QUICK TOUR · SURVIVAL KIT

Metnos

An architecture for governed, extensible and verifiable agents.

INTRODUCTORY BOOKLET · ROBERTO BRUNIALTI · 2026

QUICK TOUR & SURVIVAL KIT

Metnos

*A self-hosted architecture for a **governed, extensible agent**. The reference instance runs on a Linux home server, plans with a local LLM, **builds the tools it lacks on its own** inside a closed, verifiable vocabulary, **asks for permission** before anything that touches the world — and when it says it did something, it actually did.*

mētis (μῆτις) "cunning intelligence" + noûs (νοῦς) "mind"

I/O	closed	audit	IT + EN	0
STABLE EXECUTOR CONTRACTS	GOVERNED VOCABULARY	VERIFIABLE ACTIONS AND OUTCOMES	LOCALIZED TEXT AND PROMPTS	CLOUD REQUIRED TO THINK

status: pre-1.0, installable and operational · **installer:** included in the repo · **license:** AGPL-3.0 · **code:** github.com/brunialti/metnos

ON THIS PAGE

1 · What Metnos is, in thirty seconds

*You give it goals in natural language through a **browser**, **Telegram**, or another admitted channel. The reference installation reads mail, files, photos, calendars, the web and GitHub, but those are examples, not its boundary: every governed capability that can be expressed as a signed executor can join the same architecture. It stops to ask before actions that cannot be redone. There is no Metnos cloud and no "sign up": data and control remain with the operator.*

What happens on every request, in one drawing:

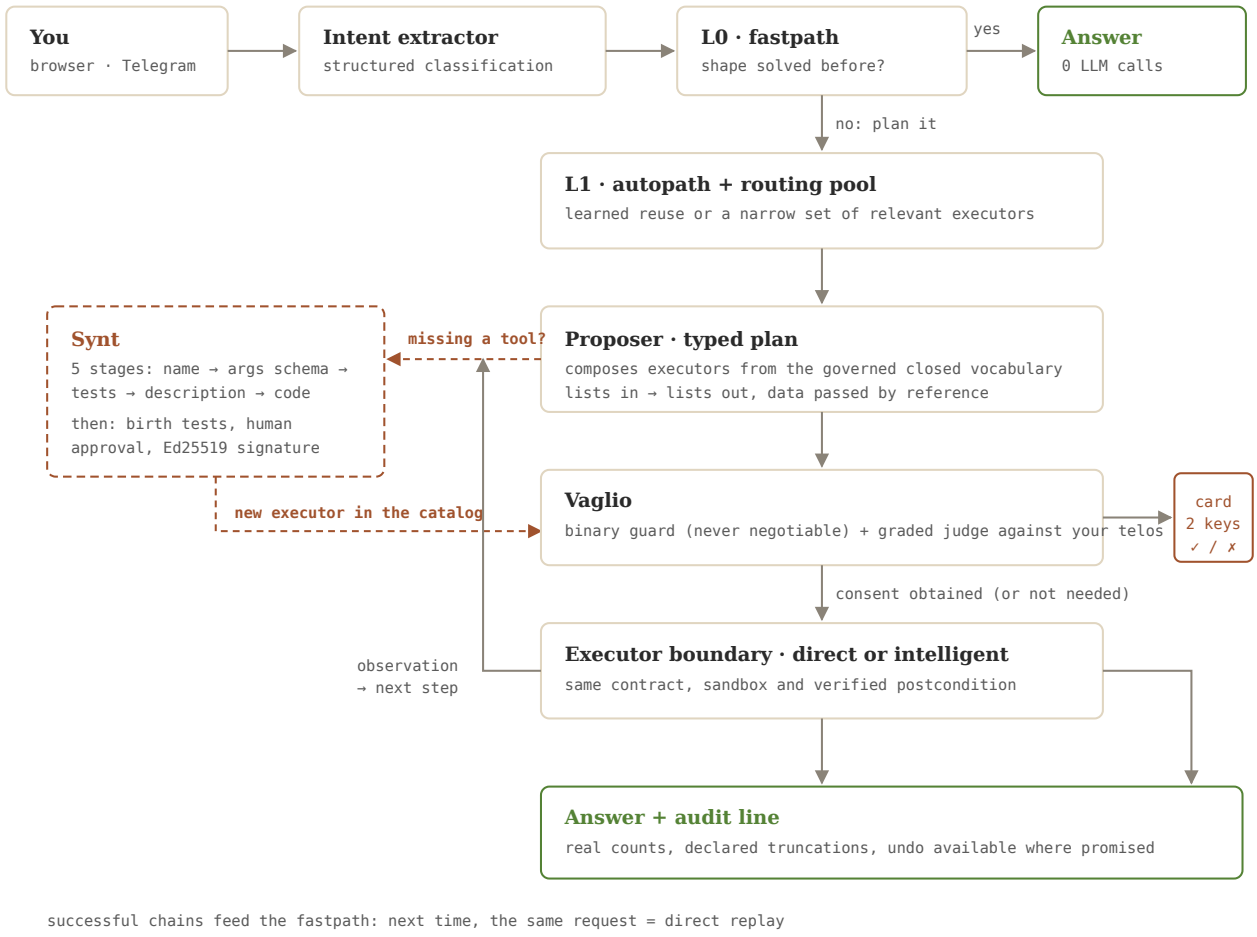


Fig. 1 — One turn end to end. L0 and L1 reuse validated plans; otherwise the proposer sees a narrow routing pool. Intelligence inside an executor does not change the contract visible to the planner.

Six properties that set it apart from a generic agent:

- **It asks before acting.** A two-phase evaluator — binary guard, then a graded judge against your telos — decides whether an action passes silently, asks for consent, or is refused. The request is three lines and two buttons. See vaglio.
- **It builds the tools it lacks.** No endless hand-written catalog: when an executor is missing, the *synthesizer* generates one in a few minutes — name from the closed vocabulary, birth tests, signature — with your approval. See synt and scene 6.
- **It can add bounded agency inside an executor without expanding the planner.** When the route is uncertain but the purpose is narrow, an internal LLM disambiguates steps within fixed actions and limits. To the planner it remains the same typed executor. See intelligent executors.

- **It takes execution to where the data lives.** A compatible executor may run on a registered PC; decision, consent, signing, and audit remain on the server. The device executes, but does not become a second agent. See remote executors.
- **It is reproducible like software, not like a prompt.** The local planner is constrained to a closed vocabulary, the generation seed is pinned, ties between sibling tools are broken by curated affinity — not by coin flip. You can test it, regression-guard it, trust it.
- **It doesn't lie about outcomes.** Counts reflect what was *actually* done; truncations are declared; every action that claims to be undoable is undoable by construction (a closed catalog of reverse patterns).

2 · Why it's different

Agents already exist, in whole families — frameworks you assemble, terminal companions, chat-borne butlers (OpenClaw and friends) and the ecosystem of importable "skills". Metnos makes different choices across several axes, and for a precise reason: ***a local planner is not a frontier model****, and it must be put in a position where it cannot go wrong.*

	TYPICAL AGENT FRAMEWORK	METNOS
Tools	Hand-written, imported, or generated free-form at runtime — then run as-is, with the assistant's privileges.	Synthesized at runtime too, but from a **closed, audited vocabulary**: signed, birth-tested, aged, admitted only through a 7-layer gate.
Routing	The model picks a tool each turn: ask twice, get two plans.	**Deterministic by construction**: pinned seed, ties broken by curated affinity. Same request, same plan, every run.
Output	Free-form, per tool.	Uniform: **lists in, lists out**, composable between steps with no glue.
Internal intelligence	Concentrated in the general agent that selects goals and tools.	May also live inside a narrow-mandate executor without changing its input, output, or authority.
Placement	Tools run where the agent lives.	A compatible executor may run on a signed device; the server retains decision and audit.
Undo	Rare, or at best "fingers crossed".	First-class: closed catalog of reverse patterns, move = COPY-then-DELETE, honest `ok_count`.
Language	English, strings hard-coded.	i18n by construction: every string and prompt is *per-language data*. IT+EN validated; other languages drop-in, not yet tested.
Security	Trust the package author.	Do *not* trust the package: the package has to pass the checks.

The map, mid-2026

A snapshot taken in July 2026: names move fast, the families barely at all.

1 · Orchestration frameworks — LangChain/LangGraph, LlamaIndex, CrewAI, AutoGen (now folded into the Microsoft Agent Framework), smolagents. Open libraries a developer *assembles* an agent from. Over the last two years they have all settled into the same shape: a graph or an event flow provides *structural* determinism — fixed routing, checkpoints, resume — but inside every node the model decides afresh on every run. Tools are free-named functions written by whoever assembles the system, and safety is that person's job, not a property of the frame. A note apart for smolagents: it has the model write executable Python directly — maximum flexibility, and all the trust moves onto the sandbox that contains it.

2 · Model-vendor stacks — OpenAI (Responses API and Agents SDK), Anthropic (Claude Code and the Agent SDK). Polished harnesses around proprietary cloud models. Claude Code's permission system — allow, ask, deny, with hooks that block outside the model's loop — is the closest mainstream relative of Metnos's approval card; but it governs free-form tool names around a remote model, where Metnos governs a closed vocabulary around a model that lives at home. On reproducibility the distance is sharp: OpenAI calls its own seed "best effort", and Anthropic's API has no seed at all.

3 · Autonomous engineering agents — Devin, OpenAI Codex, Google Jules; on the open side, OpenHands. Maximum autonomy inside an isolated virtual machine; the brake sits downstream, in the human review of the pull request. Nearly all cloud and metered. OpenHands is the self-hosted exception capable of local models — but it remains a programmer executing contained arbitrary code, not an assistant with enumerated capabilities.

4 · Self-hosted personal assistants — Metnos's own family, and this is where the comparison bites. The 2026 story is OpenClaw: the same intent — a butler in your house, reached over chat, local models welcome — and the opposite governance: free-form markdown skills plus MCP servers, permissive by default, human confirmation concentrated on payment steps. ZeroClaw, a minimal Rust rewrite, is its wary cousin: pairing, isolated workspaces, allowlists. Alongside them: Goose (Block), Letta for persistent memory, Khoj for a private index of your documents, and Home Assistant's voice assistant — which rediscovered the fastpath principle on its own: deterministic intent matching first, the model only as a reserve.

The universal socket, behind the gate — The Model Context Protocol (MCP), born at Anthropic in late 2024 and passed to the Linux Foundation in late 2025, has become the universal tool socket: OpenAI, Google and Microsoft adopted it, and thousands of servers expose ready-made connectors. A typical MCP client, though, reads the prose descriptions servers declare and trusts them: that is the vector of the *tool poisoning* the security literature documents at length. Metnos neither ignores that ecosystem nor grafts it in raw: the declared route is a dedicated executor that speaks MCP — a vocabulary-admitted name, a signature, a sandbox and an approval card like any other tool; the external server remains a mediated guest, never an authority. Thousands of connectors come within reach, but none enters without passing the gate.

Where the mainstream wins — Honesty is owed here too. Whoever grafts MCP in raw inherits thousands of connectors in an afternoon; Metnos's gate reaches that ecosystem too, just not at the same speed. A frontier model re-invoked on every turn handles novel, messy requests better than a local planner on repeatable routes. And the managed platforms offer observability, evaluation and scale that a self-hosted project does not match. Metnos does not compete on those axes: it competes on provenance, reproducibility and boundary — signed tools, a repeatable plan, judgement against your telos, data that never leaves your machine.

Nine systems, one table

SYSTEM	TOOLS	PLANNING	BRAKE · WHERE IT RUNS
LangGraph	Free-named functions + MCP	Fixed graph; decisions redone every run	Human pause in the graph · OSS, optional cloud
smolagents	The model writes Python code	Free ReAct	External sandbox · OSS, local too
OpenAI Agents SDK	Functions + hosted tools + MCP	Fresh; seed only “best effort”	Programmed guardrails · cloud-only models
Claude Code	Harness tools + MCP	Fresh; no seed	Allow·ask·deny permissions + hooks · cloud-only models
Devin	Isolated VM with shell and browser	Frontier, fresh every run	PR review · metered SaaS
OpenHands	Arbitrary code in containers	Frontier or local model	Sandbox + review · self-hosting possible
OpenClaw	Markdown skills + MCP	Fresh every run	Permissive by default · self-hosted, local
ZeroClaw	Allowlisted tools	Fresh every run	Pairing, isolated workspaces · self-hosted, local
Metnos	**Closed, signed, synthesized vocabulary; MCP mediated**	**Deterministic: pinned seed + fastpath**	**Telos-scored vaglio + approval card · self-hosted, local**

Two choices that change the boundary

These are not two more tools. They are general ways to make executors more capable without expanding the planner's language.

UNCERTAINTY INSIDE A BOUNDARY

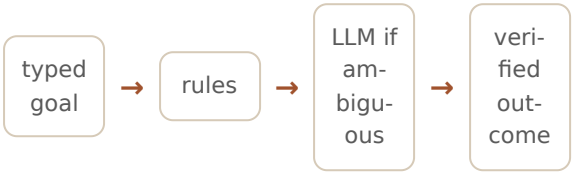
Agent inside, executor outside

An **intelligent executor** receives the same typed input and must produce the same output as a direct executor. The difference is internal: it may observe, choose one step, and observe again when the route is not known in advance.

General agent
Decomposes goals, selects tools, and may cross domains.

Intelligent executor
Has one mandate, enumerated actions, budgets, and fixed stop conditions.

The distinction is deliberate: calling both simply "agents" would hide who may change the goal or authority. Metnos puts adaptive capability *inside* a boundary that the planner can still compose and verify.



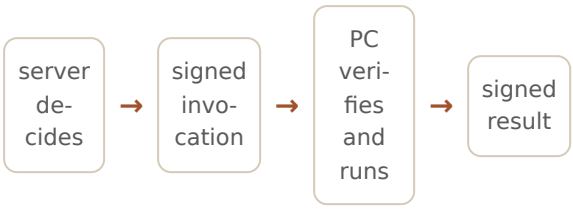
- The LLM is a fallback for the route, not the authority that chooses the purpose.
- Secrets, consent, and permissions remain outside the model.
- Without an observable postcondition it cannot report success.

Example: `login_sites` finds the sign-in area, handles one- or multi-step forms, and checks the outcome; it cannot navigate toward a different goal. How it works.

COMPUTE NEAR THE DATA

Same executor, another machine

A **remote executor** is not a copy of Metnos and not a second agent. The server still plans, applies policy and consent, signs the work, and writes the audit; a thin client executes on the registered device.



- Data can remain on the PC that owns it.
- Identity, allowed platforms, and placement belong to the contract.
- An offline or revoked PC yields an explicit error, never covert execution on the server.

Implication: the home becomes a distributed execution surface while retaining one point of decision and accountability. Details and limits.

Two independent axes: direct or intelligent describes *how* an executor solves its task; local or remote describes *where* it runs. Each executor must still declare explicitly which combinations it supports.

direct · local | intelligent · local | direct · remote | intelligent · remote

Why Metnos did not adopt the standard "skill" format

The popular skill formats — markdown skills, MCP servers — are convenient and dangerous in equal measure: you import a package and the assistant *executes its code with its own privileges*. For an assistant that touches your files, mail and shell, that is remote code execution by design: one malicious or sloppy package is enough. Metnos chooses **security by construction**: a closed vocabulary (you cannot name a tool the grammar doesn't allow), a 7-layer admission gate for every new or imported executor (signature → affinity overlap → aging → sandbox → smoke test → LLM verification → append-only audit), explicit consent before anything destructive. The slogan: *don't trust the package — the package has to earn its place*. The public skill ecosystem isn't ignored: it gets translated into this model, never executed raw.

Determinism is a choice, not an accident

Most agents treat the LLM as an oracle you re-roll every turn. Metnos makes the opposite bet: a *local* planner over a *closed* vocabulary can be made reproducible — the generation seed is pinned, and when two sibling tools tie, a curated affinity signal decides, not chance. Measurably: the internal routing bench repeats every request five times and demands the same plan, five out of five. And it compounds: a request shape solved once is replayed by the fastpath **with no LLM call at all** — lower latency, zero cost, identical outcome. Flexibility (the frontier model) stays available, but it's the reserve, not the engine.

What a terminal agent does better

Due honesty: as a real-time coding companion inside a repository, a mature terminal agent remains superior, with an extension ecosystem Metnos neither has nor chases. But Metnos is not defined by household tasks or by one vertical. It is an operational architecture that keeps planning, executable capabilities, authority, memory and audit under one governed boundary. Mail, photos and calendar are reference uses; maintaining Metnos through its GitHub workflows is already a non-household use. The two kinds of system coexist just fine.

3 · Nine cases, one system

These cases are not separate modules and do not define an exhaustive product scope: they use the same planner, contracts, and guards. The line keeps them together; each stop shows only the request, the relevant chain, and the operational boundary that changes. The GitHub maintenance case deliberately shows that the architecture is not confined to the home.

01

“Summarize today's mail”

> *Give me a summary of today's important mail.*

CATENA DI EXECUTOR

```
read_messages → classify_entries → filter_entries → describe_entries → answer
```

METNOS

3 emails today: the lawyer asks for the document by Friday; the bank confirms yesterday's transfer; your sister about Sunday lunch. Want me to try answering one of these?

capability mail:read · **consent** none — read-only verbs skip the judgment · **latency** ~3 s

02

“You answer the lawyer”

> *Ok, you answer the lawyer: tell him I'll send the document by Thursday.*

An outbound mail to an external recipient never leaves silently, at any autonomy level. Metnos drafts the reply and this bubble lands on Telegram, two buttons attached:

May I send an email?

```
to lawyer@firm.example · subject: “document delivery” · ~140 characters
reversible: no · class: mail:send
```

✓ Approve ✗ Reject

You tap Approve: the daemon checks that the token is still pending, not expired, and that *the person deciding is the same paired sender who asked.* Then: *“Sent at 11:34. Audit line #1247.”*

capability mail:send (irreversible) · **token** single-use, TTL 600 s · **identity** approver == requester

The card's three lines are not a flourish: they are *what*, *on what*, *how redoable* — the minimum to decide at a glance. By the third identical occurrence the card shrinks to two lines, then one. See approval_ux.

03

“Sort these photos by year and place” — then “undo”

> *Sort all image files in ~/images into subfolders by year, prefixing each name with shooting date and place. If the place is missing, use “unknown”.*

There is no “sort photos” tool: the planner composes four from the library, wiring each output into the next input. Raw data never transits through the model: it passes by reference (`from_step`).

```
1  find_files(base_path="~/images", patterns=["*.jpg","*.png","*.heic"],
recursive=true)
     → 98 entries (size and mtime already included)

2  filter_entries(from_step=1, kind="image")
     → 98 entries · in-memory filter, confirms the MIME-declared type

3  get_files(from_step=2, fields=["dates","place"])
     → 98 enriched entries: EXIF → date; GPS → place via reverse-geocode (local cache); no GPS
     → "unknown"

4  move_files(from_step=3, dst_template="~/images/{year}/{date}_{place}_{name}")
     → 98 moved, 0 errors · every source→destination pair recorded in the undo log
```

Four independent executors, chosen and wired by the planner. The same flow handles one or many photos.

The next day, change of heart:

> *Undo the last operation.*

> METNOS
>
> I undid the last operation: 98 files are back at their original paths.

undo multistage reverse: files first, then the created folders — only if left empty · **undo latency** ~3 s for 98 files · **idempotent** a second “undo” does nothing

> HONESTY
>
> Not everything is undoable, and Metnos doesn't pretend otherwise: a sent email cannot be recalled. Undo declares “X not undoable” and reverses the rest. Every move is COPY-then-DELETE: never a deletion without a confirmed copy.

04

“Every morning, the school register — to my daughter, not to me”

The school portal requires a login. The summary must reach the daughter's Telegram; she is paired as a guest. Three messages of setup, then it runs on its own.

> “ *Add credentials for school-portal.example: user roberto.b, password ●●●●●●●.*

Metnos recognizes a value that looks like a password and raises a dedicated card: it offers to encrypt the secret in a local vault and to *scrub the clear value from the turn log* (`<REDACTED:cred>`). Approved, it tries a test login.

> “ *Every morning at 7, look for news on the portal (homework, grades, circulars), read the attached PDFs too, and send the 5 most relevant items to my daughter on Telegram.*

CATENA DI EXECUTOR

```
scheduler 07:00 · open_sites → login_sites (intelligent executor) → act_sites →
read_urls_pdf → group_entries → describe_entries → send_messages(to_user="daughter")
```

METNOS

School — daily report

1. Teachers' strike 05/07 (circular 198) — morning classes suspended
2. Math test moved to 05/10
3. Weekend homework: chapters 4–6 + short essay
4. Coordinators' meeting Friday 05/09, 3:30 pm
5. Latin grade published

credentials encrypted locally, never in code nor in logs · **internal intelligence** login_sites adapts the route but cannot change its mandate · **cookie** reused ~14 days; on expiry, automatic re-login · **LLM** 1 call per run (the summary) — HTML is extracted deterministically · **latency** ~25–40 s/run

HONESTY

If even the re-login fails (password changed on the portal), the task pauses and the host gets a notice — not silence, not an invented report.

05

From a mixed folder to a verifiable dossier

One request crosses PDF, Word, Excel and CSV files on the paired PC, combining time filtering, deduplication, extraction, cross-checking and three outputs:

> *In the Documents/Project Atlas folder, find PDFs, documents and spreadsheets modified in the last 60 days. Logically remove duplicates by comparing signatures and content; extract deadlines, amounts, people and decisions; then create a deadline-ordered summary and a sheet with source, date, owner, amount and confidence level. Flag conflicting data and unreadable files. Save the results in a new subfolder and also create a compressed archive, but do not overwrite or delete anything without approval.*

CATENA DI EXECUTOR

```
find_files → filter_entries (60 days) → read_files (auto + signatures) → extract_entries
→ sort_entries → describe_entries → create_dirs + write_files →
create_files_spreadsheet → find_files → compress_files
```

METNOS

6 candidate files · 1 duplicate removed logically · 5 unique sources · 7 extracted rows. Conflict flagged: approved budget €120,000 due 09/30, revision €135,000 due 09/15. One PDF is unreadable. Created in a new subfolder: `riepilogo.md`, `dati_estratti.xlsx` and `risultati.zip`.

formats PDF + DOCX + XLSX + CSV, read in one flow · **checks** binary hash + signature + content hash, deterministic anomalies · **safety** create-only outputs; no overwriting or deletion · **placement** files stay on the paired Windows PC

HONESTY

Live E2E on the actual Windows PC: 11 of 11 steps succeeded in 74.9 seconds; the sheet contains one header plus 7 data rows, the ZIP contains the two requested results, and all 8 input hashes stayed unchanged. A confidence level does not turn an estimate into certainty, so unreadable files, duplicates and conflicting values remain visible in the summary. The final receipt lists the folder and all three artifacts instead of exposing the archiver's technical table.

06

“Can you read and write Excel files?” — the system explains itself

> *Can you read and write Excel files? What are the limits?*

This is not an operation: it is a question about Metnos. The Tutor keeps it away from the planner, compares it with the signed catalog and admitted local documentation, then composes an answer in the current language. It runs no executor and opens no file.

CATENA DI EXECUTOR

```
question → local embedding → admitted manifests + documentation → role filter →
grounded answer
```

METNOS

Yes. The installed catalog separates reading `.xlsx` workbooks from creating or writing spreadsheets. You can select the sheet to read; for output you can provide columns and rows produced by an earlier step. This answer describes the available capabilities: I have not opened or changed any file.

sources descriptions of executors actually admitted · **languages** IT/EN sources, cross-language retrieval, current-chat response · **authority** no tools and no execution capability

HONESTY

“Read all my Excel sheets” is not a Tutor question: it goes to the normal operational engine and its controls. The boundary prevents an explanation from becoming an action by mistake. The next chapter shows the complete design.

07

“Find the mountain photos” — visual semantics, at home

Tens of thousands of photos on a network drive. The first time, Metnos offers to index them: indexing runs in the background and pings you on Telegram when done. From then on:

> *Find the mountain photos.*

CATENA DI EXECUTOR

```
find_images_indices(query="mountain") → 20 previews + web gallery (100)
```

> *My daughter's photos at the seaside?*

CATENA DI EXECUTOR

```
find_persons_indices (faces: after a one-time “who is who” labeling) → gallery
```

models scene embeddings + face recognition + EXIF — all in-process, on your server · **privacy** no photo leaves home; the index is an inspectable SQLite file · **interface** previews in chat, full gallery in the browser

HONESTY

A search that combines person and scene must state which signals it applied. If fusion is unavailable, Metnos returns an explicit partial outcome instead of presenting it as a complete search.

08

GitHub maintenance with no daemon: two scheduled commands

The project's own support runs this way — no dedicated service, no ad-hoc code: two natural-language requests, registered as recurring tasks, that the planner turns into the usual executor chains.

> *Every 30 minutes: find the repo's new issues, skip those already in the local db, for each one search the db for similar already-resolved issues, classify and analyze it with the frontier tier, save the draft reply with status “prepared”, and notify me.*

> *After my approval: read from the db the “approved” issues not yet posted, publish the reply as a GitHub comment, mark them “posted”.*

CATENA DI EXECUTOR

```
find_issues_github → find_issues (dedup in the local db) → classify_entries +
consult_frontier → write_issues("prepared") → human approval → send_messages_github →
write_issues("posted")
```

state machine new → prepared → approved → posted: double-posting is prevented by state, not by hope · **consent** nothing reaches GitHub without an explicit “approve” · **scheduling** one sentence: “create a recurring task every 30 minutes: …”

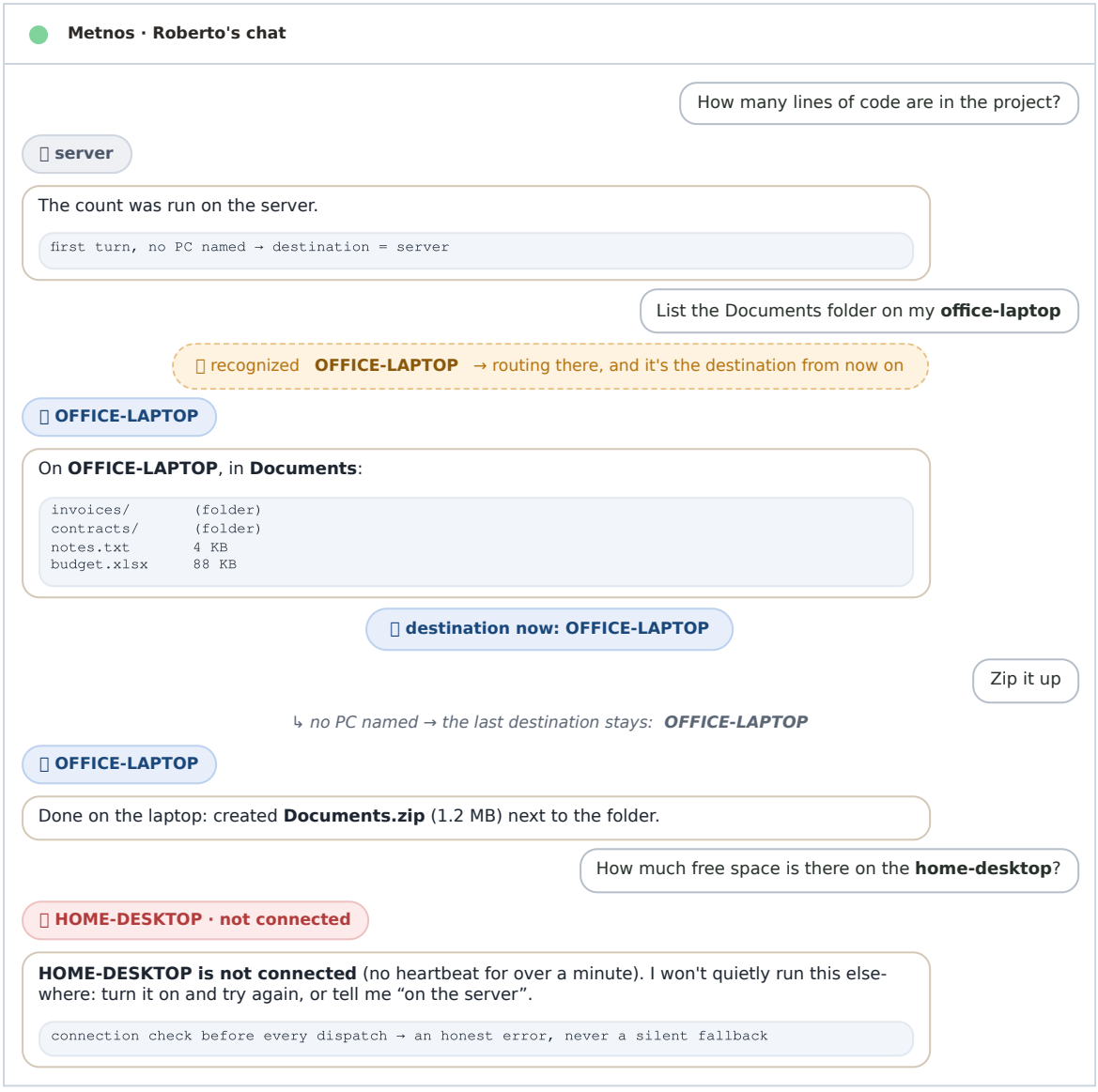
HONESTY

It's a declared dogfooding experiment: a Metnos instance helping maintain Metnos. If the frontier tier is down or over budget, the issue stays “new” and is flagged for manual treatment — never an invented reply, never a silent skip.

09

From the chat to one of your PCs, with a destination that sticks

The same natural-language commands, except execution can move to a PC you've *paired*. The chat starts with the destination on the **server**; at each change a **“destination now: …”** pill appears, telling you where the next command lands if you name no PC (the destination is sticky).



destination sticky: it stays on the last PC until you name another or say “on the server” · **connection** checked before every dispatch (heartbeat) — if the PC is off, an honest error, no covert fallback · **pairing** only the PCs you've registered yourself

> HONESTY
>
> Metnos never runs “blind” on a PC that doesn't answer: no recent heartbeat, no execution. It tells you the device is offline and leaves the choice to you — turn it on, run somewhere else, or stay on the server.

4 · The Tutor: Metnos can explain Metnos

*A governed system is useful only if its user can ask what it can do, why it stopped, and how to use a feature. The **Tutor** makes capabilities and procedures understandable without turning the manual into another all-powerful agent.*

Compiled knowledge, not an answer from memory

The Tutor indexes multilingual descriptions from manifests actually admitted by the loader, a closed set of explicitly listed public documentation, and a few high-authority curated procedures. Sources, vectors, and metadata live in one signed SQLite catalog replaced atomically. It does not scan Documents, mail, calendars, logs, ADRs, or conversations.

> CATENA DI EXECUTOR
>
> `admitted local sources → typed units → local BGE-M3 → signed catalog → answer with no tools`

Three questions it answers today

> *How do I read every mailbox without naming them one by one?*

It retrieves the current `read_messages` description and explains `account="all"`.

> *Device pairing: I would like to understand how it works.*

It uses the administrative guide only when the authenticated principal is authorized; restricted text never reaches the model for other users.

> *read_messages?*

Even a terse form can be understood semantically, without placing that phrase in a table.

Three things it does not do

It does not execute. “Read all my mail” continues through the normal engine.

It does not fill gaps. If sources are insufficient, it reports the gap.

It does not confuse roles. Audience and identity come from the authenticated channel, never from words in the question.

A new language does not require a third manual

Documentation remains Italian and English. A French, German, or Spanish question is compared by the multilingual embedding model with the best available source, usually English; the local model then answers in the current language. Fallback is per concept, so a partial translation cannot hide the rest of the catalog.

In evolution: live state and learning from gaps

> **F3/F4 roadmap — this is not released behavior yet.** Later phases add typed observations and a local maintenance loop; they do not give tools to the Tutor and do not train the model on user data.

F3, live state. Questions such as “Why did the 7 a.m. task not run?” or “Is the study laptop available for this operation?” will be able to attach bounded capsules from tasks, services, and owned devices. Every observation will have a TTL, owner, partial status, and source; no probe will be freely selected by the model.

F4, intelligent maintenance. Difficult examples such as “Can I run the digest only when the laptop comes back online?”, “How do I connect a mailbox from a provider the manual does not name?”, and “Why do two pairing explanations have different depth?” will not become three rigid cards. Recurrent gaps will be redacted, clustered locally, and classified as a missing source, stale source, conflict, or composition problem. F4 will prepare a verifiable proposal; administrative and safety content will still require human review.

> CATENA DI EXECUTOR
>
> `honest gap → local redacted clustering → knowledge-debt map → proposal → replay against past questions → signed catalog or rollback`

> HONESTY
>
> The Tutor matters precisely because it is not a RAG demo: it explains only the admitted system, knows when it does not know, and in later phases will learn where knowledge is missing without gaining operational authority.

5 · Where it lives and how you talk to it

Metnos runs on a Linux machine of yours. ***The LLM tiers are abstract roles****, not pinned models: a CPU endpoint, a local or remote model you already serve, and an optional frontier fallback are all first-class paths. Phones and laptops are clients: the mind, the memory and the audit live on one machine.*

Two surfaces, one mind

The browser: a chat served by the server itself (port 8770), with photo previews, the gallery, and the admin dashboards (synt proposals, executors, runs, turns). **Telegram**: the same brain in your pocket, with approval cards as inline buttons. For automation, the HTTP API (`/agent/turn`, streaming too) is the same door the chat uses.

The web chat also shows ✓/✗ feedback badges on every reply: your verdicts feed the care of the catalog.

Who you are, to the bot: pairing

Telegram is a public surface: anyone who knows the bot's name can write to it. Metnos binds every *(channel, sender)* pair to an autonomy level with a single-use signed code: `/pair PAIR.<code>` (Ed25519, expires in minutes) for pairings decided by the host; `/start <token>` for guests created from the admin UI. No central account, no password: a per-channel-per-sender directory only the host can edit. See pairing.

Language is data, not a constant

Every visible message, every tool description and every LLM prompt lives as *per-language data*: Italian and English are validated, a new language is added by translating the packs, no code changes (`prompts_cli add-language <code>` scaffolds the structure; the translation is reviewed and promoted by hand). Honesty: beyond IT and EN, today, it's untested ground. See multilang.

6 · What happens when it says “may I?”

This is the most important piece of the experience. Every action that changes the state of the world — sending, writing outside the granted perimeter, running shell — goes through four real components, in this order. The point is not to always ask: it is to ask well, and less and less.

PHASE	WHAT IT DOES
1 · Planner	Prepares the step (executor + arguments) in memory. Nothing is executed yet.
2 · Vaglio	First the *binary guard*: forbidden paths (`~/.ssh`, `/etc`…) and near-unrecoverable shell patterns (`rm -rf /`, `mkfs`, fork bombs) are denied outright, at every autonomy level. Then the *graded judge*: an alignment score against your telos (the aims written in `TELOS.md`).
3 · Policy	Capability class × autonomy level × persistent grants → one of *allow_silent*, *approval_required*, *deny*.
4 · Card	Three lines (what · on what · how redoable) + two buttons. Opaque single-use token, TTL 600 s, verification that the decider is the requester. Only after approval does the executor run — inside a bubblewrap sandbox with the profile declared in its signed manifest.

Less and less friction, never less control. The card modulates with recurrence: full the first times, then two lines, then one. And on the first approval of a kind you can grant a *territory* (“all writes inside `~/Documents/invoices/`”) — from then on, there, Metnos stops asking; the grant is revocable whenever you want. See approval_ux.

And once it has acted, it can walk back. Undo is not an LLM “doing its best”: it is a closed catalog of reverse patterns declared in each mutating executor's manifest (swap source/destination, delete what was created, restore the blob backup, delete by id). Honest counts: if it says it undid three things, it was three.

> **What no autonomy level unlocks.** The binary guard is not configurable from a file: forbidden paths and catastrophic patterns change only by changing the code. And generation is single-channel by principle (`SOUL.md`, six operating principles): the synt proposes *executors only*, never changes to itself, to the vaglio or to the runtime — and every proposal goes through your yes.

7 · Its memory: what it remembers and what it doesn't

Four distinct stores, with different lifetimes and write rules — SQLite tables and text files you can open, not an opaque vector somewhere.

STORE	SCOPE	WHAT IT HOLDS	RULE
Scratchpad	the turn	The observations of intermediate steps.	emptied at end of turn
Turn history	days	"What did I ask you last Tuesday?"	one record per turn, feeds the dashboards
Memories about you	forever	"Mom's birthday is April 23rd."	never written covertly: it proposes, you approve
Mnestome	about itself	Which tools worked together, with what outcome; and the attempts left halfway.	the substrate the synt reads before proposing

Memory of the world (the first three rows) and memory of itself (the mnestome), kept separate by design. The fourth store is the unusual one: the **mnestome** also records the gaps — requests no tool could close — because gaps are the engine of growth.

The detail that matters: Metnos never promotes a fact to permanent memory on its own. It proposes ("I noticed you often mention the new dentist: shall I save him as a regular contact?"), you decide. And a failed attempt is not forgotten: it remains as a *proto-mnest* — a recorded aspiration, with enough context to recognize the same shape next time. See mnestome and scratchpad.

8 · How it grows: synt, skills, backends

An executor is a small Python file that does one thing, with a manifest declaring its arguments, capabilities and sandbox profile. Once signed it is a stable artifact: it doesn't learn, it doesn't change. The growth intelligence lives elsewhere — in the ***synthesizer****.*

Synthesis in five stages

STAGE	WHAT IT PRODUCES
1 · Name	from the closed vocabulary — middle tier
2 · Signature	args schema, capabilities, reverse pattern
3 · Tests	4–6 birth tests: happy, empty, invalid args
4 · Description	the "tool's prompt" + affinity keywords
5 · Code	wise tier, local — `def invoke(args)` → lists

At the exit, the **admission gate** (7 layers): signature → affinity overlap → aging → sandbox → smoke test → LLM description-vs-code check → append-only audit · then human approval and Ed25519 signature.

Five small, focused prompts instead of one monolith separate responsibilities and make every stage verifiable. Only stage 1 sees the closed vocabulary; only stage 5 writes code.

Reactive synthesis (it happens mid-turn, scene 6) is the first step of a **cascade ordered by cost**: first *compose* existing tools, then *generate* the missing one; at night the introvert passes propose *merges* (two tools with overlapping traces), *generalizations* (three specialized tools collapse into one parametric) and *specializations* (a hot case splits off). Everything goes through your yes. Two installations, in two homes, will have different catalogs after six months — each shaped by real use. See synt and lifecycle.

Skills and backends: two axes that don't blur

Everything that ties Metnos to an external service is a **skill**: a group of capabilities *dormant until configured*, enable/disable at will (`system` · `photos` · `mail` · `web` · `geo` · `calendar` · `github` · `google-workspace` · `frontier`). Enabling a skill without its prerequisites is harmless: it stays visible and inert until its service or credential appears. You manage them from the command line or just by asking in chat ("which skills do I have?", "disable the web").

The **backend** is the other axis: *how* an action reaches a concrete service. The provider is chosen from configuration, deterministically — the planner never sees "Google": it sees `create_events`, and the resolver routes. Adding a second provider (say, GitLab next to GitHub) is *+1 backend file, zero new executors*: no per-provider photocopy tools, no provider bias in the local model. See skills_backends.

> **`system` is the skill that makes Metnos a computer assistant, not just a chatbot**: shell, sudo, packages, network mounts. Every privileged action requires a vaglio judgment, and the skill can be switched off entirely — Metnos stays out of the system until you switch it back on.

9 · People and devices: host, guests, remote executors

*Metnos is not multi-tenant in the SaaS sense: it is one household. Inside it there is a **host** — the server's owner, keeper of the signing key — and zero or more **guests**, each with their own level.*

LEVEL	FOR WHOM	WHAT IT ALLOWS
ReadOnly	A guest, a delicate channel	Read-only executors only; every write is politely refused before it even reaches the planner.
Supervised	Everyday use	The full turn; delicate actions raise the approval card on the same channel.
Full	The host	The widest perimeter the policy admits. Forbidden paths stay forbidden here too.

Scene 4 shows the model at work: the host configures, the guest receives, audit lines stay separate per pairing. Every channel has its own door, its own key, its own history.

Remote executors: how far the code can reach

One server, but not one filesystem: many things you'd want acted upon (laptop files, an open application, a screen) live elsewhere. Gateway, policy and memory stay on the server; *some executors* run on registered devices — a thin client with its own Ed25519 identity, admitted with a single-use signed code, revocable from the server in one gesture.

> HONESTY
>
> The remote client is a small Rust binary with its own sandbox and a signed build; the server invocation channel applies idempotency over flaky networks, replay protection and deferred result delivery after a disconnect. E2E tests cover the server-to-device boundary. Detail: remote executors.

10 · What the catalog enables, what the core forbids

The first list describes the current reference executor set, not an intrinsic Metnos domain. The second describes architectural boundaries that remain in force as the catalog changes.

✓ What the reference catalog enables today

- Multi-step turns from browser and Telegram, plans up to 12 steps
- Mail: reads, summarizes, archives, replies — sending always behind a card
- Files: finds, filters, moves, compresses, extracts — with declared undo
- Photos: indexes at home (scene, faces, EXIF) and searches by meaning
- Google Workspace with one OAuth: Gmail, Calendar, Drive, Docs, Sheets
- GitHub: issues, pull requests and repo files as first-class objects
- Web: search and reading via self-hosted services (search, geocoding)
- Recurring tasks in natural language (“every morning at 7…”)
- Multi-user on the same bot, with separate autonomies and audit
- Remote executors on registered Linux/Windows devices, for executors explicitly enabled for the platform
- Synthesizes new executors when missing; curates the catalog at night
- Speaks Italian and English by construction; every string is data
- Append-only audit of every action + observability dashboards

— What the core forbids, or the current catalog does not yet provide

- Act outside the house without consent: outbound capabilities go through the card
- Spend on the frontier tier beyond the configured ceiling
- Touch `~/.ssh`, `/etc`, `~/.gnupg` and the like: a list wired in code, not in a config file
- Run catastrophic patterns (`rm -rf /`, `mkfs`, fork bombs), at any autonomy
- Modify itself: the synt generates executors only, never the runtime or the vaglio
- Install an executor without your explicit approval
- Listen on a microphone (voice channel on hold, by choice)
- Use a site beyond the mandate granted to its credentials
- Run unsigned, unaudited or OS-incompatible executors on remote devices
- Present a photo search as complete when one requested signal is unavailable
- Guarantee languages beyond IT/EN: drop-in, but not yet tested

Cross-cutting rule: **no silent failure**. Cap reached → it says so (how many used, how many available) and asks before widening. Partial outcome → counted as partial. Never a “done” that doesn't match reality.

11 · Operational status

Metnos is an installable and operational pre-1.0 system, not a polished product. Contracts, security and verifiability take priority over compatibility with earlier internal choices. The generated catalog and the tests are the sources of current status.

AREA	OPERATIONAL STATUS
Capabilities	Mail, files, photos, web, Workspace, tasks and remote devices are exposed as executors with declared contracts. The generated catalog is the sole source for names and counts.
Installer	Included. The *managed* path prepares services, i18n data and signed executors, profiles the hardware and finishes with an application turn. The *custom* path connects endpoints already managed by the operator.
Public repository	A deterministic export of the components required for installation and operation, checked before publication.
Maturity	Pre-1.0: external interfaces and persistent data require care during upgrades. Partial failures, unavailable dependencies and unsupported capabilities must be stated, never hidden.

Trying it

The real requirement is hardware: a machine that can serve a capable local LLM. Tiers are abstract roles — a CPU endpoint or a model you already serve is fine; a weaker model means weaker planning, not a broken install. No GPU is required on principle.

```
$ git clone https://github.com/brunialti/metnos.git && cd metnos
$ bash install/bootstrap.sh --check   # pre-flight only — writes nothing
$ bash install/bootstrap.sh           # interactive, six phases, idempotent
```

And support is part of the experiment: the repo's issues are triaged by a Metnos instance (scene 8) — drafts prepared by the system, posted only after human approval. If the assistant can't help you run the assistant, that's a bug we want to see.

The fifty-cent description

If you take away one thing: Metnos is a self-hosted architecture for a governed agent, with five distinct commitments — **it asks before acting**, **builds tools inside verifiable rules**, **is reproducible like software**, **concentrates adaptive intelligence inside narrow-mandate executors**, and **takes signed executors to devices without moving decision and audit away from the server**. If that combination intrigues you, you are the right audience.

12 · Minimal glossary & further reading

Executor
A tool in the catalog: Python file + manifest + Ed25519 signature + sandbox profile. Vectorized by construction: lists in, lists out.

Intelligent executor
An executor with a bounded internal observation-and-action loop. It may use an LLM to disambiguate the route, while mandate, authority, input, output, and verification remain those of the public contract.

Remote executor
An executor run by a client on a registered device. The server plans, authorizes, signs, and audits; the device verifies and executes.

Synt
The synthesizer: the cascade that grows and curates the catalog (compose, generate; merge, generalize, specialize). Always behind human approval.

Vaglio
The two-phase evaluator between a proposed action and its execution: non-negotiable binary guard + graded judge against the telos.

Telos
Your aims, written in `TELOS.md` as soft tendencies. The judge measures proposed actions against them.

fastpath
The learned fast path: a request shape solved once is replayed with no LLM calls. Same outcome, minimal latency and cost. When a shape recurs across a whole group of similar requests, the generalized plan becomes an *autopath*.

Mnestome
The memory the system keeps about itself: successful co-activations (mnests) and attempts left halfway (proto-mnests). The substrate of growth proposals.

Skill / Backend
Two orthogonal axes: the skill governs *whether* a capability is active and trusted; the backend governs *how* it reaches a concrete provider, chosen by configuration and never by the model.

Pairing
The signed admission of a (channel, sender) pair to an autonomy level. Replaces login: no accounts, a directory only the host touches.

Approval card
Three lines and two buttons: what, on what, how redoable. It shortens with recurrence; it can grant revocable territories.

THE FULL GLOSSARY
`metnos.com/en/Metnos_Glossary_v1.html`

Alphabetical entries with links to canonical sources.

Keep reading

Architecture handbook

COMPLETE HANDBOOK · 20 MIN

`metnos.com/en/architecture/`

The system view and complete component atlas: topology, request flow, authority, and contracts.

Dialogue on aims and limits

REFLECTION · 25 MIN

`metnos.com/en/Metnos_Dialogue_v1.html`

The philosophical foundation of telos and vaglio, in four evenings.

A note on the code

NOTE · 5 MIN

`metnos.com/en/code.html`

The state of the code, told by the author, no varnish.

Metnos — *Quick tour & survival kit*
mētis + noûs · meant to be read in twenty minutes · the Architecture is the natural sequel
github.com/brunialti/metnos · `metnos.com` · metnos@metnos.com