TOOL EXPOSURE IS NOT FREE: ADMISSION AND EXECUTION OF TOOL CALLS IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-augmented large language model (LLM) agents are evaluated as though offering a tool were free, and work on tool overuse refines that price only in tokens. However, on a mixed workload of reasoning and exact-computation requests, merely making a Python tool available costs a large share of the accuracy on requests that never needed one, and nearly every tool-exposing method we measure pays some version of that price. Exposure is therefore a two-sided decision, not a resource to ration. Two facts follow. Benefiting from a tool implies needing one but not conversely, so even a perfect necessity predictor over-exposes, and the headroom of any admission policy equals the rate at which exposure breaks requests, which makes the value of admission measurable per workload. In this paper, we propose AGENTRT, a runtime control plane that decides exposure from serving state before the model generates a token, so the decision costs no forward pass, and that constructs and executes the payload itself for request classes admitting a typed plan. Against ten baselines, AGENTRT is the only policy that keeps accuracy on the requests that need no tool intact while solving every request that needs one. It leads every baseline on both accuracy and energy per correct answer, by up to 28 accuracy points and 69% less energy, and by 11 points against the strongest, and the lead holds on paraphrased requests and across 7B-class models. On newer models whose tool policy breaks almost nothing, admission has no headroom, as the analysis predicts, and AGENTRT remains the cheapest policy per correct answer on every one of them.

1 INTRODUCTION

Tool-augmented large language model (LLM) agents call external tools such as Python interpreters, table engines and search APIs to solve tasks that text generation alone cannot (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024; Patil et al., 2024). In a serving system, each call is a state transition: the model decodes a payload, an executor runs it, and the observation is folded into the next prompt (Kwon et al., 2023). Agent benchmarks expose a tool schema to the model, let it decide whether to call, and score the final answer (Yao et al., 2025; Lu et al., 2025; Chen et al., 2024c). We call making the schema visible for a request tool exposure, and the decision whether to do so admission. The implicit accounting is that a tool can only help and that its only price is tokens and latency, and work on tool overuse keeps that premise, predicting necessity and suppressing the rest (Qian et al., 2025b; Wang et al., 2025a; Sun et al., 2026).

However, the premise fails in a way that changes the problem. On a mixed workload of Math500 problems and exact-computation requests served by a 7B model, merely offering a Python tool costs a large share of the accuracy on the requests that never needed one, and nearly every tool-exposing baseline we measure pays some version of that price (Yao et al., 2023; Chen et al., 2024b; Zheng et al., 2024; Jiang et al., 2023b; Su et al., 2024; Zhang et al., 2026). Withholding tools is not free either, since the exact-computation half then collapses. Exposure is therefore a two-sided decision that rescues some requests and breaks others, and not an allocation problem, since exposure is cheaper than withholding on most requests. The decision has to be made per request, in the runtime, and before any token is generated.

Two facts organize what follows. Benefiting from a tool implies needing one but not conversely, so a necessity predictor over-exposes however accurate it is (Qian et al., 2025b; Wang et al., 2025a; Sun

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

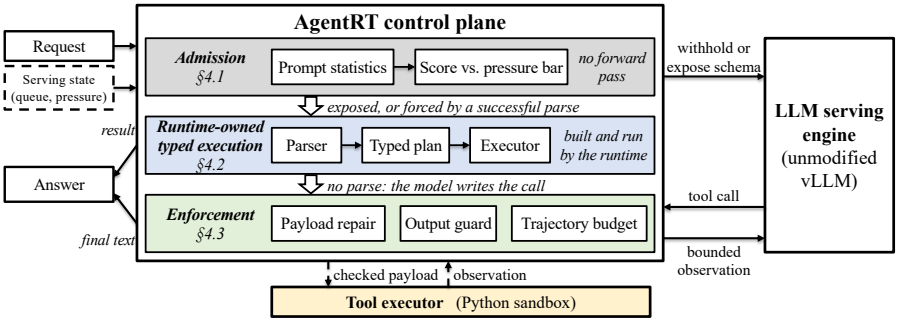


Figure 1: Overview of AGENTRT, a control plane between an unmodified serving engine and the tool executor. Admission (§4.1) decides exposure before generation, the runtime-owned route (§4.2) executes typed requests and answers them, and enforcement (§4.3) bounds the exposed loop.

et al., 2026). The headroom of even a perfect admission oracle is the share of requests that exposure breaks, since withholding the tool can save only those. The first fact says what admission should target, and the second says what admission is worth on a given workload.

Despite the attention the necessity question has received, two challenges remain. *The first challenge is to decide exposure before generation, without a second forward pass, from information the model cannot see.* Every existing way of deciding whether to call consults the model, whether by decoding first and reading token probabilities (Jiang et al., 2023b; Su et al., 2024), asking the serving model for a verdict (Jeong et al., 2024), reading its hidden states (Sun et al., 2026), or training the decision into the weights (Qian et al., 2025b; Wang et al., 2025a; Qian et al., 2025a). Each pays for the decision with a forward pass, which is the resource the decision protects, and none observes the serving state that determines what a call costs when the request arrives. The decision therefore has to move out of the model and into the runtime, where prompt statistics and serving state are readable before any token is generated. *The second challenge is to handle the requests that exposure breaks without asking the model to write the code that produces the malformed calls.* Existing methods assume that the model authors the call and that the system's job is to repair, prune, constrain or execute what it wrote (Chen et al., 2024b; Zhang et al., 2026; Zheng et al., 2024; Gao et al., 2023; Chen et al., 2023; Wang et al., 2024a; Cheng et al., 2023; Wang et al., 2024b). For the requests that exposure breaks, that authorship is the failure. The model writes a call that is malformed, prints nothing or never terminates, and admission can only withhold the tool from them. Handling them needs a route on which the runtime, not the model, constructs the call.

In this paper, we propose AGENTRT, a runtime control plane that interposes between the model API and the tool executor and modifies neither. The core idea is to make tool exposure a runtime decision, and to construct and execute the call in the runtime as well for request classes whose plan is recoverable from the request. As shown in Figure 1, an admission gate scores a request from prompt statistics and compares it with a bar that rises with serving pressure, so the decision costs no generation. For request classes that admit a typed plan, the runtime parses the request into the plan, executes it, and answers, so the payload cannot be malformed. Enforcement bounds what admission cannot predict, namely latency, context growth and failure risk.

We evaluate AGENTRT against ten baselines on a mixed workload served by Qwen2.5-7B through vLLM (Kwon et al., 2023), on paraphrased forms of that workload, on two external workloads, and on five further served models. The main contributions are summarized as follows.

- **Tool exposure as a two-sided decision.** Offering a tool costs accuracy on requests that never needed one, so exposure is a decision under two-sided error and not a resource to ration. Benefit is a strict subset of necessity, and the headroom of any admission policy equals the rate at which the tool policy breaks requests, measured here from a tenth of the workload down to nothing across six served models.

- **Admission before generation.** A gate that scores a request from prompt statistics against a bar set by serving pressure, so exposure is decided in the runtime without a forward pass.

- **Runtime-owned typed execution.** For request classes whose plan is recoverable from the request, the runtime parses, executes and answers, so the payload is well formed by construction, and enforcement bounds the exposed loop.

- **A control plane and its evaluation.** AGENTRT runs against an unmodified serving engine. Against ten baselines it alone keeps the tool-negative half intact while solving the tool-positive half, leads by up to 28 accuracy points and 69% less energy and by 11 points against the strongest baseline, keeps the energy lead on all six served models, and reports its negative results, including a gate with no discriminative power on the When2Tool benchmark (Sun et al., 2026).

2 RELATED WORK

2.1 DECIDING WHETHER TO CALL A TOOL

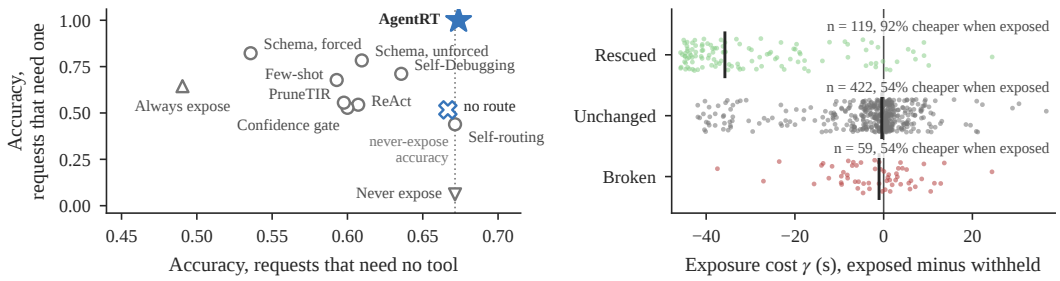
Tool-augmented agents call interpreters, retrievers and APIs from within a generation (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024; Patil et al., 2024), and when to call has been framed as predicting necessity. SMART (Qian et al., 2025b), OTC-PO (Wang et al., 2025a) and ToolRL (Qian et al., 2025a) train the policy into the weights with rewards that penalize unnecessary calls. FLARE (Jiang et al., 2023b) and DRAGIN (Su et al., 2024) trigger retrieval when token probabilities drop, at the price of a tentative generation. Adaptive-RAG (Jeong et al., 2024) routes with a small classifier, and Probe&Prefill (Sun et al., 2026) reads necessity linearly off the model's pre-generation hidden states. Model routers and cascades make a sibling decision, choosing which model answers a query and not whether a tool is exposed (Chen et al., 2024a; Ong et al., 2025; Ding et al., 2024; Aggarwal et al., 2024), and selective classification supplies the reject-option view that §3 borrows (Chow, 1970; Geifman & El-Yaniv, 2017). However, all of the tool-side methods estimate a property of the request and the model, and none of them observes the serving state that determines what a call costs when the request arrives.

2.2 WHO CONSTRUCTS THE CALL

A long line of work delegates computation to model-written code. PAL (Gao et al., 2023) and Program-of-Thoughts (Chen et al., 2023) have the model emit a program and read off its output, CodeAct (Wang et al., 2024a) makes executable code the action space, Binder (Cheng et al., 2023) emits a program in a symbolic language with model-call primitives, and Chain-of-Table (Wang et al., 2024b) emits a chain of table operations that the framework executes. Agent frameworks execute what the model wrote (Wu et al., 2024; Chase, 2022; Qiao et al., 2024; Wang et al., 2025b), and guardrails check it after the fact (Rebedea et al., 2023; Inan et al., 2023; Ruan et al., 2024). Self-Debugging (Chen et al., 2024b) repairs the program from execution feedback, PruneTIR (Zhang et al., 2026) prunes failed tool turns at inference time, and constrained decoding (Zheng et al., 2024) restricts the call to a schema. However, in every one of these methods the model constructs the call and the system acts on what it wrote, so none of them can guarantee that the call is well formed. AGENTRT moves construction into the runtime.

2.3 SERVING AGENTIC WORKLOADS

Serving systems schedule the tokens of a call the model has already requested (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024; Zhong et al., 2024). InferCept (Abhyankar et al., 2024) keeps the KV cache alive across a tool call, Conveyor (Xu et al., 2024) starts executing a tool while its call is still being decoded, Parrot (Lin et al., 2024) and Ayo (Tan et al., 2025) expose application-level dependencies to the scheduler, LLMCompiler (Kim et al., 2024) plans parallel calls, and Agentix (Luo et al., 2026) and Continuum (Li et al., 2026) schedule multi-turn agent programs and their cache. The prefill-token-equivalent proxy measures what a tool-integrated trajectory costs (Su et al., 2026). However, all of these serve or measure a call the model was already allowed to make, and none decides whether the model should see the tool at all.



(a) Accuracy on the two halves of the workload

(b) Exposure cost by what exposure did

Figure 2: **(a)** Accuracy of every policy on the two halves of the workload. The dotted line marks never-expose, and AGENTRT is the only policy in the upper right. **(b)** Exposure cost γ per request, ReAct against never-expose, grouped by what exposure did to the request, with bars at the medians.

3 PROBLEM ANALYSIS

3.1 THE TWO-SIDED COST OF EXPOSURE

Exposure damages requests that need no tool. Figure 2a plots every policy's accuracy on the requests that need no tool against its accuracy on those that need one. Withholding tools is best on the tool-negative half and catastrophic on the tool-positive half, and exposing them unconditionally inverts both. Every method that puts a schema in front of the model pays on the tool-negative half, by 3 to 18 points, and self-routing avoids the loss only by spending a generation to decide.

Exposure is not an allocation problem. Let $\Delta(x) = p_1(x) - p_0(x)$ denote the exposure gain and $\gamma(x) = c_1(x) - c_0(x)$ the exposure cost, where p_a and c_a are the solve probability and the serving cost under exposure decision $a \in \{0, 1\}$. A budgeted reading assumes $\gamma > 0$, meaning that exposure buys accuracy with resources. Figure 2b rejects that assumption. Exposure makes a request cheaper more often than costlier, and the requests it rescues are almost all among the cheaper ones. Of the 119 requests exposure rescues, 109 cost less when exposed, by a median of 36 seconds of generation. The mechanism is decode, since a model denied tools on a problem it cannot solve generates until it hits the token cap. Rationing exposure therefore solves the wrong problem on this workload. What remains is a decision under two-sided error, which we treat as selective prediction with a reject option (Chow, 1970; Geifman & El-Yaniv, 2017), where exposing plays the role of rejecting and withholding the role of accepting.

3.2 WHAT ADMISSION CAN RECOVER

Benefit is a strict subset of necessity. Let $\nu(x) = \mathbb{1}\{p_0 \text{ fails } x\}$ be the necessity label, the one that the When2Tool benchmark defines and that hidden-state probes predict.

Proposition 1 (Necessity is necessary, not sufficient)**.** $\{\Delta > 0\} \subseteq \{\nu = 1\}$*, and the inclusion is strict whenever some tool-necessary request is not rescued by exposure. Consequently an oracle policy that exposes exactly* $\{\nu = 1\}$ *attains the same accuracy as one that exposes exactly* $\{\Delta > 0\}$ *while exposing* $1/\Pr[\Delta > 0 \mid \nu = 1]$ *times as many requests.*

What matters is the ratio, measured here at $\Pr[\Delta > 0 \mid \nu = 1] = 0.371$ to 0.407. A perfect necessity oracle therefore exposes about $2.5\times$ more requests than benefit from it, so necessity is the wrong target for admission, however well it is predicted.

The value of admission. Fix the behaviour of the exposed arm, which we call the tool policy, and let R and K be the number of requests it rescues and breaks relative to withholding.

Proposition 2 (Exposure-only ceiling and its headroom)**.** *Any policy that only chooses whether to expose attains accuracy at most* $(S_0 + R)/n$*, where* S_0 *is the number of requests solved without tools. Equality holds iff it exposes exactly the rescued set. The always-expose policy attains* $(S_0 + R - K)/n$*. Hence the headroom of perfect admission over always-expose is exactly* K/n*. Admission recovers the requests the tool policy breaks and nothing else.*

Algorithm 1 AGENTRT on one request x under pressure ρ, budget B and guard limits (L_c, L_ℓ)

compute $\phi(x)$ and $e(x)$; $\pi \leftarrow P(x)$ {prompt statistics and typed parse, no forward pass}
if $\pi \neq \perp$ **then return** $E(\pi)$ {runtime-owned execution, no tool call is issued}
expose $\leftarrow \mathbb{1}[s(x) \geq \tau(\rho)] \vee \text{force}(x)$
if not expose **then return** GENERATE(x) {no schema in the prompt}
attach the tool schema to x; $c \leftarrow \mathbf{0}$; ctx $\leftarrow x$
repeat
  $y \leftarrow$ GENERATE(ctx)
  if y is a tool call **then** $o \leftarrow \text{trunc}(\text{EXECUTE}(y); L_c, L_\ell)$; append y, o to ctx; update c
  if $c_t \geq B_t$ for some t **then** withdraw the schema; $y \leftarrow$ FINALIZE(ctx)
until y is a final answer
return y

Both statements are immediate. The second is what makes the value of admission measurable, by running both arms on a calibration set and counting. It also says when admission is not worth building: where the tool policy breaks nothing, K is zero and always-expose is already optimal, which is what §5.3 finds on the newest served models.

4 DESIGN

AGENTRT sits between the model API and the tool executor and changes neither (Figure 1, Algorithm 1). A request is first parsed for a typed plan, and if one is found the runtime executes it and answers. Otherwise an admission score, computed from the prompt and the serving state before any token is generated, decides whether the model sees a tool schema, and an exposed request runs under enforcement that bounds what admission cannot predict. The three parts divide a request by what can be known before generation: a typed plan is certain, an admission score is a prediction, and enforcement covers what neither can foresee.

4.1 ADMISSION

The gate scores a request from statistics readable off the prompt in microseconds, such as counts of digits, operators and LaTeX markup and hits of two small pattern families, one for exact computation and one for output risk. The markup count separates the two halves of the workload, since a Math500 problem is dense in LaTeX and an exact-computation request carries almost none. Let $\phi(x) \in \mathbb{R}^d$ collect these statistics and let $e(x) \in \{0, \ldots, E\}$ be the exact-computation evidence, a count of pattern hits and integer-size flags. A logistic model $s(x) = \sigma(w^\top \phi(x) + b)$ over a subset of the features scores the request, with weights in Appendix B.4, and the decision is

$$\text{expose}(x) = \mathbb{1}[s(x) \geq \tau(\rho)] \;\vee\; \text{force}(x), \qquad \text{force}(x) = \mathbb{1}[e(x) \geq e^\star], \tag{1}$$

where the override forces exposure when the exact-computation evidence is strong, so the learned component can lower the exposure rate but cannot suppress a call the runtime is confident about.

The bar $\tau(\rho)$ rises with serving pressure $\rho \in [0, 1]$, read as the fraction of requests the server cannot afford to expose. At $\rho = 0$ it is the classifier's threshold, and under pressure it is the ρ-quantile of a calibration split's score distribution, which admits exactly the top $1 - \rho$ share. Either way $\tau(\rho)$ is an operating point on the two-sided error curve of Proposition 2. Raising it trades requests the tool policy would rescue against requests it would break, and in the budgeted regime it realizes the shadow price of the allocation problem in Appendix A.3.

4.2 RUNTIME-OWNED TYPED EXECUTION

For some request classes the model is not the right thing to ask. A query over a table, a statistic over a log file, and an exact arithmetic evaluation share the property that the plan is recoverable from the request text and the execution is a bounded total function of that plan. Formally, the runtime-owned route is a pair (P, E) of a parser $P : \mathcal{X} \to \Pi \cup \{\perp\}$ from request text to a typed plan and a total executor $E : \Pi \to \mathcal{V}$ from plans to values, with no code generation in the path. When $P(x) \neq \perp$ the runtime builds the program from the plan, runs it in its own sandbox, and returns $a(x) = E(P(x))$

as the answer. No tool call is issued and the model is not consulted. When $P(x) = \perp$ the request proceeds as admission decided, so coverage is measured, not asserted.

Remark 1 (Well-formedness by construction). The payloads the runtime executes on its own behalf form the image $E(P(\mathcal{X}) \setminus \{\perp\})$ of a total function on typed plans, so every one of them is well formed. A model-written payload is instead a sample from a distribution $q(\cdot \mid x)$ over strings whose support includes malformed calls, missing prints and unbounded output.

Relaying the payload or its result through the model reopens that distribution, and §5.3 measures what the relay loses on six served models.

4.3 Enforcement

Two mechanisms bound what admission cannot predict. Output isolation applies a truncation operator $\mathrm{trunc}(o; L_c, L_\ell)$ to every observation o before it enters the next prompt, keeping the head and the tail of an L_c-character, L_ℓ-line budget so that both the start of a traceback and the final value survive. A trajectory budget counts tool turns, tool calls, assistant turns and accumulated latency against an operator-configured vector B and trips as soon as any component is exhausted. When it trips, the schema is withdrawn and the model finalizes; the request is not discarded. A third mechanism is payload repair, which prints an unprinted final expression and rejects imports outside an allowlist. It never fires on the 7B workload, and §5.4 reports where it does.

5 Experiments

5.1 Setup

Models and serving. Qwen2.5-7B-Instruct (Qwen Team, 2024) is served by vLLM (Kwon et al., 2023) on an NVIDIA GB10 (128 GB unified memory) in bfloat16 with a 4096-token context, 32 concurrent requests, temperature 0 and 768 generated tokens per turn. Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Qwen3-8B (Qwen Team, 2025), Mistral-7B-Instruct-v0.3 (Jiang et al., 2023a), Qwen2.5-32B-Instruct and Qwen3.8-27B (Qwen Team, 2026) repeat the main rows under identical settings, and Qwen2.5-1.5B repeats the full matrix (Appendix C.4). All of these runs use the admission gate at its classifier threshold, which is zero serving pressure.

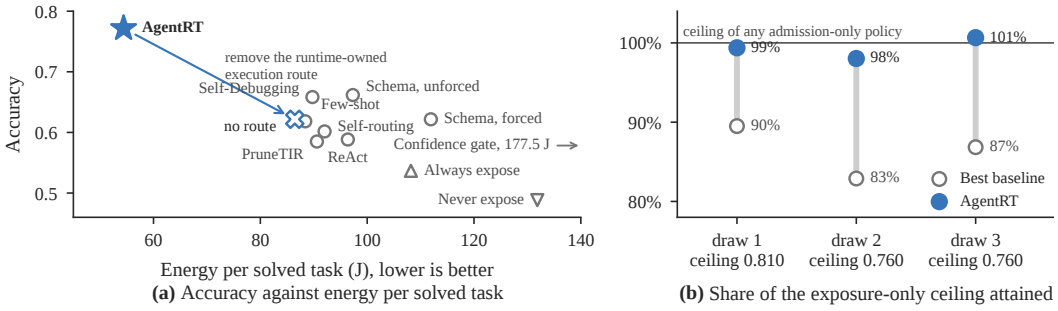
Workloads. A draw of the mixed workload samples 200 requests, 140 from Math500 and 60 synthetic exact-computation tasks spanning eight classes of exact integer and rational arithmetic, from modular powers to determinants, whose answers a 7B model cannot reliably produce by decoding alone. The Math500 half needs no tool and the exact-computation half does, which the tables call the tool-negative and tool-positive halves. Three draws redraw both halves, so the spread reflects workload sampling and not only decode nondeterminism. Two smaller workloads of 100 requests each isolate the ownership claim in §5.3: a TableQA set over one table, and a file/log set whose requests each carry a log, JSON, CSV, configuration or code artifact. When2Tool (Sun et al., 2026), a benchmark of 15 sandboxed tool environments labelled by tool necessity, is run in its own harness with Qwen3-4B against Probe&Prefill (Appendix C.5).

Baselines. Ten baselines span the two endpoints of never-expose and always-expose, four adaptations of published methods, and four standard techniques. The adaptations are ReAct-style optional exposure (Yao et al., 2023), Self-Debugging-style repair (Chen et al., 2024b), a confidence gate in the style of FLARE and DRAGIN (Jiang et al., 2023b; Su et al., 2024), and inference-time pruning in the style of PruneTIR (Zhang et al., 2026). The standard techniques are few-shot tool prompting, self-routing, and a schema-constrained call in a forced and an unforced form. Self-routing has the serving model take the routing decision of Adaptive-RAG (Jeong et al., 2024) before it answers. The forced form compels a call on every request, so the stronger unforced form is the one we compare against. Appendix B.2 gives each baseline with its prompt, measures how far the ten overlap, and reports two released fine-tuned checkpoints that confound policy with weights.

Metrics. We report accuracy and energy per solved task in joules (J). The latter is the whole-run GPU energy from board telemetry sampled at 1 Hz divided by the number of correct answers, and it compares within a platform only. Every number in the text is a mean over the three draws, and the tables carry the half-width of a Student-t 95% interval over those draws. Appendix C carries the full per-family and paraphrase tables.

Table 1: Mixed workload, Qwen2.5-7B on GB10, N=200 per draw, mean ± half-width of a Student-t 95% interval over three matched draws. The last two columns split accuracy by whether the request needs a tool.

Policy	Accuracy	J/solved	No tool	Tool
Never expose	0.488 ± 0.068	131.8 ± 19.1	0.671 ± 0.081	0.061 ± 0.063
Always expose	0.537 ± 0.133	108.2 ± 76.9	0.490 ± 0.164	0.644 ± 0.063
ReAct (Yao et al., 2023)	0.588 ± 0.029	96.4 ± 19.7	0.607 ± 0.047	0.544 ± 0.024
Few-shot prompting	0.618 ± 0.122	88.4 ± 22.3	0.593 ± 0.142	0.678 ± 0.204
Self-Debugging-style (Chen et al., 2024b)	0.658 ± 0.083	89.7 ± 30.7	0.636 ± 0.064	0.711 ± 0.157
Confidence gate (Su et al., 2024)	0.578 ± 0.052	177.5 ± 33.3	0.600 ± 0.106	0.528 ± 0.096
Self-routing	0.602 ± 0.040	92.0 ± 11.5	0.671 ± 0.047	0.439 ± 0.048
Forced schema-constrained call	0.622 ± 0.162	111.9 ± 63.2	0.536 ± 0.198	0.822 ± 0.145
PruneTIR-style (Zhang et al., 2026)	0.585 ± 0.037	90.6 ± 15.3	0.598 ± 0.074	0.556 ± 0.063
Schema-constrained, unforced	0.662 ± 0.155	97.3 ± 21.3	0.610 ± 0.159	0.783 ± 0.259
AGENTRT	0.772 ± 0.076	54.4 ± 9.1	0.674 ± 0.108	1.000
without the runtime-owned route	0.622 ± 0.031	86.5 ± 9.9	0.667 ± 0.041	0.517 ± 0.041



(a) Accuracy against energy per solved task

(b) Share of the exposure-only ceiling attained

Figure 3: **(a)** Accuracy against energy per solved task, means over three matched draws. The arrow is the ablation that disables the runtime-owned route, and the confidence gate lies off the scale. **(b)** Accuracy as a share of the exposure-only ceiling of Proposition 2, per draw. Only a policy that is not admission-only can exceed it.

5.2 ADMISSION ON THE MIXED WORKLOAD

Table 1 reports accuracy and energy per solved task for every policy, split by whether the request needs a tool, and Figure 3a plots the two axes.

The two-sided cost. On the tool-negative half the never-expose policy ties for best of all ten at 0.671, and overall it is worst at 0.488. Always-expose inverts both. Never-expose loses the tool-positive half because a thirty-digit answer cannot be produced by decoding, and always-expose loses the tool-negative half because a schema the model does not need still changes what it writes. AGENTRT is the only policy that keeps the tool-negative half intact, at 0.674 against 0.671 for never-expose, while solving all of the tool-positive half.

Accuracy and energy together. AGENTRT leads on both axes, at 0.772 accuracy and 54.4 J per solved task against 0.662 and 97.3 J for the strongest baseline. The draws are matched, so the per-draw difference is the comparison that matters, and AGENTRT is ahead of all ten baselines on both axes on every draw (Appendix C.1 gives the paired intervals). With the runtime-owned route disabled and every other component still running (last row of Table 1), accuracy falls to 0.622, below Self-Debugging-style repair and, on average, below the unforced schema baseline, while the energy lead survives on every draw. The tool-negative half stays at 0.667 and the tool-positive half collapses from all solved to 0.517, so the margin rests on the route.

Against the ceiling. Proposition 2 bounds any admission-only policy by $(S_0+R)/n$ for the exposed arm it gates, and over all nine exposed arms the strongest bound on each draw comes from gating the forced schema-constrained call. Figure 3b shows that AGENTRT reaches 98 to 101% of it, a mean of 99%, while the best baseline on each draw reaches 83 to 90%, a mean of 86%. AGENTRT

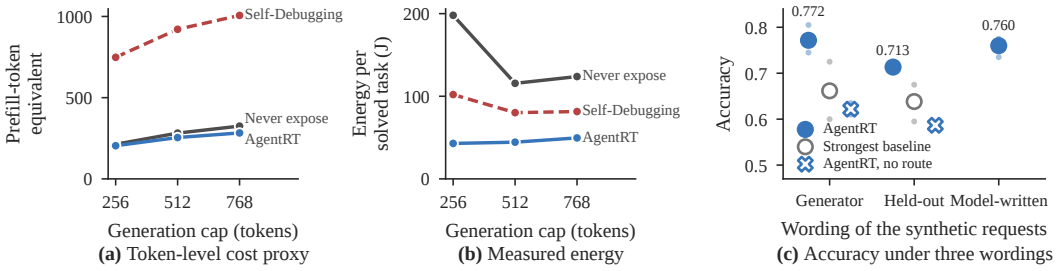


Figure 4: **(a, b)** Generation-cap sweep on one draw. The token proxy ranks never-expose far below Self-Debugging, and measured energy ranks it most expensive. **(c)** Accuracy under three wordings of the synthetic requests, dots per draw. Baselines were not re-run on the rewrites.

exceeds the bound on the third draw because where the runtime-owned route applies it replaces p_1 instead of selecting between p_0 and a fixed p_1, which admission alone cannot do.

Energy, time and the token proxy. Mean board power spans only 45.7 to 52.0 W across policies, so energy per solved task is close to accelerator time per correct answer. A token-level proxy misses the run time saved when sixty requests per draw never decode at all. The prefill-token-equivalent proxy (Su et al., 2026) ranks never-expose among the cheapest policies and Self-Debugging-style repair about three times more expensive, while measured energy ranks them in the opposite order, at 131.8 against 89.7 J per solved task. The mechanism is decode, not context. Never-expose generates the most tokens per request and hits the cap on a third of them, the repair baseline on a sixth, and AGENTRT generates the fewest on every draw (Appendix C.1). The inversion holds at every cap (Figure 4a, b), so the proxy is a context signal, not a cost predictor.

5.3 OWNERSHIP AND COVERAGE

Ownership isolated on TableQA. TableQA isolates the ownership claim with four arms over one table. Free-form Python asks the model to write pandas, a Binder-style arm (Cheng et al., 2023) has it emit a query string, a Chain-of-Table-style arm (Wang et al., 2024b) has it emit a typed plan that the runtime executes, and the runtime-owned route builds the plan without the model. Free-form Python scores 0.000 because all 167 programs are malformed, the model-produced typed plan scores 0.510, and ownership adds nothing to it, so the gain there comes from the typed interface, and ownership pays only where a parser covers the requests, as it does on every file/log request.

Parsers under paraphrase. A parser that matches its generator's phrasing overfits it, so the exact-computation parser extracts the mathematical objects a request contains and decides the operation from them, and the file/log parser takes the operation class from the artifact a request carries. Both were built on a development bank of paraphrase templates and then frozen, and a held-out bank with different vocabulary was written afterwards. On it the exact-computation parser recognizes 69% of renderings with no wrong answer and the file/log parser answers 90%, the ten pre-registered paraphrases are all recognized, and on rewrites by the served model the parser recognizes 90% with no wrong answer. The uncovered share falls through to ordinary admission.

Every policy on held-out wording. We then re-ran every policy with the synthetic requests rendered from the held-out bank, so only the wording changes (Figure 4c). The parser fires on about two thirds of the synthetic requests, and AGENTRT reaches 0.713 at 59.2 J per solved task against 0.638 and 97.1 J for the unforced call, ahead of every baseline on both axes on each draw. Disabling the route on the same wording leaves 0.587, so ownership is still worth 13 points at two-thirds coverage, and on the model-written rewrites, which the parser covers on nine in ten, AGENTRT reaches 0.760 at 51.9 J. In every configuration the runtime-owned routes emit no empty or malformed observation, against 47 for always-expose and 239 for the forced schema call.

Coverage outside the workload. How often a parser applies is a property of the workload, so we ran the frozen parsers over ten external corpora chosen in advance (Appendix C.6). They take over 0.26% of the requests, the exact-computation parser on 0.9% of DeepMind Mathematics and almost none of the general prompts, the file/log parser on 2.7% of TabMWP and 1.2% of WikiTable-

Table 2: Six served models, draw means. Exposure cost is never-expose minus always-expose on the tool-negative half, in points. †Energy from a second GB10 unit. ‡See Appendix C.2.

Model	Exposure cost	Strongest baseline	AGENTRT	No route
Qwen2.5-7B	+18.1	0.662 / 97 J (unforced schema)	0.772 / 54 J	0.622
Llama-3.1-8B	+21.9	0.533 / 80 J (unforced schema)	0.645 / 53 J	0.475
Qwen3-8B	−0.5	0.767 / 108 J (unforced schema)	0.782 / 54 J†	0.660
Mistral-7B-v0.3	+13.3	0.145 / 412 J (never-expose)	0.427 / 93 J	0.127
Qwen2.5-32B	+11.7	0.793 / 254 J (forced schema)	0.783 / 210 J	0.672
Qwen3.8-27B‡	−3.8	0.933 / 695 J (forced schema)	0.890 / 372 J	0.770

Table 3: Component ablation on the mixed workload, three draws, same gate and route throughout.

Configuration	Accuracy	Calls	Empty obs.
Admission only	0.775 ± 0.037	0.022 ± 0.026	1
+ trajectory budget and payload repair	0.777 ± 0.057	0.017 ± 0.026	0
+ output guard (full)	0.772 ± 0.076	0.022 ± 0.026	0

Questions, and both fail closed elsewhere. Runtime-owned execution is thus a guarantee about the requests it covers, not a share of traffic. End to end on DeepMind Mathematics and TabMWP the exposure cost recurs at 17 and 67 points, and AGENTRT matches never-expose on both at the lowest energy by withholding the schema, not by taking requests over.

Other served models. Table 2 repeats the main rows on five more served models. The exposure cost recurs on Llama-3.1-8B and Mistral-7B, where the route triples the best accuracy, and does not vanish with size, since forced exposure still costs Qwen2.5-32B 12 points. It vanishes with generation. Qwen3-8B and Qwen3.8-27B pay nothing, and the headroom of admission over the strongest exposing baseline falls with the cost, from a tenth of the workload on Qwen2.5-7B to a twentieth on Qwen2.5-32B and almost nothing on Qwen3.8-27B. AGENTRT's margin over that baseline falls with it, from 11 points ahead to 1 and 4 behind, but its energy lead does not. It is the cheapest policy per correct answer on every draw of every model, by 17% and 46% on the two larger ones, and relaying the payload through the model costs accuracy on every model.

5.4 ABLATIONS

Component ablation. Table 3 takes the runtime apart. Admission alone reaches 0.775, and the trajectory budget and output guard add nothing measurable, since the exposed loop handles about four requests per draw. Enforcement matters where the model writes the calls. On Qwen2.5-1.5B always-expose prints nothing on 67 of 200 requests, and payload repair fixes every case it reaches.

Two negative findings. The learned gate, fitted on Math500 only, scores 0.260 on the file/log workload where a runtime that always exposes and keeps only the trajectory budget scores 0.250, and 0.100 on the held-out paraphrases where that runtime scores 0.210, so what transfers is where the decision is made, not the predictor. Post-execution repair has nothing left after admission, since stacking it on AGENTRT moves a draw from 0.805 to 0.795 where repair alone scores 0.695.

6 CONCLUSION

In this paper, we showed that offering a tool to a model that does not need one is not free, costing 18.1 accuracy points on the requests of a mixed workload that never needed one. Two facts followed. Benefit is a strict subset of necessity, and admission is worth exactly what exposure breaks. AGENTRT acted on both, deciding exposure in the runtime without a second forward pass and executing the payload itself where a request admits a typed plan. It reached 99% of the exposure-only ceiling at 44% lower energy than the strongest baseline, and kept its lead on re-worded requests, 7B-class models and two external workloads and its energy lead on larger models that pay no exposure cost.

AI USE STATEMENT

Generative AI tools assisted with formulating the claims of §3 and drafting their proofs, with designing and reviewing the experimental methodology and code, with generating the paraphrase templates and the model-written paraphrases of Appendix C.3, and with drafting text, figures and references. Every claim, proof, number and citation was checked by the authors, every number is produced by our scripts on the hardware of §5.1, and we take full responsibility for the content of this work.

REPRODUCIBILITY STATEMENT

Appendix A gives complete proofs of Propositions 1 and 2. Appendix B specifies the serving configuration, the workload construction, the energy measurement and every baseline with its prompt. The runtime, the scripts and the per-request result files they read will be released upon acceptance.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. AutoMix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, 2024.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation*, 2024.

Harrison Chase. LangChain. https://github.com/langchain-ai/langchain, 2022.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *Proceedings of the 12th International Conference on Learning Representations*, 2024b.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-Eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024c.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Binding language models in symbolic languages. In *Proceedings of the 11th International Conference on Learning Representations*, 2023.

C. K. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970. doi: 10.1109/TIT.1970.1054406.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V.S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama Guard: LLM-based input-output safeguard for Human-AI conversations. *arXiv preprint arXiv:2312.06674*, 2023. URL https://arxiv.org/abs/2312.06674.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7036–7050. Association for Computational Linguistics, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023a.

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7969–7992. Association for Computational Linguistics, 2023b.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 24370–24391, 2024.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. OpenAssistant conversations – democratizing large language model alignment. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph E. Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2026.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *International Conference on Learning Representations (ICLR)*, 2023.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Agentix: An efficient serving engine for LLM agents as general programs. In *23rd USENIX Symposium on Networked Systems Design and Implementation*, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, 2024.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. In *Advances in Neural Information Processing Systems*, 2025a.

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. SMART: Self-aware agent for tool overuse mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621. Association for Computational Linguistics, 2025b.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, Minghua Ma, Pu Zhao, Si Qin, Xiaoting Qin, Chao Du, Yong Xu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qwen Team. Qwen3.8-27B. Model card, https://huggingface.co/Qwen/Qwen3.8-27B, 2026. Released August 2026.

Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2023.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*, 2019.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Qisheng Su, Shiting Huang, Zhen Fang, Ziyan Chen, Zehui Chen, and Feng Zhao. Beyond accuracy: Unveiling inefficiency patterns in tool-integrated reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7456–7477. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.339.

Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. DRAGIN: Dynamic retrieval augmented generation based on the real-time information needs of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12991–13013. Association for Computational Linguistics, 2024.

Chung-En Sun, Linbo Liu, Ge Yan, Zimo Wang, and Tsui-Wei Weng. LLM agents already know when to call tools – even without reasoning. *arXiv preprint arXiv:2605.09252*, 2026.

Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. Towards end-to-end optimization of LLM-based applications with Ayo. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently. *arXiv preprint arXiv:2504.14870*, 2025a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, 2024a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025b.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *Proceedings of the 12th International Conference on Learning Representations*, 2024b.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Yechen Xu, Xinhao Kong, Tingjun Chen, and Danyang Zhuo. Conveyor: Efficient tool-aware LLM serving with tool partial execution. *arXiv preprint arXiv:2406.00059*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation*, 2022.

Luan Zhang, Dandan Song, Zhijing Wu, Zhengyu Chen, Chen Zhang, Yuhang Tian, Huipeng Ma, Chenhao Li, Changzhi Zhou, Xudong Li, and Shuhao Zhang. PruneTIR: Inference-time tool call pruning for effective yet efficient tool-integrated reasoning. *arXiv preprint arXiv:2605.09931*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*, 2024.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. ToolQA: A dataset for LLM question answering with external tools. In *Advances in Neural Information Processing Systems*, 2023.

A PROOFS

Throughout, n requests $x_1, \ldots, x_n$ arrive; $a_i \in \{0, 1\}$ is the exposure decision; $p_a(x) \in [0, 1]$ is the probability the request is solved under decision a; and $c_a(x) \geq 0$ is its serving cost. Let $\Delta(x) = p_1(x) - p_0(x)$ and $\gamma(x) = c_1(x) - c_0(x)$ as in §3. In the deterministic evaluation setting of §5, $p_a(x) \in \{0, 1\}$ and $\Delta(x) \in \{-1, 0, 1\}$.

A.1 PROPOSITION 1: NECESSITY IS NECESSARY, NOT SUFFICIENT

Let $\nu(x) = \mathbb{1}\{p_0(x) = 0\}$.

Inclusion. If $\nu(x) = 0$ then $p_0(x) = 1$, so $\Delta(x) = p_1(x) - 1 \leq 0$ and $x \notin \{\Delta > 0\}$. Hence $\{\Delta > 0\} \subseteq \{\nu = 1\}$.

Strictness. The inclusion is strict exactly when some x has $p_0(x) = 0$ and $p_1(x) = 0$, i.e. a request that needs a tool and is not rescued by one. Such requests exist in every configuration we measured.

Consequence. Both the oracle that exposes $\{\nu = 1\}$ and the oracle that exposes $\{\Delta > 0\}$ solve $S_0 + |\{\Delta > 0\}|$ requests, since exposure on $\{\nu = 1\} \setminus \{\Delta > 0\}$ changes no outcome by construction. They differ only in how many requests they expose, and the ratio of those counts is $|\{\nu = 1\}|/|\{\Delta > 0\}| = 1/\Pr[\Delta > 0 \mid \nu = 1]$. □

Two remarks follow. The two oracles coincide in accuracy but not in cost, since the necessity oracle materializes tool output for requests that gain nothing from it, and each such materialization is an opportunity for the failures catalogued in §5.2. The proposition also assumes that exposure never helps a request the no-tool policy already solves, which is exact under deterministic decoding and holds in expectation otherwise.

A.2 PROPOSITION 2: THE EXPOSURE-ONLY CEILING AND ITS HEADROOM

Fix the behaviour of the exposed arm, so that p_1 is a fixed function. Let $S_0 = |\{x : p_0(x) = 1\}|$, let $R = |\{x : p_0(x) = 0,\ p_1(x) = 1\}|$ be the rescued set, and let $K = |\{x : p_0(x) = 1,\ p_1(x) = 0\}|$ be the broken set.

Upper bound. A policy π that only chooses a solves, on each request, $p_a(x)$ for its chosen a, hence at most $\max\{p_0(x), p_1(x)\}$. Summing,

$$\mathrm{Acc}(\pi) \;\leq\; \frac{1}{n} \sum_x \max\{p_0(x), p_1(x)\} \;=\; \frac{S_0 + R}{n},$$

because $\max$ differs from p_0 exactly on the rescued set. Equality holds iff π exposes every rescued request and no broken one.

Headroom. The always-expose policy solves $S_0 + R - K$ requests, since it gains the rescued set and loses the broken one. Subtracting,

$$\frac{S_0 + R}{n} - \frac{S_0 + R - K}{n} = \frac{K}{n}. \qquad \square$$

So the entire value of admission over always-expose is the requests the tool policy breaks, and it is measurable without any oracle by running both arms on a calibration set and counting. The bound is a property of the pair of a withheld and an exposed arm, so it must be recomputed for each exposed arm, and quoting the bound of a weak arm would understate what admission-only methods can achieve. §5.2 therefore reports the maximum over all nine exposed arms.

A.3 The budgeted regime

When exposure is genuinely more expensive ($\gamma > 0$), admission becomes an allocation problem:

$$\max_{a \in \{0,1\}^n} \sum_i p_{a_i}(x_i) \qquad \text{s.t.} \qquad \sum_i c_{a_i}(x_i) \le B.$$

Relaxing $a_i \in [0, 1]$ and forming the Lagrangian $\mathcal{L}(a, \lambda) = \sum_i p_{a_i}(x_i) - \lambda(\sum_i c_{a_i}(x_i) - B)$, the coefficient of a_i is $\Delta(x_i) - \lambda\gamma(x_i)$, so the maximizer sets $a_i = 1$ when $\Delta(x_i) > \lambda\gamma(x_i)$ and $a_i = 0$ when the inequality reverses, with $\lambda \ge 0$ chosen by complementary slackness so that the constraint is tight. Ordering by Δ/γ and admitting greedily until the budget is exhausted realizes this, which is the standard fractional-knapsack argument. Setting $\gamma \equiv$ const recovers a fixed posterior threshold, i.e. Chow's rule (Chow, 1970).

On the workload of §5, $\gamma \le 0$ for the majority of requests on every draw and for all but a handful of the requests exposure rescues, so the binding constraint is accuracy and not capacity. The budgeted regime would arrive under concurrency, where exposure lengthens queues instead of shortening individual requests, but a sweep to eight concurrent requests does not saturate this hardware and exposure still shortens the average request there (Appendix C.4), so the characterization describes a regime outside these measurements. Where $\gamma > 0$ and exposure lengthens queues, the runtime must ration, and the budgeted optimum is the threshold rule $\Delta(x) > \lambda\gamma(x)$ whose shadow price λ is the pressure parameter of §4.1. Δ is unknown in practice, so the design estimates only exposure gain before generation and bounds latency, context growth and failure risk afterwards.

B Experimental Details

B.1 Serving, decoding and budgets

Serving and decoding. All mixed-workload runs serve Qwen2.5-7B-Instruct through vLLM on a single GB10 unit, with greedy decoding at temperature 0, a generation cap of 768 tokens, at most 5 assistant turns and an effectively unbounded user-turn limit. The Qwen3.8-27B rows of Appendix C.2 are the one exception to the cap. vLLM runs with prefix caching, chunked prefill and CUDA graphs enabled. This is the conservative choice, since they remove much of the re-prefill cost of tool-induced context growth. These values are recorded in every result row and are identical across all ten policies and all three draws. The generation-cap sweep in §5 varies the cap to 256 and 512 and applies each value to every policy within a sweep point. The file/log workload is served at an 8192-token context, since its requests carry the artifact in the prompt. The When2Tool runs serve Qwen3-4B-Instruct with an 8192-token context and a 1024-token generation cap, one run per arm.

What each policy is allowed to spend. The tool budget is asymmetric in the baselines' favour. The trajectory budget is enabled only for AgentRT and caps it at one tool turn and one tool call, while every baseline may use tools across all five assistant turns. The output guard bounds each observation to 512 characters or 20 lines of stdout and 1024 characters or 40 lines of stderr, head-and-tail. On When2Tool the guard removed 17.5% of the prompt tokens the benchmark had already charged for. Its cost metric does not refund them, so the cost we report there is an overestimate of what the runtime actually spent.

Energy and intervals. Board telemetry is sampled once per second for the duration of each variant's run and integrated over that window. Per-variant profiles are collected separately, so no idle interval is attributed to a policy that did not incur it. Energy per solved task divides the integral by the number of correct answers, so on a workload where power is nearly invariant it is close to time over accuracy, which §5 states explicitly. Every interval in the tables is the half-width of a Student-t 95% interval over the three draws, with $t_{.975,2} = 4.30$.

B.2 Baselines in full

The ten baselines. §5.1 lists the ten. Self-Debugging-style repair returns real stdout and stderr and asks the model to explain and fix its code (Chen et al., 2024b). The confidence gate takes a tentative direct answer and accepts it when token confidence clears a threshold, in the style of FLARE and DRAGIN (Jiang et al., 2023b; Su et al., 2024). Self-routing asks the model to decide for itself before it answers, which is the routing decision of Adaptive-RAG (Jeong et al., 2024) taken by the serving model instead of an auxiliary classifier. The schema row is labelled forced because its first turn is served with `tool_choice=required` and a schema-valid call is compelled on every request, whereas published guided decoding constrains the arguments of a call the model chose to make. The unforced variant is the stronger of the two, at 0.662 against 0.622 with half the tool calls, and it is the strongest baseline in Table 1, so the reported comparison is against the better form of that method. The auxiliary-classifier form of Adaptive-RAG is implemented but not evaluated here, since it needs a second model served alongside the first.

The ten baselines overlap. ReAct-style exposure, PruneTIR-style pruning, the confidence gate and self-routing share one system prompt and differ only by a mechanism layered on it. Measured by per-request agreement with ReAct, pruning agrees 92% of the time and the confidence gate 92%, against 83% for self-routing and 79% for Self-Debugging-style repair, so on this workload the first two are ReAct with a rarely firing addition. The published methods are adapted, not reproduced. Self-Debugging was proposed for code-generation benchmarks and its authors report accuracy, not energy, so the mechanism is ported into the agent loop and every cost figure attached to it is ours; the same holds for the confidence gate, instantiated from the served model's token logprobs.

The confidence gate in detail. The gate spends a tentative generation before deciding and is the most expensive policy measured, at 177.5 ± 33.3 J per solved task. Its statistic, the minimum token probability over a several-hundred-token answer, never clears the configured threshold, so it accepted 0 of 600 requests and its row of Table 1 is ReAct plus a dead decode. At a reachable threshold it fires and scores worse, 0.550 against 0.578, since the direct answers it accepts are wrong more often than the calls they save.

Trained tool policies. Two further baselines put the tool policy in the weights instead of the runtime, and we serve their released checkpoints unmodified on one workload draw of 200 requests. ToolRL (Qian et al., 2025a) reaches 0.290 accuracy at 176 J per solved task, and SMARTAgent (Qian et al., 2025b) reaches 0.520 at 264 J. Both are more expensive per correct answer than never exposing a tool at all, which reaches 0.488 at 132 J, and both sit far below AgentRT at 0.772 and 54 J. Two qualifications apply. Only ToolRL shares our base model; SMARTAgent is trained from Mistral-7B, so its row is a reference point and not a controlled comparison. Both were also trained against tool contracts unlike the one here, ToolRL on structured function-calling corpora and SMARTAgent on its own code, search and ask triple, whereas this workload exposes a Python interpreter with a print convention, and the failure counts show that mismatch: ToolRL emits code that produces no output on 60 of 200 requests and omits a print on 49, while parsing cleanly and preserving its own reasoning across turns. The comparison that matters is therefore about deployment properties. A trained policy carries its tool contract in the weights and cannot be re-pointed at a different one without retraining, while an interposed control plane adapts to the contract it is given and observes serving state that no checkpoint can. The two are complementary in that sense, which is why the composition experiments of Appendix C.5 and §5.4 are the informative comparisons.

B.3 Prompts

The ten policies share five system prompts, reproduced here so that the reader can see what each baseline was asked; Self-Debugging-style repair receives the longest of them, the few-shot prompt.

Prompt	Policies using it
Never expose	Never expose
Always expose	Always expose
Tool-optional	ReAct, PruneTIR-style, Confidence gate, Self-routing
Few-shot	Few-shot prompting, Self-Debugging-style repair
Strict schema	Forced schema-constrained call

Prompt: Never expose

```
You are a Mathematical Problem Solving Master.  Solve the problem accurately using
reasoning only.
Policy:
1.  Do not call or request external tools.
2.  Show concise reasoning when helpful.
3.  Provide the final answer wrapped in \boxed{answer}.
```

Prompt: Always expose

```
You are a Mathematical Problem Solving Master.  Solve the problem accurately by combining
reasoning with Python verification.
Policy:
1.  You MUST call python_tool before giving the final answer.
2.  The tool call arguments MUST be a JSON object with exactly this shape:  {"code":
"..."}.  Do not pass a bare string as arguments.
3.  The Python code MUST contain the literal substring print( and must use print(...)  for
every value you need to observe.  If stdout is empty, fix the code and retry.
4.  After tool verification, provide the final answer wrapped in \boxed{answer}.
```

Prompt: Tool-optional

```
You are a Mathematical Problem Solving Master.  Solve the problem accurately.
Policy:
1.  python_tool is available, but use it only when computation, symbolic manipulation, or
verification is likely to improve correctness.
2.  If you call python_tool, arguments MUST be a JSON object with exactly this shape:
{"code":  "..."}.  The code MUST print(...)  any value you need to observe.
3.  If the problem is simple enough, solve directly without tool calls.
4.  Provide the final answer wrapped in \boxed{answer}.
```

Prompt: Few-shot

```
You are a Mathematical Problem Solving Master.  Solve the problem accurately.
Policy:
1.  python_tool is available, but use it only when exact computation or verification is
likely to change the answer.
2.  If you call python_tool, arguments MUST be a JSON object with exactly this shape:
{"code":  "..."}.  Every value you need to observe MUST be passed to print(...); a script
that ends in a bare expression returns nothing.
3.  Provide the final answer wrapped in \boxed{answer}.

Worked examples:

Example 1 (no tool needed)
Problem:  What is 12 + 5?
Assistant:  The sum is 17.  \boxed{17}

Example 2 (tool needed, correct payload)
Problem:  Compute 7^222 mod 1000.
Assistant calls python_tool with {"code":  "print(pow(7, 222, 1000))"}
Observation:  stdout:  249 / stderr:  (empty)
Assistant:  \boxed{249}

Example 3 (the mistake to avoid)
Wrong payload:  {"code":  "import math\nmath.sqrt(1444)"} -> stdout is empty, the call
returns no evidence.
Right payload:  {"code":  "import math\nprint(math.sqrt(1444))"}
```

Prompt: Strict schema

```
You are a Mathematical Problem Solving Master.  Solve the problem accurately with Python
verification when useful.
Strict tool policy:
1.  When using python_tool, the function arguments MUST be a valid JSON object and MUST NOT
```

```
be a raw string.
2.  The only allowed argument key is code.  Use exactly {"code":  "..."}.
3.  The code value must be a Python string and must include print(...).
4.  If a tool call fails, correct the JSON or code format before retrying.
5.  Provide the final answer wrapped in \boxed{answer}.
```

B.4 THE ADMISSION GATE

Two gates are implemented, and the experiments report the learned one. The auditable rule gate sums eight indicator terms over cheap lexical features of the request: prompt length at least 260 characters, at least 8 digits, at least 6 operators, at least 16 LaTeX markers, at least two keyword hits, any high-utility keyword hit, an exact-computation score capped at three, and a flag that the output would want guarding. The exact-computation evidence $e(x)$ of §4.1 takes values in $\{0, \ldots, 8\}$, and the override $\mathrm{force}(x)$ of Equation 1 fires when it reaches 3, or 2 with the modular-arithmetic pattern among the hits. The learned gate is a logistic model over six of those features, fitted on 200 Math500 rows of which 15 are positive. Its threshold is 0.5, its bias -2.59, and its fitted weights are:

Feature	Weight
Prompt length in characters	-0.00184
Digit count	$+0.00786$
Operator count	$+0.02085$
LaTeX marker count	$+0.03172$
Keyword hit count	-0.36354
High-utility keyword hit count	$+0.21868$

Neither gate consults the model, and neither costs a forward pass, which is the property admission relies on. How well the predictor separates a workload varies with the workload, and Appendix C.5 reports one on which it has no discriminative power. Pressure is read as the fraction of requests the server cannot afford and realized as that quantile of the score distribution on a held-out split, with a deterministic per-request hash breaking ties, since the score is coarse enough that a large share of requests are indistinguishable to it. Realized admission rates are reported alongside requested ones, because on When2Tool they diverge by a few points, entirely through that tie group.

B.5 THE RUNTIME-OWNED ROUTES

The exact-computation parser. The parser recognizes eight request families and emits one bounded Python program for each: modular exponentiation, a modular polynomial sum, a matrix product entry, a linear recurrence, a greatest common divisor, a sum of fractions, a determinant, and a binomial prefix sum. It normalizes notation, extracts the mathematical objects present, and decides the family from the objects plus a small lexicon of operation words. A request whose objects do not determine one family falls through to ordinary admission. Appendix C.3 describes the normalization, the held-out discipline under which coverage was measured, and the per-class coverage. On Math500 the parser fires on one problem in five hundred, a greatest-common-divisor request, and solves it correctly with the same program.

Who executes the payload. When the parser fires, the runtime executes the program it emitted in its own sandbox and returns the printed value as the final answer. The model is not consulted for that request, and no tool call is issued for it, which is why AGENTRT issues 0.02 tool calls per request on the mixed workload. Two other routes exist and are measured in Appendix C.2. In the first the runtime places the program in the system prompt and instructs the model to issue it as its tool call, so the model relays the payload. In the second the runtime executes the program and asks the model to copy the printed value into the answer, so the model copies the result. Both were abandoned as the main configuration because each hop through the model is a place where a correct payload or a correct result is altered.

The file/log parser. The parser recognizes five operation classes and dispatches to a typed executor: an error count over a log, a filtered sum over JSON records, a grouped maximum over CSV rows, a key lookup in a configuration, and the output of a deterministic snippet. The class comes from the fenced artifact the request carries, from its language tag or, when the tag is missing, from

the block's shape, and every argument is grounded in the artifact's own values. An argument the artifact cannot ground fails closed. The parser reads the request text and never the workload's category label, and deleting the label changes no answer. Appendix C.3 reports its coverage on paraphrased questions of the same workload.

The table route. The TableQA route does not parse the question. It reads a filter, metric and aggregation triple from the workload's own metadata and executes it, so its accuracy of 0.270 on that workload measures the executor under the metadata's plan and not a deployable parser. The Chain-of-Table-style arm in the same comparison drives the same executor with a model-produced plan under a typed grammar, and that arm is deployable.

B.6 Implementation

AgentRT is a Python control plane that wraps an OpenAI-compatible client and a tool executor, runs against unmodified vLLM (Kwon et al., 2023), and requires no scheduler or kernel changes. The gate, the parsers, the payload repair and the guards are CPU-side, and the largest hook costs 0.242 ms on average, which is negligible beside multi-second turns. Tool execution runs in a subprocess sandbox for profiling and in a hardened container backend for deployment. The two are interchangeable, and no reported number depends on which one ran.

C Additional Results

C.1 Statistics of the main comparison

Paired intervals. Table 1 reports two independent intervals per row, and with three draws those are wide enough to overlap even when one policy wins on every draw. The draws are matched, so the comparison that answers whether AgentRT wins is the paired per-draw difference. A paired Student-t 95% interval on that difference excludes zero against all ten baselines on accuracy and against nine of ten on energy per solved task. The exception on energy is the forced schema-constrained call, whose energy varies widely across draws. With the runtime-owned route disabled, the paired interval on accuracy excludes zero only against never-expose and self-routing, which is the sense in which the margin rests on the route.

Decode tokens. Never-expose generates 534 tokens per request and hits the generation cap on 32% of them, against 398 tokens and 17.5% for Self-Debugging-style repair, and AgentRT is lowest on every draw at 327 tokens and 13%.

The gate off its calibration distribution. The learned gate was fitted on Math500 rows only. A budget-only runtime exposes the tool to every request and keeps only the trajectory budget and its soft fallback. On the file/log workload in the generator's wording the learned gate scores 0.260 and the budget-only runtime 0.250, and on the held-out paraphrases of the same workload they score 0.100 and 0.210. Both are single runs of 100 requests, and never-expose scores 0.280 and 0.240 on the same two workloads.

Exposure gain against exposure cost. Table 4 gives the joint distribution behind Figure 2b for one draw.

C.2 Other served models

Table 5 repeats the rows of Table 1 that the claims rest on for five further served models, on the same three draws, with the same evaluator, concurrency and temperature. Llama-3.1-8B-Instruct is served with vLLM's Llama 3.1 JSON tool template and parser, Qwen3-8B with thinking disabled through its chat template, Mistral-7B-Instruct-v0.3 with vLLM's Mistral parallel tool template, Qwen2.5-32B-Instruct exactly as Qwen2.5-7B, and Qwen3.8-27B with thinking disabled, vLLM's Qwen3 XML tool parser and no multimodal inputs. Qwen3.8-27B is also the one departure from the generation cap. At 768 tokens it leaves 49% of its no-tool Math500 answers unfinished, long structured derivations and not loops, and scores 0.507 on that half, so its rows use a $2{,}048$-token cap and an

Table 4: Joint distribution of exposure gain Δ and exposure cost γ over the 200 requests of one draw, with exposure instantiated as ReAct-style optional tool use. Exposure makes a request cheaper more often than costlier, on 68%, 66% and 52% of requests across the three draws, and among the requests exposure rescues almost none cost more, at 3 of 36, 4 of 44 and 3 of 39. The requests that benefit are not the expensive ones, so admission is not an allocation problem on this workload.

	$\gamma > 0$ (exposure costs more)	$\gamma \leq 0$ (exposure costs less)
$\Delta = +1$ (rescued)	3	33
$\Delta = 0$ (no change)	54	90
$\Delta = -1$ (broken)	8	12

8,192-token context. Each model was measured on one GB10 unit, and energy is compared within a model only. The Qwen3-8B runtime-executed rows are the one exception. They were measured on the second unit, which draws 5 to 9% less board power at identical duration, so their joules read low by about that much against the Qwen3-8B baselines.

The exposure cost is a property of the model's tool policy. On the tool-negative half, always-expose loses 18.1 points to never-expose on Qwen2.5-7B, 21.9 on Llama-3.1-8B, 13.3 on Mistral-7B, where it removes three quarters of an already low accuracy, and 11.7 on Qwen2.5-32B, so the cost survives a fourfold increase in size within one generation. It does not survive a change of generation. On Qwen3-8B the loss is -0.5 points and on Qwen3.8-27B -3.8, so the newest model answers the tool-negative half better with a forced tool than without one, and on both the unforced or forced schema call is four to six points better than never-expose on that half. Proposition 2 says what that means for admission. Its headroom is the rate at which the tool policy breaks requests, and on Qwen3-8B and Qwen3.8-27B that rate is about zero, so no admission policy can do better than exposing everything. AGENTRT accordingly ties the strongest Qwen3-8B baseline on accuracy at 0.782 against 0.767, ahead on two draws of three, and finishes 0.043 behind the forced schema call on Qwen3.8-27B and 0.010 behind it on Qwen2.5-32B, on every draw in both cases. Table 6 lines the six models up. The headroom against the strongest exposing baseline falls from a tenth of the workload on the 7B-class models that pay the cost to a twentieth on Qwen2.5-32B and to at most a fortieth on Qwen3.8-27B, and AGENTRT's margin over that baseline tracks it, so the admission gate as fitted collects what the ceiling leaves it and no more. Its gain on those three models is energy. It is the cheapest policy per correct answer on every draw of each, by 46% against the strongest Qwen3.8-27B baseline and 17% on Qwen2.5-32B, because admission withholds the schema from the tool-negative half and the runtime answers the tool-positive half without decoding. On Qwen2.5-7B, Llama-3.1-8B and Mistral-7B it is ahead of every baseline on both axes on every draw.

Every hop through the model is a place where the payload is corrupted. The two indented rows measure what relaying costs. When the model re-emits the runtime's payload as its own tool call, Qwen2.5-7B copies it faithfully and keeps a tool-positive half of 0.972, but Llama-3.1-8B rewrites it on 7 of 60 synthetic requests per draw, simplifying a matrix product to a single element product, and truncates six long answers. Mistral-7B rewrites 44 of 60 into code the payload linter rejects, so its tool-positive half is zero. When the model only copies the runtime's result, Qwen2.5-7B still loses seven requests on one draw, because answers of 1,150 digits exceed the 768-token generation cap. The stronger the model, the more it rewrites. Qwen2.5-32B rewrites the runtime's payload on a third of the tool-positive requests, so the relay route reaches a tool-positive half of 0.689 where the runtime-executed route reaches all of it, and Qwen3.8-27B rewrites on a ninth. Executing in the runtime and answering directly removes both failure modes on every model and is the cheapest configuration on every model.

Mistral-7B collapses under any exposure. Every exposed baseline is below never-expose, and the exposed arms leave 98 to 121 of 200 observations empty. The runtime-owned route reaches 0.427 against 0.145 for the best arm, which triples accuracy by taking the tool-positive half away from the model. Admission keeps the tool-negative half at never-expose's level, 0.181 against 0.176.

C.3 PARSER COVERAGE ON PARAPHRASED REQUESTS

A parser that recognizes a request by matching the phrases its workload generator emits fires on every request in the generator's wording and on none of a paraphrase; the first version of both parsers

Table 5: The exposure decision on six served models, three draws each, mean ± half-width of a Student-t 95% interval. AGENTRT is the configuration of Table 1, in which the runtime executes the exact-computation payload and answers. The two indented rows instead relay that payload or its printed result through the model. Qwen3.8-27B is served with a 2,048-token generation cap and an 8,192-token context, since at the 768-token cap of the other rows half of its answers are cut off.

Policy	Accuracy	J/solved	No tool	Tool
Qwen2.5-7B				
Never expose	0.488 ± 0.068	131.8 ± 19.1	0.671 ± 0.081	0.061 ± 0.063
Always expose	0.537 ± 0.133	108.2 ± 76.9	0.490 ± 0.164	0.644 ± 0.063
Strongest baseline (unforced schema)	0.662 ± 0.155	97.3 ± 21.3	0.610 ± 0.159	0.783 ± 0.259
AGENTRT	0.772 ± 0.076	54.4 ± 9.1	0.674 ± 0.108	1.000 ± 0.000
model relays the payload	0.760 ± 0.057	68.2 ± 11.7	0.669 ± 0.074	0.972 ± 0.024
model copies the result	0.752 ± 0.095	58.0 ± 15.8	0.679 ± 0.094	0.922 ± 0.167
without the runtime-owned route	0.622 ± 0.031			
Llama-3.1-8B				
Never expose	0.348 ± 0.071	151.9 ± 33.8	0.486 ± 0.081	0.028 ± 0.048
Always expose	0.413 ± 0.040	91.5 ± 16.9	0.267 ± 0.082	0.756 ± 0.063
Strongest baseline (unforced schema)	0.533 ± 0.064	80.0 ± 32.6	0.386 ± 0.035	0.878 ± 0.133
AGENTRT	0.645 ± 0.054	53.0 ± 5.1	0.493 ± 0.077	1.000 ± 0.000
model relays the payload	0.570 ± 0.097	76.1 ± 14.8	0.476 ± 0.143	0.789 ± 0.024
model copies the result	0.640 ± 0.078	59.2 ± 17.7	0.486 ± 0.111	1.000 ± 0.000
without the runtime-owned route	0.475 ± 0.054			
Qwen3-8B				
Never expose	0.493 ± 0.019	136.5 ± 5.7	0.669 ± 0.045	0.083 ± 0.110
Always expose	0.728 ± 0.127	124.9 ± 35.2	0.674 ± 0.136	0.856 ± 0.187
Strongest baseline (unforced schema)	0.767 ± 0.068	107.9 ± 21.8	0.710 ± 0.027	0.900 ± 0.219
AGENTRT	0.782 ± 0.036	53.7 ± 6.1	0.688 ± 0.051	1.000 ± 0.000
model relays the payload	0.760 ± 0.045	73.9 ± 6.9	0.690 ± 0.057	0.922 ± 0.063
model copies the result	0.753 ± 0.075	62.4 ± 16.2	0.681 ± 0.051	0.922 ± 0.167
without the runtime-owned route	0.660 ± 0.022			
Mistral-7B-v0.3				
Never expose	0.145 ± 0.090	411.7 ± 322.5	0.176 ± 0.098	0.072 ± 0.086
Always expose	0.035 ± 0.022	2298.1 ± 1502.9	0.043 ± 0.000	0.017 ± 0.072
Strongest baseline (Self-Debugging)	0.125 ± 0.033	449.8 ± 133.7	0.152 ± 0.062	0.061 ± 0.086
AGENTRT	0.427 ± 0.126	92.6 ± 16.2	0.181 ± 0.180	1.000 ± 0.000
model relays the payload	0.112 ± 0.072	438.7 ± 198.1	0.160 ± 0.102	0.000 ± 0.000
model copies the result	0.383 ± 0.111	112.7 ± 38.4	0.162 ± 0.118	0.900 ± 0.124
without the runtime-owned route	0.127 ± 0.064			
Qwen2.5-32B				
Never expose	0.527 ± 0.044	486.1 ± 31.6	0.705 ± 0.057	0.111 ± 0.063
Always expose	0.675 ± 0.069	411.7 ± 64.5	0.588 ± 0.118	0.878 ± 0.157
Strongest baseline (forced schema)	0.793 ± 0.061	254.5 ± 27.6	0.743 ± 0.077	0.911 ± 0.157
AGENTRT	0.783 ± 0.007	210.1 ± 10.3	0.690 ± 0.010	1.000 ± 0.000
model relays the payload	0.700 ± 0.037	330.7 ± 20.8	0.705 ± 0.054	0.689 ± 0.024
model copies the result	0.763 ± 0.036	241.1 ± 49.7	0.695 ± 0.020	0.922 ± 0.167
without the runtime-owned route	0.672 ± 0.038			
Qwen3.8-27B				
Never expose	0.652 ± 0.057	807.2 ± 215.9	0.857 ± 0.108	0.172 ± 0.063
Always expose	0.905 ± 0.087	662.2 ± 188.5	0.895 ± 0.084	0.928 ± 0.120
Strongest baseline (forced schema)	0.933 ± 0.031	695.3 ± 463.1	0.919 ± 0.045	0.967 ± 0.000
AGENTRT	0.890 ± 0.076	372.1 ± 142.8	0.843 ± 0.108	1.000 ± 0.000
model relays the payload	0.857 ± 0.076	441.7 ± 144.5	0.845 ± 0.108	0.883 ± 0.000
model copies the result	0.890 ± 0.045	401.9 ± 100.2	0.843 ± 0.064	1.000 ± 0.000
without the runtime-owned route	0.770 ± 0.078			

did exactly that, recognizing none of ten paraphrases. This appendix describes the replacement parsers, the discipline under which their coverage was measured, and the per-class results behind the numbers of §5.3.

What the parsers do. The exact-computation parser normalizes notation first. Unicode superscripts and subscripts, LaTeX macros and matrix environments, MATLAB-style matrices, Python

Table 6: What admission is worth on each served model, three draws. The strongest exposing baseline is chosen by mean accuracy. Headroom is K/n of Proposition 2 against that baseline, the share of requests it breaks, given as the range over draws, and the ceiling is $(S_0 + R)/n$. Margin is AGENTRT's accuracy minus the baseline's, in points. Energy is joules per request, draw means.

Model	Strongest exposing baseline	Headroom	Ceiling	Margin	J/request		
					never	baseline	AGENTRT
Qwen2.5-7B	unforced schema (0.662)	0.07 to 0.10	0.748	+11.0	64	64	42
Llama-3.1-8B	unforced schema (0.533)	0.13 to 0.17	0.685	+11.2	53	43	34
Qwen3-8B	unforced schema (0.767)	0.04 to 0.07	0.822	+1.5	67	83	42
Mistral-7B-v0.3	Self-Debugging (0.125)	0.08 to 0.13	0.227	+30.2	57	56	39
Qwen2.5-32B	forced schema (0.793)	0.04 to 0.07	0.843	−1.0	256	202	165
Qwen3.8-27B	forced schema (0.933)	0.01 to 0.03	0.950	−4.3	525	649	330

operators, and English phrasings of powers and moduli are rewritten to one canonical form. It then extracts the mathematical objects present, namely matrix literals, fraction terms, binomial terms, summation bounds, recurrence definitions and integer lists, and decides the operation from the objects plus a small lexicon of operation words. The file/log parser takes the operation class from the fenced artifact the request carries, from its language tag or from its shape when the tag is missing. It grounds every argument in the artifact's own values: a service name that occurs in the log lines, a region value that occurs in the records, a column header that occurs in the CSV, and a section and key that occur in the config. Both fail closed. A request whose objects do not determine one operation, or whose argument the artifact cannot ground, falls through to ordinary admission. A successful parse admits the request, which is the deterministic override of §4.1.

Discipline. Coverage was measured on paraphrase banks written by hand. A development bank was used to build the parsers, the parsers were then frozen, and a held-out bank with different vocabulary and structure was written afterwards and never used to change them. Several held-out templates use phrasings the parsers were not expected to handle, so that coverage is measured, not engineered. The ten paraphrases of the original probe, written before this work, are kept as a pre-registered case. A third source of wording is the served model itself, which rewrote every request three times under instructions to preserve every number and name. Rewrites that drop an integer literal or the named entity are discarded mechanically and the rest are kept as written, so this set is not curated. Every held-out request is rendered from the same plan as the original, so it has identical parameters and an identical answer, and a synthesized program is counted as covered only if it executes to that answer.

Results. Table 8 gives the per-class numbers. On the generator's wording both parsers remain at full coverage. On the pre-registered ten the exact-computation parser recognizes 10 of 10, where the first version recognized none. On the held-out bank it recognizes 762 of 1,104 renderings with no wrong answer. That coverage is 69%, and the admission gate or the override admits 97.8% of the recognized requests. The file/log parser answers 540 of 600, which is 90% coverage. Its 20 wrong answers all come from one template in which a product value named `compute` collides with the verb of the question. On the model-written rewrites the exact-computation parser recognizes 486 of 540, again 90% with no wrong answer, and the file/log parser answers 289 of 290. The model's rewrites stay closer to the original than the held-out templates do, since they mostly change notation and word order, which is why they sit between the development and held-out figures. Replaying the measured draws through the new parser changes the routing of no synthetic request and of exactly one Math500 request per draw, a genuine greatest-common-divisor problem that the parser now solves correctly, so the numbers of Table 1 stand as measured up to that request.

End to end. Table 7 re-runs every policy on the held-out paraphrased draws, and Figure 4c plots the comparison. On those draws the exact-computation parser fires on 39 to 42 of the 60 synthetic requests, and on the model-written rewrites on 52 to 55. The AGENTRT rows of Table 1 and Table 7 were measured with the new parser. On the paraphrased file/log workload, re-run with all seven arms at the 8,192-token context of the original measurement, the runtime-owned route reaches 0.810 at 65.7 J per solved task against 0.320 at 526.9 J for the strongest other arm, the unforced schema call, and on the generator's own wording 0.740 at 55.5 J against 0.280 at 401 J. The parser's offline answer is right on 92 of the 100 paraphrased questions, and the model reproduces it on 81. The phrasings the held-out bank exposed as uncovered are instructive: a modular power stated as "base

Table 7: Held-out paraphrased workload, the draws of Table 1 with the sixty synthetic requests per draw re-worded from the held-out bank (Math500 half, parameters and answers unchanged). All rows come from one campaign on a second GB10 unit whose board power is 5 to 9% lower, so energy compares within this table only. The last row uses rewrites written by the served model.

Policy	Accuracy	J/solved	No tool	Tool
Never expose	0.487 ± 0.026	121.1 ± 11.4	0.662 ± 0.054	0.078 ± 0.104
Always expose	0.497 ± 0.057	117.9 ± 31.8	0.498 ± 0.121	0.494 ± 0.104
ReAct (Yao et al., 2023)	0.553 ± 0.094	94.9 ± 19.5	0.593 ± 0.141	0.461 ± 0.024
Few-shot prompting	0.600 ± 0.062	90.3 ± 14.0	0.619 ± 0.057	0.556 ± 0.133
Self-Debugging-style (Chen et al., 2024b)	0.612 ± 0.061	94.9 ± 30.2	0.621 ± 0.053	0.589 ± 0.104
Confidence gate (Su et al., 2024)	0.557 ± 0.068	172.9 ± 18.4	0.581 ± 0.089	0.500 ± 0.041
Self-routing	0.562 ± 0.104	98.3 ± 26.7	0.667 ± 0.088	0.317 ± 0.149
Forced schema-constrained call	0.587 ± 0.162	112.1 ± 37.3	0.519 ± 0.138	0.744 ± 0.316
PruneTIR-style (Zhang et al., 2026)	0.542 ± 0.101	97.0 ± 16.1	0.579 ± 0.108	0.456 ± 0.104
Schema-constrained, unforced	0.638 ± 0.100	97.1 ± 16.0	0.621 ± 0.108	0.678 ± 0.157
AGENTRT	0.713 ± 0.019	59.2 ± 2.5	0.671 ± 0.053	0.811 ± 0.157
without the runtime-owned route	0.587 ± 0.014	87.0 ± 8.1	0.671 ± 0.031	0.389 ± 0.104
on model-written rewrites	0.760 ± 0.054	51.9 ± 7.0	0.681 ± 0.051	0.944 ± 0.063

b, exponent e, modulus m" with no operator; a binomial sum stated as "n choose 0 through n choose k"; a matrix given as prose rows; an entry of a product written $(AB)_{rc}$ without a comma or as `C[r][c]`; a greatest common divisor asked for as "the largest positive integer dividing each of" or "greatest common factor"; a nested `gcd(a, gcd(b, c))`; a summation index that "ranges over" its values; and, for the file/log parser, a question that asks which team "leads" without naming an extremum, or which team "failed the most" without naming the column.

Table 8: Parser coverage by request class. A rendering counts as covered only if the synthesized program executes to the plan's answer (exact computation) or the parser's boxed answer equals the ground truth (file/log). Held-out templates were written after the parsers were frozen. Model-written rewrites are by the served Qwen2.5-7B-Instruct, three per request, counting only rewrites that preserve every integer literal (all 540) or the named entity (290 of 300).

Class	Generator	Development	Held-out	Model-written
Modular power	24/24	192/192	72/168	45/72
Binomial prefix sum	24/24	192/192	96/144	72/72
Determinant	24/24	192/192	120/144	72/72
Matrix product entry	24/24	168/168	96/144	59/72
Linear recurrence	21/21	147/147	126/126	63/63
Greatest common divisor	21/21	147/147	42/126	62/63
Fraction sum	21/21	147/147	105/126	63/63
Modular polynomial sum	21/21	147/147	105/126	50/63
exact computation, total	180/180	1332/1332	762/1104	486/540
Log error count	20/20	120/120	120/120	60/60
JSON filtered sum	20/20	120/120	100/120	50/50
CSV grouped maximum	20/20	120/120	80/120	59/60
Config lookup	20/20	120/120	120/120	60/60
Code output	20/20	120/120	120/120	60/60
file/log, total	100/100	600/600	540/600	289/290

C.4 GENERALITY

Repeating the full ten-policy matrix on Qwen2.5-1.5B, on the same workload draw and the same hardware, widens the gap instead of closing it. AGENTRT reaches 0.550 against 0.410 for the strongest baseline and 0.340 for never-expose, at 429 prefill-token-equivalents against 956. The ordering of the baselines is not preserved across model sizes. The forced schema-constrained row falls from 0.622 to 0.160, but that is mostly the forcing and not the schema, since compelling a call on every request sends the smaller model into the regime it is worst at. Unforced, the same

schema constraint reaches 0.290 at 1.5B, better than the forced version by 13 points and still below never-expose at 0.340. That is the exposure result again at a smaller scale. On this model, offering a schema at all costs more than it returns, whether or not the call is compelled. A smaller model is worse at producing a well-formed payload, so more of the accuracy depends on who constructs it.

The payload repair is the clearest case of a mechanism whose value is model-dependent. It never fires on the 7B mixed workload, because admission has already removed the requests that would produce unprintable code. At 1.5B it fires, and the failure it repairs is everywhere. Under always-expose 67 of 200 requests emit code that prints nothing, and 161 do so under the forced schema-constrained call, against 2 under AGENTRT, both of which are repaired. Empty observations follow the same order, at 67, 190 and 0.

Concurrency. Appendix A.3 characterizes admission as a knapsack once the accelerator is saturated, and says that regime should appear under concurrency. We swept one, four and eight concurrent requests on 60-request draws and did not reach it. Throughput scales almost linearly, with wall-clock falling 8.4× from concurrency one to eight, and mean board power is flat at 40 to 42 W throughout, so the accelerator is not saturated at eight and exposure still shortens the average request instead of lengthening the queue. Always-expose finishes faster than never-expose at every concurrency we ran. Energy per solved task falls for every policy as concurrency rises and the ordering does not change. At concurrency eight it is 184 J for AGENTRT against 281 for always-expose and 387 for never-expose, the same roughly two-to-one ratio as at concurrency one. So the budgeted characterization remains a statement about a regime outside these measurements.

C.5 THE WHEN2TOOL COMPARISON

When2Tool (Sun et al., 2026) is built for the necessity question. It has 15 sandboxed tool environments at three difficulty levels, a task is labelled tool-necessary when the no-tool arm fails it, and Probe&Prefill reads that label off hidden states at AUROC 0.89 to 0.96. By Proposition 1, such a probe and a resource-aware runtime are complementary, since each supplies a different half of the same decision. The probe estimates whether a request needs a tool, and the runtime decides whether serving can afford one at the moment the request arrives.

We run AGENTRT inside the upstream harness as a controller, so that every policy shares one evaluation loop and one task file, a stratified 750-task subset with 250 per difficulty across all 15 environments, and differences are attributable to the policy and not to bookkeeping. Two design points matter for the comparison to mean anything. First, the combined arm must differ from Probe&Prefill in one respect only. That method enforces its verdict by prefilling the assistant turn while leaving the tool schema exposed, so the model may disregard the steer, whereas the runtime's own denial is binding. The combined arm therefore enforces the probe's verdict with the probe's mechanism and reserves revocation for the runtime's own refusals, and we verify that the exported probabilities reproduce Probe&Prefill's own decisions exactly at every threshold. Second, the benchmark supplies its own cost metric, decoded tokens plus a fifth of prompt tokens, and we report on that metric instead of one of our own.

Pressure and tie-breaking. Pressure is read as the fraction of requests the server cannot afford, realized as a quantile of the affordability score on the training split. On this benchmark 59% of requests score identically, so the raw quantile is degenerate over half the range and admission breaks ties by a deterministic hash of the request, which is the correct greedy choice among requests the gate cannot separate. Three pressure settings resolved to the same policy before the tie-break was added, which gives a noise-floor measurement. The same policy run three times scores 0.8813, 0.8773 and 0.8800, a standard deviation of 0.0020, because batched decoding is not bitwise deterministic even at temperature zero. Differences below half a point on this benchmark are noise.

The composition fails for lack of a signal. Proposition 1 says a necessity probe and a resource-aware runtime answer different questions and so should compose. On this benchmark they do not, because one signal is unavailable. The gate cannot separate this workload. Scoring every request with the same gate the runtime uses, the mixed workload spreads over ten or eleven distinct values with no value holding more than 21% of requests, while on When2Tool 59% of requests land on a single value, in the calibration and the evaluation split alike. Admission above zero pressure is

Table 9: When2Tool, Qwen3-4B, stratified 750-task subset, one run per arm. Token cost is the benchmark's own metric: decoded tokens plus a fifth of prompt tokens. *Admitted* counts requests the runtime exposed tools to; Probe&Prefill exposes all 750 and steers with a prefill instead, and the combined arms enforce the probe's verdict the same way unless marked revoked.

Policy	Accuracy	Tool calls	Token cost	Admitted
Probe&Prefill, τ=0.3	0.896	473	102,727	—
Probe&Prefill, τ=0.5	0.887	443	120,205	—
Probe&Prefill, τ=0.7	0.885	406	122,363	—
Probe&Prefill, τ=0.9	0.877	350	119,631	—
AGENTRT, p=0.0	0.879	710	84,021	750
AGENTRT, p=0.25	0.815	576	95,316	590
AGENTRT, p=0.5	0.740	386	109,142	395
AGENTRT, p=0.75	0.669	186	164,404	183
AGENTRT, p=0.9	0.631	81	248,568	80
AGENTRT, p=1.0	0.564	0	330,651	0
+ probe τ=0.5 (revoked), p=0.0	0.859	376	127,432	386
+ probe τ=0.5 (steered), p=0.0	0.879	444	117,679	750
+ probe τ=0.5 (steered), p=0.5	0.729	227	123,264	395
+ probe τ=0.5 (steered), p=0.75	0.679	133	169,432	183
+ probe τ=0.3 (steered), p=0.0	0.893	480	106,647	750
+ probe τ=0.7 (steered), p=0.0	0.875	406	111,323	750

therefore mostly choosing among requests it considers identical. At pressures 0.25 and 0.5, every one of the 160 and 355 denials falls inside that tie group, and at 0.75, 446 of 567 do. The accuracy it gives up tracks random denial.

The composition follows from that, and Table 9 shows it. At zero pressure, where admission admits everything, the combined arm reproduces Probe&Prefill at every threshold we swept. It scores 0.893 against 0.896 for τ=0.3, 0.879 against 0.887 for 0.5 and 0.875 against 0.885 for 0.7, with tool-call counts matching to within seven, one and zero. That residual is the output guard and the tool-turn budget, not the gate. Turning the gate up makes both arms worse than either alone, at 0.740 for admission by itself at half pressure and 0.729 for the two together. A signal the gate cannot compute is not information that the gate can contribute.

One arm does carry over, and it is the one with no admission decision in it. Running the runtime at zero pressure without the probe reaches 0.879, within two points of the best probe setting, at 84,021 token cost against that setting's 102,727, which is 127 per correct answer against 153. It does so while making 710 tool calls, more than any probe setting, which is the same two-sided fact §5.2 reports on a different workload. Calls are cheap and decode is not, so steering a model away from a tool it would have used converts a cheap cost into an expensive one. This is a negative result for the gate on this benchmark, not for the framing. It also sharpens what §5.2 shows, since there the same gate does separate requests and admission is worth 18 points on the tool-negative half. The contribution that transfers is where the decision is taken and what the runtime does with the trajectory afterwards, not the discriminative power of this particular scorer, which is the same conclusion the out-of-distribution gate result reaches in §5.4.

The last block of Table 9 isolates the enforcement mechanism. Enforcing the probe's verdict by withdrawing the tool schema instead of by steering, at the same threshold and the same zero pressure, costs 0.020 accuracy and 68 tool calls, so the gap between the revoked and the steered row is the enforcement mechanism and not composition. Every arm here is a single run; the three pressure settings that resolved to one policy before the tie-break was added give the noise floor quoted above.

C.6 COVERAGE ON EXTERNAL WORKLOADS

Both parsers were frozen at the commit that introduced them, before any corpus in this section was inspected, and the census that follows ran them unchanged. The corpora were chosen by a rule fixed in advance. Each is public, was not constructed by us, is scored by an exact answer where one exists, was released before this work began, and was never consulted while the parsers were

Table 10: Coverage census of the frozen parsers. For each corpus, how often the exact-computation parser and the file/log parser fire and take over a request (a takeover is a parse that executes), how many takeovers answer correctly and wrongly where ground truth exists, and how many requests the admission gate of the main configuration would expose. Every workload of this paper is listed for reference. Fires of the code-output category on foreign prompts were counted and not executed.

		Exact computation		File/log		
Corpus	n	takeover	right/wrong	takeover	right/wrong	Gate exposes
Workloads of this paper						
Mixed workload, draw 1	200	61 (30.5%)	61/0	0 (0.0%)	0/0	67 (33.5%)
Mixed workload, held-out wording	200	40 (20.0%)	40/0	0 (0.0%)	0/0	61 (30.5%)
File/log workload	100	0 (0.0%)	0/0	100 (100.0%)	100/0	20 (20.0%)
File/log workload, held-out wording	100	0 (0.0%)	0/0	97 (97.0%)	92/5	20 (20.0%)
TableQA workload	100	0 (0.0%)	0/0	0 (0.0%)	0/0	20 (20.0%)
API workload	100	0 (0.0%)	0/0	0 (0.0%)	0/0	0 (0.0%)
External benchmarks						
Math500	500	1 (0.2%)	1/0	0 (0.0%)	0/0	14 (2.8%)
AIME 2024	30	0 (0.0%)	0/0	0 (0.0%)	0/0	0 (0.0%)
DeepMind Mathematics, 56 modules	5,600	51 (0.9%)	51/0	0 (0.0%)	0/0	672 (12.0%)
ToolQA, questions only	1,530	0 (0.0%)	0/0	0 (0.0%)	0/0	10 (0.7%)
ToolQA, with a table sample	1,230	0 (0.0%)	0/0	0 (0.0%)	0/0	831 (67.6%)
TabMWP test	1,000	0 (0.0%)	0/0	27 (2.7%)	24/3	0 (0.0%)
WikiTableQuestions dev	2,563	0 (0.0%)	0/0	30 (1.2%)	13/17	151 (5.9%)
Instruction data and traffic proxies						
WebInstruct	200	0 (0.0%)	0/0	0 (0.0%)	0/0	0 (0.0%)
SimpleQA	200	0 (0.0%)	0/0	0 (0.0%)	0/0	0 (0.0%)
ShareGPT, first user turns	20,000	2 (0.0%)	-	0 (0.0%)	-	590 (3.0%)
OASST1, root prompts	10,364	0 (0.0%)	-	0 (0.0%)	-	8 (0.1%)

developed. Sampling is seeded and uniform over a corpus's own modules or splits, so no module and no question was selected by how the parsers behave on it. The phrasings the census found uncovered are listed below and were not used to change either parser.

Corpora. DeepMind Mathematics (Saxton et al., 2019) contributes its interpolation test split, 100 questions drawn from each of its 56 modules. ToolQA (Zhuang et al., 2023) contributes every easy and hard question, once as a bare question and once with a seeded sample of 30 rows of the dataset's table attached as a fenced block. The sample is drawn independently of the question and is the artifact form the file/log parser expects. TabMWP (Lu et al., 2023) and the development split of WikiTableQuestions (Pasupat & Liang, 2015) carry their tables as fenced CSV, keeping tables of at most 60 rows in the latter. First user turns of ShareGPT (Zheng et al., 2023) and root prompts of OASST1 (Köpf et al., 2023) stand in for real traffic and carry no ground truth. Every request runs through the exact-computation parser, through the file/log parser and through the admission gate of the main configuration. A synthesized program is executed in the runtime's sandbox, and the code-output category of the file/log parser is counted but not executed on foreign prompts.

Results. Table 10 gives every corpus and Table 11 every module. The ten external corpora hold 43,217 requests, of which the two traffic proxies contribute 30,364, and the two parsers take over 0.26% of them. The exact-computation parser answers every takeover that has a ground truth correctly. On DeepMind Mathematics all of its takeovers are greatest-common-divisor questions. The other half of that module asks for the greatest common factor or the highest common divisor, and its lexicon lacks both phrasings. The gate exposes 190 of the 200 questions in the two remainder modules, but the parser takes over none of them because it recognizes a remainder only under a power. The file/log parser's wrong answers all fall on open-domain tables where the question names an extremum but the parser grounds the wrong column and returns a name where a count was asked or a value where a name was asked. On ToolQA it fails closed on every request that carries a table sample, most often because it cannot ground the columns the question refers to. On the two traffic proxies it takes over nothing and the exact-computation parser takes over two ShareGPT exercises in which a greatest common divisor is a step of an RSA problem and not the answer, so both takeovers would have answered the wrong question. The gate's exposure rate on the ToolQA requests that carry a table sample is high because a table of numbers scores as exact-computation evidence.

End to end. Table 12 runs four arms of the main comparison on three seeded draws of 300 requests from DeepMind Mathematics, uniform over its modules, and from TabMWP, under the settings of

Table 11: DeepMind Mathematics, interpolate test split, 100 questions sampled uniformly from each of the 56 modules. Runtime-owned is the share the frozen exact-computation parser takes over (every takeover answered correctly), and exposed is the share the admission gate would expose a tool to, including the parser's override.

Module	Runtime-owned	Exposed	Module	Runtime-owned	Exposed
algebra__linear_1d	0	0	measurement__time	0	0
algebra__linear_1d_composed	0	0	numbers__base_conversion	0	0
algebra__linear_2d	0	0	numbers__div_remainder	0	90
algebra__linear_2d_composed	0	0	numbers__div_remainder_composed	0	100
algebra__polynomial_roots	0	55	numbers__gcd	51	55
algebra__polynomial_roots_composed	0	12	numbers__gcd_composed	0	0
algebra__sequence_next_term	0	8	numbers__is_factor	0	11
algebra__sequence_nth_term	0	6	numbers__is_factor_composed	0	0
arithmetic__add_or_sub	0	7	numbers__is_prime	0	0
arithmetic__add_or_sub_in_base	0	6	numbers__is_prime_composed	0	1
arithmetic__add_sub_multiple	0	0	numbers__lcm	0	29
arithmetic__div	0	72	numbers__lcm_composed	0	28
arithmetic__mixed	0	15	numbers__list_prime_factors	0	0
arithmetic__mul	0	0	numbers__list_prime_factors_composed	0	2
arithmetic__mul_div_multiple	0	8	numbers__place_value	0	0
arithmetic__nearest_integer_root	0	38	numbers__place_value_composed	0	1
arithmetic__simplify_surd	0	0	numbers__round_number	0	4
calculus__differentiate	0	24	numbers__round_number_composed	0	16
calculus__differentiate_composed	0	3	polynomials__add	0	1
comparison__closest	0	0	polynomials__coefficient_named	0	0
comparison__closest_composed	0	22	polynomials__collect	0	4
comparison__kth_biggest	0	2	polynomials__compose	0	1
comparison__kth_biggest_composed	0	15	polynomials__evaluate	0	0
comparison__pair	0	21	polynomials__evaluate_composed	0	0
comparison__pair_composed	0	10	polynomials__expand	0	0
comparison__sort	0	0	polynomials__simplify_power	0	0
comparison__sort_composed	0	5	probability__swr_p_level_set	0	0
measurement__conversion	0	0	probability__swr_p_sequence	0	0

Table 12: End to end on two external workloads, three seeded draws of 300 requests each, mean $\pm$ half-width of a Student-t 95% interval, under the settings of Table 1. Calls are model-issued tool calls per request. Each block ends with the takeover count of the frozen parsers over the three draws and how many of those takeovers the parser answered correctly.

Policy	Accuracy	J/solved	Calls/req
DeepMind Mathematics, 300 questions per draw, 56 modules			
Never expose	0.632 ± 0.063	71.5 ± 8.1	0.00 ± 0.00
Always expose	0.460 ± 0.000	80.1 ± 22.6	0.74 ± 0.10
Unforced schema	0.588 ± 0.042	85.4 ± 43.4	1.72 ± 0.12
AGENTRT	0.639 ± 0.041	66.0 ± 5.9	0.08 ± 0.03
taken over: 6 of 900, parser right on 6			
TabMWP, 300 questions per draw			
Never expose	0.950 ± 0.008	24.0 ± 1.6	0.00 ± 0.00
Always expose	0.276 ± 0.029	103.8 ± 43.0	0.41 ± 0.07
Unforced schema	0.850 ± 0.093	48.2 ± 5.1	1.52 ± 0.18
AGENTRT	0.951 ± 0.017	22.5 ± 1.7	0.00 ± 0.00
with the file/log parser	0.942 ± 0.005	23.4 ± 1.8	0.00 ± 0.00
taken over: 29 of 900, parser right on 27			

Table 1. The exposure cost recurs on both. Always-expose loses 17 points on DeepMind Mathematics and 67 on TabMWP, and the unforced schema call still loses 4 and 10. The mechanism is the malformed call of §5.3. Of the 486 and 652 requests always-expose fails on the two workloads, 356 and 588 end without a call the serving engine can parse, because the model writes the code argument with single quotes inside the JSON. Proposition 2 puts the headroom of admission over always-expose at 28 and 68 points and over the unforced call at 17 and 13, and AGENTRT collects it by withholding the schema. It matches never-expose on both workloads at the lowest energy per solved task, and it rescues and breaks requests in equal measure relative to never-expose, 43 against 37 on DeepMind Mathematics over the three draws, because the gate cannot tell the requests the unforced call would rescue from the rest, so the ceiling of 0.76 there stays out of reach. The

parsers take over 6 of 900 requests on the first workload, all correctly, and none on the second. The file/log parser arm, in which the runtime computes the answer and the model restates it, takes over 29 TabMWP requests. The parser is right on 27 and the model restates 5 of those wrongly, which is the relay loss of Appendix C.3 again, so that arm finishes one point below never-expose.

The scope of the guarantee. The runtime-owned route is a guarantee about the requests it takes over, not a claim about how many there are. Where a parser fires and executes, the payload is well formed by construction. How often that happens is a property of the workload, and it ranges from all of the file/log workload and 30% of the mixed workload down to under 1% of DeepMind Mathematics and effectively none of general traffic. The file/log parser's precision on tables it was not built for is 37 of 57, so on open-domain tables its extremum operation needs the fail-closed check that ToolQA exercised. Beyond a parser's reach the runtime is admission alone.

C.7 LIMITATIONS

Coverage is the sharpest limitation. The runtime-owned route recognizes 69% of held-out paraphrases of the exact-computation requests and 90% of the file/log requests, but on ten external corpora the frozen parsers take over 0.26% of requests, so runtime-owned execution is a guarantee about the requests it covers and not a share of a general workload, and outside a parser's reach the runtime is ordinary admission. Energy is GPU-domain on one platform, the cost-proxy inversion was swept on one model, the analysis treats exposure as binary, and the mixed-workload runs exercise the admission gate only at zero serving pressure, so the budgeted regime of Appendix A.3 remains unmeasured. Future work will extend both parsers to the phrasings that the held-out bank of Appendix C.3 exposed as uncovered.