User Input]. Use
@[Search Web] to find a
historical event that day."
```

You choose the model per step. Google documents that the output type depends on the model selected — "a text response, video, or image."

This is where the model, the tools and the referenced steps all combine. The entire assembled prompt — text + tools + step references + assets — is sent to the model.

Cost and latency live here. Every Generate node is a model call. Fewer, sharper Generate steps beat many thin ones.

TYPE 03 · OUTPUT

Control what the user sees.

```
output type:
webpage · auto-layout
webpage · manual layout
export → Drive spreadsheet
```

Auto-layout hands page structure to the model. Fast, but non-deterministic between runs.

Manual layout means you specify the layout in the prompt — use it the moment output consistency matters.

Multiple output steps are supported in one Opal, so a single run can render a page *and* write a spreadsheet.

Assets are context you **attach**. Tools are capabilities the model **calls**. Don't confuse them.

ASSETS · STATIC REFERENCE CONTEXT

Uploaded once, referenced many times.

```
what you can attach
· uploaded files (e.g. images)
· a link to a YouTube video

how you use it
@ mention the asset inside any step's prompt
```

Google's stated rule: assets "are best for static content that you want to provide as context to the AI model when prompting."

Practical read: brand guidelines, a style sheet, a reference image, a source document that rarely changes. Anything that changes per run should be a *User Input* or a tool call — not an asset.

Watch the cost: an asset referenced in five Generate steps is sent to the model five times.

TOOLS · CAPABILITIES GRANTED PER STEP

Granted by mentioning them.

```
documented examples
· search the web
· search a map
· get the current weather

invoke with @ → opens the tools menu
```

The mechanic that matters: Google states tools "are made available to the model along with the prompt for each step. Including a tool into a prompt gives the model access to that tool."

So tool access is per-step, not global. A step that doesn't mention `@Search Web` cannot search — that is your blast-radius control.

Grant the minimum. Extra tools in a prompt widen the model's option space and make runs less repeatable.

Two ways to connect, **one underlying truth**: the reference inside the prompt.

TWO EQUIVALENT GESTURES

A • drag from a step's connection point
onto another step
→ Opal ADDS the reference to the target's prompt

B • type @ inside the prompt and pick the step
→ the canvas DRAWS a dotted line for you

// identical result. A is discovery, B is speed.

assembled prompt sent to the model =
your text + referenced steps + tools + assets

Fan-in is native. Google states you can have "one step that references the output of multiple steps together." A synthesis step that @-mentions three upstream Generate steps is the standard pattern.

Execution order is derived, not drawn. The docs note "some steps will require previous steps to be run first" — dependency comes from references, so a step with no upstream reference can run immediately.

FOUR CONSEQUENCES FOR HOW YOU BUILD

01 • Name your steps deliberately. Step names are what you @-mention. "Generate 2" tells you nothing at reference time; "Draft LinkedIn Post" reads correctly inside a downstream prompt.

02 • A drawn line is not a guarantee. If you later rewrite a prompt and drop the @ reference, the data dependency is gone. Verify in the Console (slide 11), not on the canvas.

03 • Reference deliberately, not defensively. Every referenced step's full output is injected into the prompt. Referencing five upstream steps "just in case" bloats the prompt and dilutes the instruction.

04 • Independent branches can run in parallel. If two Generate steps don't reference each other, nothing forces them to be sequential — this is the basis of the latency pattern on slide 13.

Remix a Gallery Opal. You get a **working reference implementation** before you write a single prompt.

THE EXACT SEQUENCE

01 opal.google → Gallery

02 open any Opal → view-only visual editor
you can inspect how it was built, not edit it

03 click **Remix** → creates YOUR editable copy
changes never affect the Gallery original

04 natural-language editor → describe one change
"Add a step to generate a 256x256 social image
based on the banner, and include it in the post."

05 Preview in the sidebar → verify
Google's stated worked example uses the **Blog Post Writer** Opal — "a mini app designed to get a topic from the user and generate a full blog post, with multiple generation steps."

Who curates the Gallery: the Opal team populated it with apps chosen to "best illustrate the product's capabilities" — so it is a capability catalogue, not a community marketplace.

WHY THIS BEATS STARTING FROM SCRATCH

01 · You see idiomatic prompt structure. The Gallery Opals show how Google chains steps, phrases model instructions, and wires tools — patterns that take hours to discover alone.

02 · One change at a time is debuggable. Starting from a working graph means any breakage is attributable to your single edit. Starting from empty means everything is suspect at once.

03 · Remix is non-destructive. Your copy is independent. You can remix the same Gallery app five times to compare five approaches side by side.

The limitation: the Gallery is Google-curated and finite. Once your use case diverges from a template, you move to path B or C.

Describe the whole app in one prompt. Opal generates the graph. Then you fix it.

GOOGLE'S OWN EXAMPLE PROMPTS · VERBATIM

> "Create a Google Doodle that inputs a date, looks up a significant historical event, and generates a Google Logo to educate and inspire users"

> "Create an app that take the screenshot of a math problem, provides a detailed description of the concept behind it, ask for user's feedback, and provide a detailed solution"

> "Given a mood or preferences, generate a DJ track. Each song should have a witty DJ intro, plus a link to the videos on youtube."

Read the structure, not the topic. Every one of these names an *input*, a *transformation*, and an *output artefact* — and the third one even specifies per-item structure ("each song should have..."). That is why they produce usable graphs.

THE PROMPT FORMULA THAT WORKS

Create an app that:
takes <specific input + type>
uses <named tool> to <action>
produces <artefact> containing
<explicit fields>

It can make multiple changes at once. Google notes the natural-language editor "can make multiple changes from one prompt, so it can be a good way to get started or try out new ideas."

That is also the risk. A vague prompt rewrites more of the graph than you intended. Once the skeleton exists, switch to narrow edits: "change only the second step's prompt to..."

Equivalence to know: it makes "the same types of changes that you can make with manual edits using the visual editor" — no hidden extra power.

Google's twelve-step walkthrough, **compressed to what matters.**

THE BIRTHDAY APP · GOOGLE'S WORKED EXAMPLE

01 Create New → name it (top-left)

02 + User Input → prompt: "When is your birthday?"

03 + Generate → drag input's connector onto it
prompt:

"The user's birthday is: [User Input].
Use [Search Web] and google.com to find a
historical event that happened on the user's
birthday. Try to find events that are fun
and positive, like celebrations."

04 + Output → connect → type: webpage w/ auto-layout

05 deselect all → sidebar Preview → Start

Note the prompt style Google models here: it states the data source, names the tool inline, and gives a qualitative constraint ("fun and positive"). That last clause is doing real work — it is the difference between a usable app and a random one.

EDITOR MECHANICS WORTH KNOWING

01 · New steps can overlap. Google explicitly warns "You may need to drag the steps so that they are not overlapping." Layout is manual and purely cosmetic — it does not affect execution.

02 · The sidebar is context-sensitive. Nothing selected → Preview / Theme / Console. A step selected → that step's title and prompt. Click the canvas background to get back to Preview.

03 · You can run a single step. Each step has a play button. Steps with upstream dependencies will require those to have been run first.

04 · Mix the paths freely. Google's stated model is that you can build "using conversational natural language commands, a visual editor, or a combination of both." The mature workflow is: NL for scaffolding, visual editor for precision.

Auto-layout is a demo feature. Manual layout is a product feature.

OUTPUT 01 · WEBPAGE, AUTO-LAYOUT

The model designs the page.

output type:
webpage with auto-layout

"the AI model decides the layout"

Use when: prototyping, demos, exploratory tools where "looks reasonable" is the bar.

The cost: the model chooses structure each run. Two runs of the same app can render differently — headings, ordering, and emphasis can all shift.

Symptom you'll hit: a stakeholder says "it looked different yesterday." That is auto-layout working as designed, not a bug.

OUTPUT 02 · WEBPAGE, MANUAL LAYOUT ★

You specify the structure.

output type:
manual layout
then state layout in the prompt:
"H1 title, then a 3-column
table, then sources list"

Use when: anyone other than you will use the app more than once.

Google's framing: "If you want more control over the layout, try selecting the manual layout option and then specify what you want the page to look like in your prompt."

Practical rule: name every section and its order explicitly. Anything you leave unspecified, the model will decide for you — differently each time.

OUTPUT 03 · DRIVE SPREADSHEET

Export to Google Sheets.

output type:
export → Drive spreadsheet

Google Labs cite this as a
top observed pattern

Use when: the output feeds something else — a dashboard, a review process, another tool.

Why it matters: this is Opal's main escape hatch into the rest of your stack. Google's own expansion post highlights apps that "extract data from the web, analyze findings and save results directly into Google Sheets."

Combine outputs: render a page for the human *and* write the row for the system, in one run.

Source: developers.google.com/opal/overview ("Output steps allow you to control what you see... dynamic webpage where the AI model decides the layout, or exporting to a Google Drive spreadsheet") · [/quickstart](#) (manual-layout guidance) · [blog.google](#) Opal 160-country post (Sheets as an observed pattern).

Preview shows you *what* broke. The Console shows you *why*.

WHAT THE CONSOLE EXPOSES · PER STEP

```
sidebar → Console → Start

▶ step 1 · User Input 0.2s ▶
▼ step 2 · Generate 4.1s ▶
├─ model inputs ← the ASSEMBLED prompt
├─ thinking steps
├─ tool calls ← did @Search Web fire?
└─ model outputs
▶ step 3 · Output 1.8s ▶

▶ = run just this step
```

Google documents exactly this: steps appear "in the order that they are executed in, along with how long each step has been executing," and expanding a step shows "model inputs and outputs, thinking steps, and tool calls."

Preview and Console are synchronised — the app state mirrors the console, so you watch the UI and the internals together.

THE FOUR-QUESTION DEBUG LOOP

Q1 · Did the reference resolve?

Open *model inputs* on the failing step. If the upstream step's text isn't in there, your @ reference is missing — the drawn line lied to you.

Q2 · Did the tool actually fire?

Check *tool calls*. A step that "should have searched" but shows no tool call didn't have the tool granted in its prompt.

Q3 · Where is the latency?

Per-step timings tell you which Generate node dominates. That is the node to split, simplify, or parallelise (slide 13).

Q4 · Is the asset being read correctly?

Google's example: for a step referencing a Google Doc asset, the console "will give you the request details and the output of the doc that will get used in the prompt."

Source: developers.google.com/opal/overview — "Debug with the console" (Start button, execution order + elapsed time, per-step play icon, expandable details showing model inputs/outputs, thinking steps and tool calls, Preview mirrors Console, Google Doc asset example).

Restoring an old version permanently deletes everything newer. There is no redo.

HOW IT WORKS

access

top bar → three-dot menu → Show version history

what you get

- automatic saves as you edit
- a list of previous versions
- per-version view of how steps were arranged
- "Restore this version" at the top of the editor

Saves are automatic. Google states Opal "will automatically save a version history of your edits" — you don't commit anything manually.

You can inspect before restoring. Selecting a version shows you that version's step arrangement in the visual editor, so you can confirm it's the one you want first.

THE DESTRUCTIVE EDGE • READ CAREFULLY

*"Any earlier versions of your Opal app will still be available, but **any later versions than the one you are restoring to will be permanently deleted** and no longer available for viewing or restoring."*

Mitigation 01 • Remix before you experiment. A remix is a full independent copy. Branch first, then break things — the original is untouched and its history is intact.

Mitigation 02 • Treat restore as a last resort. Prefer fixing forward with a narrow edit. Restore only when the graph is genuinely unrecoverable — and accept you are discarding the branch.

Independent steps don't need each other. Stop making them wait.

SAME WORK • TWO GRAPH SHAPES

```
// x CHAINED — each step waits for the last
[Input] → [LinkedIn] → [Substack] → [X] → [Out]
6s + 6s + 6s = ~18s

// ✓ FANNED OUT — no cross-references
r→ [LinkedIn] ↯
[Input] —+→ [Substack] +→ [Output]
      ↳ [X post] ⌞
max(6,6,6) = ~6s

// the ONLY change: each branch references
// [Input] directly, never a sibling branch
```

Why this works: execution order in Opal is derived from references (slide 06). If a step doesn't @-mention a sibling, nothing forces it to wait for that sibling. Parallel execution is a documented Opal capability, added by Google Labs as the product matured.

HOW TO REFACTOR AN EXISTING OPAL

- 01 • Find the chain in the Console.** Per-step timings (slide 11) show you the serial path. Add up the Generate steps — if the total $\approx$ the run time, you are fully serialised.
- 02 • Ask: does B genuinely need A's output?** If B only needs the original input, it was never a real dependency — it was an accident of how you drew the graph.
- 03 • Rewrite the prompt, not the canvas.** Remove the sibling @ reference and point it at the input step instead. The dotted line follows automatically.
- 04 • Converge in one synthesis step.** A final Generate or Output step that @-mentions all branches is the join. This is the fan-in Google documents explicitly.

Inside a Generate step, stop picking a model — pick an agent.

WHAT THE AGENT STEP CHANGES

```
// before — a Generate step
model := pick(gemini-*) // fixed
out := model(prompt, @refs) // one shot

// after — an Agent step
agent := select(Gemini 3 Flash)
agent.memory // survives sessions
agent.route(input) // decides at runtime
agent.chat(user) // interactive, mid-run
```

Memory. Google's own framing: "Whether it's a user's name ... preferences ... a running shopping list." State that persists *across sessions* — the first durable state Opal has ever had.

Dynamic routing. The agent decides which path to take at runtime. This is Opal's answer to the missing conditional primitive from slide 03 — the branch lives inside the agent, not on the canvas.

Interactive chat. The workflow can talk back mid-run instead of only rendering a final Output.

WHAT IT DOES NOT FIX

Routing is not inspectable as a graph. A canvas branch you can read is replaced by a decision inside a prompt. You lose the diagram; you keep the Console trace (slide 11) as your only view.

Still no loop primitive. Dynamic routing lets the agent choose a path; it is not documented as a bounded iteration construct with a counter you control.

Still no API, no trigger. The agent step changes what happens *inside* a run. It does not give you a way to start a run from outside (slide 29).

Memory scope is undocumented. Whether memory is per-user or per-Opal is not stated in the announcement. Test it before you put anything personal in it.

An Opal run is a **single pass through a graph**. Plan your persistence accordingly.

THREE TIERS OF PERSISTENCE

TIER 1 · Within a run — step outputs.

Every step's output is available to any downstream step that @-mentions it. This is your working memory, and it dies when the run ends.

A Memory block also holds data temporarily during a single run.

TIER 2 · Across runs — assets.

Uploaded files and YouTube links persist with the Opal itself. Same content on every run, for every user. Good for reference material; useless for accumulating results.

TIER 3 · Beyond Opal — the Sheets export.

The Drive spreadsheet output is the only documented way to make a run's results outlive the run in a structured, queryable form. If you need history, this is the mechanism.

WHAT THIS RULES OUT — AND THE WORKAROUND

TIER 0 · Agent memory (Feb 2026). An Agent step persists state *across sessions* — names, preferences, a running list. This is the one tier that breaks the stateless model. Scope is undocumented: test it.

X No scheduled or triggered runs. There is no documented cron or webhook entry point. A human opens the app and presses start — that is the trigger model.

✓ The workaround: Sheets as the state store. Write results out to a spreadsheet, and let the next run read that same sheet back in as context.

X No durable grounding — yet. Agent memory holds small facts, not a corpus. The Gemini Notebook tile that would fix this is built into Opal but gated off (slide 25).

You don't pin the model. Google does — and Google retires models.

THE 2026 INCIDENT — A REAL POST-MORTEM

```
// reported symptom
user opens a working Opal → error
prompt unchanged, graph unchanged → error

// root cause
Opal backend pinned to
gemini-3.1-flash-lite-preview
Google officially retired that model
→ the app broke with no user action
```

Read the architecture, not the bug. The model surface inside Opal is *curated by Google Labs*, not declared by you. There is no version pin, no `model=` parameter you own, no deprecation window you get notified about.

Compare with the API path. On the Gemini API you name an exact model ID and you get a documented retirement schedule. In Opal you get a dropdown whose contents change.

The second-order effect: prompt drift. Even without a hard error, a prompt tuned against one backend degrades silently when the backend rotates.

MITIGATIONS THAT ACTUALLY WORK

1 • Keep a golden-input test. One fixed input with a known-good output, run before every demo. It is the only regression signal you have — there is no CI hook.

2 • Write prompts that don't depend on a model's quirks. Explicit output structure, explicit constraints. Prompts that rely on a specific model's "style" are the ones that drift.

3 • Never put an Opal on a critical path. No SLA, no deprecation policy, no version pin. If an outage costs money, the workload belongs on Vertex AI (slide 28).

4 • Screenshot the Console on a good run. When it breaks you will need to know what the working trace looked like. Version history (slide 12) does not capture backend state.

Private by default. But sharing an Opal **also shares the underlying Drive file.**

THE SHARING CONTROLS

```
default state
private · visible only to you

Share app →
· select specific individuals
· or allow anyone with the link

toggle: allow editor view + remix
ON → they see your GRAPH and ALL PROMPTS
OFF → they see only the app view
```

Editor view ≠ edit rights. Google is precise here: people you share editor view with "cannot make edits to your Opal, but they can remix" — i.e. take their own copy.

Publishing creates a link where users try the app themselves. You can restrict who, and you can unpublish at any time — after which "the link will lead to an error page."

Users run it with their own Google account.

THE CAVEAT GOOGLE PUTS IN WRITING

*"Please note, any time you share your Opal with others you are sharing the drive file with them as well which may give others workarounds to view details about your Opal. If it's important to you to ensure no one has any way to access details about your Opal including your prompts, **please do not share your Opal.**"*

Translate that for your organisation: the "hide editor view" toggle is a UI convenience, *not* a security boundary. Treat every prompt in a shared Opal as disclosed to everyone you shared with.

Practical rules: never embed secrets, API keys, or confidential business logic in a prompt you intend to share. If the prompt itself is the IP, don't distribute the Opal — distribute its output.

Not used for training. But humans may review a small subset.

GOOGLE'S DATA STATEMENT · QUOTED

*"We don't use content from Opal data to train our generative AI models. This includes your prompts and any output Opal generates for you. **In some cases, humans may review a small subset of prompts in order to troubleshoot or better understand use cases.**"*

Read both halves. The training exclusion is clear and covers prompts *and* outputs. The human-review clause is equally explicit and is the part that matters for a compliance review.

Governing terms: Google directs you to the standard Terms of Service and Privacy Policy for what is collected and how it is used — there is no separate enterprise data addendum documented for Opal.

Storage location: your Opals are files in your Google Drive.

THE OPERATING RULES THIS IMPLIES

01 · Don't paste regulated data into an Opal. No PII, no PHI, no cardholder data, no material non-public information. A Labs experiment is not the venue for it.

02 · Assume prompts are readable by humans. Google says so directly. Write every prompt as if a Google engineer might read it for troubleshooting — because that is the documented behaviour.

03 · Use synthetic data for demos. Prototype the *shape* of the workflow with fake records, then rebuild on a governed platform if it needs real data.

04 · Check your Workspace posture separately. Opals live in Drive, so your organisation's Drive sharing policy and DLP rules apply to the app files themselves.

Workspace shipped an AI control center in May 2026. **Opal is not in it.**

AI CONTROL CENTER · SHIPPED 4 MAY 2026

Admin console → AI control center

covered Gmail Drive Docs
covered Sheets Slides Meet
covered Calendar Chat
covered Gemini App

absentOpal

SKUs: Enterprise Standard / Enterprise Plus

What the control center gives you for the covered surfaces: a centralised view of security and governance settings for generative AI and agent actions across Workspace data.

What that means for Opal: an employee can build an Opal, wire it to a prompt containing internal content, and share it by link — and none of that appears in the admin surface you just rolled out. Your AI governance posture has a hole exactly the shape of a Labs experiment.

THE FOUR QUESTIONS YOU CANNOT ANSWER

"Who in my tenant is using it?" No admin inventory of Opals exists. Discovery is manual — you ask people.

"Can I turn it off for the org?" Not documented as an AI control center toggle. Labs access control is a separate, coarser lever.

"Where are the audit logs?" No public audit-log hook for Opal runs is documented. Compare: NotebookLM Enterprise has IAM + Cloud audit logging (slide 24).

"Which region does it process in?" No data-residency option is documented for Opal. If residency is a hard requirement, the answer is Gemini Notebook Enterprise (us / eu / global), not Opal.

Opal workflows now surface inside Gemini's Gems Manager as "Gems from Labs".

WHAT THE INTEGRATION ACTUALLY DOES

Gemini → Gems Manager

- └─ **Gems from Labs** ← NEW section
 - pre-built Opal workflows
 - **My Gems from Labs** ← your existing Opals
- └─ **Gem Manager** ← the familiar one
 - personal / pre-built custom Gems

new gem → Workflow Builder → auto-generates
steps + system prompts + visual elements

Migration is automatic. Users who already had Opal workflows find them listed under *My Gems from Labs*.

The preview gained a voice path — standard text input plus voice dictation — and can be launched full screen.

Sharing improved materially: publish publicly with a shareable link, rather than relying on Google Drive permissions (the caveat from slide 17).

Escape hatch preserved: an advanced option links straight into the Opal Builder for detailed editing.

STATUS · AND WHAT IT MEANS FOR YOU

Rolled out from 17 Dec 2025, initially to the US and a subset of users — matching Opal's own regional availability. Treat presence in your account as not guaranteed.

The strategic read: Opal is being folded into Gemini as the workflow layer, rather than remaining a standalone Labs site. Build with the expectation that the Gemini surface becomes the primary entry point.

Practical consequence: the same artefact now has two front doors — opal.google for building, Gemini for distribution. Your users may never see the Opal canvas at all.

Verify before you present: rollout state changes weekly. Check your own Gems Manager rather than assuming this slide still holds.

NotebookLM is now **Gemini Notebook**. Same product, same URL, new name.

WHAT CHANGED, AND WHAT DID NOT

```
// name
NotebookLM → Gemini Notebook

// product URL — unchanged, still works
notebooklm.google.com

// support / docs domain — new
support.google.com/gemininotebook/...
```

Still a standalone product. Google's wording: "It remains a standalone product focused on being your premier research tool" — the rename "reflects how it will evolve to do more across the Google ecosystem." It was not absorbed into the Gemini app.

It also lives inside the Gemini app. "You can already access and create notebooks directly within the Gemini app, with full cross-app syncing between the Gemini app and the standalone Gemini Notebook experience."

Model backing moved too. The June 2026 update describes it running on **Gemini 3.5** and Antigravity, with a "secure cloud computer" for code execution.

WHY A RENAME IS A TECHNICAL PROBLEM

Every tutorial you find is pre-rename. Anything written before 16 Jul 2026 says "NotebookLM". Anything about limits or features from 2025 predates two model generations. Date-check before you trust it.

Two live domains, one product. Product at `notebooklm.google.com`, documentation under `gemininotebook`. Bookmark both; neither is deprecated.

The enterprise SKU is a different product. "Gemini Notebook for Enterprise" is sold and documented under Google Cloud, with its own API and controls (slide 24). Do not conflate the two.

Naming convention for this deck: "Gemini Notebook (formerly NotebookLM)" on first mention, "Gemini Notebook" thereafter.

A notebook is **sources + chat + studio outputs**. The sources define everything.

ACCEPTED SOURCE TYPES

`files` PDF · DOCX · TXT · Markdown · CSV
PPTX · EPUB
`google` Docs · Slides (≤100 slides)
Sheets (≤100k tokens) · Drive (auto-sync)
`web` URLs, text only — paywalled pages fail
`youtube` public videos `with captions`
`audio` MP3 · WAV — transcribed at import
`images` png jpg webp heic tiff gif bmp avif ...
`other` pasted text · Play Books · `Gemini chats`

`per-source ceiling: ≤500,000 words OR ≤200 MB`

Drive sources auto-sync — but Drive *footnotes and comments are not imported*. If your review history lives in comments, it is invisible to the notebook.

A Gemini chat can become a source. That is the one clean bridge from a conversational session into a durable, cited knowledge base.

THE INGESTION TRAPS

YouTube sources are volatile. If the source video is removed, the imported source is **auto-deleted within 30 days**. A notebook grounded on YouTube can silently lose its evidence.

Captions are a hard dependency. No captions, no import. Non-speech audio is rejected outright. Transcription happens once, at import — not on demand.

Web URLs are text-only. Tables rendered by JavaScript, diagrams, and anything behind a paywall do not arrive. Export to PDF and upload that instead.

Sheets cap is a token cap, not a row cap. ≤100k tokens. A wide sheet hits it far earlier than you expect — pre-aggregate before importing.

Sources per notebook is the constraint that will bite you: 50 on free, 600 at the top.

Capability	Standard	Plus	Pro	Ultra 20 TB	Ultra 30 TB
Notebooks / user	100	200	500	500	500
Sources / notebook	50	100	300	500	600
Chats	50 / day	200 / day	500 / day	2.5K / day	5K / day
Audio Overviews	3 / day	6 / day	20 / day	100 / day	200 / day
Video Overviews	3 / day	6 / day	20 / day	100 / day	200 / day
↳ cinematic	—	—	2 / day	10 / day	20 / day
Reports	10 / day	20 / day	100 / day	500 / day	1K / day
Flashcards · Quizzes · Mind Maps	10 / day	20 / day	100 / day	500 / day	1K / day
Deep Research	10 / month	3 / day	20 / day	75 / day	200 / day
Watermark removal	no	no	yes*	yes*	yes*

How to read this

Daily quotas reset after 24 h; monthly after 30 days. Deep Research on the free tier is *10 per month* — not per day. That single line kills most free-tier team pilots.

Sharing does not raise a collaborator's ceiling. Limits are per user. A Pro user sharing into a free user's account does not lift that user's caps.

Plan pricing (consumer): AI Plus ~~\$4.99~~/mo · AI Pro ~~\$19.99~~/mo · AI Ultra ~~\$99.99~~/mo. Advanced Sharing is a paid feature; custom chat and analytics are available to everyone.

* Visible watermarks are **force-applied in India, South Korea and Vietnam on every tier**. SynthID invisible watermarking applies to generated media regardless of tier.

Source: support.google.com/gemininotebook/answer/16213268 — official plan-comparison table (all rows above, quota reset windows, sharing does not change collaborator limits, watermark regional rules). Pricing: one.google.com/about/google-ai-plans. Community "NotebookLM limits" blogs round these numbers — cite the support page, not them.

Opal has no API. Gemini Notebook Enterprise does. That asymmetry decides your design.

REST SURFACE · `discoveryengine.googleapis.com`

```
# endpoint shape
https://{ENDPOINT_LOCATION}-discoveryengine.googleapis.com/v1alpha
/projects/{PROJECT_NUMBER}/locations/{LOCATION}/notebooks

# ENDPOINT_LOCATION ∈ { us, eu, global }

# methods
notebooks.create notebooks.get notebooks.share
notebooks.batchDelete notebooks.listRecentlyViewed
```

```
# create a notebook
curl -X POST \
-H "Authorization:Bearer $(gcloud auth print-access-token)" \
-H "Content-Type: application/json" \
"https://us-discoveryengine.googleapis.com/v1alpha/projects/\
PROJECT_NUMBER/locations/global/notebooks" \
-d '{"title": "My Notebook"}'
```

Roles on the notebook resource: `PROJECT_ROLE_OWNER · WRITER · READER · NOT_SHARED`. The caller must additionally hold the **Cloud NotebookLM User** IAM role. A sibling doc covers podcast / audio-overview generation as an API method.

WHAT THE ENTERPRISE SKU BUYS

Price: **\$9 USD per licence per month** after a 30-day trial, **minimum 15 licences**. Also reachable via Gemini Enterprise Standard / Plus / Frontline add-ons.

Data boundary: "Your uploaded files will remain within your GCP project. Data regionalization is honored." Files, chats and model outputs are **not reviewed by humans and not used to improve generative AI models** — unconditionally, unlike the consumer tier.

Controls: VPC Service Controls, CMEK, IAM roles, Model Armor, audit logs. Microsoft 365 *and* Workspace sources. 5× more Audio Overviews, notebooks and sources per notebook.

The consumer product has no public API. `notebooklm.google.com` offers no self-serve API key. Don't design against it.

Sources: `docs.cloud.google.com/gemini/enterprise/notebooklm-enterprise/docs/api-notebooks` (26 Aug 2026 – endpoint shape, us/eu/global, the five methods, cURL example verbatim, `PROJECT_ROLE_*` values, Cloud NotebookLM User IAM role) · `.../docs/podcast-api` (audio-overview API method) · `cloud.google.com/resources/notebooklm-enterprise` (\$9/licence/mo, 15-seat minimum, VPC-SC / CMEK / Model Armor, M365 + Workspace sources, GCP-project data boundary quote).

The connector is **already in the product**. It is switched off — and you can prove it yourself.

THE EVIDENCE · VIEW-SOURCE ON OPAL.GOOGLE, 1 SEP 2026

```
// runtime config embedded in the production shell HTML
"flags": {
  "mcp": false,
  "opalAdk": false,
  "enableNotebookLm": false, ← the code path exists
  "enableAgentWorkbench": false,
  "enableGraphEditorAgent": false, ...
}
```

Read the flag correctly — it says two things at once. The key's *presence* proves Google has built a notebook code path into Opal and ships it to every user. Its *value* proves the gate is closed. This is a launch waiting on a switch, not a rumour.

The mechanism, from build teardowns: you pick a notebook tied to your Google account; it appears as "a dedicated tile within the Opal canvas" and "connects to the rest of the workflow blocks just like any other node." In a Generate step you then "pull or reference data from the linked notebook via a natural-language prompt."

Momentum is real: a 4 Aug 2026 report states Google "will soon integrate NotebookLM into the Opal workflow builder." Still no dated Google announcement — so treat it as imminent, not available.

WHAT YOU CAN ACTUALLY USE TODAY

✓ **A real notebook connector shipped — for Gemini Enterprise, not Opal.** "Enable Gemini Notebook Enterprise as a search source (Preview)", documented 26 Aug 2026. Create a Data Store → source type **Gemini Notebook**. Caveat: users "search on notebook titles" — title-level, not full-text.

✓ **In Opal, the export path still works.** Notebook → Report → export PDF / DOCX / Markdown → upload as an Opal **Asset** → @-reference it in a Generate step. Grounding is frozen at export time; re-export when sources change.

✓ **Region rule to plan for now.** The Enterprise connector pins notebook, data store and app to the *same* region — EU notebook needs an EU data store and an EU app.

How to say it in the room: "Built and gated, not announced." Design the workflow for the tile now — swap out the manual export when the flag flips.

Sources: opal.google production shell HTML, fetched 1 Sep 2026 — runtime flags object containing "enableNotebookLm":false (PRIMARY; verify it yourself) · testingcatalog.com "Google test NotebookLM integration for Opal workflows" — canvas tile, node wiring, natural-language pull in a Generate step (SECONDARY) · TestingCatalog, 4 Aug 2026: "Google will soon integrate NotebookLM into the Opal workflow builder" (SECONDARY) · docs.cloud.google.com/gemini/enterprise/docs/connectors/connect-notebooklm — "Enable Gemini Notebook Enterprise as a search source (Preview)", 26 Aug 2026, title-level search + same-region rule (PRIMARY). No Google announcement of the Opal tile between 24 Feb and 1 Sep 2026.

Gemini Notebook is **source-grounded**. Opal is **generative**. Use each for what it does.

THE DIVISION OF LABOUR

Gemini Notebook — the grounding layer

- upload the real sources (PDFs, docs, transcripts)
- ask it for a distilled, cited **Report**
- answers are anchored in YOUR documents

↓ export PDF / DOCX / Markdown / text

Opal — the generation + distribution layer

- paste the brief in as an **Asset** (static)
- or a **User Input** step (per-run)
- fan out to N audience-specific Generate steps
- render a page + write a Sheet
- publish one link to the team

Why this split is the right one. Gemini Notebook's value is that it will not stray from your sources. Opal's value is that it turns one input into many artefacts and hands you a hosted app. Asking Opal to be source-faithful without grounding is asking the wrong tool.

Asset vs User Input: asset if the brief is stable for weeks; input step if it changes every run.

THE GROUNDING GUARDRAIL

```
// put this in EVERY Generate step
Use ONLY the content of @[Research Brief].
```

Do not add facts, figures, dates or names that do not appear in it.

If the brief does not cover something the output needs, write: **NOT IN SOURCE** — do not fill the gap.

Verify it in the Console. Open *model inputs* on a Generate step and confirm the brief's text is actually present. If it isn't, your reference didn't resolve and the model is improvising.

Grant tools narrowly. If a step must stay inside the brief, do *not* give it `@Search Web` — otherwise it can pull in outside claims that break your grounding.

The native tile is built but gated off (slide 25). Until the flag flips, export-and-attach is the integration — and the guardrail prompt above stays identical either way.

Mechanics per developers.google.com/opal/overview (assets as static context, @ references, per-step tool grants, Console model-inputs view) · blog.google "Do better research with NotebookLM" (report export formats: PDF, DOCX, Markdown, text). The manual export step is required because Opal's notebook code path is still gated off in production (flag "enableNotebookLm":false, opal.google, 1 Sep 2026) — see slide 25. Prompts above are this workshop's worked example, not Google-published text.

The pattern Google Labs cites most: **extract** → **analyse** → **write to Sheets**.

THE GRAPH · SIX NODES

```
[1] User Input · type: text
"Which companies should I research? One per line."

[2] Generate · "Funding" ↵
@[1] @Search Web |
[3] Generate · "Headcount" | all three
@[1] @Search Web | reference
[4] Generate · "Recent news" | ONLY [1]
@[1] @Search Web ↵ → parallel

[5] Generate · "Normalise to rows"
@[2] @[3] @[4] → strict column contract

[6] Output · export → Drive spreadsheet @[5]
```

Why node [5] exists. Three independent researchers produce three different shapes of prose. Node [5] is the only node whose job is *structure* — it converts free text into a fixed column contract before anything touches the spreadsheet.

THE TWO PROMPTS THAT DECIDE QUALITY

Node [2] — a research step

For each company in @[1], use @[Search Web] to find total funding raised and last round date.

Return one line per company:
company | total_funding | last_round | source_url

If a value is not found, write UNKNOWN.
~~Never estimate or infer a figure.~~

Node [5] — the normaliser

Merge @[2] @[3] @[4] into ONE table.
Exactly these columns, in this order:
company, funding, last_round, headcount,
news_headline, news_date, sources
One row per company. No commentary, no preamble.

The UNKNOWN rule is the important one. Without it the model fills gaps plausibly, and a spreadsheet of confident wrong numbers is worse than an empty cell.

Opal wins on time-to-shareable-prototype. It loses on everything operational.

REQUIREMENT	GOOGLE OPAL	GEMINI GEM	N8N / ZAPIER	CUSTOM CODE
Multi-step AI chain	Native — the entire product	Single persona, not a graph	Yes, but you wire each model call	Yes, with full control
Time to shareable app	Minutes • hosting included	Minutes, but chat-shaped only	Hours + a front end to build	Days
Scheduled / triggered runs	Not documented — human-initiated	No	Core strength	Yes
Programmatic API	No public endpoint documented	No	Webhooks in/out	By definition
Multimodal generation	Text / image / video per step	Depends on the model	Via third-party API nodes	Whatever you integrate
Regulated data	No — Labs experiment, human review	Per your Workspace terms	Self-host possible	Full control

THE DECISION RULE

Use Opal when a human will press start, the data is not sensitive, and you need it shareable today. That covers prototypes, internal tools, demos, and validating whether a workflow is worth building properly. Graduate off Opal when you need a schedule, an API, an audit trail, or regulated data. Prototype in Opal, then rebuild the winners on a platform with operational guarantees.

Opal column sourced from developers.google.com/opal (+ /overview, /faq) as cited throughout this deck. "Not documented" means absent from Google's Opal documentation as read for this workshop – not a statement that Google has ruled it out. Other columns are this workshop's assessment for orientation, not vendor claims.

Three things Google documents clearly — and four I will not invent numbers for.

DOCUMENTED CONSTRAINTS · PLAN AROUND THESE

01 · Geographic availability is an explicit allow-list.

Google publishes the full country list in the FAQ — 160+ countries including Israel, the US, India, Japan, Brazil, Canada. If a country is absent, users there cannot access Opal. Check the list before you promise a rollout.

02 · The editor is desktop-only in practice.

"Opal's editor view is optimized for a desktop computer. You may use your mobile phone to use or view an already created app." Building on mobile is not a supported path.

03 · Correctness is your responsibility.

The FAQ leads with "Opal can make mistakes so it's important to check your prompts and test your app." There is no evaluation harness, no regression testing, and no output validation built in.

NOT DOCUMENTED · ASK, DON'T ASSUME

X Rate limits, step caps, timeouts. Google publishes no per-user run limits, no max-steps-per-app, no runtime or file-size ceilings. Third-party blogs assert figures; none trace to Google. Assume limits exist, unstated.

X Pricing and commercial terms. No pricing page is published for Opal. It is a Labs experiment; the absence of a price today is not a commitment about tomorrow.

X The exact model list per step. The docs say you "select which AI model you want to use" without enumerating them, and the roster changes underneath you (slide 16). The one model Google names in writing is **Gemini 3 Flash**, for the agent step.

X Longevity. Labs experiments get promoted, absorbed, or retired. Opal is being absorbed into Gemini (slide 20) — that is a positive signal, but it is not a support commitment. Do not make an Opal a single point of failure in a business process.

Every one of these produces an app that demos well and **fails on the second run.**

ANTI-PATTERN 01 · THE BIG ONE

Trusting the drawn line.

Symptom: a step ignores its upstream data even though the canvas shows a connection.

Cause: the prompt is the real edge. You rewrote it and dropped the @ reference.

Fix: Console → *model inputs*. If the upstream text isn't there, the wire is gone.

ANTI-PATTERN 02

Auto-layout in production.

Symptom: "it looked different yesterday." Sections reorder between runs.

Cause: auto-layout hands page structure to the model, per run.

Fix: switch to manual layout and name every section and its order in the prompt.

ANTI-PATTERN 03

Secrets in prompts.

Symptom: an API key, internal pricing rule, or client name sitting in a Generate step.

Cause: assuming "hide editor view" is a security boundary. Google says sharing an Opal shares the Drive file too.

Fix: nothing confidential in any prompt you will share.

ANTI-PATTERN 04

The one-mega-prompt node.

Symptom: one Generate step doing research, analysis and formatting. Output quality is erratic and undebuggable.

Cause: treating Opal like a chatbox instead of a graph.

Fix: one responsibility per node. You get per-step timings and isolated re-runs for free.

ANTI-PATTERN 05

Accidental serialisation.

Symptom: a 4-step app takes 25 seconds when each step takes 6.

Cause: each branch references the previous branch instead of the shared input.

Fix: point independent branches at the input step, then join once at the end (slide 13).

ANTI-PATTERN 06

Restore as undo.

Symptom: you restore an older version to check something — and lose a day of work permanently.

Cause: restoring deletes every later version, irreversibly.

Fix: remix to branch before experimenting. Fix forward; restore only as a last resort.

Tick all fifteen and you have a tool people will actually reuse.

BUILD · HOUR ONE

- 01 Remixed a Gallery Opal before building from scratch
- 02 Every step renamed to something you'd want to read inside an @ reference
- 03 One responsibility per Generate node — no mega-prompts
- 04 Tools granted per step, minimum necessary
- 05 Static context as Assets; per-run context as User Input

VERIFY · BEFORE YOU SHARE

- 06 Console run: every reference resolves in *model inputs*
- 07 Console run: every expected tool call actually fired
- 08 Ran it 3× with the same input — output shape is stable
- 09 Ran it with deliberately bad input — degrades gracefully
- 10 Manual layout set if anyone but you will use it twice
- 11 No secrets, no client names, no regulated data in any prompt

SHIP · AND KNOW WHEN TO LEAVE

- 12 Latency checked — independent branches fanned out, not chained
- 13 Share scope decided: individuals vs link, editor view on/off
- 14 Recipients' countries are on Google's Opal availability list
- 15 **Graduation trigger written down:** the moment this needs a schedule, an API, an audit trail, or real customer data — it gets rebuilt off Opal

Primary sources for this deck: [developers.google.com/opal](#) · [/overview](#) · [/quickstart](#) · [/faq](#) (updated 24 Feb 2026) · [developers.googleblog.com/introducing-opal](#) (24 Jul 2025) · [blog.google](#) Google Labs — Opal 160-country expansion (6 Nov 2025) and "Build dynamic agentic workflows in Opal" (24 Feb 2026) · [blog.google](#) + [workspaceupdates](#) "NotebookLM is now Gemini Notebook" (16 Jul 2026) · [support.google.com/gemininotebook/answer/16213268](#) · [docs.cloud.google.com/gemini/enterprise/notebooklm-enterprise](#) (26 Aug 2026) · [workspaceupdates](#) AI control center (4 May 2026). Current as of 1 Sep 2026 — Opal is an experiment; verify in your own account before relying on any slide here.