QUICK TOUR · SURVIVAL KIT

Metnos

Un’architettura per agenti governati, estensibili e verificabili.

BOOKLET INTRODUTTIVO · ROBERTO BRUNIALTI · 2026

QUICK TOUR & SURVIVAL KIT

Metnos

*Un'architettura self-hosted per un **agente governato ed estensibile**. L'istanza di riferimento vive su un server Linux domestico, pianifica con un LLM locale, **si costruisce da sola gli strumenti che le mancano** dentro un vocabolario chiuso e verificabile, **chiede il permesso** prima di ogni azione che tocca il mondo — e quando dice di aver fatto una cosa, l'ha fatta davvero.*

mētis (μῆτις) «intelligenza astuta» + noûs (νοῦς) «mente»

I/O	chiuso	audit	IT + EN	0
CONTRATTI EXECUTOR STABILI	VOCABOLARIO GOVERNATO	AZIONI ED ESITI VERIFICABILI	TESTO E PROMPT LOCALIZZATI	CLOUD RICHIESTO PER PENSARE

stato: pre-1.0, installabile e operativo · **installer:** incluso nel repo · **licenza:** AGPL-3.0 · **codice:** github.com/brunialti/metnos

IN QUESTA PAGINA

1 · Cos'è Metnos, in trenta secondi

*Gli si affidano obiettivi in linguaggio naturale dal **browser**, da **Telegram** o da un altro canale ammesso. L'installazione di riferimento legge mail, file, foto, calendario, web e GitHub, ma sono esempi, non il confine: ogni capacità governabile che possa essere espressa come executor firmato può entrare nella stessa architettura. Si ferma a chiedere prima delle azioni che non si possono rifare. Non c'è un cloud di Metnos e non c'è un «registrati»: dati e controllo restano all'operatore.*

Quello che succede a ogni richiesta, in un disegno:

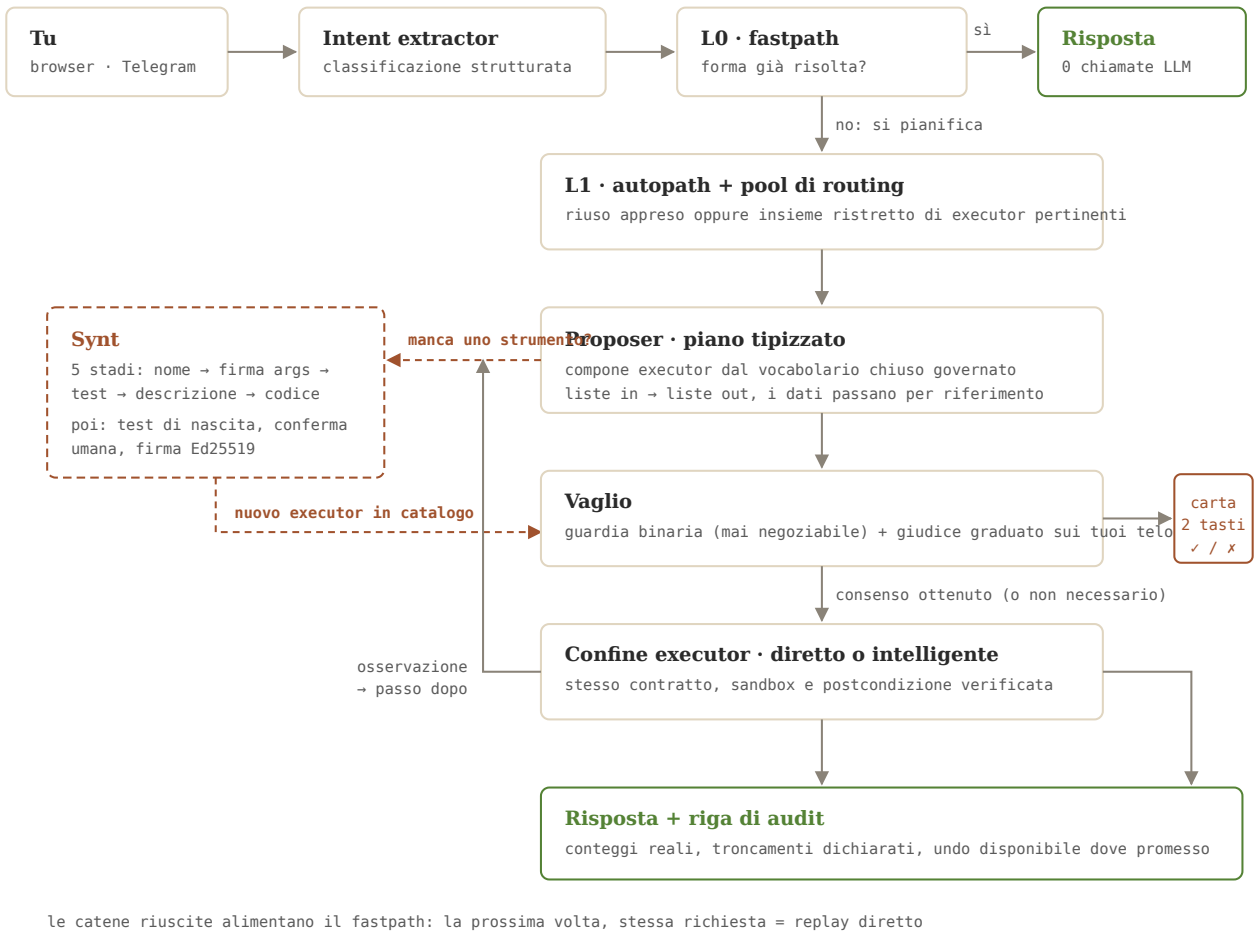


Fig. 1 — Il giro di un turno. L0 e L1 riusano piani validati; negli altri casi il proposer lavora su un pool ristretto. L'intelligenza interna di un executor non cambia il contratto visto dal planner.

Sei proprietà che lo distinguono da un agente generico:

- **Chiede prima di agire.** Un valutatore in due fasi — guardia binaria, poi giudice graduato sui vostri telos — decide se un'azione passa in silenzio, chiede conferma o viene rifiutata. La richiesta è tre righe e due bottoni. Vedi vaglio.
- **Costruisce gli strumenti che gli mancano.** Niente catalogo infinito scritto a mano: quando manca un executor, il *sintetizzatore* ne genera uno in pochi minuti — nome dal vocabolario chiuso, test di nascita, firma — con la vostra approvazione. Vedi synt e la scena 6.

- **Può agentificare un executor senza agentificare il piano.** Se il percorso è incerto ma lo scopo è stretto, un LLM interno disambigua i passi entro azioni e limiti fissati. Per il planner resta lo stesso executor tipizzato. Executor intelligenti.
- **Porta l'esecuzione dove vivono i dati.** Un executor compatibile può girare su un PC registrato; decisione, consenso, firma e audit restano sul server. Il dispositivo esegue, non diventa un secondo agente. Executor remoti.
- **È riproducibile come il software, non come un prompt.** Il pianificatore locale è vincolato a un vocabolario chiuso, il seme di generazione è fisso, i pareggi fra strumenti simili si rompono con affinità curate — non a testa o croce. Si può testare, verificare le regressioni, fidarsi.
- **Non mente sull'esito.** I conteggi riflettono ciò che è stato *davvero* fatto; i troncamenti vengono dichiarati; ogni azione che si dichiara annullabile lo è per costruzione (catalogo chiuso di pattern di reverse).

2 · Perché è diverso

Di agenti ce ne sono già, a famiglie intere — i framework da assemblare, gli agenti da terminale, i maggiordomi in chat (OpenClaw e simili) e l'ecosistema delle «skill» importabili. Metnos fa scelte diverse su più assi, e le fa per una ragione precisa: ***un pianificatore locale non è un modello di frontiera****, e va messo nelle condizioni di non sbagliare.*

	FRAMEWORK AGENTICO TIPICO	METNOS
Strumenti	Scritti a mano, importati o generati al volo in forma libera — poi eseguiti così come sono, coi privilegi dell'assistente.	Sintetizzati anche a runtime, ma da un **vocabolario chiuso e verificato**: firmati, testati alla nascita, invecchiati, ammessi solo dopo un cancello a 7 strati.
Instradamento	Il modello sceglie uno strumento a ogni turno: chiedete due volte, ottenete due piani.	**Deterministico per costruzione**: seme fisso, pareggi rotti da affinità curate. Stessa richiesta, stesso piano, a ogni esecuzione.
Output	Forma libera, per strumento.	Uniforme: **liste in ingresso, liste in uscita**, componibili fra passi senza colla.
Intelligenza interna	Concentrata nell'agente generale, che sceglie obiettivi e strumenti.	Può vivere anche dentro un executor a mandato ristretto, senza cambiarne input, output o autorità.
Collocazione	Gli strumenti girano dove vive l'agente.	Un executor compatibile può girare su un dispositivo firmato; il server conserva decisione e audit.
Undo	Raro, o nella migliore delle ipotesi «si spera».	Di prima classe: catalogo chiuso di pattern di reverse, move = COPY-poi-DELETE, `ok_count` onesto.
Lingua	Inglese, stringhe cablate nel codice.	i18n per costruzione: ogni stringa e prompt è *dato per-lingua*. IT+EN validate; altre lingue a incastro, non ancora testate.
Sicurezza	Ci si fida dell'autore del pacchetto.	*Non* ci si fida del pacchetto: è il pacchetto a dover passare i controlli.

La mappa, a metà 2026

Fotografia di luglio 2026: i nomi corrono, le famiglie restano.

1 · Framework di orchestrazione — LangChain/LangGraph, LlamaIndex, CrewAI, AutoGen (oggi confluito nel Microsoft Agent Framework), smolagents. Librerie aperte con cui uno sviluppatore si costruisce il proprio agente. Negli ultimi due anni hanno preso tutte la stessa forma: un grafo o un flusso di eventi garantisce il determinismo *della struttura* — instradamento fisso, punti di salvataggio, ripresa dopo un'interruzione — ma dentro ogni nodo il modello decide da capo a ogni esecuzione. Gli strumenti sono funzioni dai nomi liberi, scritte dallo sviluppatore; la sicurezza è affar suo, non una proprietà del telaio. Caso a sé smolagents, che fa scrivere al modello direttamente il codice Python da eseguire: flessibilità massima, e tutta la fiducia si sposta sulla sandbox che lo contiene.

2 · Stack dei fornitori di modelli — OpenAI (Responses API e Agents SDK), Anthropic (Claude Code e Agent SDK): prodotti curatissimi, costruiti attorno a modelli proprietari che vivono nel cloud. Il sistema di permessi di Claude Code — allow, ask, deny, più hook che bloccano fuori dal ciclo del modello — è il parente più prossimo della carta di conferma di Metnos; ma governa strumenti dai nomi liberi attorno a un modello remoto, mentre Metnos governa un vocabolario chiuso attorno a un modello che abita in casa. Sulla riproducibilità la distanza è netta: OpenAI dichiara il suo seme «best effort»; l'API di Anthropic, un seme, non ce l'ha proprio.

3 · Agenti autonomi di ingegneria — Devin, OpenAI Codex, Google Jules; sul versante aperto, OpenHands. Massima autonomia dentro una macchina virtuale isolata; il freno sta a valle, nella revisione umana della pull request. Quasi tutti in cloud e a consumo. OpenHands è l'eccezione — self-hosted, gira anche su modelli locali — ma resta un programmatore che esegue codice arbitrario dentro un contenitore, non un assistente dalle capacità enumerate.

4 · Assistenti personali self-hosted — la famiglia di Metnos, ed è qui che il confronto morde. Il fenomeno del 2026 è OpenClaw: stessa intenzione — un maggiordomo in casa vostra, che risponde in chat, anche con modelli locali — e governo opposto: skill libere in markdown più server MCP, permissivo in partenza, conferma umana riservata ai pagamenti. ZeroClaw, riscrittura minimale in Rust, ne è il cugino guardingo: pairing, spazi di lavoro isolati, liste di ammissione. Accanto a loro Goose (Block), Letta per la memoria persistente, Khoj per l'indice privato dei propri documenti, e l'assistente vocale di Home Assistant — che ha riscoperto per conto proprio il principio del fastpath: prima il riconoscimento deterministico dell'intento, il modello solo come riserva.

La presa universale, dietro il cancello — Il Model Context Protocol (MCP), nato in Anthropic a fine 2024 e passato a fine 2025 alla Linux Foundation, è diventato la presa universale degli strumenti: OpenAI, Google e Microsoft l'hanno adottato, migliaia di server espongono connettori pronti. Un client MCP tipico, però, legge le descrizioni in prosa che i server dichiarano e se ne fida: è il vettore dell'*avvelenamento degli strumenti* ben documentato dalla letteratura di sicurezza. Metnos non ignora quell'ecosistema e non lo innesta crudo: la via dichiarata è un executor dedicato che parla MCP — nome ammesso dal vocabolario, firma, sandbox e carta di conferma come per ogni altro strumento; il server esterno resta un ospite mediato, mai un'autorità. Migliaia di connettori diventano raggiungibili, ma nessuno entra senza passare dal cancello.

Dove il mainstream vince — Onestà dovuta anche qui. Chi innesta MCP senza mediazione eredita in un pomeriggio migliaia di connettori; il cancello di Metnos quell'ecosistema lo raggiunge, ma non alla stessa velocità. Un modello di frontiera interpellato a ogni turno affronta le richieste inedite e disordinate meglio di un pianificatore locale ancorato a rotte ripetibili. E le piattaforme gestite offrono osservabilità, strumenti di valutazione e una scalabilità che un progetto self-hosted non pareggia. Metnos non gareggia su questi assi: gareggia su provenienza, riproducibilità e confine — strumenti firmati, piano ripetibile, giudizio sui vostri telos, dati che non lasciano mai la vostra macchina.

Nove sistemi, una tavola

SISTEMA	STRUMENTI	PIANIFICAZIONE	FRENO · DOVE GIRA
LangGraph	Funzioni dai nomi liberi + MCP	Grafo fisso; decisioni rifatte a ogni esecuzione	Pausa umana nel grafo · OSS, cloud opzionale
smolagents	Il modello scrive codice Python	ReAct libero	Sandbox esterna · OSS, anche locale
OpenAI Agents SDK	Funzioni + strumenti ospitati + MCP	Nuova a ogni turno; seme «best effort»	Guardrail nel codice · modelli solo cloud
Claude Code	Strumenti di serie + MCP	Nuova a ogni turno; nessun seme	Permessi allow·ask·deny + hook · modelli solo cloud
Devin	VM isolata, con shell e browser	Frontiera, nuova a ogni esecuzione	Revisione della PR · SaaS a consumo
OpenHands	Codice arbitrario in container	Frontiera o modello locale	Sandbox + revisione · anche self-hosted
OpenClaw	Skill in markdown + MCP	Nuova a ogni esecuzione	Permissivo in partenza · self-hosted, locale
ZeroClaw	Strumenti con liste di ammissione	Nuova a ogni esecuzione	Pairing, spazi isolati · self-hosted, locale
Metnos	**Vocabolario chiuso, firmato, sintetizzato; MCP mediato**	**Deterministica: seme fisso + fastpath**	**Vaglio sui telos + carta di conferma · self-hosted, locale**

Due scelte che cambiano il perimetro

Non sono due nuovi strumenti: sono due modi generali di far lavorare gli executor senza allargare il linguaggio del planner.

INCERTEZZA DENTRO UN CONFINE

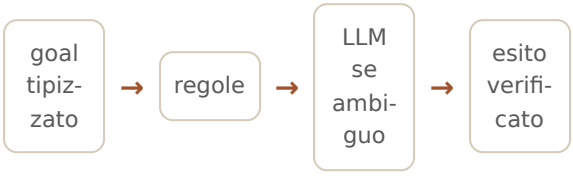
Agente dentro, executor fuori

Un executor **agentificato** riceve lo stesso input tipizzato e deve produrre lo stesso output di un executor diretto. La differenza è interna: può osservare, scegliere un passo e riosservare quando il percorso non è noto in anticipo.

Agente generale
Scompone obiettivi, sceglie strumenti e può attraversare domini.

Executor agentificato
Ha un solo incarico, azioni enumerate, budget e arresti fissati.

La distinzione è deliberata: chiamarlo semplicemente «agente» nasconderebbe chi può cambiare obiettivo o autorità. Metnos mette la capacità adattiva *dentro* un confine che il planner può ancora comporre e verificare.



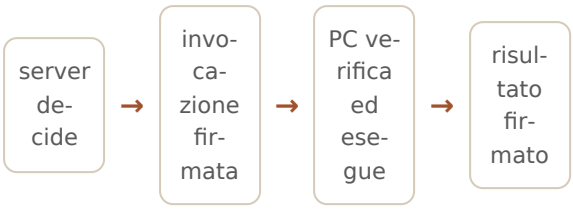
- Il LLM è un fallback per il percorso, non l'autorità che decide lo scopo.
- Segreti, consenso e permessi restano fuori dal modello.
- Senza una postcondizione osservabile non può dichiarare successo.

Esempio: `login_sites` cerca l'area di accesso, gestisce form a uno o più passi e controlla l'esito; non può navigare verso un altro obiettivo. Come funziona.

CALCOLO VICINO AI DATI

Stesso executor, un'altra macchina

Un **executor remoto** non è una copia di Metnos e non è un secondo agente. Il server continua a pianificare, applicare policy e consenso, firmare il lavoro e scrivere l'audit; un client sottile esegue sul dispositivo registrato.



- I dati possono restare sul PC che li possiede.
- Identità, piattaforme ammesse e collocazione fanno parte del contratto.
- PC spento o revocato significa errore esplicito, mai esecuzione nascosta sul server.

Implicazione: la casa diventa una superficie di esecuzione distribuita, ma conserva un solo punto di decisione e responsabilità. Dettagli e limiti.

Due assi indipendenti: diretto o agentificato descrive *come* l'executor risolve il compito; locale o remoto descrive *dove* gira. Ogni executor deve comunque dichiarare esplicitamente quali combinazioni supporta.

diretto · locale | agentificato · locale | diretto · remoto | agentificato · remoto

Perché Metnos non ha adottato il formato «skill» standard

I formati di skill in voga — le skill in markdown, i server MCP — sono comodi e pericolosi in pari misura: importate un pacchetto e l'assistente *ne esegue il codice coi propri privilegi*. Per un assistente che tocca file, mail e shell di casa vostra, è esecuzione remota di codice per disegno: basta un pacchetto malevolo o sciatto. Metnos sceglie la **sicurezza per costruzione**: vocabolario chiuso (non potete nominare uno strumento che la grammatica non ammette), cancello di ammissione a 7 strati per ogni executor nuovo o importato (firma → sovrapposizione di affinità → invecchiamento → sandbox → test di fumo → verifica LLM → audit append-only), consenso esplicito prima di ogni azione distruttiva. Lo slogan: *il pacchetto non si fida, il pacchetto si guadagna il posto*. L'ecosistema pubblico delle skill non viene ignorato: viene tradotto dentro questo modello, mai eseguito crudo.

Il determinismo è una scelta, non un caso

La maggior parte degli agenti tratta l'LLM come un oracolo da rilanciare a ogni turno. Metnos fa la scommessa opposta: un pianificatore *locale* su vocabolario *chiuso* si può rendere riproducibile — il seme di generazione è fissato, e quando due strumenti fratelli pareggiano decide un segnale di affinità curato, non il caso. Risultato misurabile: il banco di prova interno ripete ogni richiesta cinque volte e pretende lo stesso piano, cinque su cinque. E si compone: una forma di richiesta risolta una volta viene rigiocata dal fastpath **senza alcuna chiamata LLM** — latenza più bassa, costo zero, esito identico. La flessibilità (il modello di frontiera) resta disponibile, ma è la riserva, non il motore.

Cosa fa meglio un agente da terminale

Onestà dovuta: come compagno in tempo reale di chi scrive codice dentro un repository, un agente da terminale maturo resta superiore, con un ecosistema di estensioni che Metnos non ha né insegue. Ma Metnos non è definito dalle attività domestiche né da un solo verticale. È un'architettura operativa che tiene pianificazione, capacità eseguibili, autorità, memoria e audit dentro un unico confine governato. Posta, foto e calendario sono usi di riferimento; manutenere Metnos attraverso i suoi flussi GitHub è già un uso non domestico. I due tipi di sistema convivono benissimo.

3 · Nove casi, un solo sistema

I casi che seguono non sono moduli separati e non definiscono un perimetro esaustivo del prodotto: usano lo stesso planner, gli stessi contratti e le stesse guardie. La linea li tiene insieme; ogni tappa mostra soltanto la richiesta, la catena rilevante e il limite operativo che cambia. Il caso di manutenzione GitHub mostra deliberatamente che l'architettura non è confinata alla casa.

01

«Riassumi le mail di oggi»

> *Dammi il riassunto delle mail importanti di oggi.*

CATENA DI EXECUTOR

`read_messages → classify_entries → filter_entries → describe_entries → risposta`

METNOS

3 mail oggi: l'avvocato chiede il documento entro venerdì; la banca conferma il bonifico di ieri; la sorella sul pranzo di domenica. Vuoi che provi a rispondere a una di queste?

capability mail:read · **conferme** nessuna — i verbi di sola lettura saltano il giudizio · **latenza** ~3 s

02

«Rispondi tu all'avvocato»

> *Ok, rispondi tu all'avvocato: digli che invio il documento entro giovedì.*

Una mail in uscita verso un destinatario esterno non parte mai in silenzio, a nessun livello di autonomia. Metnos compone la bozza e su Telegram arriva questa bolla con due bottoni:

Posso inviare una mail?

`a avvocato@studio.example · oggetto: «consegna documento» · ~140 caratteri`

`reversibile: no · classe: mail:send`

✓ Approva ✗ Rifiuta

Toccate Approva: il daemon verifica che il token sia ancora pendente, non scaduto, e che *chi decide sia lo stesso mittente accoppiato che ha chiesto*. Poi: *«Spedita alle 11:34. Riga di audit #1247.»*

capability mail:send (irreversibile) · **token** monouso, TTL 600 s · **identità** chi approva == chi ha chiesto

Le tre righe della carta non sono un vezzo: sono *cosa*, *su che cosa*, *quanto è rifacibile* — il minimo per decidere a colpo d'occhio. Alla terza occorrenza identica la carta scende a due righe, poi a una. Vedi approval_ux.

03

«Ordina queste foto per anno e luogo» — e poi «annulla»

> *Ordina tutti i file immagine in ~/images creando sottocartelle per anno, premettendo al nome data e luogo di scatto. Se il luogo non c'è, metti «unknown».*

Non esiste uno strumento «ordina foto»: il pianificatore ne compone quattro della libreria, collegando l'uscita di ognuno all'ingresso del successivo. I dati non transitano dal modello: passano per riferimento (`from_step`).

1 `find_files(base_path="~/images", patterns=["*.jpg","*.png","*.heic"], recursive=true)`

→ 98 entries (con size e mtime già inclusi)

2 `filter_entries(from_step=1, kind="image")`

→ 98 entries · filtro in memoria, conferma il tipo dichiarato dal MIME

3 `get_files(from_step=2, fields=["dates","place"])`

→ 98 entries arricchite: EXIF → data; GPS → luogo via reverse-geocode (cache locale); senza GPS → "unknown"

4 `move_files(from_step=3, dst_template="~/images/{anno}/{data}_{luogo}_{nome}")`

→ 98 spostati, 0 errori · ogni coppia sorgente→destinazione registrata nel registro di undo

Quattro executor indipendenti, scelti e collegati dal pianificatore. Lo stesso flusso gestisce una o molte foto.

Il giorno dopo, cambio di idea:

> *Annulla l'ultima operazione.*

> METNOS
>
> Ho annullato l'ultima operazione: ho riportato 98 file ai loro percorsi originali.

undo reverse multistage: prima i file, poi le cartelle create — solo se rimaste vuote · **latenza undo** ~3 s su 98 file · **idempotente** un secondo «annulla» non fa nulla

> ONESTÀ
>
> Non tutto è annullabile, e Metnos non finge il contrario: una mail spedita non si richiama. L'undo dichiara «X non annullabili» e ribalta il resto. Ogni move è COPY-poi-DELETE: mai una cancellazione senza copia confermata.

04

«Ogni mattina, il registro scolastico — alla figlia, non a me»

Il portale della scuola richiede login. Il riassunto deve arrivare sul Telegram della figlia, accoppiata come ospite. Tre messaggi di preparazione, poi va da solo.

> *Aggiungi credenziali per portale-scuola.example: utente roberto.b, password ●●●●●●●.*

Metnos riconosce un valore che somiglia a una password e apre una carta dedicata: propone di cifrare il dato in una cassaforte locale e di *cancellare il valore in chiaro dal log del turno* (`<REDACTED:cred>`). Approvato, prova un login di test.

> *Ogni mattina alle 7 cerca le novità sul portale (compiti, voti, circolari), leggi anche i PDF allegati, e manda il riassunto delle 5 più rilevanti alla figlia su Telegram.*

CATENA DI EXECUTOR

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

METNOS

Scuola — riepilogo del giorno

1. Sciopero docenti (circolare 198) — lezioni sospese al mattino
2. Verifica di matematica spostata alla settimana successiva
3. Compiti del fine settimana: capitoli 4–6 + saggio breve
4. Riunione coordinatori venerdì 9/05, ore 15:30
5. Pubblicato il voto di latino

credenziali cifrate in locale, mai nel codice né nei log · **intelligenza interna** login_sites adatta il percorso, ma non può cambiare mandato · **cookie** riusato ~14 giorni; alla scadenza re-login automatico · **LLM** 1 sola chiamata per esecuzione (il riassunto) — l'HTML si estrae in modo deterministico · **latenza** ~25–40 s a esecuzione

ONESTÀ

Se anche il re-login fallisce (password cambiata sul portale), il task si mette in pausa e l'host riceve un avviso — non un silenzio, non un report inventato.

05

Da una cartella eterogenea a un dossier verificabile

Una sola richiesta attraversa PDF, Word, Excel e CSV sul PC associato; unisce ricerca temporale, deduplicazione, estrazione, controllo incrociato e tre output:

> *Nella cartella Documenti/Progetto Atlas, trova PDF, documenti e fogli di calcolo modificati negli ultimi 60 giorni. Elimina logicamente i duplicati confrontando firme e contenuto, estrai scadenze, importi, persone e decisioni, quindi crea un riepilogo ordinato per scadenza e un foglio con origine, data, responsabile, importo e livello di confidenza. Segnala dati contraddittori e file illeggibili. Salva i risultati in una nuova sottocartella, genera anche un archivio compresso, ma non sovrascrivere né cancellare nulla senza approvazione.*

CATENA DI EXECUTOR

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

METNOS

6 file candidati · 1 duplicato eliminato logicamente · 5 sorgenti uniche · 7 righe estratte. Contraddizione segnalata: budget approvato €120.000 con scadenza 30/09, revisione €135.000 con scadenza 15/09. Un PDF è illeggibile. Creati in una nuova sottocartella: `riepilogo.md`, `dati_estratti.xlsx` e `risultati.zip`.

formati PDF + DOCX + XLSX + CSV, letti in un unico flusso · **controlli** hash binario + firma + hash del contenuto, anomalie deterministiche · **sicurezza** output create-only; nessuna sovrascrittura o cancellazione · **collocazione** i file restano sul PC Windows associato

ONESTÀ

Prova E2E sul PC Windows reale: 11 passaggi su 11 riusciti in 74,9 s; foglio con intestazione più 7 righe, ZIP con i due risultati richiesti e hash degli 8 input invariati. Il livello di confidenza non trasforma una stima in certezza: per questo file illeggibili, duplicati e valori in conflitto restano visibili nel riepilogo. La ricevuta finale elenca cartella e tre artefatti; non espone la tabella tecnica dell'archiviatore.

06

«Posso leggere e scrivere file Excel?» — il sistema spiega se stesso

> *È possibile leggere e scrivere file Excel? Quali limiti ci sono?*

Questa non è un'operazione: è una domanda su Metnos. Il Tutor la separa dal pianificatore, confronta la domanda con il catalogo firmato e con la documentazione locale ammessa, quindi compone una risposta nella lingua corrente. Non esegue executor e non apre file.

CATENA DI EXECUTOR

`domanda → embedding locale → manifest ammessi + documentazione → filtro del ruolo → risposta fondata`

METNOS

Sì. Il catalogo installato distingue la lettura di cartelle di lavoro `.xlsx` dalla creazione o scrittura di fogli. Puoi indicare il foglio da leggere; per l'output puoi fornire colonne e righe provenienti da un passo precedente. Questa risposta descrive le capacità disponibili: non ho aperto o modificato file.

fonti descrizioni degli executor realmente ammessi · **lingue** fonti IT/EN, retrieval cross-lingua e risposta nella lingua della chat · **autorità** nessuno strumento e nessuna capacità di esecuzione

ONESTÀ

«Leggi tutti i miei fogli Excel» non è una domanda Tutor: passa al normale motore operativo con i suoi controlli. Il confine evita che una spiegazione diventi un'azione per equivoco. Il capitolo seguente mostra il disegno completo.

07

«Trova le foto in montagna» — semantica visiva, in casa

Decine di migliaia di foto su un disco di rete. La prima volta Metnos propone di indicizzarle: l'indicizzazione gira in background e avvisa su Telegram a lavoro finito. Da quel momento:

> *Trova le foto in montagna.*

CATENA DI EXECUTOR

`find_images_indices(query="montagna") → 20 anteprime + galleria web (100)`

> *Le foto della figlia al mare?*

CATENA DI EXECUTOR

`find_persons_indices (volti: dopo un'etichettatura una-tantum «chi è chi») → galleria`

modelli embedding di scena + riconoscimento volti + EXIF — tutto nel processo, sul vostro server · **privacy** nessuna foto lascia casa; l'indice è un file SQLite ispezionabile · **interfaccia** anteprime in chat, galleria completa nel browser

ONESTÀ

Una ricerca che combina persona e scena deve indicare quali segnali ha applicato. Se la fusione non è disponibile, Metnos restituisce un esito parziale esplicito invece di presentarlo come ricerca completa.

08

Manutenzione GitHub senza demone: due comandi schedulati

Il supporto del progetto stesso è gestito così — niente servizio dedicato, niente codice ad hoc: due richieste in linguaggio naturale, registrate come task ricorrenti, che il pianificatore trasforma nelle solite catene di executor.

> *Ogni 30 minuti: trova le issue nuove del repo, scarta quelle già nel db locale, cerca per ciascuna le issue simili già risolte, classificala e analizzala col tier frontier, salva la bozza di risposta con stato «prepared», e avvisami.*

> *Dopo la mia approvazione: leggi dal db le issue «approved» non ancora pubblicate, pubblica la risposta come commento su GitHub, segnala «posted».*

CATENA DI EXECUTOR

```
find_issues_github → find_issues (dedup nel db locale) → classify_entries +
consult_frontier → write_issues("prepared") → approvazione umana → send_messages_github
→ write_issues("posted")
```

macchina a stati new → prepared → approved → posted: il doppio invio lo impedisce lo stato, non la speranza · **consenso** niente raggiunge GitHub senza un «approva» esplicito · **scheduling** una frase: «crea un task ricorrente ogni 30 minuti: …»

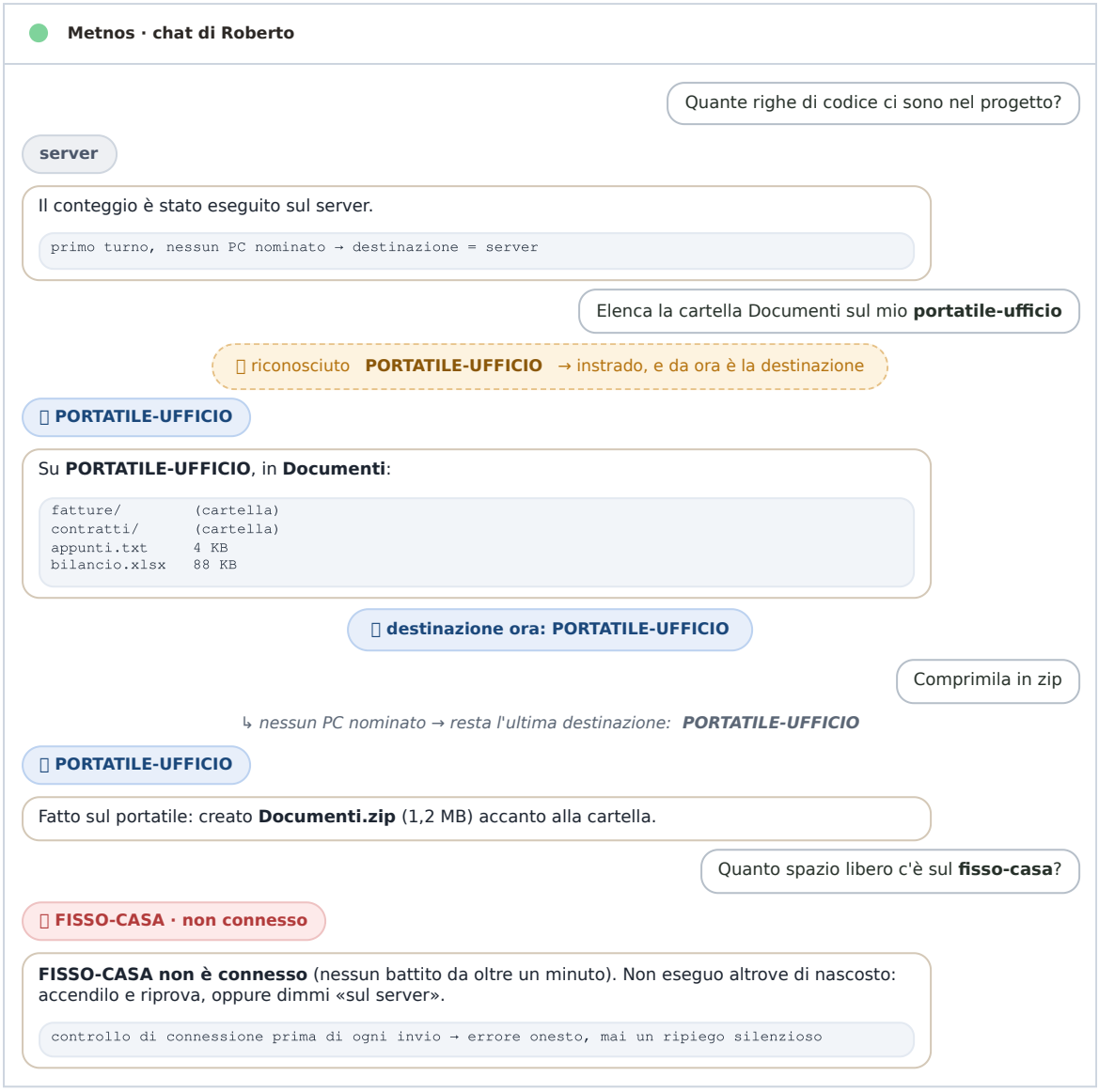
ONESTÀ

È un esperimento dichiarato: un'istanza di Metnos che aiuta a mantenere Metnos. Se il tier frontier è giù o fuori budget, l'issue resta «new» e viene marcata per trattamento manuale — mai una risposta inventata, mai uno skip silenzioso.

09

Dalla chat a un tuo PC, con la destinazione che resta

Gli stessi comandi in linguaggio naturale, ma l'esecuzione può spostarsi su un PC che avete *appaiato*. La chat parte con destinazione **server**; a ogni cambio compare una pillola **«destinazione ora: …»** che dice dove finisce il prossimo comando se non nominate nessun PC (destinazione appiccicosa).



destinazione appiccicosa: resta sull'ultimo PC finché non ne nominate un altro o non dite «sul server» · **connessione** verificata prima di ogni invio (battito) — se il PC è spento, errore onesto, nessun ripiego di nascosto · **appaiamento** solo i PC che avete registrato voi

> ONESTÀ
>
> Metnos non esegue mai «alla cieca» su un PC che non risponde: niente battito recente, niente esecuzione. Vi dice che il dispositivo è scollegato e vi lascia scegliere — accenderlo, ripetere altrove, o restare sul server.

4 · Il Tutor: Metnos sa spiegare Metnos

*Un sistema governato serve davvero solo se l'utente può chiedergli che cosa sa fare, perché si è fermato e come si usa una funzione. Il **Tutor** rende leggibili capacità e procedure senza trasformare il manuale in un altro agente onnipotente.*

Conoscenza compilata, non una risposta a memoria

Il Tutor indicizza le descrizioni multilingue dei manifest che il loader ha realmente ammesso, un insieme chiuso di documentazione pubblica esplicitamente elencata e poche procedure curate ad alta autorità. Fonti, vettori e metadati vivono in un catalogo SQLite firmato e sostituito atomicamente. Non scandisce Documenti, mail, calendari, log, ADR o conversazioni.

> CATENA DI EXECUTOR
>
> `fonti locali ammesse → unità tipizzate → BGE-M3 locale → catalogo firmato → risposta senza strumenti`

Tre domande che risolve oggi

> *Come leggo tutte le caselle email senza indicarle una per una?*

Recupera la descrizione corrente di `read_messages` e spiega il valore `account="all"`.

> *Pairing dei dispositivi: vorrei capirne la logica.*

Usa la guida amministrativa solo se il principal autenticato è autorizzato; il testo riservato non raggiunge il modello per gli altri utenti.

> *read_messages?*

Anche una forma telegrafica può essere riconosciuta semanticamente, senza inserire quella frase in una tabella.

Tre cose che non fa

Non esegue. «Leggi tutte le mail» prosegue nel motore normale.

Non completa i vuoti. Se le fonti non bastano, dichiara la lacuna.

Non confonde i ruoli. Audience e identità arrivano dal canale autenticato, mai dalle parole della domanda.

Una lingua nuova non richiede un terzo manuale

La documentazione resta italiana e inglese. Una domanda francese, tedesca o spagnola viene confrontata dal modello di embedding multilingue con la migliore fonte disponibile, di norma inglese; il modello locale formula poi la risposta nella lingua corrente. Il fallback avviene per singolo concetto, così una traduzione parziale non nasconde il resto del catalogo.

In evoluzione: stato live e apprendimento delle lacune

> **Roadmap F3/F4 — non è ancora comportamento rilasciato.** Le fasi successive aggiungono osservazioni tipizzate e un ciclo locale di manutenzione; non danno strumenti al Tutor e non addestrano il modello sui dati dell'utente.

F3, stato live. Domande come «Perché il task delle 7 non è partito?» o «Il portatile studio è disponibile per questa operazione?» potranno allegare capsule bounded da task, servizi e dispositivi posseduti. Ogni osservazione avrà TTL, owner, stato parziale e fonte; nessun probe sarà scelto liberamente dal modello.

F4, manutenzione intelligente. Esempi oggi difficili come «Posso far partire il riepilogo solo quando il portatile torna online?», «Come collego una mailbox di un provider che il manuale non nomina?» e «Perché due spiegazioni sul pairing hanno profondità diverse?» non diventeranno tre schede rigide. Le lacune ricorrenti saranno redatte, raggruppate localmente e classificate come fonte mancante, fonte obsoleta, conflitto o problema di composizione. F4 preparerà una proposta verificabile; contenuti amministrativi e di sicurezza resteranno soggetti a review umana.

> CATENA DI EXECUTOR
>
> `lacuna onesta → clustering locale redatto → mappa del debito conoscitivo → proposta → replay contro domande passate → catalogo firmato o rollback`

> ONESTÀ
>
> Il Tutor è importante proprio perché non è una demo di RAG: spiega soltanto il sistema ammesso, sa quando non sa, e nelle fasi future imparerà dove manca conoscenza senza acquisire autorità operativa.

5 · Dove vive e come gli si parla

*Metnos gira su una macchina Linux vostra. **I livelli LLM sono ruoli astratti**, non modelli fissati: un endpoint su CPU, un modello locale o remoto che già servite e un eventuale ripiego di frontiera sono percorsi di prima classe. Telefoni e laptop sono client: la mente, la memoria e l'audit vivono su una macchina sola.*

Due superfici, una mente

Il browser: una chat servita dal server stesso (porta 8770), con le anteprime delle foto, la galleria, e i cruscotti di amministrazione (proposte del synt, executor, esecuzioni, turni). **Telegram**: lo stesso cervello in tasca, con le carte di conferma come bottoni in linea. Per chi automatizza, l'API HTTP (`/agent/turn`, anche in streaming) è la stessa porta che usa la chat.

La chat web mostra anche i badge di feedback ✓/✗ per ogni risposta: il gradimento alimenta la cura del catalogo.

Chi siete, per il bot: il pairing

Telegram è una superficie pubblica: chiunque conosca il nome del bot può scrivergli. Metnos lega ogni coppia *(canale, mittente)* a un livello di autonomia con un codice firmato monouso: `/pair PAIR.<codice>` (Ed25519, scade in pochi minuti) per gli accoppiamenti decisi dall'host; `/start <token>` per gli ospiti creati dall'interfaccia di amministrazione. Niente account centrale, niente password: una rubrica per-canale-per-mittente che solo l'host modifica. Vedi pairing.

La lingua è un dato, non una costante

Ogni messaggio visibile, ogni descrizione di strumento e ogni prompt LLM vive come *dato per-lingua*: italiano e inglese sono validati, una lingua nuova si aggiunge per traduzione dei pacchetti, senza toccare codice (`prompts_cli add-language <codice>` prepara la struttura; la traduzione si rivede e si promuove a mano). Onestà: oltre IT e EN, oggi, è terreno non testato. Vedi multilang.

6 · Cosa succede quando dice «posso?»

È il pezzo di esperienza più importante del progetto. Ogni azione che cambia lo stato del mondo — inviare, scrivere fuori dal perimetro concesso, eseguire shell — passa per quattro componenti reali, in quest'ordine. Il punto non è chiedere sempre: è chiedere bene, e sempre meno.

FASE	COSA FA
1 · Planner	Prepara il passo (executor + argomenti) in memoria. Niente è ancora eseguito.
2 · Vaglio	Prima la *guardia binaria*: percorsi vietati (`~/.ssh`, `/etc`…) e pattern di shell quasi irrecuperabili (`rm -rf /`, `mkfs`, fork bomb) sono negati a prescindere, a ogni livello di autonomia. Poi il *giudice graduato*: un punteggio di allineamento ai vostri telos (i fini scritti in `TELOS.md`).
3 · Policy	Classe di capability × livello di autonomia × concessioni persistenti → uno fra *allow_silent*, *approval_required*, *deny*.
4 · Carta	Tre righe (cosa · su che cosa · quanto è rifacibile) + due bottoni. Token opaco monouso, TTL 600 s, verifica che chi decide sia chi ha chiesto. Solo dopo l'approvazione l'executor parte — dentro una sandbox bubblewrap col profilo dichiarato nel suo manifest firmato.

Sempre meno fastidio, mai meno controllo. La carta si modula sulla ricorrenza: piena le prime volte, poi due righe, poi una. E alla prima approvazione di una tipologia potete concedere un *territorio* («tutte le scritture dentro `~/Documenti/fatture/`») — da quel momento, lì, Metnos smette di chiedere; la concessione è revocabile quando volete. Vedi approval_ux.

E quando ha agito, può tornare indietro. L'undo non è un LLM che «ci prova»: è un catalogo chiuso di pattern di reverse dichiarati nel manifest di ogni executor mutante (scambia sorgente/destinazione, cancella i creati, ripristina il backup del blob, cancella per id). Conteggi onesti: se dice che ha annullato tre cose, erano tre.

> **Ciò che nessuna autonomia sblocca.** La guardia binaria non è configurabile da file: percorsi vietati e pattern catastrofici si cambiano solo cambiando il codice. E la generazione è monocanale per principio (`SOUL.md`, sei principi operativi): il synt propone *solo executor*, mai modifiche a se stesso, al vaglio o al runtime — e ogni proposta passa dal vostro sì.

7 · La sua memoria: cosa ricorda e cosa no

Quattro depositi distinti, con durate e regole di scrittura diverse — tabelle SQLite e file di testo che potete aprire, non un vettore opaco da qualche parte.

DEPOSITO	ORIZZONTE	COSA CONTIENE	REGOLA
Scratchpad	il turno	Le osservazioni dei passi intermedi.	si svuota a fine turno
Storico turni	giorni	«Cosa ti ho chiesto martedì scorso?»	un file per turno, alimenta i cruscotti
Memorie su di voi	per sempre	«Il compleanno della mamma è il 23/4.»	mai scritte di nascosto: propone, voi approvate
Mnestoma	su di sé	Quali strumenti hanno collaborato, con che esito; e i tentativi rimasti a metà.	il substrato che il synt legge per proporre

Memoria del mondo (le prime tre righe) e memoria di sé (il mnestoma), tenute separate per disegno. Il quarto deposito è il più insolito: il **mnestoma** registra anche i vuoti — le richieste che nessuno strumento ha saputo chiudere — perché sono il motore della crescita.

Il dettaglio che conta: Metnos non promuove mai un fatto a memoria permanente da solo. Propone («ho notato che parli spesso del nuovo dentista: lo salvo come contatto abituale?»), voi decidete. E un tentativo fallito non viene dimenticato: resta come *proto-mnest* — un'aspirazione registrata, con abbastanza contesto per riconoscere la stessa forma la prossima volta. Vedi mnestoma e scratchpad.

8 · Come cresce: synt, skill, backend

Un executor è un piccolo file Python che fa una cosa sola, con un manifest che ne dichiara argomenti, capability e profilo sandbox. Una volta firmato è un artefatto stabile: non impara, non cambia. L'intelligenza di crescita sta altrove — nel ***sintetizzatore****.*

La sintesi in cinque stadi

STADIO	COSA PRODUCE
1 · Nome	dal vocabolario chiuso — tier middle
2 · Firma	schema args, capability, pattern di reverse
3 · Test	4–6 prove di nascita: felice, vuoto, invalido
4 · Descrizione	il «prompt dello strumento» + parole di affinità
5 · Codice	tier wise, locale — `def invoke(args)` → liste

All'uscita, il **cancello di ammissione** (7 strati): firma → sovrapposizione di affinità → invecchiamento → sandbox → test di fumo → verifica LLM descrizione-vs-codice → audit append-only · poi conferma umana e firma Ed25519.

Cinque prompt piccoli e focalizzati invece di uno monolitico separano le responsabilità e rendono ogni stadio verificabile. Solo lo stadio 1 vede il vocabolario chiuso; solo il 5 scrive codice.

La sintesi reattiva (capita in mezzo a un turno, scena 6) è il primo gradino di una **cascata per costo crescente**: prima *componi* gli esistenti, poi *genera* il mancante; di notte le passate introvertive propongono *fusioni* (due strumenti con tracce sovrapposte), *generalizzazioni* (tre specializzati collassano in uno parametrico) e *specializzazioni* (un caso caldo si stacca). Tutto passa dal vostro sì. Due installazioni, in due case, dopo sei mesi avranno cataloghi diversi — ognuno plasmato dall'uso reale. Vedi synt e lifecycle.

Skill e backend: due assi che non si confondono

Tutto ciò che lega Metnos a un servizio esterno è una **skill**: un gruppo di capacità *dormiente finché non configurata*, attivabile e disattivabile a piacere (`system` · `photos` · `mail` · `web` · `geo` · `calendar` · `github` · `google-workspace` · `frontier`). Abilitare una skill senza i prerequisiti è innocuo: resta visibile e inerte finché il suo servizio o la sua credenziale non compaiono. Si gestiscono da riga di comando o chiedendolo in chat («quali skill ho?», «disattiva il web»).

Il **backend** è l'altro asse: *come* un'azione raggiunge un servizio concreto. Il provider si sceglie da configurazione, deterministicamente — il pianificatore non vede mai «Google»: vede `create_events`, e il resolver instrada. Aggiungere un secondo provider (es. GitLab accanto a GitHub) è *+1 file di backend, zero executor nuovi*: niente strumenti fotocopia per provider, niente pregiudizio di scelta nel modello locale. Vedi skills_backends.

> **`system` è la skill che rende Metnos un assistente del computer, non solo un chatbot**: shell, sudo, pacchetti, mount di rete. Ogni azione privilegiata richiede un giudizio del vaglio, e la skill si può spegnere del tutto — Metnos resta fuori dal sistema finché non la riaccendete voi.

9 · Persone e dispositivi: host, ospiti, executor remoti

*Metnos non è multi-tenant in senso SaaS: è una casa sola. Dentro quella casa c'è un **host** — il proprietario del server, custode della chiave di firma — e zero o più **ospiti**, ognuno col suo livello.*

LIVELLO	PER CHI	COSA PUÒ
ReadOnly	Un ospite, un canale delicato	Solo executor di lettura; ogni scrittura viene rifiutata con garbo prima ancora di arrivare al pianificatore.
Supervised	L'uso quotidiano	Tutto il turno; le azioni delicate alzano la carta di conferma sul canale stesso.
Full	L'host	Il perimetro più ampio che la policy ammette. I percorsi vietati restano vietati anche qui.

La scena 4 mostra il modello al lavoro: l'host configura, l'ospite riceve, le righe di audit restano separate per accoppiamento. Ogni canale ha la sua porta, la sua chiave, la sua storia.

Executor remoti: dove può arrivare il codice

Un solo server, ma non un solo filesystem: tante cose su cui vorreste agire (i file del laptop, un'applicazione aperta, uno schermo) vivono altrove. Gateway, policy e memoria restano sul server; *alcuni executor* girano su dispositivi registrati — un client sottile con identità Ed25519 propria, ammesso con un codice firmato monouso, revocabile dal server in un gesto.

> ONESTÀ
> Il client remoto è un piccolo binario Rust con sandbox e build firmata; sul server il canale di invocazione applica idempotenza su rete intermittente, protezione dai replay e consegna differita del risultato dopo una disconnessione. I test E2E coprono il confine server-dispositivo. Dettaglio: executor remoti.

10 · Cosa abilita il catalogo, cosa vieta il core

Il primo elenco descrive il set di executor dell'istanza di riferimento, non un dominio intrinseco di Metnos. Il secondo descrive confini architetturali che restano validi quando il catalogo cambia.

✓ Cosa abilita oggi il catalogo di riferimento

- Turni multi-passo da browser e Telegram, con piani fino a 12 passi
- Mail: legge, riassume, archivia, risponde — l'invio sempre dietro carta
- File: cerca, filtra, sposta, comprime, estrae — con undo dichiarato
- Foto: indicizza in casa (scena, volti, EXIF) e cerca per significato
- Google Workspace con un solo OAuth: Gmail, Calendar, Drive, Docs, Sheets
- GitHub: issue, pull request e file del repo come oggetti di prima classe
- Web: ricerca e lettura via servizi self-hosted (ricerca, geocoding)
- Compiti ricorrenti in linguaggio naturale («ogni mattina alle 7…»)
- Multi-utente sullo stesso bot, con autonomie e audit separati
- Executor remoti su dispositivi Linux/Windows registrati, per executor esplicitamente abilitati alla piattaforma
- Sintetizza executor nuovi quando mancano; cura il catalogo di notte
- Parla italiano e inglese per costruzione; ogni stringa è un dato
- Audit append-only di ogni azione + cruscotti di osservabilità

— Cosa vieta il core, o il catalogo corrente non offre ancora

- Agire fuori casa senza consenso: le capability in uscita passano dalla carta
- Spendere sul tier di frontiera oltre il tetto configurato
- Toccare `~/.ssh`, `/etc`, `~/.gnupg` e simili: lista cablata nel codice, non in un file di configurazione
- Eseguire pattern catastrofici (`rm -rf /`, `mkfs`, fork bomb), a qualunque autonomia
- Modificare se stesso: il synt genera solo executor, mai il runtime o il vaglio
- Installare un executor senza la vostra conferma esplicita
- Ascoltare da un microfono (canale voce in attesa, per scelta)
- Usare un sito oltre il mandato concesso alle sue credenziali
- Eseguire su dispositivi remoti executor non firmati, non auditati o non dichiarati compatibili col sistema operativo
- Presentare come completa una ricerca foto quando uno dei segnali richiesti non è disponibile
- Garantire lingue oltre IT/EN: a incastro, ma non ancora testate

Regola trasversale: **nessun fallimento silenzioso**. Cap raggiunto → lo dice (quanti usati, quanti disponibili) e chiede se allargare. Esito parziale → contato come parziale. Mai un «fatto» che non corrisponde alla realtà.

11 · Stato operativo

Metnos è un sistema pre-1.0 installabile e operativo, non un prodotto rifinito. Contratti, sicurezza e verificabilità hanno priorità sulla compatibilità con scelte interne precedenti. Il catalogo generato e i test sono le fonti dello stato corrente.

AREA	STATO OPERATIVO
Capacità	Mail, file, foto, web, Workspace, task e dispositivi remoti sono esposti come executor con contratti dichiarati. Il catalogo generato è la fonte unica per nomi e conteggio.
Installer	Incluso. Il percorso *managed* prepara servizi, dati i18n ed executor firmati, profila l'hardware e chiude con un turno applicativo. Il percorso *custom* collega endpoint già gestiti dall'operatore.
Repo pubblico	Esportazione deterministica dei componenti necessari all'installazione e all'esercizio, sottoposta a controlli prima della pubblicazione.
Maturità	Pre-1.0: interfacce esterne e dati persistenti richiedono cautela negli aggiornamenti. Fallimenti parziali, dipendenze non disponibili e capacità non supportate devono essere dichiarati, mai nascosti.

Provarlo

Il requisito vero è l'hardware: una macchina che regga un LLM locale capace. I livelli sono ruoli astratti — un endpoint su CPU o un modello già in servizio vanno bene; un modello più debole significa pianificazione più debole, non un'installazione rotta. Nessuna GPU è richiesta per principio.

```
$ git clone https://github.com/brunialti/metnos.git && cd metnos
$ bash install/bootstrap.sh --check    # solo verifica: non scrive nulla
$ bash install/bootstrap.sh            # interattivo, sei fasi, idempotente
```

E il supporto è parte dell'esperimento: le issue del repo vengono istruite da un'istanza di Metnos (scena 8) — bozze preparate dal sistema, pubblicate solo dopo approvazione umana. Se l'assistente non sa aiutarvi a far girare l'assistente, quello è un bug che vogliamo vedere.

La descrizione da mezzo euro

Se portate via una cosa sola: Metnos è un'architettura self-hosted per un agente governato, con cinque impegni peculiari — **chiede prima di agire**, **costruisce strumenti dentro regole verificabili**, **è riproducibile come il software**, **concentra l'intelligenza adattiva dentro executor a mandato ristretto** e **porta executor firmati sui dispositivi senza spostare decisione e audit dal server**. Se questa combinazione vi incuriosisce, siete il pubblico giusto.

12 · Glossario minimo & letture successive

Executor
Uno strumento del catalogo: file Python + manifest + firma Ed25519 + profilo sandbox. Vettoriale per costruzione: liste in ingresso, liste in uscita.

Executor agentificato
Un executor con un ciclo interno limitato di osservazione e azione. Può usare un LLM per disambiguare il percorso, ma mandato, autorità, input, output e verifica restano quelli del contratto pubblico.

Executor remoto
Un executor eseguito da un client su un dispositivo registrato. Il server pianifica, autorizza, firma e registra; il dispositivo verifica ed esegue.

Synt
Il sintetizzatore: la cascata che fa crescere e cura il catalogo (componi, genera; fondi, generalizza, specializza). Sempre dietro conferma umana.

Vaglio
Il valutatore in due fasi fra un'azione proposta e la sua esecuzione: guardia binaria non negoziabile + giudice graduato sui telos.

Telos
I vostri fini, scritti in `TELOS.md` come tendenze morbide. Il giudice misura le azioni proposte contro di essi.

fastpath
Il sentiero veloce appreso: una richiesta già risolta viene rigiocata senza chiamate LLM. Stesso esito, latenza e costo minimi. Quando una forma si ripete per un intero gruppo di richieste simili, il piano generalizzato diventa un *autopath*.

Mnestoma
La memoria che il sistema tiene su di sé: co-attivazioni riuscite (mnest) e tentativi rimasti a metà (proto-mnest). Il substrato delle proposte di crescita.

Skill / Backend
Due assi ortogonali: la skill governa *se* una capacità è attiva e fidata; il backend governa *come* raggiunge un provider concreto, scelto da configurazione e mai dal modello.

Pairing
L'ammissione firmata di una coppia (canale, mittente) a un livello di autonomia. Sostituisce il login: niente account, una rubrica che solo l'host tocca.

Carta di conferma
Tre righe e due bottoni: cosa, su che cosa, quanto è rifacibile. Si accorcia con la ricorrenza; può concedere territori revocabili.

IL GLOSSARIO COMPLETO
`metnos.com/it/Metnos_Glossario_v1.html`

Voci alfabetiche con rimandi alle fonti canoniche.

Continua a leggere

Manuale di architettura

MANUALE COMPLETO · 20 MIN

`metnos.com/it/architecture/`

La visione d'insieme e l'atlante completo dei componenti: topologia, flusso, autorità e contratti.

Dialogo sui fini e i limiti

RIFLESSIONE · 25 MIN

`metnos.com/it/Metnos_Dialogo_v1.html`

La fondazione filosofica di telos e vaglio, in quattro serate.

Una nota sul codice

NOTA · 5 MIN

`metnos.com/it/code.html`

Lo stato del codice raccontato dall'autore, senza trucco.

Metnos — *Quick tour & survival kit*
mētis + noûs · pensato per essere letto in venti minuti · l'Architettura è il seguito naturale
github.com/brunialti/metnos · `metnos.com` · metnos@metnos.com