indaloop

AI × PROCESSI × AZIENDE

CASO STUDIO · REPORT TECNICO

MISURARE PRIMA DI OTTIMIZZARE

Come una scelta di modello basata sulla reputazione è fallita in produzione in MadTools, cosa ha rivelato un harness di valutazione costruito sul compito reale, e perché abbiamo separato l'LLM dall'aritmetica.

DI DANIELE MARCHETTI · MADTOOLS · GIUGNO–LUGLIO 2026 · 25 MINUTI DI LETTURA

72,5%	6,23 g	38/38	0
timeout nella run di produzione fallita (37 su 51)	errore medio del modello scelto, da 41,81 g	item del golden set entro l'1% dall'oracolo	exit code della prima run ibrida non presidiata

IN SINTESI

COSA È SUCCESSO, IN UNA PAGINA

A fine giugno 2026 ho messo in produzione notturna una pipeline di stima del peso basata su LLM in MadTools, produttore italiano di utensili da taglio. La scelta del motore si basava sul consenso della community: **Gemma-4-12B**, all'epoca ampiamente reputato il miglior modello open-weight di piccola taglia per l'output JSON strutturato. L'ho usato end-to-end: una chiamata al modello per item, in cui l'LLM interpretava la descrizione libera, sceglieva famiglia geometrica e formula, eseguiva il calcolo e restituiva un peso.

Il fallimento verificato è arrivato con la run del **14 luglio 2026**: 51 item inviati al modello, 37 timeout definitivi, 6 pesi scritti, 8 solo motivazione. La run è durata 2 ore e 13 minuti ed è uscita con exit code 1. *La reputazione non aveva selezionato un sistema funzionante.*

Questa pagina documenta cosa ho costruito dopo: un harness di valutazione specifico per il compito, un benchmark cross-modello sotto un contratto di runtime identico, e la riprogettazione architetturale che ne è seguita. L'harness — un golden set di 54 item con ground truth intermedia strutturata, un oracolo aritmetico indipendente, un runner con metadati riproducibili e cinque gruppi di metriche (validità tecnica, performance, triage su `ToolCalculator`, decisione, accuratezza rispetto alla formula) — ha fatto emergere due risultati che hanno guidato tutto il resto del progetto.

Primo: un default silenzioso di Ollama (`num_ctx=4096`) stava troncando un system prompt di circa 6.900 token a quanto il runtime riusciva a contenere. Impostare `num_ctx=8192` sullo stesso benchmark di `qwen3:14b` ha fatto scendere il MAE da 41,81 g a 6,23 g e ha portato gli output confrontabili da 30 a 38 su 38. *Quella misura riguardava l'infrastruttura, non il modello.* Qualunque candidato valutato prima di quella correzione sarebbe stato misurato contro un prompt mutilato.

Secondo: nel benchmark cross-modello su sei configurazioni ospitate in locale, `qwen3:14b` in modalità `system-runtime` è stata l'unica run a combinare copertura numerica completa (38/38 confrontabili), errore basso rispetto alla formula (MAE 6,23 g, 35/38 entro l'1% dall'oracolo) e nessun timeout o errore di output. Tutte le altre configurazioni hanno fallito su una porzione consistente del set, e il sistema in carica `madtools-peso` (Gemma-4-12B embedded) restava indietro rispetto al candidato di migrazione sia su copertura sia su accuratezza.

Quell'evidenza, insieme a discrepanze moltiplicative eterogenee osservate in un successivo replay di 108 item di produzione, ha motivato una decisione architetturale precisa: **separare il parser LLM dal calcolatore deterministico**. L'LLM estrae parametri semantici strutturati; un calcolatore C# deterministico applica le formule versionate. Da quando è stata implementata, la pipeline ibrida ha superato un controllo di sviluppo sul golden set (38/38 entro l'1%, MAE 0,00 g rispetto all'oracolo), un replay di 108 item fuori campione e una run di produzione notturna non presidiata il 24 luglio (46 pesi numerici e 23 rifiuti espliciti su 69 item eleggibili, exit code 0).

1. IL PROBLEMA OPERATIVO

UN PESO, DA UNA RIGA DI TESTO LIBERO

MadTools è un produttore italiano di utensili da taglio: punte, frese, seghe, carotatori, frese a profilo speciale e utensili con inserti riportati per la lavorazione industriale. Ne sono il titolare e lo sviluppatore principale del sistema qui descritto. Gli ordini cliente sono gestiti su una board Monday.com chiamata PRODOTTI ORDINATI; ogni riga è un utensile, descritto in italiano, in un breve campo di testo libero.

Il problema vive nella descrizione. Ecco un esempio pulito:

```
PUNTA A ELICA HSS Ø12x100 Z=4 RIF. 58YT69
```

Una punta elicoidale, acciaio super rapido, diametro 12 mm, lunghezza totale 100 mm, 4 taglienti, riferimento cliente 58YT69. Un attrezzista la legge in meno di un secondo. Un parser a espressioni regolari la legge solo se il formato è vicino a un piccolo insieme di forme canoniche.

Non tutti gli utensili sulla board sono prodotti internamente. Due categorie escono dalla nostra linea di produzione, e sono la ragione per cui questo progetto esiste.

Lavorazioni conto terzi

I clienti ci inviano utensili di loro proprietà per rigenerazione, affilatura, sostituzione placchetta o modifica profilo — `RIGENERATURA`, `AFFILATURA`, `SOSTITUZIONE PLACCHETTA`, `MODIFICA PROFILO`. Non produciamo questi utensili: lavoriamo su quelli che il cliente ci ha mandato, e li restituiamo.

Rivendite di produzione esterna

Un piccolo numero di utensili che non produciamo noi ma acquistiamo da terzi e rivendiamo.

Per gli utensili che produciamo, il peso è noto dal processo produttivo. Per queste due categorie no. Ed è esattamente il peso che serve per costruire il *DDT* (documento di trasporto) e per dimensionare il sovrappacco di ogni spedizione di ritorno al cliente. L'alternativa ingenua è la pesatura manuale al

momento dell'imballo: prendere l'utensile dal vassoio in uscita, metterlo sulla bilancia, registrare il numero, riprendere. L'obiettivo del sistema è produrre una stima numerica del peso direttamente dalla descrizione libera, con cadenza notturna. Dove la descrizione non consente un calcolo, il sistema scrive un null esplicito con un codice motivo, e per quell'item si torna alla pesatura manuale.

Il compito ha una struttura. Ogni utensile da taglio appartiene a una tra poche famiglie geometriche — cilindrico pieno, carotatore, prismatico, sega, barra, inserto riportato — ognuna con la propria formula di peso. Ogni materiale ha una densità nota (metallo duro integrale 14,50 g/cm³; acciaio super rapido 8,20 g/cm³; assemblaggi brasati e meccanici 7,85 g/cm³). Le formule sono polinomi di dimensioni e coefficienti di taratura. In linea di principio, un parser ben scritto può risolvere la maggioranza delle descrizioni in un peso senza alcuna modellazione statistica.

Il codice ne conteneva già uno. `ToolCalculator`, parser e calcolatore deterministico, gestisce correttamente i casi ben formati, in millisecondi, senza alcuna dipendenza esterna. *La sua debolezza è la variabilità dell'input, non l'aritmetica.*

Il resto interessante — descrizioni che arrivano da ordini d'acquisto cliente, da operatori che digitano in fretta, da fogli di calcolo importati, da formati storici precedenti alle convenzioni attuali, con notazioni incoerenti (`D=12`, `Ø12`, `12mm`), abbreviazioni ambigue (`INT` quasi sempre significa `INTEGRALE`, ma occasionalmente `INTERO`) e termini di servizio accanto a quelli geometrici — è dove un parser a regole smette di scalare. Lì ho messo un LLM nella pipeline.

Una nota sul metodo. Il lavoro di ingegneria è stato realizzato con una disciplina Writer/Verifier. Una sessione di coding con GPT-5.6 Sol (via Codex di OpenAI) ha scritto il codice; una sessione separata di Claude Code (Claude Opus 4.7) ha revisionato ogni commit prima del merge. Le decisioni strategiche, l'interpretazione dei benchmark e questo testo sono nati in una sessione parallela con Claude Opus 4.7 come partner di brainstorming e stesura, con me come umano nel loop. La stessa affermazione procedurale che questa pagina fa sul codice di produzione — valutare prima di scegliere, usare l'AI come componente sotto misura — è valsa anche in fase di scrittura.

2. IL BASELINE END-TO-END

SEMPLICE DA GESTIRE, IMPOSSIBILE DA DIAGNOSTICARE

Il primo motore LLM di produzione era `madtools-peso`, un modello Ollama custom derivato da `gemma4:12b` con l'intero system prompt incorporato via `Modelfile`. Il client C# inviava un messaggio

utente per item, richiedeva `format=json` e `temperature=0`, e riceveva:

```
{
  "peso_stimato": 54.66,
  "ragionamento": "HM integrale, cilindro Ø10x60, K=0.80."
}
```

Una chiamata HTTP per item. Un prompt. Un solo punto in cui aggiornare la logica. Operativamente semplice.

Diagnosticamente debole. Quella singola risposta combinava quattro responsabilità: interpretare la descrizione, scegliere materiale e famiglia geometrica, scegliere il coefficiente e applicare la formula, formattare l'involucro JSON. *Quando un peso era sbagliato non potevo sapere quale delle quattro avesse fallito.* Quando molti pesi erano sbagliati non avevo un modo rigoroso di confrontare una modifica al prompt con la run precedente.

La scelta di Gemma-4-12B è stata fatta sul consenso della community: all'epoca era ampiamente reputato il miglior modello open-weight di piccola taglia per l'output JSON strutturato. L'ho scelto su quella reputazione — nessuna valutazione specifica sul compito, nessun confronto con alternative, nessuna misura sul mio dominio.

La run di produzione del 14 luglio

Dal report di produzione persistito:

CAMPO	RISULTATO
Item letti dalla board	139
Item inviati all'LLM	51
Timeout LLM definitivi	37
Pesi scritti	6
Solo motivazione	8
Output non validi	0
Fallimenti di scrittura	0
Durata	2h 13m 12s
Exit code	1

Il tasso di timeout definitivi era 37/51 = **72,5%**. Non è un'affermazione sull'accuratezza numerica. È un'affermazione sulla copertura dentro la finestra operativa: tre item su quattro non ricevevano risposta entro un timeout di 120 secondi con tre ritentativi.

Prima di quella run avevo già provato correzioni a livello di prompt — accorciare il budget di ragionamento, aggiungere istruzioni anti-loop esplicite, elencare le parole chiave delle lavorazioni conto terzi perché il modello non le confondesse con operazioni di produzione. Nulla ha funzionato. La correzione successiva sarebbe stata valutata allo stesso modo: rilasciare, aspettare una notte, contare i timeout. *Questo non è un ciclo diagnostico.* Se volevo sapere quale modello, o quale prompt, funzionasse davvero, dovevo poter far girare gli stessi input su candidati diversi e misurare i risultati su assi confrontabili.

3. L'HARNESS DI VALUTAZIONE

CINQUE ARTEFATTI, CINQUE DOMANDE DIVERSE

3.1 Il golden set

54 item, curati a mano, ciascuno etichettato con la decisione corretta e — dove applicabile — i valori intermedi corretti (materiale, famiglia, dimensioni, parametri di formula) e il peso derivato dalla formula.

- 38 item con `decisione = "valore"`: descrizioni che devono produrre un peso.
- 16 item con `decisione = "null"`: descrizioni che devono produrre un rifiuto, ciascuna etichettata con una categoria tratta da dodici motivi di rifiuto predefiniti.

La ricchezza a livello di item conta più del numero di item. Permette all'harness di dire non solo *se* un candidato ha sbagliato, ma *dove* nella pipeline l'errore è stato introdotto.

> **Il golden set è un artefatto di sviluppo.** L'ho usato per iterare il prompt del parser durante lo sviluppo, correggendo il prompt sui suoi stessi item. Le metriche calcolate su di esso verificano la coerenza interna dopo lo sviluppo del prompt; non sono una stima non distorta della generalizzazione.

3.2 L'oracolo aritmetico

Un modulo Python, `oracolo.py`, che calcola il peso dell'utensile a partire da parametri strutturati usando le formule della specifica di dominio.

L'oracolo e il `ToolCalculator` in C# sono implementati separatamente dalla stessa specifica. Il loro accordo è evidenza utile contro errori di implementazione indipendenti. *Non è una prova di verità fisica.* Due implementazioni costruite dalla stessa specifica possono condividere un errore comune nella specifica, nei coefficienti o nelle assunzioni di dominio. Per questo la pagina usa **concordanza con l'oracolo** per l'accuratezza rispetto alla formula e riserva **accuratezza fisica** al confronto con la bilancia.

L'oracolo è validato da una suite di 27 test pytest che codificano direttamente la specifica.

3.3 Il runner

Il runner parla con il modello. Punta a modelli ospitati su Ollama e supporta due modalità di payload:

- `embedded`: per ogni chiamata viene inviata solo la descrizione utente; il system prompt vive nel `Modelfile`, riproducendo il pattern di produzione originale di `madtools-peso`.
- `system-runtime`: il system prompt viene letto da disco all'avvio della run e iniettato come ruolo `system` a ogni chiamata, come va usato un modello base puro.

Ogni run registra sei identificatori: SHA-256 del system prompt, digest del modello, tag del modello, modalità di payload, timestamp UTC della run, SHA-256 del golden set. Due run che differiscono su uno qualsiasi di questi non sono direttamente confrontabili.

Sei identificatori sono necessari ma non sufficienti. Anche le opzioni di runtime come `num_ctx`, la temperatura, la politica di timeout e la gestione della thinking mode incidono sul contratto effettivamente consegnato. La sezione 4 mostra cosa succede quando una di queste resta al default di piattaforma senza essere tracciata.

3.4 I cinque gruppi di metriche

La scelta di cinque gruppi di metriche — e non uno — riflette il fatto che un modello può fallire in modi strutturalmente diversi. Comprimerli in un unico scalare nasconde proprio il segnale che serve vedere.

1. **Validità tecnica.** Parsabilità del JSON, conformità allo schema, intervallo numerico plausibile (da 0,1 g a 50 kg), ragionamento non vuoto. Un modello che fallisce qui è rotto a livello di output, prima ancora di poter porre una domanda semantica.
2. **Performance.** Latenza mediana, p95, p99. Numero di timeout. Numero di loop di ragionamento, definiti operativamente come risposte il cui `ragionamento` supera gli 800 token di output (circa dieci volte il budget di 30 parole indicato nel prompt).
3. **Concordanza con `ToolCalculator` (solo triage).** Dove entrambi i motori producono un peso numerico, quanto distano? Deliberatamente non decisionale: `ToolCalculator` non è ground truth. Scostamenti oltre il 20% segnalano item da ispezionare a mano.

4. **Decisione null/valore.** Una classificazione binaria: il modello ha rifiutato quando doveva rifiutare, e risposto quando doveva rispondere? TP, TN, FP, FN, precision, recall. Una sotto-metrica distingue "corretto per la ragione giusta" da "corretto per caso" tramite keyword matching sul testo del ragionamento.
5. **Concordanza con l'oracolo.** Per gli item in cui il modello ha prodotto un peso e l'oracolo ne prevedeva uno, quanto è vicino il numero del modello a quello dell'oracolo? MAE, MAPE e conteggi entro l'1%, entro il 5% e oltre il 20%. È la metrica decisionale primaria dell'harness end-to-end.

> **Una nota sui denominatori.** Ogni risultato sulle metriche 3, 4 e 5 va riportato con il numero di item su cui è calcolato. Un candidato che rifiuta la maggior parte degli item e risponde solo ai più facili può mostrare una media ottima sui pochi a cui risponde, coprendo però solo una frazione del carico. La sezione 5 mostra un caso concreto in cui ignorarlo fa sembrare vincente un candidato che va scartato.

Ogni metrica è salvata per item in un database SQLite (`eval_results.db`), indicizzata per `run_id` e collegata ai metadati versionati della run. Una run è un artefatto di prima classe: riproducibile per SHA, confrontabile tra modelli, interrogabile a posteriori.

4. TRONCAMENTO DEL CONTESTO

IL BUG CHE HA INVALIDATO OGNI MISURA

Prima di far girare l'harness sui sei modelli candidati, ho fatto uno smoke test manuale su un solo candidato con una sola descrizione — il rapido controllo di sanità che chiunque fa prima di dedicare una serata a un benchmark. Ho scelto `qwen3:14b`, gli ho mandato una descrizione canonica del golden set, e ho aspettato.

La risposta ha impiegato 148 secondi ed è tornata con un peso sbagliato di più di un ordine di grandezza.

Invece di leggere il JSON, ho guardato il log del server Ollama per quella chiamata. In mezzo alla gestione della richiesta c'era una riga che segnalava che il prompt di input era stato **troncato** per entrare in una finestra di contesto più piccola del prompt stesso. Il modello aveva ricevuto una frazione di quello che intendevo mandargli. Nessun errore sollevato; nessun avviso arrivato al client HTTP; richiesta completata; risposta JSON ben formata. *La pipeline non aveva alcun modo di sapere che stava lavorando su un prompt mutilato.*

La causa era un default che non avevo mai ispezionato. Il parametro `num_ctx` di Ollama — la finestra di contesto usata in inferenza — vale 4.096 token quando il client non specifica altrimenti. Quel valore copre sia input sia output; il runtime ne riserva una parte per la generazione. Il mio system prompt canonico — compilato da file skill modulari che contengono ogni densità dei materiali, ogni formula geometrica, ogni mappatura di keyword e ogni categoria di rifiuto — era di circa 6.900 token.

Ho corretto il client aggiungendo `num_ctx=8192` alle opzioni della richiesta. Nessuna configurazione globale di Ollama modificata; nessun `Modelfile` rigenerato. La run di benchmark successiva di `qwen3:14b` sul golden set ha prodotto questo confronto prima/dopo:

METRICA	NUM_CTX=4096	NUM_CTX=8192
Output numerici confrontabili	30	38
MAE	41,81 g	6,23 g
MAPE	18,06%	1,76%
Entro l'1% dall'oracolo	14	35
Oltre il 20% dall'oracolo	10	1
Timeout	2	0
Errori di output	7	0
Durata della run	2.243 s	1.067 s

Questo non dimostra che ogni errore precedente alla correzione fosse causato dal troncamento. Dimostra che la run di benchmark senza contesto adeguato *non era una misura valida del sistema prompt-più-modello che intendevo misurare*. Qualunque candidato avessi valutato prima di scoprirlo sarebbe stato misurato contro un prompt mutilato. Avrei concluso che tutti e sei i modelli erano scadenti, che Gemma-4-12B era leggermente meno scadente degli altri, e che la mossa sensata era restare col modello che avevo. Ogni confronto non avrebbe detto nulla se non sul troncamento, che era una proprietà dell'infrastruttura e non dei candidati.

Ne seguono due lezioni operative.

I default di piattaforma che rimodellano silenziosamente l'input sono il tipo di bug più pericoloso

Producono output plausibili, non sollevano errori e invalidano ogni metrica a valle senza mai essere visibili nella metrica stessa.

Registrare sei identificatori per run è necessario ma non sufficiente

`prompt_sha` identifica il file di prompt che il client intende inviare. Non può rivelare cosa il runtime consegna davvero dopo troncamento, quantizzazione o riscrittura del chat template. In questo caso la scoperta è arrivata da un'ispezione manuale del log, non dai metadati di fingerprint. L'harness attuale registra anche `num_ctx` e le opzioni correlate per ogni run.

5. IL BENCHMARK CROSS-MODELLO

SEI CONFIGURAZIONI, UN CONTRATTO IDENTICO

Con `num_ctx=8192` impostato sul client, ho finalmente fatto girare l'harness sui sei modelli candidati con input identici e misura identica. La lista dei candidati è stata scelta tra ciò che era praticamente disponibile nel registry Ollama locale all'epoca. Cinque candidati sono stati usati in modalità `system-runtime`; il sesto era il sistema in carica `madtools-peso`, eseguito in modalità `embedded` come lo usava la produzione.

La tabella riporta la metrica decisionale primaria (concordanza con l'oracolo) calcolata sui 38 item del golden set con peso numerico atteso.

MODELLO	MODALITÀ PAYLOAD	CONFRONTABILI	MAE (G)	MAPE (%)	ENTRO L'1%
`qwen3:14b`	system-runtime	38/38	6,23	1,76	35/38 (92,11%)
`qwen2.5:14b`	system-runtime	35/38	101,09	45,54	4/35 (11,43%)
`gemma4:e4b`	system-runtime	38/38	97,25	47,38	5/38 (13,16%)
`gemma4:12b`	system-runtime	25/38	129,63	42,28	2/25 (8,00%)
`qwen3.5:9B` †	system-runtime	4/38	0,04	0,01	4/4 (100,00%)
`madtools-peso`	embedded	21/38	78,78	44,98	2/21 (9,52%)

† `qwen3.5:9B` è un artefatto locale nel registry Ollama del server di deploy, non un rilascio ufficiale confermato. I suoi risultati non sono un confronto di qualità valido; sono inclusi solo per completezza.

`qwen3:14b` in modalità `system-runtime` con `num_ctx=8192` è stata l'unica run a combinare copertura numerica completa (38 output confrontabili su 38), errore basso rispetto alla formula e nessun timeout o errore di output. *Tutti gli altri candidati hanno fallito su una porzione consistente del set* tramite output null, timeout o pesi discordanti dall'oracolo.

La trappola del denominatore di qwen3.5:9B

`qwen3.5:9B` compare in tabella con il MAE più basso (0,04 g) e il 100% entro l'1%. Quei numeri sono calcolati su **quattro item**. Trentanove dei cinquantaquattro item del golden set sono finiti in stato di errore — il runner ha segnalato loop di ragionamento o errori di output — prima che fosse possibile un confronto valido sul peso. Causa radice: il modello usa un wrapper di reasoning mode `<think>…</think>` che il runner non sapeva rimuovere. I quattro item sopravvissuti sono quelli su cui il modello non è entrato in quella modalità.

Non è un buon modello che va bene su un compito. È un'incompatibilità client/modello che lascia per caso una piccola popolazione residua su cui il modello va bene. E quella popolazione non è un campione casuale del golden set: *è esattamente l'insieme degli item che non hanno innescato il fallimento.*

La forma generale della lezione: **le metriche condizionate alle risposte del modello favoriranno sempre un candidato che risponde di rado.** La copertura — il denominatore — fa parte del numero.

Prompt embedded contro prompt a runtime

Il benchmark includeva anche un confronto a parità di modello base. `madtools-peso` e `gemma4:12b` hanno gli stessi pesi base: l'unica differenza è la consegna del prompt. La soglia PASS pre-registrata `|ΔMAE| ≤ 3,0 g` non è stata rispettata. `madtools-peso` (embedded) ha registrato MAE 78,78 g; `gemma4:12b` (system-runtime) MAE 129,63 g. Lo scarto è stato di 50,85 g, più di sedici volte la soglia. Entrambe le configurazioni erano scadenti sulla metrica primaria, ma quella embedded lo era sensibilmente meno.

Non ho una spiegazione definitiva. La consegna del prompt può differire in modi che dal lato client non posso ispezionare — il chat template nativo applicato al messaggio `system` a runtime, lo stato di tokenizzazione, il comportamento di warm-up. L'implicazione pratica: *migrare da un Modelfile custom a un modello base puro con prompt a runtime non è un semplice cambio di trasporto.* Va rivalutato con l'harness da un capo all'altro.

Decisione di selezione

```
qwen3:14b
payload mode: system-runtime
num_ctx: 8192
```

Era una selezione tra le configurazioni locali testate sotto il contratto misurato. *Non* era evidenza che `qwen3:14b` sia in generale il miglior modello per l'output strutturato.

6. LA DEBOLEZZA RIMASTA

RAGIONAMENTO CORRETTO, NUMERO SBAGLIATO

`qwen3:14b` ha prodotto 35 pesi entro l'1% dall'oracolo. Ne ha prodotti anche tre che non lo erano.

ITEM	ATTESO	MODELLO	ERRORE % ASSOLUTO
`GS-015`	28,26 g	29,23 g	3,43%
`GS-022`	230,50 g	212,71 g	7,72%
`GS-049`	405,74 g	193,03 g	52,43%

- **GS-015** era un errore di coefficiente/interpretazione: il modello ha applicato un valore di K incoerente con la geometria che aveva identificato.
- **GS-022** aveva la famiglia geometrica generale corretta ma ha prodotto un output numerico incoerente con la formula che ne sarebbe derivata.
- **GS-049** era il caso interessante. Il modello ha scritto correttamente la formula della sega circolare (`π/4 × (D_est² − D_int²) × spessore × densità`), ha scritto correttamente ogni dimensione, e poi ha emesso un numero finale che da quella formula non discendeva. Circa metà del peso atteso, su una descrizione in cui ogni altro passaggio scritto era in ordine.

Tre item non sono un corpus. Ma su `GS-049` in particolare avevo un'ipotesi. Il rapporto tra il peso corretto e l'output del modello era vicino a un fattore due, e uno sguardo successivo a casi simili su utensili cilindrici con gambo suggeriva che alcuni errori fossero vicini a un fattore π. Ho formulato un'ipotesi diagnostica di lavoro: forse il modello scriveva la formula simbolicamente con un fattore π e poi lasciava cadere quel fattore nel risultato numerico. Una firma da "fattore π mancante".

Il pattern che non era un pattern

Dei 108 item del replay (vedi sezione 8.2), nove hanno prodotto un peso della nuova pipeline che differiva dal valore memorizzato dal vecchio sistema end-to-end di più del 20%. Ho calcolato il rapporto tra i due su tutti e nove.

I rapporti erano eterogenei. Media 2,61, deviazione standard 2,09, intervallo da 0,19 a 7,89. Un solo codice prodotto distinto (`RG_AL0142A1`, presente due volte) aveva un rapporto entro l'1% da π. Un secondo codice rientrava entro il 10% da π. *Né π da solo né π×K descrivevano l'insieme.*

L'ipotesi del "fattore π mancante" era una proiezione da un caso memorabile su un fenomeno aritmeticamente più disordinato. La lettura più stretta e difendibile: il vecchio sistema end-to-end faceva **errori moltiplicativi eterogenei senza una struttura di fattore condivisa**, su item il cui ragionamento simbolico aveva scritto correttamente. Il fallimento specifico non aveva una firma diagnostica pulita. La classe di fallimento sì.

La classe di fallimento rimasta

Messa da parte l'ipotesi diagnostica, un'osservazione ha retto su tutti i casi in cui un modello ha prodotto un peso sbagliato scrivendo un ragionamento corretto:

> Il modello sa produrre un ragionamento simbolico fluente — nominare materiale e densità, identificare la famiglia geometrica, scrivere la formula con π al posto giusto, scrivere correttamente le dimensioni — e poi, **nel campo numerico finale, restituire un valore che non discende dall'impianto simbolico che ha appena esposto.**

Non ho una spiegazione meccanicistica definitiva. La forma del fallimento è coerente con una proprietà documentata dei modelli linguistici basati su transformer: sono generatori di token. Quando un modello scrive un passaggio simbolico, emette il token `π` perché il completamento della frase lo richiede. Quando emette un risultato numerico, produce una sequenza di cifre associata al contesto recente. Se il calcolo è multi-fattore — densità per raggio al quadrato per lunghezza per coefficiente per π — il numero emesso può sbagliare di uno qualsiasi di quei fattori senza che alcuna incoerenza emerga nel testo visibile. *Il livello simbolico e quello numerico non sono sempre sincronizzati.*

Non ho testato modelli di frontiera su questo compito, e l'evidenza che ho non prova che fallirebbero. Ma il fallimento non è primariamente un *non conoscere il dominio*: il ragionamento mostrava che il modello conosceva la formula. È il fatto che la pipeline non è in grado di accorgersi quando l'output numerico non discende dal ragionamento simbolico che lo circonda. Non è un problema di scala risolvibile passando a un modello più grande. **È un problema di confine** — si risolve mettendo codice deterministico a governare il passaggio numerico, e dando alla pipeline una struttura osservabile in cui la decisione semantica dell'LLM e il calcolo deterministico siano ispezionabili separatamente.

7. L'ARCHITETTURA IBRIDA

L'LLM INTERPRETA. IL CODICE CALCOLA.

Il cambio architetturale è abbastanza diretto da stare in una frase: usare l'LLM per il livello in cui eccelle — leggere una descrizione libera disordinata e produrne un'interpretazione strutturata e pulita — e affidare l'aritmetica a un componente che calcola in modo deterministico.

7.1 Parser semantico

`qwen3:14b` riceve il prompt del parser tramite un messaggio `system` a runtime. **Non restituisce alcun peso.** Il suo output è un record strutturato di parametri semantici estratti:

```
{
  "decisione": "valore",
  "motivo_null": null,
  "materiale_primario": "HM",
  "material_type": "HMIntegrale",
  "tipo_utensile": "cilindrico_pieno",
  "dimensioni": {
    "D_tot": 10, "L_tot": 60, "Z": 2,
    "D_gambo": null, "L_gambo": null,
    "lato1": null, "lato2": null,
    "D_esterno": null, "D_interno": null, "spessore": null
  },
  "riduzione_gambo": false,
  "ragionamento_parser": "Punta HM integrale Ø10x60 Z=2."
}
```

Oppure, quando la descrizione non consente un calcolo, `decisione = "null"` con un `motivo_null` tratto dall'enum controllato di dodici valori.

7.2 Calcolatore deterministico

L'applicazione C# passa i parametri accettati al `MadToolsCalcBatch.exe` deployato. Il calcolatore applica le formule versionate e valida le dimensioni obbligatorie e gli invarianti geometrici. Restituisce un peso in grammi oppure un errore di calcolo esplicito. Per il caso di test canonico — cilindro pieno HM integrale, Ø10 × 60, Z=2 — il calcolatore restituisce **54,66 g**.

7.3 Persistenza e audit

Il client C# scrive su Monday:

- peso e giustificazione leggibile quando il calcolo riesce;
- un null esplicito e un messaggio diagnostico quando il parser rifiuta (con un `motivo_null` dall'enum di dodici valori), quando il calcolatore respinge i parametri, o quando la chiamata all'LLM va in timeout.

Il null semantico del parser porta con sé un codice motivo leggibile a macchina. Il rifiuto del calcolatore e il timeout dell'LLM portano messaggi diagnostici in linguaggio naturale: leggibili operativamente, ma non parte di un enum fisso. Ogni chiamata produce una traccia di audit strutturata.

7.4 Cosa compra questa struttura

L'architettura non garantisce un parsing corretto. Un record semantico plausibile ma sbagliato può comunque produrre a valle un peso plausibile e sbagliato. È un rischio reale e l'harness deve continuare a misurarlo.

Quello che l'architettura compra, e che il precedente disegno end-to-end non aveva, è la **separazione osservabile**. Quando un peso è sbagliato, si può dire se il parser ha assegnato la geometria sbagliata, se i parametri erano corretti e il calcolatore ha applicato la formula come da specifica, o se il calcolatore ha fallito su un input per il resto valido. *Quella diagnosi per livello è ciò che il regime end-to-end nascondeva dentro un unico JSON opaco.*

Un'alternativa considerata e scartata

Prima di implementare l'ibrido ho considerato il **function calling** — dare all'LLM uno strumento `calculate_weight` e lasciare che lo invocasse quando riteneva di aver estratto abbastanza per calcolare. Sulla carta somiglia all'ibrido: l'LLM interpreta, lo strumento calcola.

L'ho scartato per due ragioni. Primo, il function calling raddoppia grossomodo i round-trip verso l'LLM per item — più latenza e più superficie di fallimento, su un compito in cui entrambe erano già sotto pressione.

Secondo, e più fondamentale, *il function calling non eliminerebbe il fallimento documentato nella sezione 6. Lo sposterebbe un passo a monte.* Il modello potrebbe comunque allucinare un diametro, un coefficiente K, un'assegnazione di materiale, e passare il parametro allucinato allo strumento. Lo strumento calcolerebbe un peso sbagliato da parametri sbagliati e lo restituirebbe. Tornerei alla stessa classe di errore, ma più difficile da attribuire a posteriori. Il determinismo dello strumento non rende corretti i parametri che gli arrivano.

L'architettura ibrida evita questo perché i parametri vivono in un **record JSON ispezionabile** prima che il livello deterministico li consumi. Possono essere confrontati con una ground truth di riferimento per il solo passaggio di parsing. È stata la proprietà di progetto che contava di più.

8. EVIDENZE DOPO LA DECOMPOSIZIONE

TRE PASSAGGI, A PESO EPISTEMICO CRESCENTE

L'architettura ibrida è stata implementata nella settimana successiva al benchmark, e da allora è stata valutata in tre passaggi. Ognuno misura una proprietà diversa, e nessuno da solo è una prova di accuratezza.

8.1 Controllo di sviluppo in-sample

Sul golden set di 54 item — lo stesso set usato per iterare quattro volte il prompt del parser durante lo sviluppo.

METRICA	RISULTATO
Item con peso atteso entro l'1% dall'oracolo	38/38
Esatti alla precisione memorizzata	38/38
Precision sulla decisione	0,974
Recall sulla decisione	1,000
MAE deterministico rispetto all'oracolo	0,00 g

Sono numeri forti. Vanno letti per quello che sono: *un controllo che le modifiche al prompt del parser atterrino dove volevano atterrare.* Non sono una validazione su input mai visti, perché il prompt è stato iterato proprio su questi item. In termini di holdout, questo è in-sample.

8.2 Replay fuori campione su 108 item di produzione

Un replay di 108 item presi dal traffico di produzione reale — item che il modello non aveva mai visto in sviluppo. Il replay non scrive su Monday: esegue la pipeline ibrida offline e confronta il suo output con il valore che il vecchio sistema end-to-end aveva scritto in precedenza.

ESITO	ITEM
Peso vecchio e nuovo entrambi numerici	92
Vecchio e nuovo entrambi null	11
Peso vecchio presente, nuovo null	3
Rifiuto del calcolatore	1
Timeout LLM	1
Totale	**108**

All'interno delle 92 righe numericamente confrontabili: **81 concordavano entro l'1%**; 11 divergevano di più dell'1%; 9 di quelle 11 divergevano di più del 20%.

Le nove divergenze grandi sono quelle che ho ispezionato. Nella maggior parte il vecchio sistema end-to-end aveva scritto un peso numerico incompatibile con la formula che lo stesso modello aveva registrato nel proprio campo di ragionamento. Il risultato della pipeline ibrida, deterministico una volta

estratti i parametri, ha prodotto un peso coerente con la formula su ognuno di quegli item — *correzioni aritmetiche del sistema precedente.* Su quattro item con divergenza meno chiara, un esperto di dominio ha rivisto il caso: tre erano rifiuti legittimi della nuova pipeline, e uno era una **regressione del parser** su una descrizione di sega circolare, il cui formato canonico a tre numeri (`Ø_esterno × spessore × Ø_interno`) il parser ha scambiato per una terna di dimensioni cilindriche.

Il replay sostiene quindi due affermazioni:

1. Sulle 92 righe confrontabili, il livello deterministico ha eliminato la classe di discrepanze aritmetiche documentata nella sezione 6 ogni volta che i parametri semantici erano corretti.
2. Le regressioni di parsing restano possibili e richiedono una revisione a parte; nessuna metrica di sintesi da sola avrebbe riconosciuto il caso della sega circolare come regressione anziché come correzione.

Il replay non stabilisce l'accuratezza fisica. Le letture di bilancia non erano disponibili per tutti i 108 item; la classificazione dell'esperto ha lavorato su conoscenza di geometria e materiali, non su misure.

8.3 Prima run di produzione non presidiata — 24 luglio 2026

FASE	RISULTATO
Item letti dalla board	152
Già pesati	55
Codice prodotto non valido o mancante	28
Item eleggibili	69
Risposte LLM valide	68
Null semantici	19
Decisioni "valore" del parser	49
Rifiuti del calcolatore	3
Pesi scritti	46
Timeout LLM definitivi	1
Output non validi	0
Fallimenti di scrittura su Monday	0
Durata	31m 11s
Exit code	0

Copertura numerica: 46 item su 69 eleggibili hanno ricevuto un peso (66,7%). Gli altri 23 hanno ricevuto un esito esplicito — 19 null semantici del parser (ciascuno con un codice motivo da enum), 3 rifiuti del calcolatore, 1 timeout LLM.

A quel 66,7% si attaccano due cautele. Primo, i 69 item erano il *backlog residuo* della board — item precedentemente esclusi dall'elaborazione — non un campione rappresentativo del volume giornaliero ordinario, che si aggira sui 5–10 item. Il backlog concentra le descrizioni più difficili da risolvere, quindi il dato va letto come copertura sulla coda difficile della produzione. Secondo, questa run non aveva ground truth fisica indipendente: i 46 pesi calcolati sono geometricamente plausibili (intervallo osservato 25,00–2408,41 g, nessuno zero, nessun outlier), ma non esiste una lettura di bilancia per valutarli singolarmente. **La notte del 24 luglio è evidenza di stabilità tecnica, non di accuratezza numerica.**

La run ha anche fatto emergere due pattern che vale la pena nominare, perché sono il tipo di scoperta che una valutazione sintetica non avrebbe prodotto.

Descrizioni di lame non normalizzate alla famiglia sega circolare

Tre dei 19 null semantici (`FR1000B0`, `VA0563B0`, `VA0562B0`) erano lame da taglio descritte con `lama` o `lama da taglio`. Il parser non ha mappato in modo coerente questi termini alla famiglia `sega_circolare`. Quello che nel replay era un caso singolo è ora un piccolo pattern con tre occorrenze in una sola run.

Barre rettangolari che arrivano al calcolatore senza un L_tot valido

I tre rifiuti del calcolatore (`BA1384A1`, `BA1388A1`, `BA0068A1-1`) erano utensili in barra rettangolare nel formato `H×larghezza×lunghezza`. Il formato quadrato equivalente viene calcolato correttamente. Il report notturno non conserva il JSON grezzo del parser, quindi la causa non è stata localizzata in modo conclusivo tra output del parser e interfaccia del calcolatore.

Sulla categoria degli errori

Nessuno dei tre passaggi stabilisce l'accuratezza numerica fisica. Quello che stabiliscono, presi insieme, è che l'architettura ibrida cambia la **categoria** di fallimento disponibile al sistema.

Con il disegno end-to-end, quando l'LLM produceva un peso sbagliato, scriveva un numero plausibile sulla board. Gli operatori a valle non avevano alcun segnale che il numero fosse sbagliato finché l'utensile non arrivava alla bilancia. Con il disegno ibrido, quando l'LLM non riesce a produrre parametri coerenti, o il calcolatore respinge parametri non validi, o la chiamata va in timeout, la pipeline scrive un esito esplicito con una diagnostica. Le tre barre rettangolari non hanno prodotto silenziosamente pesi plausibili: hanno prodotto rifiuti espliciti che indicavano `L_tot` come mancante.

Se questa asimmetria — numeri sbagliati silenziosi da un lato, rifiuti espliciti e rumorosi dall'altro — valga il suo costo dipende da cosa succede a valle. In questo progetto ogni utensile viene pesato sulla bilancia prima dell'inscatolamento, per necessità operativa dell'officina. Quella rete di sicurezza assorbe il costo degli errori silenziosi a posteriori, ma solo dopo che si sono già propagati nella pianificazione e nella preparazione. Un rifiuto rumoroso intercetta lo stesso item prima, in un punto in cui l'intervento di un operatore è già previsto. *È un compromesso specifico del contesto.*

9. LIMITI

COSA QUESTA EVIDENZA NON STABILISCE

9.1 Riferimento derivato dalla formula

L'oracolo aritmetico valida la conformità alle formule e ai coefficienti mantenuti. Non sostituisce le misure fisiche con la bilancia. Un MAE di 0,00 g sul golden set significa che la pipeline concorda con il riferimento derivato dalla formula — non che i pesi corrispondano alla massa fisica degli utensili.

9.2 Riuso del golden set

Il prompt del parser è stato iterato quattro volte sullo stesso golden set di 54 item che poi riporta il baseline finale di sviluppo. Il suo 38/38 non va descritto come accuratezza su holdout.

9.3 Singole run di benchmark

La tabella cross-modello riporta una run per configurazione. L'inferenza locale a `temperature=0` non è del tutto deterministica. La pagina non riporta varianza o stabilità tra run; un benchmark replicato (tre-cinque run per configurazione) renderebbe il confronto più stretto.

9.4 Metriche numeriche condizionate

MAE, MAPE e i conteggi entro l'1% escludono gli item in cui il modello ha prodotto un null, un timeout o un errore di output. Su configurazioni con alti tassi di null o errori queste metriche possono sembrare migliori di quanto il quadro completo giustifichi.

9.5 Provenienza di runtime incompleta

I sei identificatori registrati per run sono necessari ma non sufficienti a riprodurre una run. Contano anche opzioni di runtime e comportamento del server: `num_ctx`, temperatura, thinking mode, impostazioni di timeout e retry, versione del client, versione di Ollama. L'harness attuale li registra; le run precedenti possono non averli.

9.6 Ambito ristretto di modelli

Il benchmark copre modelli disponibili localmente nell'intervallo 4B–14B, più un'incompatibilità client/modello. Non copre modelli di frontiera ospitati e non può sostenere conclusioni su tutti gli LLM. L'argomento per cui il fallimento aritmetico non è un problema di scala è strutturale, non empirico.

10. CONCLUSIONI

DUE CONCLUSIONI, E I LORO LIMITI

Primo: la scelta del modello per questo compito di produzione richiedeva una valutazione sotto il contratto di runtime reale. La reputazione generalista non ha predetto il risultato — la scelta più reputata (Gemma-4-12B embedded via `Modelfile`) è risultata tra le peggiori nel benchmark cross-modello. E una finestra di contesto insufficiente ha invalidato il primo tentativo di confronto prima ancora che si potesse valutare la qualità dei modelli. La valutazione doveva misurare non solo il modello, ma l'input consegnato, le opzioni di runtime e il contratto di output persistito.

Se questo generalizzi a *tutti* i compiti LLM in produzione è una domanda per le loro valutazioni. Ciò che generalizza è l'affermazione procedurale: *costruisci l'harness prima di far girare il benchmark, e misura quello che il modello vede davvero.*

Secondo: l'aritmetica generata era una fonte non necessaria di opacità in questa pipeline. Una volta convertita la descrizione in parametri strutturati, un calcolatore deterministico ha applicato le stesse formule in modo riproducibile e ha respinto esplicitamente le geometrie non valide. Separare parsing e calcolo non ha, di per sé, reso il sistema numericamente più accurato sulla coda difficile — le regressioni del parser sono reali e restano irrisolte. Quello che ha fatto è cambiare la forma del fallimento in qualcosa di *diagnosticabile*: la decisione semantica e l'aritmetica sono diventate due artefatti ispezionabili in modo indipendente.

Se la stessa decomposizione migliori altre applicazioni dipende da quanto il passaggio aritmetico sia definito abbastanza bene da poter essere affidato a codice deterministico. In questo progetto lo era.

Qualche lezione minore

- **Il denominatore fa parte del numero.** Una percentuale "entro l'1% dall'oracolo" che non riporta il numero di item su cui è calcolata può ingannare in entrambe le direzioni. Ogni metrica condizionata — MAE, MAPE, precision sui rifiuti — ha un denominatore che fa parte del numero.
- **I golden set usati per iterare i prompt sono artefatti di sviluppo, non set di validazione.** Le metriche che ne derivano sono oneste come controllo di sviluppo e, se riportate senza quella qualificazione, disoneste sulla generalizzazione.
- **Le interpretazioni di un segnale spesso cambiano quando vengono verificate sistematicamente.** La lettura del "fattore π mancante" è sopravvissuta a un caso memorabile; non è sopravvissuta a un controllo sistematico dei rapporti su nove casi. Chiamare ipotesi un'interpretazione finché non è verificata su un campione è una piccola disciplina che ripaga dove conta.

Lavoro futuro

- **Il dataset di validazione alimentato dalla bilancia** è la voce singola a leva più alta. Ogni utensile spedito a un cliente viene già pesato prima dell'inscatolamento. Convogliare quella lettura in un dataset unito a descrizione, parametri estratti e risultato deterministico converte una necessità operativa in un set di valutazione continuamente aggiornato a costo marginale.
- **Metriche che separino parsing e calcolo.** I cinque gruppi di metriche erano pensati per il regime end-to-end. L'architettura ibrida crea una struttura a due assi che richiede un set di metriche complementare, capace di valutarli in modo indipendente.
- **Le due regressioni del parser dalla run del 24 luglio** — il buco di normalizzazione sulla sega circolare e il bug di interfaccia su `L_tot` per le barre rettangolari — restano da localizzare con precisione e correggere.
- **Logging retrospettivo.** Il report notturno non conserva il JSON grezzo del parser, e un incidente precedente ha sovrascritto il proprio report durante una riesecuzione. Il logging di produzione per un sistema con ML nel loop va progettato pensando alla valutazione retrospettiva.

- **Robustezza del deploy.** Il deploy attuale dipende da un token di sessione utente che rende la pipeline più fragile di quanto un'integrazione interna dovrebbe essere.

La catena di evidenze è partita da un fallimento di produzione verificato, diagnosticabile solo tramite misura, ed è arrivata a un'architettura che tiene la misura nel loop per costruzione. *Alla fine, è di questo che il progetto ha parlato.*

RIPRODUCIBILITÀ

GLI ARTEFATTI DIETRO I NUMERI

I principali artefatti interni usati per i valori riportati:

```
eval/tests/golden_set_v0_1.json
eval/reports/baseline_2026-07-15.md
eval/reports/note_context_truncation.md
eval/reports/report_qwen3_14b_20260715_083409_750415Z.json
eval/reports/report_qwen3_14b_20260715_093521_334357Z.json
eval/reports/report_qwen3_14b_20260720_161125_952953Z.json
eval/reports/replay_ibrido_108_item_2026-07-20.json
eval/reports/analisi_divergenze_replay_108_2026-07-23.md
eval/reports/analisi_pipeline_notturna_2026-07-24.md
logs/madtools_peso_report_2026-07-14.json
logs/madtools_peso_report_2026-07-24.json
```

CONTATTI

STESSA DOMANDA SULLA VOSTRA PIPELINE?

Una chiamata di 30 minuti per capire cosa andrebbe davvero misurato. Senza demo precostituite.

Contatti e altri casi su indaloop.ai.

indaloop

AI × PROCESSI × AZIENDE

AI che legge, regole che controllano, persone che decidono.

Altri casi e strumenti gratuiti su indaloop.ai