TOKENCAST: FORECASTING TOKEN CONSUMPTION DURING LLM AGENT EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

When a large language model (LLM) agent executes the same task, token consumption can vary by over an order of magnitude across runs. The agent chooses its next steps based on tool feedback and intermediate results, while the growing context steadily inflates the input size of every subsequent call. The total consumption of a task is therefore hard to predict before execution and the prediction must be revised as the run unfolds. In this paper, we propose TokenCast, which learns a composable cost representation for each execution segment, recording its own consumption and the context growth it introduces. Composing adjacent segments yields a cumulative estimate that captures the extra input cost incurred when context from earlier segments is re-read by every later call. As execution unfolds, newly observed evidence refreshes the forecast, requiring no additional LLM calls and incurring a mean cumulative prediction time of 32.8 ms per run on SWE-bench Verified. Across 4 task suites and 6 agent models, TokenCast's mean absolute error reduction against the strongest comparator averages 14.5% over 96 evaluated combinations. In offline budget-control replay, TokenCast uses 21.3% fewer tokens on average than a fixed-budget policy at matched trace completion. Our implementation is available at `https://anonymous.4open.science/r/TokenCast-1F8B`.

1 INTRODUCTION

The applications of large language models (LLMs) are expanding from simple question answering to complex tasks such as software engineering and deep research. Completing these tasks typically relies on LLM-based agents that repeatedly plan, modify code, invoke tools, and verify results (Yao et al., 2023; Yang et al., 2024; Wang et al., 2025). A given task may be resolved in a single edit or may require multiple rounds of retries before converging, and the dialogue and tool outputs generated in each round accumulate into the context of subsequent requests (Yang et al., 2024; Xiao et al., 2026; Zhu et al., 2026). The final token consumption of a task is therefore difficult to predict before execution completes, making it hard for users to forecast costs and plan usage (Bai et al., 2026).

The challenge of predicting tokens lies in the fact that every action an agent takes can trigger new model calls, and consumption across steps is interdependent: the longer the context left by earlier steps, the larger the input to every subsequent call, causing consumption to accumulate and amplify (Salim et al., 2026; Zhu et al., 2026; Xiao et al., 2026). The model's generative behavior further compounds the difficulty. Given the same input, the model may choose different courses of action, generate outputs of different lengths, and consequently undergo different numbers of verification or retry cycles. In practice, token consumption across different executions of the same task can differ by up to 30$\times$ (Bai et al., 2026), making accurate prediction challenging.

Existing work can be organized by prediction scope and observation time. The scope ranges from a single response to an entire agent task; the forecast can be made before execution or updated during it. These dimensions give four settings (Figure 1): response length before generation, remaining response length during generation, total task consumption before execution, and remaining task consumption as an agent runs. Prior methods address these settings with different targets and access assumptions (Shahout et al., 2025; Xie et al., 2026; Bai et al., 2026).

At the request level, prior work studies how to predict the response length of a single model call. Because the input is known before generation begins, these methods typically estimate length from

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.

input features for resource allocation (Jin et al., 2023; Qiu et al., 2024; Fu et al., 2024; Zheng et al., 2026), while some also refine the estimate during generation using intermediate states or entropy statistics (Shahout et al., 2025; Xie et al., 2026; Merzouk et al., 2026).

At the agent-task level, prior work estimates total token consumption before execution, either from the task description or through the model's own cost assessment (Bai et al., 2026). Multi-step LLM execution frameworks organize or schedule requests according to program structures, semantic dependencies, or workflow paths available before the corresponding requests are executed (Khattab et al., 2023; Lin et al., 2024; Zheng et al., 2024; Ni et al., 2026; Yu et al., 2026). Open-ended agents present a different situation: their behavior depends on tool feedback, environment state, and intermediate results, and no complete dependency graph exists before execution begins (Yao et al., 2023; Yang et al., 2024; Wang et al., 2025).

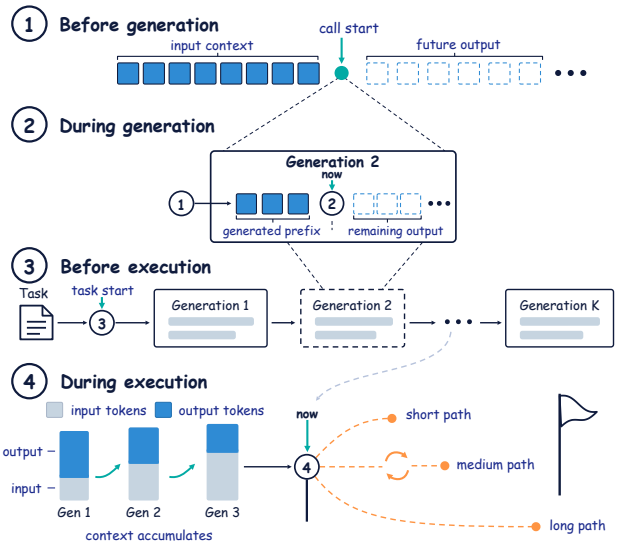


Figure 1: Four token-consumption prediction settings. At the call level, the input context supports a forecast before generation (1), and the output prefix updates it during generation (2). At the task level, total consumption is forecast before execution (3), and remaining consumption is updated as calls complete and context accumulates (4).

Multi-step forecasting has also been studied in structured workflows (Ni et al., 2026). We focus on remaining provider-accounted input and output consumption in sequential agent execution. Each future request can bill retained context again, coupling the remaining cost to both future call count and input lengths. TokenCast updates this forecast from observed execution information without querying an LLM for the cost estimate.

TokenCast represents each execution segment by call count, net input-length change, and a cost residual. An exact composition identity exposes how context growth in one segment affects the cost baseline of later calls. The task predictor combines a direct forecast with a prefix–suffix forecast conditioned on a predicted boundary state, and refreshes its features as execution proceeds. Calibrated quantile models provide prediction intervals. The predictor makes no additional LLM calls.

Our contributions are as follows:

- We introduce a segment-cost factorization and an exact composition identity that accounts for repeated input consumption across calls.
- We use this identity in a staged prefix–suffix predictor that combines direct and compositional forecasts and updates from execution evidence without additional LLM calls.
- Across four benchmarks and six agent models, TokenCast's MAE reduction averages 14.5% across 96 comparisons with the strongest comparator in each. Budget-control replay saves 21.3% of tokens at matched trace completion.

Figure 2 shows how observed execution and unresolved future segments support the four settings.

2 Related Work

Request-level output length prediction. Early methods extract features from the prompt to estimate response length for memory reservation, batching, or shortest-job-first scheduling (Jin et al., 2023; Qiu et al., 2024). Prompt-only estimates remain highly uncertain, and prompt-conditioned response lengths can exhibit broad or heavy-tailed distributions (Perez-Ramirez et al., 2025; Wang

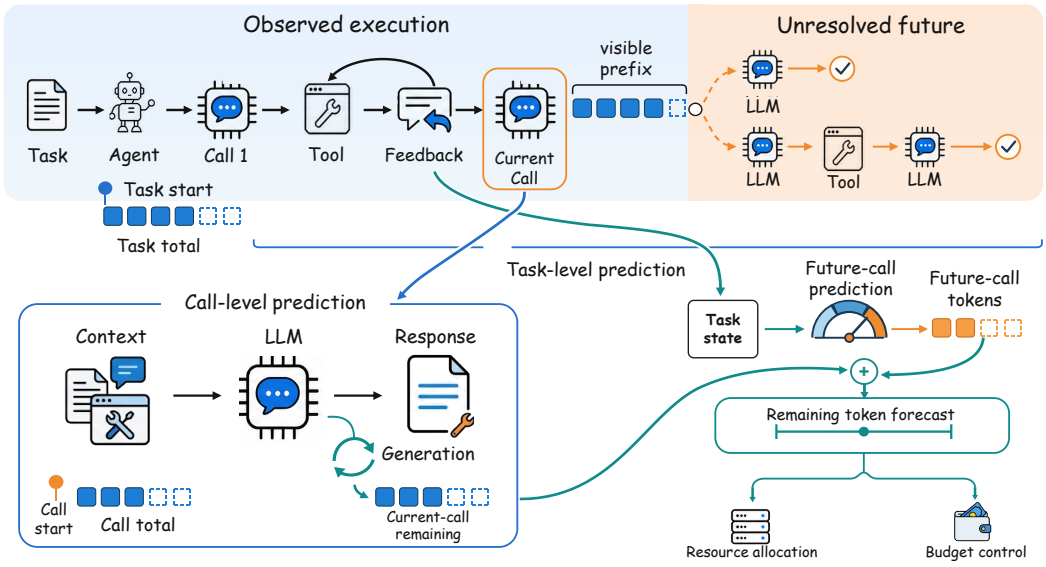


Figure 2: Overview of TokenCast. Observed execution evidence supports call- and task-level forecasts, while the compositional path propagates predicted context growth into later-call costs.

et al., 2026). Subsequent work therefore models more than a single point estimate: one line of work learns pairwise length orderings among requests to set scheduling priorities (Fu et al., 2024), and another fits a heavy-tailed log-t distribution that lets the scheduler trade off average and tail latency (Zheng et al., 2026). Once generation begins, the decoder's own intermediate states become available. Several recent studies show that mid-layer representations or entropy statistics can continuously refine the remaining-length estimate during decoding, narrowing the gap between the initial guess and the actual length (Shahout et al., 2025; Merzouk et al., 2026; Xie et al., 2026). These methods share a common set of assumptions: the prediction target is a single response, the prompt is known, and the system has access to the prompt or to model internals. In agent tasks, later requests do not exist until earlier actions and tool calls complete, so none of these assumptions holds.

Task-level and multi-step prediction. Prior work has extended prediction to entire tasks. Self-Prediction has a coding agent inspect its environment before execution and estimate input, output, and total token consumption by stage (Bai et al., 2026). DSPy (Khattab et al., 2023), Parrot (Lin et al., 2024), and SGLang (Zheng et al., 2024) represent multi-step LLM applications through program structures or semantic dependencies available to serving runtimes. Open-ended agents expose no such structure before execution.

Chimera predicts remaining workflow output with a CPU-based quantile random forest (Ni et al., 2026). Pythia profiles historical traces to infer likely workflow paths and role-level output lengths (Yu et al., 2026). TokenCast's segment identity addresses the repeated input cost induced by carried context, and its forecasts use no additional LLM calls.

Trace studies of deployed agents report extensive iterative review loops in multi-agent software pipelines, and long contexts with short outputs and heavy prefix reuse in real coding-agent sessions (Salim et al., 2026; Zhu et al., 2026). Broader analyses organize token use and efficiency across single-agent, multi-agent, and agent-ecosystem settings (Chen et al., 2026). These studies characterize consumption after the fact and do not forecast it for a running task.

3 Method

3.1 Problem Formulation

An agent executes a task through a sequence of LLM calls interleaved with tool interactions. Let C_k denote the provider-accounted input and output token consumption of call k. A task terminates upon completion, failure, or when an execution limit is reached. For a task with K calls, its total con-

sumption is $T = \sum_{j=1}^{K} C_j$. After call k completes, the confirmed consumption is $S_k = \sum_{j=1}^{k} C_j$, and the remaining consumption is $R_k = T - S_k$. Each forecast uses only the task and execution information available at its prediction point.

We consider four prediction settings along the execution trajectory.

- **Task Start** predicts the total consumption T before execution begins.
- **Call Start** predicts the consumption C_k of the current call after its request has been assembled and before any output token is generated.
- **In-call Update** updates the prediction of C_k as the generated prefix becomes available.
- **Task Update** predicts the remaining consumption R_k after call k completes, yielding an updated forecast of the task total, $\widehat{T}_k = S_k + \widehat{R}_k$.

3.2 CALL–TASK FORECASTING

TokenCast represents segment cost relative to its starting input length. The next segment inherits the context produced by the preceding one, yielding an exact composition rule for adjacent segments.

Segment representation. A contiguous block of one or more calls forms a segment. Let segment A start with input length L_A, span n_A calls, and consume C_A tokens in total. Its representation is $\phi_A = (n_A, g_A, b_A)$, where g_A is the net change in input length across the segment and $b_A = C_A - n_A L_A$ is the residual after subtracting the starting-input baseline $n_A L_A$, covering generation and within-segment context growth. The next segment starts with input length $L_A + g_A$. Reasoning tokens billed by the provider contribute to b_A; when they do not persist in the conversation context, they do not contribute to the context change g_A.

When segment B immediately follows A, the combined representation is

$$\phi_{A \circ B} = \big(n_A + n_B,\ g_A + g_B,\ b_A + b_B + n_B\, g_A\big). \tag{1}$$

This identity follows from the definitions and the boundary condition $L_B = L_A + g_A$. The third term $n_B\, g_A$ arises from realigning the baseline: b_B is defined relative to B's actual starting point $L_A + g_A$, whereas the combined residual is relative to L_A. The difference on each call is exactly g_A, and n_B calls accumulate to $n_B\, g_A$.

Forecasting with composition. The decomposition converts aggregate remaining-cost prediction into three sub-problems: the prefix segment's representation, the boundary state, and the suffix segment's representation. TokenCast fits LightGBM models for these predictions and updates after completed calls. Appendix D.6 compares alternative base predictors. Input features are drawn from the task and execution information available at the prediction point. Appendix B.1 details these features and when each becomes available.

Call-level forecasts predict the current call's cost from the features visible at Call Start or In-call Update. Task-level forecasts maintain two paths. The direct path predicts remaining total cost as a single target. The compositional path separates the current or next segment from the subsequent suffix. Their cost representations are predicted separately and combined using Eq. (1).

The compositional path predicts the prefix representation $(\widehat{n}_A, \widehat{g}_A, \widehat{b}_A)$ and its ending state. The predicted context change sets the suffix input baseline, while the predicted ending state and features visible at the original prediction point condition the suffix model. The suffix representation is converted to a remaining-cost forecast through Eq. (1).

Updating forecasts. Each completed step during execution produces new observations, and TokenCast refreshes its forecasts accordingly. At Call Start, the current request has been assembled and the actual input length is known, so the call-level forecast can be based on it directly. At In-call Update, the committed generated prefix and streaming timing supply additional features for the current-call forecast. At Task Update, the completed call supplies confirmed cumulative cost S_k and any completed tool outcomes. The next request's input length remains predicted until that request is assembled. TokenCast uses the available evidence to re-predict $\widehat{R}_k$ and reports $\widehat{T}_k = S_k + \widehat{R}_k$.

3.3 COMPOSITIONAL LEARNING

Staged fitting. TokenCast fits the direct, prefix, and suffix predictors as separate LightGBM models using labels extracted from completed traces. The direct path predicts remaining total cost. The compositional path predicts the prefix and suffix representations together with the prefix-ending boundary variables. Prefix context-change and suffix call-count errors can affect downstream cost through the composition identity, motivating the following weights on their local absolute losses:

$$\ell_g = n_B \left| \widehat{g}_A - g_A \right|, \qquad \ell_n = L_B \left| \widehat{n}_B - n_B \right|. \tag{2}$$

Cross-fitting. The suffix model is trained on predicted prefix boundaries. Tasks are partitioned into F folds, and each fold's prefix predictions are generated by models trained on the remaining folds. The predicted context change sets the suffix input baseline, so its residual label is recomputed as $b_B^{\text{train}} = C_B - n_B \widehat{L}_B$, where $\widehat{L}_B = L_A + \widehat{g}_A^{\text{oof}}$. The loss weights in Eq. (2) are motivated by the composition identity; Appendix B.3 distinguishes this motivation from the error decomposition under the rebased suffix training label.

Correction model. After the direct, prefix, and suffix models are fixed, a correction model is trained on out-of-fold outputs of the complete forecasting pipeline:

$$\psi^* = \arg\min_{\psi} \sum_i \left| y_i - \widehat{C}_{\text{comp},i} - h_\psi(\mathbf{q}_i) \right|. \tag{3}$$

The input $\mathbf{q}_i$ contains the visible features, the direct and compositional forecasts, their difference, and the predicted boundary variables. The final task-level forecast is $\widehat{C}_{\text{comp}} + h_\psi(\mathbf{q})$.

Prediction intervals. Separate LightGBM quantile models produce the 0.05 and 0.95 endpoints at each prediction point. The endpoints are widened symmetrically by a quantile of interval residuals on held-out calibration tasks and are bounded below by consumption already confirmed within the prediction scope. Appendix B.4 gives the procedure.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Tasks and execution traces. We evaluate TokenCast on SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024), Search-R1 (Jin et al., 2025a), MMLU-Pro (Wang et al., 2024b), and LongBench-v2 (Bai et al., 2025), covering software engineering, retrieval-based question answering, knowledge-based reasoning, and long-context understanding. We collect 11,712 execution traces from 240 benchmark tasks with six agent LLMs: GPT-5.4 (OpenAI, 2026), Claude Opus 4.6 (Anthropic, 2026), Gemini 3.1 Pro (Google DeepMind, 2026), DeepSeek-V4-Pro (DeepSeek-AI, 2026), Qwen3.8-27B (Qwen Team, 2026), and Llama-3.2-3B-Instruct (Meta AI, 2024). For reproducibility, we specify *gpt-5.4-2026-03-05* for the GPT-5.4 API and *qwen3.8-27b-20260815* for the self-hosted Qwen checkpoint; Table 5 lists model access and reasoning configurations. Traces are collected using DeepSeek Harness (DeepSeek, 2026) and OpenHands (Wang et al., 2025). Repeated executions support the analysis of run-to-run variation, with additional repeats for anchor tasks. Table 2 specifies the collection design for each benchmark and harness. All runs of the same task remain in one partition. Training, validation, calibration, and test tasks are separated as described in Appendix C. Generalization experiments also use independently released trajectories (Appendix C.1).

Baselines. We compare TokenCast with three output-length predictors, TRAIL (Shahout et al., 2025), EGTP (Xie et al., 2026), and TIE (Zheng et al., 2026), and the agent-level consumption estimator Self-Prediction (Bai et al., 2026). We adapt these methods to the prediction targets and observations available at the four prediction points. At Task Start, Self-Prediction inspects the task environment before estimating total consumption. At the other prediction points, it uses the observed execution prefix. Appendix C.3 details each adaptation. Appendix D.6 compares direct regression, compositional forecasting, their average, and the full correction pipeline; Appendix D.6 evaluates the segment representation and composition procedure.

Metrics and implementation. We report mean absolute error (MAE) and weighted absolute percentage error (WAPE) at the four prediction points. At Call Start, the target includes the assembled

Table 1: MAE ↓ in tokens at four prediction points. Norm. Avg. is the macro-average of MAE normalized at each prediction point by the MAE of the corresponding history-median predictor. k denotes thousands of tokens. Best and second-best results are in bold and underlined, respectively. Marks are assigned using unrounded values.

Agent LLM	Prediction point	TokenCast (Ours)	TRAIL	EGTP	TIE	Self-Pred.
		SWE-bench Verified				
GPT-5.4	Task Start	**144.0k**	165.3k	160.9k	157.0k	<u>152.0k</u>
	Call Start	**64.5**	71.1	70.3	<u>69.5</u>	70.9
	In-call Update	**38.9**	78.9	<u>74.6</u>	77.4	80.2
	Task Update	**80.0k**	<u>115.0k</u>	117.0k	119.0k	126.0k
	Norm. Avg.	**0.69**	0.94	<u>0.92</u>	0.92	0.94
Qwen3.8-27B	Task Start	**192.0k**	232.0k	<u>209.0k</u>	215.0k	221.0k
	Call Start	**87.0**	109.0	102.9	<u>99.0</u>	105.5
	In-call Update	**81.9**	120.0	116.0	111.4	<u>108.9</u>
	Task Update	**124.0k**	176.0k	171.0k	<u>164.0k</u>	173.0k
	Norm. Avg.	**0.65**	0.86	0.81	<u>0.79</u>	0.82
		Search-R1				
GPT-5.4	Task Start	**34.2k**	37.8k	37.6k	<u>36.4k</u>	36.9k
	Call Start	<u>34.5</u>	37.3	36.5	**34.0**	39.8
	In-call Update	**31.3**	39.4	38.5	<u>36.8</u>	39.9
	Task Update	**17.6k**	<u>23.1k</u>	23.7k	23.3k	24.2k
	Norm. Avg.	**0.75**	0.89	0.88	<u>0.85</u>	0.91
Qwen3.8-27B	Task Start	**39.0k**	53.5k	49.8k	<u>47.6k</u>	51.0k
	Call Start	**56.7**	<u>65.9</u>	71.4	67.7	76.1
	In-call Update	**57.0**	84.6	80.7	<u>75.5</u>	83.2
	Task Update	**24.0k**	39.5k	<u>34.0k</u>	35.4k	38.4k
	Norm. Avg.	**0.59**	0.83	0.79	<u>0.77</u>	0.84

request's known input tokens. For 90% prediction intervals, we report empirical coverage, mean width, and mean interval score (MIS) (Gneiting & Raftery, 2007). Tasks receive equal weight, with that weight distributed across their runs and evaluated checkpoints. Cross-configuration summaries divide MAE by that of a history-median predictor, which outputs the median training target for the same benchmark, agent LLM, and prediction point. The local platform provides eight NVIDIA A100 GPUs. Internal-state baselines use the agent model when accessible and a proxy for API-based models. Appendix C gives metrics, data partitions, training settings, and timing procedures.

4.2 Experimental Results and Analysis

4.2.1 Main Results

Table 1 shows SWE-bench Verified and Search-R1 results for GPT-5.4 and Qwen3.8-27B; Figure 3 covers all four benchmarks and six agent LLMs. On SWE-bench Verified with GPT-5.4, TokenCast reduces MAE relative to the strongest comparator by 47.9% at In-call Update, from EGTP's 74.6 to 38.9 tokens, and by 30.4% at Task Update, from TRAIL's 115k to 80k tokens. At Task Start, the strongest comparator is Self-Prediction, and the reduction is 5.3%, from 152k to 144k tokens. Across the 96 benchmark–model–prediction-point combinations in Appendix D.2, its MAE reduction against the lowest comparator MAE averages 14.5%. The averages are -2.2% at Task Start, 1.9% at Call Start, 30.4% at In-call Update, and 27.8% at Task Update. TokenCast trails the strongest comparator in 24 combinations: 15 at Task Start and nine at Call Start. None of these losses occurs at In-call Update or Task Update, where execution evidence accumulates.

Error relative to consumption. WAPE complements MAE by expressing absolute error relative to mean target consumption. For GPT-5.4 at Task Start, TokenCast's MAE of 32.2k tokens on LongBench-v2 corresponds to a WAPE of 4.2%, whereas its MAE of 8.0k on MMLU-Pro corresponds to 15.8%. Their mean target consumptions are 771.2k and 50.6k tokens, respectively. Table 11 in Appendix D.3 reports the full WAPE results.

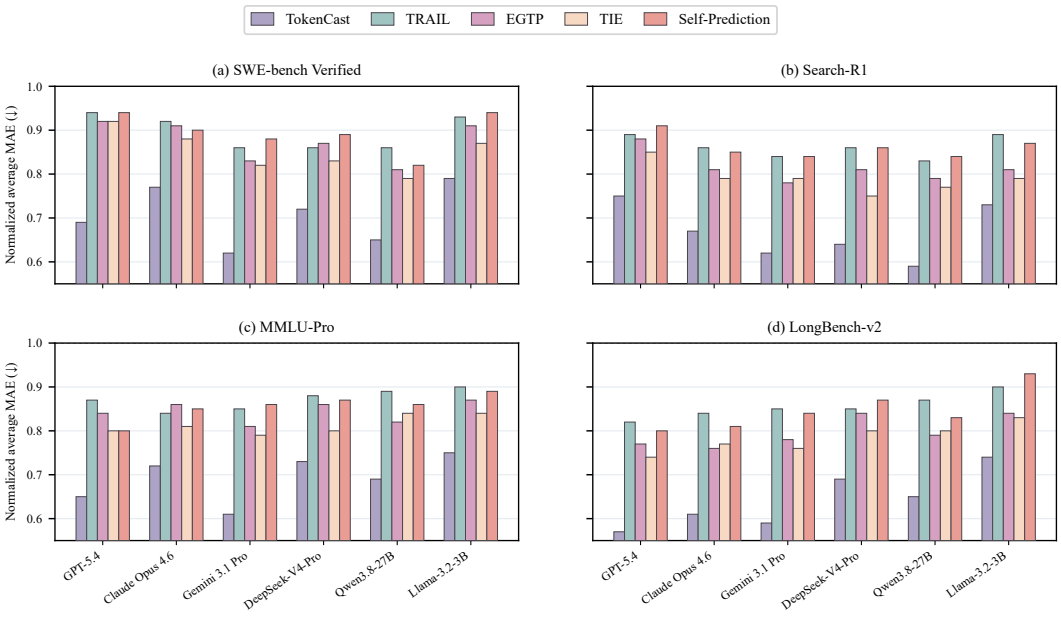


Figure 3: Normalized average MAE across four benchmarks and six agent LLMs. Each panel corresponds to one benchmark, and each bar group corresponds to one agent LLM. For each method, MAE is normalized by the corresponding history-median MAE at each prediction point and macro-averaged over the four prediction points. Lower is better.

Table 2: Execution traces per benchmark and harness, over six agent LLMs.

Benchmark	Harness	Tasks	Runs	Anchors	Anchor runs	Traces
SWE-bench Verified	DeepSeek Harness	144	2	24	8	2,592
SWE-bench Verified	OpenHands	144	2	24	8	2,592
Search-R1	DeepSeek Harness	32	4	8	16	1,344
Search-R1	OpenHands	32	4	8	16	1,344
MMLU-Pro	DeepSeek Harness	32	4	8	8	960
MMLU-Pro	OpenHands	32	4	8	8	960
LongBench-v2	DeepSeek Harness	32	4	8	8	960
LongBench-v2	OpenHands	32	4	8	8	960
Total						11,712

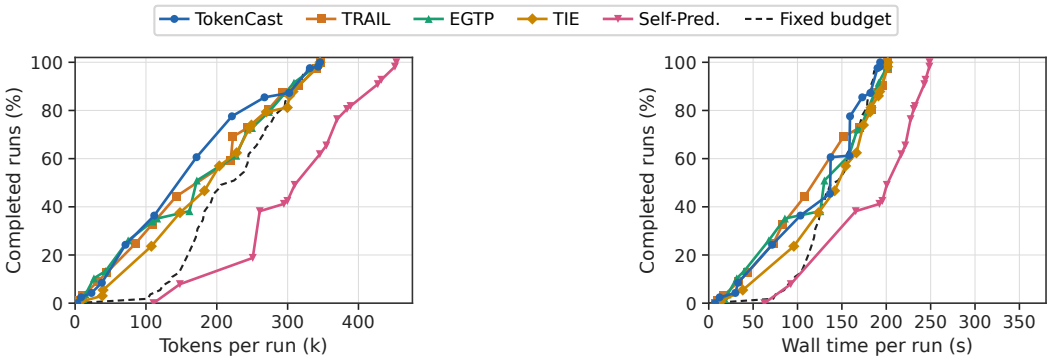
4.2.2 Prediction Reliability

Run-to-run variation. Repeated GPT-5.4 executions of the same task show substantial consumption spread across all four benchmarks. Figure 6 in Appendix D.1 shows the distributions for 48 anchor tasks and gives the repetition counts per benchmark.

Interval reliability. On the anchor tasks, TokenCast's calibrated intervals reduce MIS relative to Self-Prediction's native intervals at all four prediction points, by 32.0% on average. At Task Start, TokenCast covers 82.0% of outcomes against a nominal 90% level, while Self-Prediction covers 52.7%. Table 3 reports interval width, coverage, and MIS at every prediction point.

4.2.3 Generalization

Unseen task types. On independently released LiveClawBench trajectories (Long et al., 2026), zero-shot leave-one-domain-out transfer yields MAE ratios of 1.31 at Call Start and 1.47 at Task Update relative to Self-Prediction. With 20 target-domain tasks for adaptation, the ratios fall to 0.82 and 0.85. Appendix D.4.1 gives the protocol and intermediate results.



(a) Tokens (b) Wall time

Figure 4: Trace completion under stopping limits on SWE-bench Verified. Panel (a) plots trace completion against mean tokens per run, and panel (b) plots it against replay-accounted wall time per run. Prediction overhead is included. The dashed curve is the fixed-budget baseline.

Table 3: 90% prediction intervals over repeated executions of anchor tasks. TokenCast uses held-out calibration; Self-Prediction uses its reported 5th and 95th percentiles.

	TokenCast			Self-Prediction		
Prediction point	Coverage (%)	Width	MIS ↓	Coverage (%)	Width	MIS ↓
Task Start	82.0	752.5k	**1547.9k**	52.7	354.3k	3334.6k
Call Start	89.8	315	**549**	73.0	223	638
In-call Update	92.8	319	**576**	75.2	237	732
Task Update	91.2	379.1k	**574.3k**	77.0	232.0k	941.6k

Unseen agent LLMs. At Call Start, transfer to a held-out agent LLM yields 1.02 times the target history-median MAE without target-model tasks, improving to 0.95 with 20 tasks. With 3–10 target tasks, transfer outperforms target-only training. Appendix D.4.2 gives the protocol and a separate remaining-output-token Task Update analysis; other generalization results appear in Appendix D.4.

4.3 Case Study

TokenCast's forecasts are revised as execution unfolds. In a code-repair run, a verification failure raises the task-total forecast, which decreases after a compatibility workaround passes the reported assertions. In long-context QA, file operations after answer generation consume further tokens and raise the forecast. Both traces are in Appendix E. We next examine token-budget control driven by these online forecasts on SWE-bench Verified.

Budget control. On 288 GPT-5.4 runs from 144 SWE-bench Verified tasks, Figure 4 shows that TokenCast uses 21.3% fewer tokens on average across seven replay budgets while matching fixed-budget trace completion at every budget. A run is trace-complete when it reaches its recorded terminal state; the replay does not measure task resolution. Prediction costs are included. Appendix D.7 gives the stopping rules, per-budget results, and overhead (Table 21).

4.4 Ablation Study

Feature analysis. Figure 5 shows that the useful evidence changes over a run. Request length leads at Task Start and Call Start, with relative importance of 0.24 and 0.30. During generation, committed prefix length dominates at 0.58. After a call completes, last-call input tokens, remaining plan items, and calls without progress carry similar importance at 0.28, 0.25, and 0.23.

Segment representation. The segment triple improves both task-level and call-level forecasts. Replacing it with total token count, input length, and output length raises their normalized MAEs from 0.71 to 0.88 and from 0.68 to 0.82. The no-composition variant has a normalized average MAE of

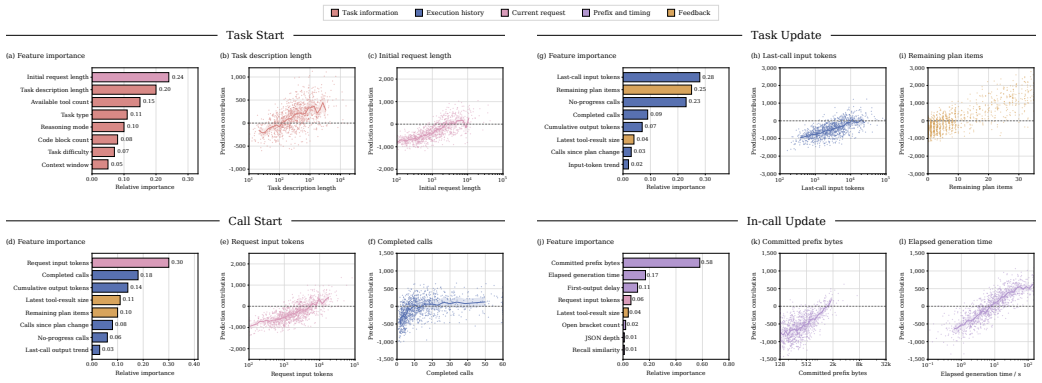


Figure 5: Feature analysis at the four prediction points. For each point, the panels show the relative importance of selected features and contribution patterns for two representative features. Colors indicate the feature groups defined in Appendix B.1.

0.78 and call-level MAE of 0.74, compared with 0.69 and 0.68 for the full method. Within the triple, omitting context growth yields 0.76, while omitting the residual yields 0.73.

Forecasting strategy. Before execution, direct regression has lower normalized MAE than composition, 0.82 versus 0.87. The ranking reverses at Task Update after calls have completed: composition reaches 0.66 and direct regression 0.72. Averaging the two forecasts reduces Task Update MAE to 0.64, and the full correction model reaches 0.62 while raising interval coverage from 88.1% under direct regression to 90.6%. Appendix D.6 gives the full comparison.

Pipeline components. The largest degradation comes from dropping both cross-fitting and cost weighting. Normalized average MAE rises from 0.69 to 0.78, while 90% interval coverage falls from 90.6% to 87.4%. Dropping cross-fitting alone gives 0.73 MAE, and dropping cost weighting alone gives 0.72. The correction model and predicted boundary variables also contribute: their separate ablations give 0.74 and 0.72 MAE. Among six base predictors, LightGBM gives the lowest error and the highest coverage with 0.8 ms model time. Appendix D.6 reports the complete component and base-predictor results in Tables 19 and 16.

Update frequency. Updating after every call takes 32.8 ms of mean cumulative prediction time and makes 19.7 forecasts per run. Updating every three calls cuts the time to 11.9 ms and the forecast count to 6.9, and a single forecast at Task Start takes 2.1 ms. These full-pipeline times include feature extraction and model inference, so every-call updating adds under 0.03% to the 129 s median wall time of a GPT-5.4 run. Appendix D.5 reports the full frequency sweep with per-run standard deviations for all four update intervals.

5 Conclusion

TokenCast addresses the problem of forecasting token consumption during LLM agent execution, where context accumulation causes each call's cost to depend on the entire preceding trajectory. The method represents each execution segment with a composable cost triple that separates the starting-input baseline from incremental consumption, and composes adjacent segments to propagate context growth into downstream cost estimates. Across four benchmarks and six agent LLMs, TokenCast's MAE reduction against the strongest comparator averages 14.5% over 96 evaluated combinations and, in offline budget-control replay, it matches a fixed budget's trace-completion rate while consuming 21.3% fewer tokens. Its mean cumulative task-level prediction time is 32.8 ms per run on SWE-bench Verified. Because the predictors are learned from recorded traces, transfer to a new task domain or agent LLM improves with a small set of target tasks from the new setting. This work provides a lightweight prediction layer for agent token consumption, and we leave the integration of these forecasts into a runtime decision framework that actively manages execution under budget constraints as a direction for future work.

AI USE STATEMENT

During manuscript preparation, large language models were used solely as general-purpose writing assistants for grammar checking, word refinement, and improving clarity. LLMs did not contribute to the research ideation, methodological design, or experimental execution. All suggestions produced by the LLMs were reviewed, edited, and vetted by the authors, who take full responsibility for the entire content of the paper.

ETHICS STATEMENT

Our experiments use public benchmarks and recorded agent trajectories to study token consumption in LLM agents. The study involves no human subjects or collection of private user data. Models and datasets are used under their respective licenses and terms of use.

REPRODUCIBILITY STATEMENT

Code associated with this study is linked at `https://anonymous.4open.science/r/TokenCast-1F8B`. Section 3 presents the forecasting formulation and training objective, with implementation details in Appendix B. Appendix C documents benchmark sampling, trajectory collection, agent and harness configurations, data splits, baseline implementations, and evaluation metrics. The budget-control replay protocol and complete Self-Prediction prompt are provided in Appendices D.7 and F, respectively.

REFERENCES

Anthropic. Introducing Claude Opus 4.6, 2026. URL `https://www.anthropic.com/news/claude-opus-4-6`.

Longju Bai, Zhemin Huang, Xingyao Wang, Jiao Sun, Rada Mihalcea, Erik Brynjolfsson, Alex Pentland, and Jiaxin Pei. How do AI agents spend your money? analyzing and predicting token consumption in agentic coding tasks. *CoRR*, abs/2604.22750, 2026. doi: 10.48550/ARXIV.2604.22750. URL `https://doi.org/10.48550/arXiv.2604.22750`.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 3639–3664. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.183. URL `https://doi.org/10.18653/v1/2025.acl-long.183`.

Yuxi Chen, Junming Chen, Chenyu He, Yiwei Li, Yicheng Ji, Yifan Wu, Dingyu Yang, Lansong Diao, Lidan Shou, Hongliang Zhang, et al. Token economics for llm agents: A dual-view study from computing and economics. *arXiv preprint arXiv:2605.09104*, 2026.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified, 2024. URL `https://openai.com/index/introducing-swe-bench-verified/`.

DeepSeek. DeepSeek Harness, 2026. URL `https://www.deepseek.com/harness/en/`.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient llm scheduling by learning to rank. *Advances in Neural Information Processing Systems*, 37:59006–59029, 2024.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

Google DeepMind. Gemini 3.1 Pro model card, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025a. doi: 10.48550/ARXIV.2503.09516. URL https://doi.org/10.48550/arXiv.2503.09516.

Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. In Guodong Long, Michale Blumestein, Yi Chang, Liane Lewin-Eytan, Zi Helen Huang, and Elad Yom-Tov (eds.), *Companion Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025 - 2 May 2025*, pp. 737–740. ACM, 2025b. doi: 10.1145/3701716.3715313. URL https://doi.org/10.1145/3701716.3715313.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. S^3: Increasing GPU utilization during generative inference for higher throughput. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/3a13be0c5dae69e0f08065f113fb10b8-Abstract-Conference.html.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. *CoRR*, abs/2310.03714, 2023. doi: 10.48550/ARXIV.2310.03714. URL https://doi.org/10.48550/arXiv.2310.03714.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

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 Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

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 Ada Gavrilovska and Douglas B. Terry (eds.), *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pp. 929–945. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

Xiang Long, Li Du, Yilong Xu, RongJian Xu, Qiyanhui Lu, Ying Gao, Qinhua Xie, Fangcheng Liu, Ning Ding, Haoqing Wang, et al. Liveclawbench: Benchmarking llm agents on complex, real-world assistant tasks. *arXiv preprint arXiv:2604.13072*, 2026.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 9802–9822, 2023.

Mohamed Amine Merzouk, Dmitri Carpov, Mirko Bronzi, Damiano Fornasiere, and Adam Oberman. How much is left? llms linearly encode their remaining output length. *CoRR*, abs/2607.05316, 2026. doi: 10.48550/ARXIV.2607.05316. URL https://doi.org/10.48550/arXiv.2607.05316.

Meta AI. Llama-3.2-3B-Instruct. Hugging Face, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct.

Kangqi Ni, Wenyue Hua, Xiaoxiang Shi, Jiang Guo, Shiyu Chang, and Tianlong Chen. Chimera: Latency- and performance-aware multi-agent serving for heterogeneous llms. *CoRR*, abs/2603.22206, 2026. doi: 10.48550/ARXIV.2603.22206. URL https://doi.org/10.48550/arXiv.2603.22206.

OpenAI. Introducing GPT-5.4, March 2026. URL https://openai.com/index/introducing-gpt-5-4/.

Daniel F. Perez-Ramirez, Dejan Kostic, and Magnus Boman. CASTILLO: characterizing response length distributions of large language models. *CoRR*, abs/2505.16881, 2025. doi: 10.48550/ARXIV.2505.16881. URL https://doi.org/10.48550/arXiv.2505.16881.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Basar, and Ravishankar K. Iyer. Efficient interactive LLM serving with proxy model-based sequence length prediction. *CoRR*, abs/2404.08509, 2024. doi: 10.48550/ARXIV.2404.08509. URL https://doi.org/10.48550/arXiv.2404.08509.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174, 2009.

Mohamad Salim, Jasmine Latendresse, SayedHassan Khatoonabadi, and Emad Shihab. Tokenomics: Quantifying where tokens are used in agentic software engineering. *arXiv preprint arXiv:2601.14470*, 2026.

Rana Shahout, Eran Malach, Chunwei Liu, Weifan Jiang, Minlan Yu, and Michael Mitzenmacher. Don't stop me now: Embedding based scheduling for LLMS. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=7JhGdZvW4T.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Jing Wang, Yu-Yang Qian, Ke Xue, Chao Qian, Peng Zhao, and Zhi-Hua Zhou. Robust length prediction: A perspective from heavy-tailed prompt-conditioned distributions. *CoRR*, abs/2604.07931, 2026. doi: 10.48550/ARXIV.2604.07931. URL https://doi.org/10.48550/arXiv.2604.07931.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 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, and et al. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In Grace Hui Yang, Hongning Wang, Sam Han, Claudia Hauff, Guido Zuccon, and Yi Zhang (eds.), *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024, Washington DC, USA, July 14-18, 2024*, pp. 641–649. ACM, 2024. doi: 10.1145/3626772.3657878. URL https://doi.org/10.1145/3626772.3657878.

Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. Reducing cost of LLM agents with trajectory reduction. *Proc. ACM Softw. Eng.*, 3(FSE):1241–1263, 2026. doi: 10.1145/3797084. URL https://doi.org/10.1145/3797084.

Huanyi Xie, Yubin Chen, Liangyu Wang, Lijie Hu, and Di Wang. Predicting LLM output length via entropy-guided representations. *CoRR*, abs/2602.11812, 2026. doi: 10.48550/ARXIV.2602.11812. URL https://doi.org/10.48550/arXiv.2602.11812.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shan Yu, Junyi Shu, Yuanjiang Ni, Kun Qian, Xue Li, Yang Wang, Jinyuan Zhang, Ziyi Xu, Shuo Yang, Lingjun Zhu, et al. Pythia: Exploiting workflow predictability for efficient agent-native llm serving. *arXiv preprint arXiv:2604.25899*, 2026.

Haoyu Zheng, Yongqiang Zhang, Fangcheng Fu, Xiaokai Zhou, Hao Luo, Hongchao Zhu, Yuanyuan Zhu, Hao Wang, Xiao Yan, and Jiawei Jiang. Scheduling LLM inference with uncertainty-aware output length predictions. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=I5IMkvVKd7.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. Tracelab: Characterizing coding agent workloads for llm serving. *arXiv preprint arXiv:2606.30560*, 2026.

APPENDIX

A COMPARISON WITH RELATED METHODS

Table 4 compares prediction targets, evidence, and update points. TokenCast forecasts call and remaining-task consumption from execution features and a segment-cost representation.

Table 4: Prediction targets and estimators in related work. Appendix C.3 describes our adaptations.

Work	Prediction target	Evidence and estimator	Update point
TRAIL	Response length	Last-token hidden state and length bins	Before generation
EGTP	Response and remaining-response length	Hidden states and entropy pooling	Pre/mid generation
TIE	Response length distribution	Text embedding and log-t model	Before generation
Chimera	Remaining workflow output	Prompt and workflow features; quantile forest	Workflow request
Pythia	Workflow path and role output length	Historical trace profiles	Incoming request
Self-Prediction	Total task tokens	Agent inspection of task environment	Before execution
TokenCast	Provider-accounted call and remaining-task tokens	Execution features; direct and compositional predictors	Four prediction points

B METHOD DETAILS

B.1 EXECUTION EVIDENCE

The observation record $\mathbf{v}$ contains only information available at the prediction point. Token usage comes from provider records, and a logical call aggregates its initial request and any retries. The input length of an assembled request is measured directly; a future request's input length remains predicted until assembly. The visible prefix contains streamed text and tool-call arguments committed so far, with a checkpoint every 128 UTF-8 bytes. Unavailable fields are masked. Prediction-point observations remain separate from the prefix-ending predictions passed to the suffix model.

Task representation. The task representation $\mathbf{d}$ uses TF-IDF features over word unigrams and bigrams and character 3- to 5-grams, with 12,000 terms for each representation. When a benchmark provides a task-difficulty label, the label is prepended to the task statement before encoding.

Action types. Reads include file reads, glob matches, and searches. Edits write or replace file content. Tracking updates the to-do list. A shell command is labeled as a test when it invokes pytest, tox, nox, or unittest, or refers to a test directory, and other shell commands are labeled as runs. An action is marked as failed when a tool reports an error or a shell command exits with a nonzero status. The stage of a call is determined by its most recent action.

Execution history. The progress record contains the numbers of planned, completed, and remaining to-do items, together with the number of calls since the list was last updated. The consumption history contains the input, output, and reasoning usage of the latest completed call, their changes from the preceding call, their mean, median, and least-squares trends over the last five and ten calls, and running totals of calls, model requests, tool actions, and confirmed tokens. Task Update additionally records the numbers of failed requests and tool errors in the latest call, together with the numbers of consecutive calls that failed, required a retry, or made no progress. Progress is defined as an edit, test, tracking update, or completed plan item. A window statistic is masked when the required observations are unavailable.

Recent tool actions. The w most recent tool actions fill the slots in recency order. Each slot records the action type, status, result size, calls since execution, and the reported result: an exit code, the number of lines returned by a read, the number of matches returned by a search, or the passed and failed counts of a test run. We select w per fold from $\{3, 5, 8\}$ using the validation tasks.

Generated prefix. The lowercased prefix is hashed by words and word pairs into 256 signed buckets with weight $1 + \log n$ for n occurrences, and its last line is hashed into 64 buckets. A single scan records whether the prefix is a valid JSON prefix, its nesting depth, whether it ends inside a string and under which tool-argument key, and, within string contents, unmatched brackets, quote parity, open code fences, heredocs, and the termination pattern of the last line.

Prefix recall. Earlier completed model requests from the same run serve as references, each represented by its hashed prefix at every checkpoint and its final generated length. The current prefix is compared with these references using cosine similarity, and the record includes the final and remaining lengths associated with the most similar requests and checkpoints.

Stream timing. For the current request, let s denote its start time and let $c_1 < \cdots < c_m$ denote the output-checkpoint times. The timing record contains the time to the first checkpoint $c_1 - s$, elapsed time $c_m - s$, streaming time $c_m - c_1$, committed bytes divided by streaming time, time since the previous checkpoint, the longest gap between checkpoints, and the initial-delay fraction $(c_1 - s)/(c_m - s)$. Fields with undefined denominators or too few observations are masked.

B.2 Predictors and Training

Predictors. TokenCast uses separate LightGBM models for the four prediction settings. Call Start and In-call Update predict the complete consumption of the current call. Task Start and Task Update use a direct path and a compositional path. The direct path predicts remaining task consumption as one target. The compositional path predicts the current or next segment, its ending boundary, and the subsequent suffix, then combines the two segment representations using Eq. (1).

Training instances. Labels are extracted from completed traces. Call-level instances use the realized complete-call consumption. Task-level instances contain the remaining-consumption target, the prefix representation and ending boundary, and the suffix representation. Model inputs contain only the evidence available at the corresponding prediction point.

Task-level fitting. The direct and prefix models are fitted first. Task-level cross-fitting then produces out-of-fold prefix boundaries for suffix training. The suffix residual label is recomputed against the predicted input baseline as described in Section 3.3. After these models are fixed, the complete pipeline produces out-of-fold forecasts used to fit the correction model in Eq. (3).

Inference. Call-level forecasts use the corresponding fitted model directly. Task-level forecasts compute the direct prediction, the prefix boundary, the suffix prediction, and the compositional forecast before applying the correction model. The point models remain fixed during execution. Raw interval endpoints are produced by the quantile models in Appendix B.4.

B.3 Theoretical Analysis

Composition properties. Let L_A and L_B be the initial input lengths of adjacent segments A and B. Since $L_B = L_A + g_A$ and $C_X = n_X L_X + b_X$ for $X \in \{A, B\}$,

$$C_{A \circ B} = (n_A + n_B)L_A + b_A + b_B + n_B g_A = C_A + C_B.$$

The cross-term $n_B g_A$ arises when the cost of B is expressed relative to the initial input length of A. For three consecutive segments A, B, and D, either grouping yields the residual $b_A + b_B + b_D + n_B g_A + n_D(g_A + g_B)$. The composition is therefore associative.

Single-call representation. A single call has representation $\phi_j = (1,\, L_{j+1} - L_j,\, C_j - L_j)$. For the terminal call, set $L_{K+1} = L_K$ to close the notation; no suffix follows it. Composing these representations in execution order recovers the representation of any contiguous segment. For a complete trace of K calls, the resulting residual is $\sum_{j=1}^{K} C_j - K L_1$. Hence

$$T = K L_1 + b_{\phi_1 \circ \cdots \circ \phi_K},$$

regardless of how the trace is partitioned.

Error propagation and loss weights. In a compositional forecast, the suffix is predicted from the estimated boundary of the prefix. Let $L_B = L_A + g_A$ be the true input length at the start of the suffix, and let δg_A and δn_B denote prediction errors in the prefix's context change and the suffix's

call count. With L_A fixed, and writing δb_B for the error in the suffix residual, the suffix cost error is

$$\Delta C_B = n_B \delta g_A + L_B \delta n_B + \delta n_B \delta g_A + \delta b_B.$$

The coefficients n_B and L_B motivate the local weights in Eq. (2) under a true-boundary residual. Suffix training instead uses $b_B^{\text{train}} = C_B - n_B \widehat{L}_B$. Writing $e_b = \widehat{b}_B - b_B^{\text{train}}$ and $e_n = \widehat{n}_B - n_B$ gives $\widehat{C}_B - C_B = \widehat{L}_B e_n + e_b$. Boundary prediction can affect the learned residual and the suffix model's inputs. Cross-fitting exposes suffix training to out-of-fold boundary errors.

B.4 Interval Calibration

After the point models are fixed, separate LightGBM quantile models are trained for the 0.05 and 0.95 endpoints at each prediction setting. They use the evidence available at that prediction point and are fitted on the training tasks. Their outputs form the raw interval $[\ell_i, u_i]$.

Intervals are calibrated separately for Task Start, Call Start, In-call Update, and Task Update using the held-out calibration tasks of each fold. For target y_i, the absolute interval residual is

$$s_i = \max\{\ell_i - y_i,\ y_i - u_i,\ 0\}. \tag{4}$$

For each prediction setting, the corresponding calibration quantile of $\{s_i\}$ is added symmetrically to the raw endpoints. At runtime, the calibrated endpoints are bounded below by consumption already confirmed within the prediction scope. Calibration parameters remain fixed during inference.

C Experimental Setup

C.1 Trace Collection

Benchmarks. From the 500 instances of SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024), we take 144. The four smallest repositories, flask, seaborn, requests, and pylint, enter in full. pytest, xarray, astropy, scikit-learn, and matplotlib contribute 15 each, and sphinx, sympy, and django 16 each. Search-R1 (Jin et al., 2025a) contributes 32 questions from seven FlashRAG evaluation sets (Jin et al., 2025b), half single-hop from NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), and PopQA (Mallen et al., 2023), and half multi-hop from HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). MMLU-Pro (Wang et al., 2024b) contributes 32 test questions with two or three per category, and LongBench-v2 (Bai et al., 2025) has 32 questions spread over its six domains and three length labels. Within each stratum, we take instances in the stable-hash order of their id, and the anchors are the first two per SWE-bench Verified repository, one for flask and three for django, and the first eight in each other benchmark.

Task packs. A task is a statement and a seed workspace. For SWE-bench, the seed is the repository at the base commit. The agent may not edit tests and finishes by writing `submission.patch`, the diff of its working tree. For the other benchmarks, the seed holds `README.md` and an empty `answer.md`, the statement provides the question and its options, for LongBench-v2 together with the context document, and the agent writes its answer into `answer.md`. A Search-R1 statement also names a shell command that queries a local BM25 server (Robertson & Zaragoza, 2009; Jin et al., 2025a), the official Search-R1 retriever over the 21,015,324-passage wiki-18 corpus, and prints three passages. Correctness is scored after collection, with the SWE-bench Verified harness for patches, letter match for MMLU-Pro and LongBench-v2, and alias match for Search-R1.

Agent LLMs. Table 5 lists the models and their reasoning configurations. GPT-5.4 (OpenAI, 2026), Claude Opus 4.6 (Anthropic, 2026), and Gemini 3.1 Pro (Google DeepMind, 2026) use the lowest reasoning setting available through their respective APIs. DeepSeek-V4-Pro (DeepSeek-AI, 2026) and Qwen3.8-27B (Qwen Team, 2026) run with thinking enabled, while Llama-3.2-3B-Instruct (Meta AI, 2024) has no reasoning mode. Temperature is 0 where supported. The self-hosted models run on vLLM (Kwon et al., 2023) in bf16 on eight A100 GPUs with full context windows.

Harnesses. DeepSeek Harness (DeepSeek, 2026) runs at version 0.1.1-rc.2, commit `b150a551`, in its headless profile with 12 tools: `read`, `write`, `edit`, `str_replace_editor`, `glob`, `grep`,

Table 5: Agent LLMs used for trace collection, with access modes and reasoning configurations.

Model	Vendor	Access	Reasoning configuration
GPT-5.4	OpenAI	API	Low reasoning effort
Claude Opus 4.6	Anthropic	API	Adaptive thinking; low effort
Gemini 3.1 Pro	Google	API	Low thinking level
DeepSeek-V4-Pro	DeepSeek	API	Thinking enabled
Qwen3.8-27B	Alibaba	Self-hosted	Thinking enabled
Llama-3.2-3B-Instruct	Meta	Self-hosted	No reasoning mode

`pwsh`, `job_list`, `job_output`, `job_kill`, `todo_write`, and `skill`. Commands are executed without a sandbox or approval prompts. Context compaction, result pruning, generated session titles, web access, and subagents are turned off, so every provider request is an agent call. Each run starts a fresh session in a fresh copy of the seed workspace on Windows.

OpenHands (Wang et al., 2025) runs at version 1.18.0 with the CodeAct (Wang et al., 2024a) agent and its `execute_bash`, `str_replace_editor`, and task-tracking tools, with browsing and the condenser disabled. It runs SWE-bench Verified in the official image of each instance and the other benchmarks in a plain Linux image and calls the same retrieval script.

Both harnesses cap a run at 7,200 s, 500 calls, 4 retries per call, 65,536 output tokens per model request, and 30,000,000 tokens. A run ends when the agent submits, the model gives a final response without submitting, or a cap is reached. A logical call consists of its initial model request and any retry requests issued before tool feedback is received. The observer timestamps every model request and emits a checkpoint every 128 bytes of committed output. Token usage is taken from the provider response for each request and aggregated at the logical-call level.

Repeated executions. Table 2 lists the runs per task. SWE-bench Verified tasks run twice per model, and the other tasks run four times. Anchors run eight times, or 16 for Search-R1.

Trace statistics. Table 6 lists the median tokens and calls per run and the share of correct runs for each benchmark and model. For GPT-5.4 on SWE-bench Verified, the input makes up 99% of the tokens, the median run takes 129 s, and the median call emits two checkpoints.

The code-repair execution in Appendix E.1 illustrates the four prediction points and the usage records along a complete run (Figure 9).

Additional trajectories. The generalization experiments use independently released trajectories from LiveClawBench (Long et al., 2026), which records multiple agent LLMs across task domains under a shared agent framework. We retain runs with complete interaction records and per-call input and output token counts, and construct targets with the accounting convention of Section 3.1.

For the reasoning-configuration experiment in Appendix D.4.5, we rerun 69 SWE-bench Verified tasks with GPT-5.4 reasoning effort set to off and compare them with the same tasks under the low setting. The API route reports no separate reasoning-token count. Measured against visible output, the median request bills 0.46 output tokens per byte under the low setting and 0.34 under off.

Table 6: Median tokens in thousands and calls among runs with complete usage, and correct runs as a percentage of all attempted runs, under each harness.

	DeepSeek Harness			OpenHands		
Model	Tokens	Calls	Correct	Tokens	Calls	Correct
SWE-bench Verified						
GPT-5.4	245.4	21	40.7	314.5	20	36.3
Claude Opus 4.6	258.1	21	41.7	470.7	22	54.2
Gemini 3.1 Pro	333.5	21	43.5	253.5	21	47.5
DeepSeek-V4-Pro	280.9	20	34.3	336.8	28	33.8
Qwen3.8-27B	526.0	22	29.6	211.5	15	31.2
Llama-3.2-3B-Instruct	508.8	31	9.5	619.7	33	11.8
Search-R1						
GPT-5.4	38.1	8	76.3	37.5	8	75.0
Claude Opus 4.6	58.3	9	59.8	58.5	10	69.6
Gemini 3.1 Pro	55.8	10	68.8	48.3	8	69.6
DeepSeek-V4-Pro	94.8	12	65.2	41.2	9	70.5
Qwen3.8-27B	60.3	10	54.0	84.0	9	58.9
Llama-3.2-3B-Instruct	136.4	16	27.2	98.7	15	28.6
MMLU-Pro						
GPT-5.4	21.6	4	81.9	46.1	6	81.9
Claude Opus 4.6	27.3	5	87.5	29.3	4	88.1
Gemini 3.1 Pro	54.6	9	75.6	59.7	9	77.5
DeepSeek-V4-Pro	35.2	6	70.6	38.6	5	76.9
Qwen3.8-27B	41.9	7	76.9	52.7	7	67.5
Llama-3.2-3B-Instruct	30.1	5	42.5	48.7	5	43.8
LongBench-v2						
GPT-5.4	762.5	6	58.8	608.0	5	58.1
Claude Opus 4.6	361.7	4	66.9	716.1	6	75.0
Gemini 3.1 Pro	832.5	8	64.4	610.0	8	69.4
DeepSeek-V4-Pro	523.7	4	60.6	1,030.7	9	52.5
Qwen3.8-27B	884.4	8	45.0	581.0	6	45.6
Llama-3.2-3B-Instruct	645.9	7	25.6	592.1	4	13.1

C.2 Data Splits

Tasks are assigned to five folds, balanced over benchmarks and agent LLMs, and all runs of a task remain in their fold. Each fold serves once as the test fold. The next fold calibrates intervals, the one after selects hyperparameters, and the remaining two train. We repeat the partitioning with three random seeds. For each seed, we pool out-of-fold test predictions and compute task-weighted metrics over all evaluated tasks. Tables 1 and 7–10 report means across the three seeds.

C.3 Baselines

Output-length predictors. TRAIL, EGTP, and TIE are originally designed to predict the length of a single response. We adapt each method to the target associated with a prediction point while preserving its core representation and estimator. For Qwen3.8-27B and Llama-3.2-3B-Instruct, the internal-state methods read the last layer of the agent LLM itself. For the API models, they use Llama-3.2-3B-Instruct as a proxy encoder over the text visible at the prediction point: the task statement at Task Start, the statement and transcript of completed calls at Call Start and Task Update, and the statement and generated prefix at In-call Update. The window keeps the first 384 tokens of the statement and the last 1,536 of the remaining text. TRAIL classifies the last-token state into 8 log-spaced target-length bins and predicts the expected bin median. EGTP fits a ridge regression of log target length on the entropy-weighted mean state and entropy statistics of the window. TIE encodes the window with bge-small-en-v1.5 (Xiao et al., 2024) and fits the location and scale of a log-t distribution with 3.5 degrees of freedom on the concatenated CLS, mean, and max poolings. Its point prediction is the median, and its interval spans the 0.05 and 0.95 quantiles.

Agent usage estimators. Self-Prediction queries the agent LLM for the 5th, 50th, and 95th percentiles of token consumption for the same prediction target. The median serves as the point prediction, and the 5th and 95th percentiles define a nominal 90% prediction interval. We use these predictions directly, without fitting or calibrating them on held-out tasks. At Task Start, the agent first inspects the seed workspace with its full tool set and estimates the tokens of a complete run. At the other prediction points, we adapt the prompt to the observed execution record and corresponding prediction target. At Call Start, the agent estimates the complete consumption of the current call. At In-call Update, it estimates the same complete-call quantity using the observed generation prefix. At Task Update, it estimates the token consumption of the remaining execution. Appendix F provides the complete prompt template for all four points.

Fitting. Each learned baseline uses TokenCast's partition: it is fitted on each fold's training tasks, tuned on the validation tasks, and calibrated on the calibration tasks.

C.4 Metrics

For each prediction setting, let i index the evaluated prediction points, with target y_i, point prediction $\hat{y}_i$, 90% prediction interval $[\ell_i, u_i]$, and normalized weight w_i. We assign equal weight to each evaluated task, divide that weight equally among its evaluated runs, and divide each run's weight equally among its evaluated prediction points. Thus, $\sum_i w_i = 1$, and

$$\mathrm{MAE} = \sum_i w_i |\hat{y}_i - y_i|, \qquad \bar{y} = \sum_i w_i y_i, \qquad \mathrm{WAPE} = \frac{\mathrm{MAE}}{\bar{y}} \times 100\%. \tag{5}$$

MAE and mean target consumption $\bar{y}$ use the same evaluation samples and task–run–checkpoint weights. WAPE expresses the mean absolute error as a percentage of the mean actual target consumption. For a nominal $1 - \alpha$ prediction interval, we use the interval score

$$\mathrm{IS}_i = (u_i - \ell_i) + \frac{2}{\alpha}(\ell_i - y_i)\mathbf{1}\{y_i < \ell_i\} + \frac{2}{\alpha}(y_i - u_i)\mathbf{1}\{y_i > u_i\}, \tag{6}$$

with $\alpha = 0.1$. Mean interval score is

$$\mathrm{MIS} = \sum_i w_i \, \mathrm{IS}_i. \tag{7}$$

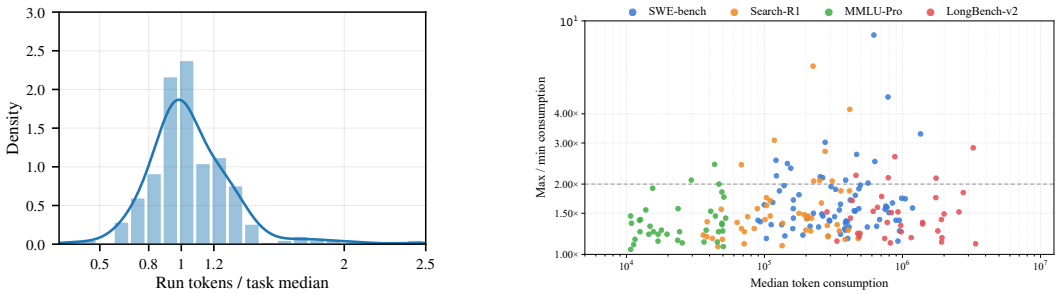
Empirical coverage and mean interval width are aggregated using the same weights. The Norm. Avg. column of Table 1 is the arithmetic mean over the four prediction points of a method's MAE divided by the MAE of the corresponding history-median predictor. This predictor outputs the median target of the training tasks in the same fold for the same benchmark, agent LLM, and prediction point, and uses no task features or execution evidence.

D Extended Evaluation

D.1 Run-to-Run Variation and Interval Reliability

Figure 6 summarizes repeated GPT-5.4 executions from 48 anchor tasks across four benchmarks and two harnesses. Search-R1 has 16 repetitions per task–harness pair, while the others have eight. Repeated executions of a task show wide consumption spread across all four benchmarks.

Table 3 compares TokenCast's calibrated intervals with the native intervals returned by Self-Prediction at the four prediction points. MIS combines interval width with a penalty for how far outcomes fall outside the interval. Lower values indicate better interval forecasts. TokenCast's wider intervals reduce the missed-outcome penalty enough to improve MIS. This comparison includes TokenCast's held-out calibration and Self-Prediction's native, uncalibrated intervals.



(a) Run-to-run variation

(b) Consumption spread

Figure 6: Run-to-run variation in token consumption across repeated executions of the same task. The panels show the distribution across repeated runs and the within-task consumption spread as a function of task consumption, over 48 GPT-5.4 anchor tasks.

D.2 Results Across Benchmarks and Agent LLMs

Tables 7 to 10 report the MAE for each benchmark and agent LLM on the same runs, with the history median as the normalization reference. MAE is reported in tokens, with k denoting thousands. Norm. Avg. follows the normalization in Appendix C.4.

D.3 Error Relative to Consumption

Table 11 reports WAPE and mean target consumption for GPT-5.4 and Qwen3.8-27B across four benchmarks and four prediction points. For each benchmark, agent LLM, and prediction point, we use the same evaluation weights as MAE:

$$\mathrm{WAPE}(\%) = 100\,\frac{\sum_i w_i|\widehat{y}_i - y_i|}{\sum_i w_i y_i} = 100\,\frac{\mathrm{MAE}}{\bar{y}}, \qquad \bar{y} = \sum_i w_i y_i, \quad \sum_i w_i = 1. \tag{8}$$

Here, y_i is the target consumption: total task consumption T at Task Start, full-call consumption C_k at Call Start and In-call Update, and remaining task consumption R_k at Task Update. The weights follow Appendix C.4. All methods within a row share the same target mean and evaluation weights. A WAPE of 15% means that the weighted MAE is 15% of the weighted mean target.

Table 7: MAE on SWE-bench Verified. Norm. Avg. is the macro-average of MAE normalized at each prediction point by the history-median MAE. k denotes thousands of tokens. Best and second-best results are in bold and underlined, respectively. Marks are assigned using unrounded values.

Prediction point	TokenCast	TRAIL	EGTP	TIE	Self-Pred.	History median
GPT-5.4						
Task Start	**144.0k**	165.3k	160.9k	157.0k	152.0k	179.0k
Call Start	**64.5**	71.1	70.3	69.5	70.9	74.2
In-call Update	**38.9**	78.9	74.6	77.4	80.2	80.3
Task Update	**80.0k**	115.0k	117.0k	119.0k	126.0k	130.0k
Norm. Avg.	**0.69**	0.94	0.92	0.92	0.94	1.00
Claude Opus 4.6						
Task Start	150.7k	157.5k	153.8k	146.8k	**138.9k**	171.0k
Call Start	**64.1**	69.4	69.9	66.8	68.8	73.5
In-call Update	**55.0**	80.2	71.7	74.4	78.1	83.4
Task Update	**88.7k**	112.7k	122.4k	116.4k	122.0k	133.0k
Norm. Avg.	**0.77**	0.92	0.91	0.88	0.90	1.00
Gemini 3.1 Pro						
Task Start	**157.8k**	173.5k	170.2k	176.4k	163.6k	193.0k
Call Start	**68.3**	74.7	71.6	68.4	79.8	88.5
In-call Update	**41.0**	84.3	79.4	77.2	87.2	96.2
Task Update	**69.6k**	126.3k	121.4k	118.6k	132.2k	151.0k
Norm. Avg.	**0.62**	0.86	0.83	0.82	0.88	1.00
DeepSeek-V4-Pro						
Task Start	212.9k	204.9k	211.5k	198.4k	**194.5k**	232.0k
Call Start	**80.0**	89.0	84.6	85.9	91.3	101.5
In-call Update	**65.5**	96.1	94.9	86.3	104.8	112.8
Task Update	**106.2k**	143.4k	155.6k	148.1k	158.4k	177.0k
Norm. Avg.	**0.72**	0.86	0.87	0.83	0.89	1.00
Qwen3.8-27B						
Task Start	**192.0k**	232.0k	209.0k	215.0k	221.0k	284.0k
Call Start	**87.0**	109.0	102.9	99.0	105.5	119.0
In-call Update	**81.9**	120.0	116.0	111.4	108.9	147.0
Task Update	**124.0k**	176.0k	171.0k	164.0k	173.0k	198.0k
Norm. Avg.	**0.65**	0.86	0.81	0.79	0.82	1.00
Llama-3.2-3B-Instruct						
Task Start	272.3k	277.6k	269.7k	**258.2k**	264.8k	302.0k
Call Start	**133.8**	140.7	134.6	140.0	141.6	147.0
In-call Update	**100.3**	141.8	137.7	126.4	150.2	158.0
Task Update	**152.2k**	205.3k	213.1k	188.2k	213.7k	218.0k
Norm. Avg.	**0.79**	0.93	0.91	0.87	0.94	1.00

Table 8: MAE on Search-R1. Norm. Avg. is the macro-average of MAE normalized at each prediction point by the history-median MAE. k denotes thousands of tokens. Best and second-best results are in bold and underlined, respectively. Marks are assigned using unrounded values.

Prediction point	TokenCast	TRAIL	EGTP	TIE	Self-Pred.	History median
GPT-5.4						
Task Start	**34.2k**	37.8k	37.6k	36.4k	36.9k	44.9k
Call Start	34.5	37.3	36.5	**34.0**	39.8	44.0
In-call Update	**31.3**	39.4	38.5	36.8	39.9	42.0
Task Update	**17.6k**	23.1k	23.7k	23.3k	24.2k	25.0k
Norm. Avg.	**0.75**	0.89	0.88	0.85	0.91	1.00
Claude Opus 4.6						
Task Start	31.1k	33.9k	32.9k	32.2k	**29.8k**	39.8k
Call Start	**30.4**	33.6	32.0	30.4	34.9	40.2
In-call Update	**22.8**	33.4	31.6	29.7	34.0	38.5
Task Update	**12.9k**	20.8k	18.7k	19.5k	21.4k	23.6k
Norm. Avg.	**0.67**	0.86	0.81	0.79	0.85	1.00
Gemini 3.1 Pro						
Task Start	38.0k	42.3k	41.0k	40.7k	**37.1k**	49.8k
Call Start	40.0	44.7	**39.8**	42.3	47.0	54.5
In-call Update	**25.5**	46.8	44.2	42.6	47.5	57.0
Task Update	**16.1k**	27.8k	24.6k	25.2k	28.6k	31.5k
Norm. Avg.	**0.61**	0.84	0.78	0.79	0.84	1.00
DeepSeek-V4-Pro						
Task Start	46.5k	50.0k	47.5k	44.2k	**42.9k**	58.5k
Call Start	50.7	57.3	54.0	**50.6**	60.6	68.0
In-call Update	**35.1**	60.8	56.2	50.4	62.3	71.5
Task Update	**19.9k**	33.7k	31.1k	29.8k	34.5k	37.2k
Norm. Avg.	**0.64**	0.86	0.81	0.75	0.86	1.00
Qwen3.8-27B						
Task Start	**39.0k**	53.5k	49.8k	47.6k	51.0k	72.9k
Call Start	**56.7**	65.9	71.4	67.7	76.1	91.0
In-call Update	**57.0**	84.6	80.7	75.5	83.2	93.0
Task Update	**24.0k**	39.5k	34.0k	35.4k	38.4k	41.0k
Norm. Avg.	**0.59**	0.83	0.79	0.77	0.84	1.00
Llama-3.2-3B-Instruct						
Task Start	59.0k	60.0k	57.9k	53.6k	**52.4k**	68.0k
Call Start	79.0	80.9	**70.4**	75.4	84.6	95.0
In-call Update	**64.0**	99.4	91.2	84.7	95.6	108.0
Task Update	**31.0k**	43.8k	40.2k	38.6k	45.9k	49.0k
Norm. Avg.	**0.73**	0.89	0.81	0.79	0.87	1.00

Table 9: MAE on MMLU-Pro. Norm. Avg. is the macro-average of MAE normalized at each prediction point by the history-median MAE. k denotes thousands of tokens. Best and second-best results are in bold and underlined, respectively. Marks are assigned using unrounded values.

Prediction point	TokenCast	TRAIL	EGTP	TIE	Self-Pred.	History median
GPT-5.4						
Task Start	8.0k	8.8k	8.5k	8.2k	**7.4k**	9.7k
Call Start	**14.8**	16.7	15.5	15.8	17.6	19.6
In-call Update	**9.2**	19.6	18.3	16.7	17.4	24.3
Task Update	**4.3k**	6.2k	6.5k	6.0k	5.8k	6.9k
Norm. Avg.	**0.65**	0.87	0.84	0.80	0.80	1.00
Claude Opus 4.6						
Task Start	7.5k	8.0k	7.7k	7.3k	**6.9k**	8.9k
Call Start	14.9	14.8	15.6	**14.3**	16.1	18.3
In-call Update	**11.5**	15.5	16.8	16.0	18.4	22.0
Task Update	**4.4k**	6.0k	6.1k	5.8k	5.6k	6.3k
Norm. Avg.	**0.72**	0.84	0.86	0.81	0.85	1.00
Gemini 3.1 Pro						
Task Start	**8.6k**	10.1k	9.6k	9.3k	8.9k	11.5k
Call Start	**18.0**	18.3	20.1	19.4	21.3	23.9
In-call Update	**10.5**	24.7	20.6	21.8	23.6	28.7
Task Update	**4.7k**	7.2k	6.8k	6.4k	7.5k	8.0k
Norm. Avg.	**0.61**	0.85	0.81	0.79	0.86	1.00
DeepSeek-V4-Pro						
Task Start	11.6k	12.3k	11.9k	11.0k	**10.6k**	13.7k
Call Start	24.4	26.2	24.8	**23.5**	26.9	29.6
In-call Update	**19.5**	31.8	28.7	25.4	30.4	35.9
Task Update	**6.9k**	8.5k	9.2k	8.9k	9.4k	9.8k
Norm. Avg.	**0.73**	0.88	0.86	0.80	0.87	1.00
Qwen3.8-27B						
Task Start	13.0k	14.3k	**12.5k**	13.7k	13.3k	16.2k
Call Start	**26.5**	30.3	28.6	27.9	31.1	35.1
In-call Update	**21.1**	37.7	35.5	34.3	32.8	42.4
Task Update	**8.4k**	11.2k	10.3k	10.7k	11.5k	12.0k
Norm. Avg.	**0.69**	0.89	0.82	0.84	0.86	1.00
Llama-3.2-3B-Instruct						
Task Start	16.4k	17.5k	16.9k	**15.4k**	15.8k	19.6k
Call Start	33.5	36.5	**33.2**	33.8	37.9	41.8
In-call Update	**30.7**	46.0	43.7	42.3	47.3	50.2
Task Update	**10.8k**	13.2k	13.8k	13.5k	12.9k	14.4k
Norm. Avg.	**0.75**	0.90	0.87	0.84	0.89	1.00

Table 10: MAE on LongBench-v2. Norm. Avg. is the macro-average of MAE normalized at each prediction point by the history-median MAE. k denotes thousands of tokens. Best and second-best results are in bold and underlined, respectively. Marks are assigned using unrounded values.

Prediction point	TokenCast	TRAIL	EGTP	TIE	Self-Pred.	History median
GPT-5.4						
Task Start	**32.2k**	37.1k	36.0k	35.1k	32.7k	44.0k
Call Start	**53.2**	60.6	56.9	53.7	61.2	74.0
In-call Update	**35.5**	73.5	68.8	62.7	70.1	91.0
Task Update	**13.9k**	26.9k	24.2k	25.1k	28.2k	33.0k
Norm. Avg.	**0.57**	0.82	0.77	0.74	0.80	1.00
Claude Opus 4.6						
Task Start	**28.7k**	33.4k	32.0k	30.7k	29.4k	40.5k
Call Start	**48.1**	57.2	50.0	54.5	56.0	67.5
In-call Update	**41.3**	69.5	57.9	61.3	65.4	82.0
Task Update	**14.9k**	23.8k	22.4k	21.5k	25.1k	28.5k
Norm. Avg.	**0.61**	0.84	0.76	0.77	0.81	1.00
Gemini 3.1 Pro						
Task Start	39.9k	43.2k	41.9k	**38.2k**	38.9k	50.0k
Call Start	**60.6**	72.3	66.8	66.0	75.0	85.0
In-call Update	**43.7**	88.4	80.6	75.1	85.8	103.0
Task Update	**15.8k**	31.2k	27.2k	28.6k	33.0k	38.0k
Norm. Avg.	**0.59**	0.85	0.78	0.76	0.84	1.00
DeepSeek-V4-Pro						
Task Start	52.3k	55.0k	54.9k	51.2k	**49.6k**	61.5k
Call Start	**84.5**	96.0	89.6	85.6	97.9	108.0
In-call Update	**77.2**	115.6	110.2	101.7	118.4	132.0
Task Update	**26.1k**	35.7k	38.7k	37.4k	42.2k	47.5k
Norm. Avg.	**0.69**	0.85	0.84	0.80	0.87	1.00
Qwen3.8-27B						
Task Start	**50.8k**	60.7k	55.9k	54.6k	52.6k	70.0k
Call Start	94.4	107.8	**94.1**	99.7	110.8	124.0
In-call Update	**95.0**	136.4	128.9	121.2	118.6	151.0
Task Update	**26.9k**	47.6k	42.7k	44.5k	49.0k	56.0k
Norm. Avg.	**0.65**	0.87	0.79	0.80	0.83	1.00
Llama-3.2-3B-Instruct						
Task Start	70.6k	73.6k	69.2k	**66.3k**	71.4k	82.0k
Call Start	116.8	128.0	**116.2**	121.8	133.7	142.0
In-call Update	**111.9**	156.2	141.9	145.2	162.7	171.0
Task Update	**38.6k**	56.8k	55.4k	52.1k	60.2k	64.0k
Norm. Avg.	**0.74**	0.90	0.84	0.83	0.93	1.00

Table 11: WAPE (%) at four prediction points. Best and second-best results are in bold and underlined, respectively. Marks are assigned using unrounded values.

Agent LLM	Prediction point	Mean target (k)	TokenCast	TRAIL	EGTP	TIE	Self-Pred.
			SWE-bench Verified				
GPT-5.4	Task Start	475.6	**30.3**	34.8	33.8	33.0	32.0
	Call Start	19.6	**0.329**	0.363	0.359	0.355	0.362
	In-call Update	20.1	**0.194**	0.393	0.371	0.385	0.399
	Task Update	286.3	**27.9**	40.2	40.9	41.6	44.0
Qwen3.8-27B	Task Start	611.5	**31.4**	37.9	34.2	35.2	36.1
	Call Start	25.4	**0.343**	0.429	0.405	0.390	0.415
	In-call Update	25.9	**0.316**	0.463	0.448	0.430	0.420
	Task Update	366.3	**33.9**	48.0	46.7	44.8	47.2
			Search-R1				
GPT-5.4	Task Start	53.5	**63.9**	70.7	70.3	68.0	69.0
	Call Start	6.0	0.575	0.622	0.608	**0.567**	0.663
	In-call Update	6.1	**0.513**	0.646	0.631	0.603	0.654
	Task Update	31.6	**55.7**	73.1	75.0	73.7	76.6
Qwen3.8-27B	Task Start	98.1	**39.8**	54.5	50.8	48.5	52.0
	Call Start	9.5	**0.597**	0.694	0.752	0.713	0.801
	In-call Update	9.8	**0.582**	0.863	0.823	0.770	0.849
	Task Update	57.6	**41.7**	68.6	59.0	61.5	66.7
			MMLU-Pro				
GPT-5.4	Task Start	50.6	15.8	17.4	16.8	16.2	**14.6**
	Call Start	8.2	**0.180**	0.204	0.189	0.193	0.215
	In-call Update	8.4	**0.110**	0.233	0.218	0.199	0.207
	Task Update	27.9	**15.4**	22.2	23.3	21.5	20.8
Qwen3.8-27B	Task Start	63.8	20.4	22.4	**19.6**	21.5	20.8
	Call Start	10.0	**0.265**	0.303	0.286	0.279	0.311
	In-call Update	10.3	**0.205**	0.366	0.345	0.333	0.318
	Task Update	35.2	**23.9**	31.8	29.3	30.4	32.7
			LongBench-v2				
GPT-5.4	Task Start	771.2	**4.2**	4.8	4.7	4.6	4.2
	Call Start	135.7	**0.039**	0.045	0.042	0.040	0.045
	In-call Update	136.6	**0.026**	0.054	0.050	0.046	0.051
	Task Update	391.2	**3.6**	6.9	6.2	6.4	7.2
Qwen3.8-27B	Task Start	856.5	**5.9**	7.1	6.5	6.4	6.1
	Call Start	112.1	0.084	0.096	**0.084**	0.089	0.099
	In-call Update	112.6	**0.084**	0.121	0.114	0.108	0.105
	Task Update	434.2	**6.2**	11.0	9.8	10.2	11.3

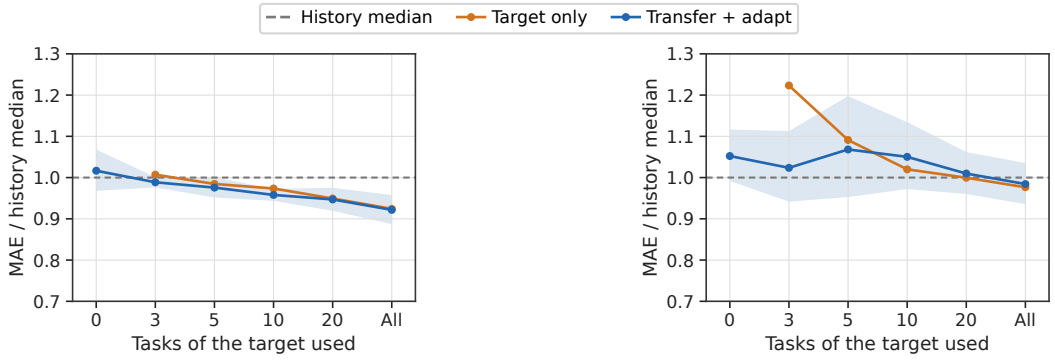
D.4 GENERALIZATION

D.4.1 UNSEEN TASK TYPES

We use the publicly released LiveClawBench trajectories (Long et al., 2026), which contain tasks from 10 application domains executed under a shared agent framework. For each agent LLM, we perform leave-one-domain-out evaluation, treating one domain as the unseen task type and training on the remaining domains. Zero-shot transfer uses no trajectories from the held-out domain. For adaptation, we add k held-out-domain tasks to the training set and evaluate on the remaining held-out tasks. The target-only baseline is trained on the same k tasks without source-domain data. All runs of a task remain in one partition. Relative to Self-Prediction, zero-shot MAE is 1.31 at Call Start and 1.47 at Task Update. With 10 target tasks, the ratios fall to 0.86 and 0.91; with 20, they reach 0.82 and 0.85. The trajectory-selection criteria are in Appendix C.1.

D.4.2 UNSEEN AGENT LLMS

We hold out each agent LLM in turn and train on trajectories from the remaining models under a shared task-collection and agent framework. Adaptation uses k tasks executed by the held-out model, with task partitions shared across models and adaptation tasks disjoint from the test tasks. At Call Start, zero-shot MAE is 1.02 times the target history median; this ratio falls to 0.96 with 10 target-model tasks and 0.95 with 20. With 3–10 tasks, the transferred predictor achieves lower MAE than training on the same target-model tasks alone. Figure 7 also reports an auxiliary Task Update evaluation of remaining output tokens, a different target from total remaining consumption. Its normalized MAE ranges from 1.01 to 1.07 with up to 20 target-model tasks and reaches 0.98 when all target-model tasks are available. The target-only predictor starts at 1.22 with three tasks and approaches the history median as additional target-model trajectories are provided.



(a) Call Start

(b) Task Update, output

Figure 7: Transfer to unseen agent LLMs with increasing amounts of target-model data. The panels report Call Start and remaining output-token prediction at Task Update. MAE is normalized by the target history median; lower is better.

D.4.3 CROSS-HARNESS TRANSFER

Table 12 reports TokenCast's MAE relative to the history median under each test harness. It also reports transfer from DeepSeek Harness training runs to OpenHands runs of held-out tasks.

D.4.4 LENGTH EXTRAPOLATION

We sort the SWE-bench Verified tasks by their median number of calls into five bands, train on the four shorter bands, and evaluate on the longest band. TokenCast achieves lower MAE than Self-Prediction at all four prediction points on these longer tasks, with reductions of 12.4% at Task Start, 15.7% at Call Start, 44.9% at In-call Update, and 26.8% at Task Update. Table 13 additionally compares this length-based split with random splits of the same fold sizes. Relative to the random splits, MAE differs by $+7.8\%$ at Task Start, $+1.9\%$ at Call Start, -2.6% at In-call Update, and $+7.1\%$ at Task Update, so the longer test band raises MAE by at most 7.8%.

Table 12: TokenCast MAE divided by the history-median MAE on the test harness. Lower is better. The two OpenHands rows use the same test runs.

Training harness	Test harness	Task Start	Call Start	In-call Update	Task Update
DeepSeek Harness	DeepSeek Harness	0.90	0.71	0.64	0.82
OpenHands	OpenHands	0.83	0.77	0.68	0.75
DeepSeek Harness	OpenHands	1.04	0.88	0.79	0.96

Table 13: Length extrapolation to tasks longer than those observed during training. MAE changes are reported relative to Self-Prediction and matched random splits.

Prediction point	vs. Self-Prediction ↓	vs. random split ↓
Task Start	-12.4%	$+7.8\%$
Call Start	-15.7%	$+1.9\%$
In-call Update	-44.9%	-2.6%
Task Update	-26.8%	$+7.1\%$

D.4.5 Reasoning Configuration Shift

We evaluate a shift in GPT-5.4's reasoning configuration by rerunning the same 69 SWE-bench Verified tasks with reasoning effort set to off and comparing them with the corresponding runs under the low setting. Median run consumption decreases from 273k to 238k tokens. When trained and evaluated within the off configuration, TokenCast reduces MAE relative to Self-Prediction by 20.7% at Call Start, 50.8% at In-call Update, and 32.2% at Task Update (Table 14). We also transfer the predictor trained under the low configuration directly to the off configuration and adapt it using 20 target-configuration tasks. After adaptation, MAE decreases to 37.9 at Call Start, 23.6 at In-call Update, and 76.0k at Task Update.

Table 14: MAE under a GPT-5.4 reasoning-configuration shift on SWE-bench Verified at the three online prediction points. Results include evaluation within the off configuration, direct transfer from low to off, and adaptation with 20 target-configuration tasks.

	Off		Low → Off	
Prediction point	TokenCast	Self-Pred.	Direct	20 tasks
Call Start	37.1	46.8	50.7	37.9
In-call Update	22.5	45.7	33.0	23.6
Task Update	82.0k	121.0k	82.0k	76.0k

D.5 Online Update Frequency

We measure the prediction overhead of refreshing the Task Update forecast at different intervals on SWE-bench Verified with GPT-5.4. Starting from the same initial forecast, the 1-, 3-, and 5-call settings refresh the prediction after every 1, 3, or 5 completed calls, respectively, and End makes only the initial forecast. At skipped checkpoints, the latest forecast of final task consumption is retained, the tokens confirmed so far are subtracted, and the remaining estimate is clipped at zero.

Table 15 reports the mean number of forecasts and cumulative prediction time per run. Counts include the initial Task Start forecast, and times are reported as mean $\pm$ standard deviation across runs. Updating every call makes 19.7 forecasts and takes 32.8 ± 30.5 ms per run. The three-call and five-call schedules reduce these to 6.9 forecasts and 11.9 ± 10.4 ms, and 4.3 forecasts and 7.7 ± 6.4 ms, respectively. End makes only the initial forecast and takes 2.1 ± 0.8 ms. Every-call updating therefore remains a small fraction of the 129 s median run time reported in Appendix C.1.

Table 15: Prediction counts and cumulative overhead at online update intervals on SWE-bench Verified. Predictions/run includes the initial Task Start forecast and later Task Update refreshes.

Update interval	Predictions/run ↓	Time/run (ms) ↓
1 call	19.7	32.8 ± 30.5
3 calls	6.9	11.9 ± 10.4
5 calls	4.3	7.7 ± 6.4
End	1.0	2.1 ± 0.8

D.6 Ablation Study

Base predictor comparison. Table 16 compares base predictors in the same pipeline. Ridge regression and KNN reach normalized average MAEs of 0.93 and 0.97. Random Forest reaches 0.81, and the MLP reaches 0.78 with 87.2% interval coverage. The three gradient-boosting methods reach 0.75 or lower, and LightGBM gives the lowest error, 0.69, and the highest coverage, 90.6%.

The base-predictor comparison reports 0.8 ms for LightGBM inference, compared with 3.9 ms for XGBoost and 5.2 ms for CatBoost. Under every-call updating, the complete pipeline, including feature extraction and composition, makes 19.7 task-level forecasts and takes 32.8 ms per SWE-bench Verified run on average. We use LightGBM as the base predictor.

Table 16: Base predictor comparison on SWE-bench Verified (GPT-5.4). Norm. Avg. is the normalized MAE averaged over the four prediction points. 90% Cov. is the empirical coverage of the 90% prediction interval. Best and second-best results in each column are in bold and underlined, respectively. Coverage is ranked high to low and other metrics low to high.

Predictor	Norm. Avg. ↓	90% Cov. (%)	Model time (ms)
Ridge Regression	0.93	82.4	**0.2**
KNN	0.97	79.6	14.3
Random Forest	0.81	86.8	2.7
XGBoost	<u>0.73</u>	<u>89.3</u>	3.9
CatBoost	0.75	88.7	5.2
MLP	0.78	87.2	6.1
LightGBM	**0.69**	**90.6**	<u>0.8</u>

Prediction strategy. At Task Start, no segment has finished, so the prefix–suffix decomposition has no observed anchor. The compositional path has a normalized MAE of 0.87, compared with 0.82 for direct regression. At Task Update, observed segment boundaries provide information about the input baseline and context growth. The compositional path improves to 0.66, below direct regression's 0.72. Averaging the two paths reduces MAE to 0.81 at Task Start and 0.64 at Task Update. The correction model uses the direct and compositional forecasts together with the predicted boundary variables, reducing these values to 0.80 and 0.62, with 90.6% coverage (Table 17). It is trained on out-of-fold outputs of the complete pipeline.

Table 17: Effect of forecasting strategy on SWE-bench Verified (GPT-5.4). Task Start and Task Update columns report the normalized MAE at these two task-level prediction settings. Best and second-best results in each column are in bold and underlined, respectively. Coverage is ranked high to low and other metrics low to high.

Strategy	Task Start ↓	Task Update ↓	90% Cov. (%)
Direct only	0.82	0.72	88.1
Compositional only	0.87	0.66	89.3
Direct–compositional average	<u>0.81</u>	<u>0.64</u>	<u>89.8</u>
Full correction pipeline	**0.80**	**0.62**	**90.6**

Segment representation ablation. Regressing directly on raw execution features raises the normalized average MAE to 0.85, with task-level and call-level MAE of 0.88 and 0.82. Retaining the triple

while removing composition gives 0.78. This intervention also changes call-level MAE from 0.68 to 0.74, so the ablation affects more than the task-level composition readout.

Removing g raises the normalized average MAE to 0.76 and task-level MAE from 0.71 to 0.79. Removing b raises it to 0.73 (Table 18). The larger effect of removing g is consistent with its role in the cross-term $n_B g_A$; the ablations also change call-level predictions and do not isolate this term.

Table 18: Ablation of the segment representation on SWE-bench Verified (GPT-5.4). Task-level and Call-level columns report normalized MAE averaged over the two prediction points within each level. Best and second-best results in each column are in bold and underlined, respectively.

Variant	Norm. Avg. ↓	Task-level ↓	Call-level ↓
Raw features	0.85	0.88	0.82
No composition	0.78	0.81	0.74
Drop g	0.76	0.79	0.72
Drop b	<u>0.73</u>	<u>0.75</u>	<u>0.70</u>
Full	**0.69**	**0.71**	**0.68**

Component ablation. The suffix model takes predictions from the prefix model as input. A shift in this input between training and inference can therefore propagate errors through the cascade. Cross-fitting reduces the shift, while the cost-weighted loss scales errors in the coupled variables g and n by their downstream impact. Removing either component raises the normalized average MAE by 0.03–0.04. Removing both raises MAE to 0.78 and lowers coverage to 87.4% (Table 19).

Without the correction model, MAE rises to 0.74 and coverage falls to 89.0%. Removing the boundary variables raises MAE to 0.72, consistent with their role in conditioning the suffix model.

Table 19: Component ablation on SWE-bench Verified (GPT-5.4). Each row removes one component from the full pipeline. Best and second-best results in each column are in bold and underlined, respectively. Coverage is ranked high to low and other metrics low to high.

Configuration	Norm. Avg. ↓	90% Cov. (%)
Full	**0.69**	**90.6**
− Cross-fitting	0.73	88.9
− Cost weighting	<u>0.72</u>	89.4
− Correction	0.74	89.0
− Boundary vars	<u>0.72</u>	<u>90.1</u>
− Cross-fit. & cost-wt.	0.78	87.4

D.7 Budget Control

Replay and stopping rules. We replay 288 GPT-5.4 runs from 144 SWE-bench Verified tasks, with two recorded runs per task. Seven budgets are defined by the 0.3 to 0.9 quantiles of recorded run consumption. At each Task Update checkpoint, the controller adds confirmed consumption to a selected quantile of predicted remaining consumption and stops the run when the sum exceeds the budget. A replay is trace-complete if it reaches its recorded terminal state. For each method, budget, and test fold, the controller selects the 0.05, 0.5, or 0.95 forecast quantile on runs outside the test fold, choosing the one using the fewest tokens while reaching at least the fixed-budget trace-completion rate. Results are pooled over out-of-fold test runs and averaged over three split seeds.

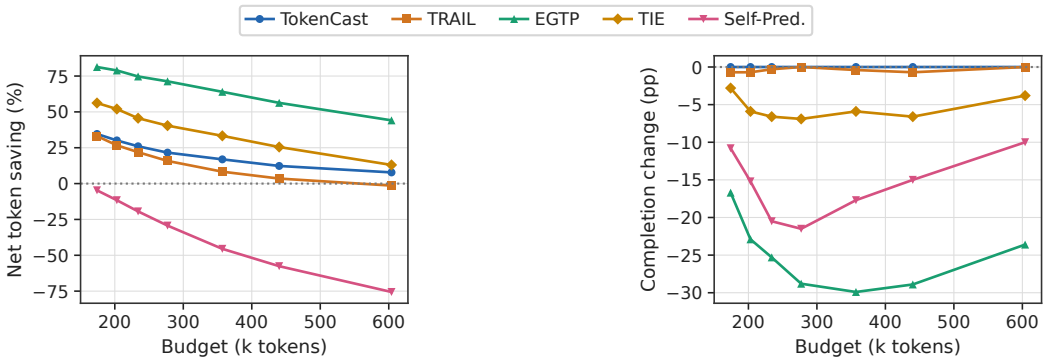
Cost accounting. A controller-stopped run is charged the consumption recorded at its stopping checkpoint. A run that finishes before the fixed cap is charged its recorded total; otherwise it is stopped at the cap, with wall time scaled by the recorded token progress. Prediction processing and time are included in the replay accounting. For encoder-based baselines, prediction tokens count locally processed text; Self-Prediction's count includes additional provider-recorded LLM usage. Savings are relative to the fixed-budget strategy at the same budget. Table 20 lists the complete results, Figure 8 shows the budget-wise response, and Table 21 compares prediction overhead.

Table 20: Budget control on 288 GPT-5.4 runs from 144 SWE-bench Verified tasks. Tokens are in thousands per run, wall time is in seconds per run, and trace completion and savings are in percent. Only TokenCast matches the fixed-budget trace completion at every budget.

Budget (k tokens)	Method	Trace-complete (%)	Execution (k/run)	Prediction (k/run)	Total (k/run)	Time (s/run)	Saving (%)
174	Fixed budget	30.2	154.7	0.0	154.7	113.7	–
	TokenCast	30.2	101.1	0.0	101.1	76.5	34.6
	TRAIL	29.5	91.0	12.5	103.5	73.0	33.1
	EGTP	13.5	25.3	3.6	28.9	22.6	81.3
	TIE	27.4	59.8	8.0	67.8	45.6	56.2
	Self-Pred.	19.4	70.0	92.0	162.0	303.5	-4.7
203	Fixed budget	39.9	173.5	0.0	173.5	125.3	–
	TokenCast	39.9	121.2	0.0	121.2	90.0	30.1
	TRAIL	39.2	112.6	14.5	127.1	87.6	26.7
	EGTP	17.0	31.8	4.8	36.6	27.6	78.9
	TIE	34.0	73.5	9.7	83.2	54.9	52.0
	Self-Pred.	24.7	84.5	109.2	193.7	355.6	-11.6
234	Fixed budget	50.0	190.7	0.0	190.7	135.8	–
	TokenCast	50.0	141.2	0.0	141.2	103.1	26.0
	TRAIL	49.7	132.3	16.7	149.0	101.0	21.9
	EGTP	24.7	42.7	5.6	48.3	34.8	74.7
	TIE	43.4	92.3	11.4	103.7	67.1	45.6
	Self-Pred.	29.5	100.2	127.5	227.7	413.3	-19.4
277	Fixed budget	59.7	210.4	0.0	210.4	148.5	–
	TokenCast	59.7	165.0	0.0	165.0	118.1	21.6
	TRAIL	59.7	157.3	19.9	177.2	118.7	15.8
	EGTP	30.9	53.4	6.9	60.3	42.1	71.3
	TIE	52.8	111.5	13.8	125.3	80.1	40.4
	Self-Pred.	38.2	120.4	151.7	272.1	481.7	-29.3
357	Fixed budget	69.8	237.4	0.0	237.4	165.1	–
	TokenCast	69.8	197.2	0.0	197.2	138.3	16.9
	TRAIL	69.4	193.9	23.8	217.7	144.2	8.3
	EGTP	39.9	76.0	9.5	85.5	58.9	64.0
	TIE	63.9	141.2	17.1	158.3	98.8	33.3
	Self-Pred.	52.1	154.8	190.5	345.3	593.9	-45.5
440	Fixed budget	79.9	258.2	0.0	258.2	177.2	–
	TokenCast	79.9	226.5	0.0	226.5	156.4	12.3
	TRAIL	79.2	222.3	26.9	249.2	163.1	3.5
	EGTP	51.0	100.2	12.7	112.9	77.3	56.3
	TIE	73.3	171.5	20.8	192.3	119.6	25.5
	Self-Pred.	64.9	184.3	222.5	406.8	688.3	-57.6
604	Fixed budget	89.9	280.7	0.0	280.7	191.3	–
	TokenCast	89.9	258.8	0.0	258.8	176.4	7.8
	TRAIL	89.9	254.1	30.4	284.5	184.2	-1.4
	EGTP	66.3	140.3	16.6	156.9	104.3	44.1
	TIE	86.1	218.6	25.6	244.2	150.4	13.0
	Self-Pred.	79.9	224.6	268.3	492.9	820.8	-75.6

Table 21: Budget-equal-weighted replay means. Trace completion is shown alongside overhead because lower processing totals can result from stopping more runs. Encoder-based baselines count locally processed input tokens; Self-Prediction counts additional LLM usage. These token counts therefore represent different resources.

Method	Trace-complete (%)	Prediction tokens	Total tokens	Total time
TokenCast	59.9	0.0k	173.0k	122.7 s
TRAIL	59.5	20.7k	186.9k	124.5 s
EGTP	34.8	8.5k	75.6k	52.5 s
TIE	54.4	15.2k	139.3k	88.1 s
Self-Prediction	44.1	166.0k	300.1k	522.4 s



(a) Net token saving

(b) Completion-rate change

Figure 8: Trace completion and token consumption across seven budgets in the offline replay.

E Case Studies

Token consumption reflects both the actions needed to finish a task and the context carried into each call. Two executions connect these quantities to observable events: verification after an edit and artifact preparation with a long context. Task Update estimates are displayed as final totals, $\widehat{T}_k = S_k + \widehat{R}_k$, where S_k is confirmed consumption. Call-level estimates retain the full current-call target C_k. Dashed lines mark retrospective targets. Grouped bars show Call Start forecasts from TokenCast and Self-Prediction for full current-call consumption.

E.1 Code Repair

GPT-5.4 repairs a `swap_dims()` mutation issue in `pydata__xarray-6938` with DeepSeek Harness. The run contains 21 calls and 39 checkpoints, consumes 243,371 tokens, and lasts 130 s. The following timeline, detail plots, and table show how forecasts change during source editing, verification failure, retry, and completion.

Execution timeline. Figure 9 plots confirmed consumption over wall time. Task Start is at 0 s and the first Call Start 1 ms later, when the first request is assembled. The In-call Update shown is checkpoint 5 of call 12 at 65.3 s, with 657 bytes committed and 95,591 tokens confirmed. The Task Update after call 12 is at 68.7 s, with 109,011 tokens confirmed and 134,360 still to come.

Generated prefix. Figure 10 and Table 22 report forecasts at selected prediction points. Call 12 performs the source edit and consumes 13,420 tokens. At 795 generated bytes, the prefix exposes the original source span. At 1,450 bytes, it reveals the replacement text. Across these checkpoints, the 90% interval contracts from 911 tokens at Call Start to 65 tokens at 1,450 bytes, while the forecast remains close to the recorded 13,420-token call cost.

Verification feedback. At call 14, reproduction encounters the removed NumPy `unicode_alias`, and the predicted task total rises to 281,746 tokens. After a compatibility workaround allows the non-mutation assertions to pass at call 15, the forecast becomes 252,381 tokens, close to the recorded

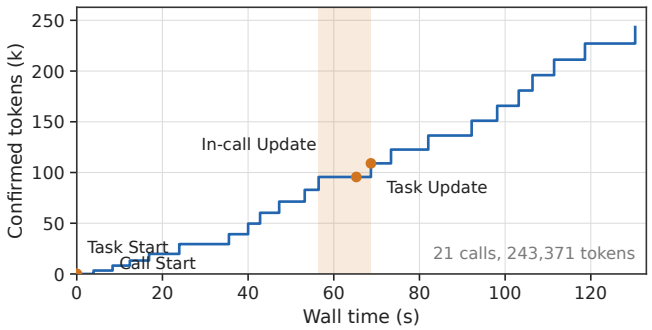


Figure 9: Confirmed tokens over wall time for one run. The shaded span is call 12.

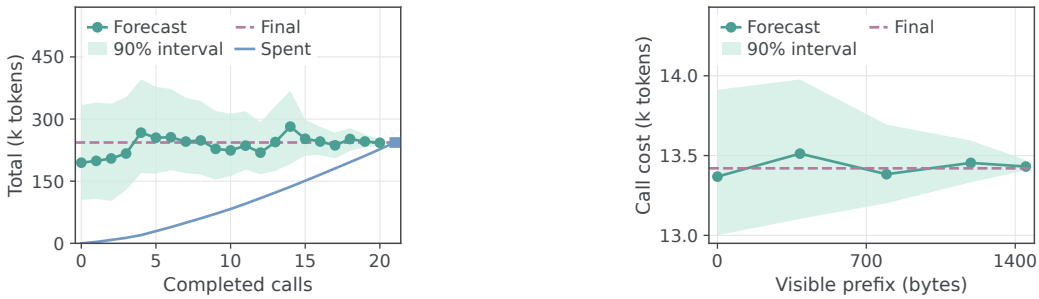


Figure 10: Forecast updates during code repair. Left: task-level forecasts following verification outcomes. Right: current-call forecasts during edit call 12 as generation becomes visible.

total of 243,371. The execution still consumes another 92,329 tokens while inspecting the diff, preparing artifacts, and submitting the response, accounting for 37.9% of the final total.

Table 22: Forecasts at selected points in the code-repair execution. Token quantities are in thousands. Recorded targets are shown retrospectively.

Prediction point	Available evidence	Target	Forecast [90% interval]	Recorded
Task Start	Issue and configuration	T	194.638 [104.782, 333.912]	243.371
Call Start, 12	Edit request assembled	C_{12}	13.368 [13.001, 13.912]	13.420
In-call, 12	795 bytes, old source span visible	C_{12}	13.383 [13.201, 13.694]	13.420
In-call, 12	1,450 bytes, replacement text visible	C_{12}	13.431 [13.407, 13.472]	13.420
Update, $k = 14$	NumPy alias prevents reproduction	T	281.746 [191.304, 368.259]	243.371
Update, $k = 15$	Compatibility workaround, assertions pass	T	252.381 [211.623, 298.576]	243.371

E.2 LONG-CONTEXT QA

File operations. Qwen3.8-27B answers a LongBench question about OpenLRM and Instant3D using OpenHands. During call 1, it generates option C and its justification, then encounters a file-creation error because `answer.md` already exists. Inspecting and replacing the placeholder, preparing the patch, and checking the artifacts extend the execution to six calls. Successful replacement at call 3 is followed by another 395,353 tokens of consumption. Figure 11 and Table 23 show the corresponding task-level and call-level forecasts.

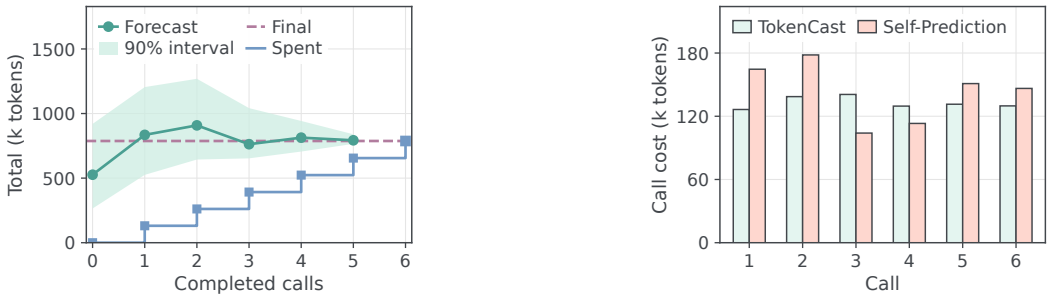


Figure 11: Repeated input makes additional calls expensive. Left: task updates after a failed write and artifact preparation. Right: full-call Call Start forecasts from TokenCast and Self-Prediction.

Table 23: Long-context stages. All token quantities are in thousands. The same answer is carried through the subsequent file operations.

Point	Newly available evidence	S_k	Forecast T [90% interval]
Task Start	Long-context question and output requirements	0.000	526.384 [264.731, 919.648]
Update, $k = 1$	Answer generated, file creation fails	131.059	834.719 [525.193, 1204.762]
Update, $k = 2$	Existing answer placeholder inspected	261.368	908.362 [643.881, 1268.997]
Update, $k = 3$	Answer file successfully replaced	392.112	762.541 [652.967, 1041.863]
Update, $k = 5$	Answer and patch read back	655.063	792.813 [765.208, 839.426]
Termination	Finish action recorded	787.465	—

Repeated input. The first call processes 129,714 input tokens. Subsequent calls process 130,181–132,209 input tokens and emit 128–350 output tokens each. Input contributes 785,146 of 787,465 tokens (99.7%). For the recorded execution, the segment decomposition gives

$$T = nL_1 + b = 6 \times 129{,}714 + 9{,}181 = 787{,}465.$$

The repeated-input term contributes 778,284 tokens. Each additional call therefore carries roughly 130,000 input tokens, so the remaining call count and the input boundary jointly determine the task cost. The failed write at call 1 signals extra calls at this context length, and the Task Update forecast rises from 526k at Task Start to 835k tokens.

F SELF-PREDICTION PROMPT

The following prompt template adapts the zero-shot agent self-prediction protocol of Bai et al. (2026) to the four prediction points used in our evaluation. It retains environment inspection, workload analysis, and phase-wise input/output token estimation. At Task Start, the estimator may inspect the initial environment before predicting complete-run consumption. At the other three prediction points, it receives the observed execution record and the target-specific evidence available at that point without advancing the task. Missing fields are marked as `not available`.

Self-Prediction prompt template

```
Estimate the token consumption of the agent execution described below. Base your
    prediction on the task requirements, the specified model and agent configuration,
    and the evidence available at the supplied prediction point. Your estimate should
    describe the consumption of the agent operating under its existing workflow and
    execution limits. Your deliverable is a token-cost estimate. Do not implement a
    solution, modify task files, or submit an answer to the underlying task.

Prediction target

First, determine which part of the execution the estimate must cover. At Task Start,
    estimate all input and output tokens that a complete run would consume from the
    original task state until termination. The estimate includes the exploration,
    solution development, and verification that the run would require. Any inspection
    performed specifically to prepare this estimate belongs to the estimation session
    and is excluded from the predicted consumption. This inspection may improve your
    understanding of the task; however, the predicted run still begins from its
    original state.

At Task Update, estimate the input and output tokens that will be consumed after the
    latest completed call until the task ends. Completed calls provide evidence of
    progress and consumption, but their recorded usage is outside of this remaining-
    task target. Their messages and tool results may still appear in later requests;
    reading that retained content in a later request contributes new input usage within
     the prediction scope.

At Call Start, estimate the complete input and output consumption of the current call,
    including retries of its request. At In-call Update, estimate the same complete-
    call quantity using the generation observed so far. This includes the input,
    supplied output prefix, continuation, and any usage attributable to retries within
    the call. A call comprises one initial model request and any retry requests issued
    before tool feedback is received. A subsequent request made after receiving tool
    feedback belongs to a later call and is outside the call-level estimate.

A task ends when the agent finishes, fails, or reaches an execution limit. Account for
    the termination behavior supported by the available evidence. Successful completion
     is one possible outcome, and repeated failures or exhausted limits may end the run
     earlier. Execution limits constrain the forecast, but they are not estimates of
    how much work the agent will perform.

Understanding the task and execution evidence

Read the task statement together with the agent instructions and completion conditions.
     Determine what the agent must establish, produce, or verify before it can complete
     the task. Consider how the available tools and workflows organize the work into
    model calls. Distinguish the number of tool actions from the number of model calls:
     several actions may be issued in one response, and a single unresolved issue may
    require multiple responses.

At Task Start, use the available inspection tools to understand the initial environment
     while leaving task files unchanged. For a coding task, inspect the relevant source
     files, their dependencies, and the tests associated with the requested behavior.
    Assess whether the work appears localized or spans several components, whether the
    expected behavior is clearly specified, and how much investigation is likely to
    precede a change. For retrieval or document-based tasks, examine the supplied
    materials and resource descriptions to assess what evidence is already available
    and what additional information the agent would need to obtain. Focus the
    inspection on uncertainties that materially affect the expected workload.

At the other prediction points, use the supplied execution record without advancing the
     task or obtaining additional tool results. Read the actions together with their
    observed outcomes. A proposed edit, a completed edit, and a successful test provide
     different evidence of progress. Identify what has been established, what remains
    unresolved, and which results are still pending. Treat unavailable information as
    unknown; the absence of a recorded failure does not establish success.
```

```
For a task-level forecast, form a plausible continuation from the current state. A
    coding run may still need to investigate a failure, revise an implementation, run
    relevant checks, and prepare its submission. A retrieval task may require further
    searches because the current evidence covers only part of the question. A reasoning
     task with sufficient information may complete in one response. Use phases that fit
     the actual task and the observed state. At Task Update, include only phases or
    portions of phases that remain.

Use the execution history to assess both progress and the cost of further work. Recent
    calls can indicate typical response lengths, request growth, and the amount of work
     the agent accomplishes per interaction. Compare calls with similar roles where
    possible. Repeated searches, recurring test failures, or several calls without
    resolving an outstanding issue may indicate additional investigation or revision. A
     sequence of completed checks may indicate that only final verification or
    submission remains. Relate these observations to the work still required before
    estimating the remaining call count.

For a call-level forecast, focus on what the current request asks the model to generate
    . Determine whether the response is likely to contain a short tool invocation,
    several tool-call arguments, a substantial code fragment, an explanation, or a
    final answer. At the In-call Update, examine how much of that response has already
    been produced and what remains unfinished. Use the supplied timing information as
    supporting evidence when it is informative, while keeping elapsed time distinct
    from token counts.

Constructing the token estimate

For task-level predictions, the forecast execution is divided into a small number of
    non-overlapping phases. For each phase, the likely number of model calls, the input
     those calls will receive, and the output they will generate should be assessed.
    Let the phase estimates reflect the actual continuation you expect. Final
    verification or submission may require only one additional call. For the call-level
     prediction, a single current_call phase is sufficient.

Estimate the input consumption from the content submitted for each model request. This
    can include system and agent instructions, task statements, tool definitions,
    retained conversation history, content generated from earlier calls, and tool
    results. Use an exact supplied input token count when one is available. When later
    requests have not yet been assembled, estimate their size from the current context,
     expected additions, and specified context retention behavior.

Account for retained content each time it is submitted. A tool result added early in a
    run may appear in several subsequent requests; therefore, its contribution depends
    on both its size and the number of later calls that retain it. Likewise, additional
     investigation can increase the call count and enlarge the context of those calls.
    Reflect both effects in the phase estimates. Apply context pruning, summarization,
    or compaction only when supported by the supplied workflow or observations.

Estimate output consumption from the responses expected within each phase. Include
    generated text, code, tool-call arguments, and reasoning tokens according to the
    supplied accounting convention. The final answer may be short, even when
    intermediate responses consume substantial tokens. When reasoning tokens are
    already included in the reported output total, count them only once. Tool-generated
     search results, file contents, and test logs contribute model input when submitted
     to a later request; their production by a tool is not itself an LLM output.

Use confirmed usage as an accounting anchor. Within the selected scope, include each
    confirmed input or output count exactly once. At In-call Update, the visible prefix
     is already part of the complete output being predicted; add only the expected
    continuation to its counted contribution. Do not add the prefix again when a
    supplied cumulative output count already includes it. Character and byte counts are
     observations about text length, not exact token counts; use the supplied token
    measurements or accounting information where available.

Account for retries within the call according to the request history, observed failures
    , and the configured retry policy. The recorded usage from retries that have
    already occurred within the prediction scope is included. Estimate further retry
    consumption only to the extent supported by the current conditions. The maximum
    retry allowance defines a limit and does not imply that every retry will occur.

Combine these estimates into the best-supported forecast. Consider whether the evidence
     supports a straightforward continuation, additional revision cycles, or early
    termination, and let this assessment inform the predicted workload. Avoid adding
    unexplained safety margins to each phase.

Preparing the final output
```

```
Return a non-negative integer for each token estimate. The phase input estimates must
    sum to predicted_input_tokens, and the phase output estimates must sum to
    predicted_output_tokens. Their combined sum must equal the predicted_total_tokens.
    Check that the result covers the requested scope and includes its confirmed usage
    without duplication. A complete call estimate must not fall below the confirmed
    consumption of that call. A Task Update estimate excludes completed call usage and
    is zero when termination is confirmed and no further model calls remain.

Set predicted_total_tokens to the median (50th percentile) of token consumption for the
     specified prediction target. Set lower_total_tokens and upper_total_tokens to its
    5th and 95th percentiles, respectively, so that the interval targets 90% coverage.
    These quantiles describe uncertainty in token consumption within the same
    prediction scope. Return non-negative integers satisfying lower_total_tokens <=
    predicted_total_tokens <= upper_total_tokens. When confirmed consumption is
    supplied for that scope, all three estimates must be at least that amount. At Task
    Update, the scope includes only future consumption; set all three estimates to zero
     when termination is confirmed and no further model calls remain. Use one
    breakdown_by_phase entry for each phase included in the forecast, with names that
    describe the expected work. For either call-level prediction point, use
    current_call when no further decomposition is needed.

Submit a JSON object with the following fields:

{
  "predicted_input_tokens": <integer>,
  "predicted_output_tokens": <integer>,
  "predicted_total_tokens": <integer>,
  "lower_total_tokens": <integer>,
  "upper_total_tokens": <integer>,
  "breakdown_by_phase": [
    {
      "phase": "<phase name>",
      "input_tokens": <integer>,
      "output_tokens": <integer>
    }
  ]
}

Use the completion interface specified below. Submit the JSON estimate without a task
    solution or additional explanatory text. Fields marked not available provide no
    additional evidence and must not be treated as zero-valued observations.

Prediction point:
{{prediction_point}}

Task:
{{task_description}}

Difficulty, when provided:
{{task_difficulty}}

Model and agent configuration:
{{model_and_agent_configuration}}

Agent workflow and available tools:
{{agent_instructions_and_tool_descriptions}}

Execution limits and token accounting:
{{execution_limits_and_token_accounting}}

Initial environment:
{{initial_environment}}

Observed execution history and tool feedback:
{{execution_history_and_tool_feedback}}

Current request:
{{current_request}}

Generated prefix and stream timing:
{{generated_prefix_and_stream_timing}}

Confirmed usage within the prediction scope:
{{confirmed_usage_in_scope}}

Completion instruction:
{{completion_instruction}}
```

```
Estimate the consumption for the specified prediction point and submit the JSON
   estimate.
```

The `prediction_point` field selects one of the four scopes defined above. When the harness exposes a `finish` tool, `completion_instruction` requests submission of the JSON estimate through that tool. Otherwise, it requests the same object as the final response. Estimation calls and inspections performed solely for estimation are recorded as prediction overhead. The input and output fields refer to the selected scope, and therefore, `predicted_total_tokens` represents complete-task consumption at Task Start, complete-call consumption at both call-level points, and remaining-task consumption at Task Update.