RECURSIVE INFERENCE FOR TEST-TIME COMPUTE ALLOCATION IN TIME-SERIES FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

In time-series forecasting tasks, some forecasts require carrying seasonal demand *months* into the future whereas in other settings, one must anticipate changes in electricity load over the *next few hours*. Still, a pretrained forecasting model typically runs the same sequence of blocks *once* for each task. In this paper, we ask whether a small amount of additional inference budget can be utilized for tangible performance gains, in a way that leaves the model's parameters frozen. A fixed budget can be variously used to distribute compute. We explore these choices through compute programs that revisit selected consecutive blocks, control the size of each update, and determine which token representations can change. Their cost in encoder-block evaluations is known in advance. But their impact on forecast error for the particular task must be measured. Recursive inference uses a reasoning model to propose programs, read their results on historical forecasting windows, and revise subsequent proposals. Held-out later windows decide whether to deploy a program, and it then stays *fixed* throughout testing. We find that this simple idea, across 56 tasks from 14 datasets, reduces test MSE by 4.02% on average, at only 1.28 times the baseline encoder-block evaluations per forecast.

1 INTRODUCTION

Let us consider two settings in which pretrained time-series foundation models (Ansari et al., 2024; Das et al., 2024; Woo et al., 2024; Ansari et al., 2025) are already being used or evaluated on real operational data. A company (e.g., Decathlon) can use a model like Chronos-2 to forecast weekly sales across thousands of products. This information can be used both for replenishment over twelve weeks and for planning over a full year (Bruned et al., 2026). We know that ski-gloves and surfboards follow different seasonal patterns. The year-long forecast must figure out how to carry those patterns much farther forward. Now, let us consider a different temporal scale. Time-series models like Chronos-2 are also being evaluated on fifteen-minute measurements from 200 real electricity feeders (Kaas et al., 2026). Local demand and solar generation often change net-load within hours. This produces peaks and fluctuations against the usual daily pattern. The two settings put very different demands on what a model digests from the observed history.

The differences discussed above affect the inputs as well as the requested forecast horizon. But how this data is routed through the model is almost always *unchanged*. For example, each encoder block is applied once in a prescribed order. The specific setting of whether the model is following demand across seasons or responding to changes in local electricity load makes no difference. But now suppose we could give the frozen model a modest additional inference budget (Snell et al., 2025). And give it the flexibility to spend it on selected parts of the data-processing route. For example, historical windows from each task could let us try those choices where the values in intervals outside the window are already known. We could then check which use of the extra compute helps before choosing how to run the model on later forecasts (Tashman, 2000). The main question is how much unharvested flexibility is already present in the model's learned blocks.

The general principle outlined above is known to work well. Several existing proposals suggest different ways to distribute compute. Recurrent networks can learn how many steps to take (Graves, 2017). Shared transformations refine representations through repeated application (Dehghani et al., 2019; Yang et al., 2024b; Geiping et al., 2025). Pretrained language-model layers can be skipped or revisited at inference time (Sun et al., 2025; Li et al., 2026; Kapl et al., 2026). If we consider

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

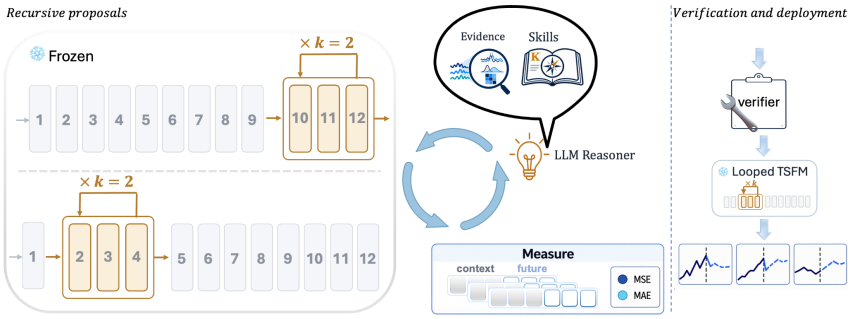


Figure 1: Overview of recursive inference. (Left) Recursive proposals: a reasoning model proposes and revises compute programs for a frozen forecaster. (Right) Verification and deployment.

applying this principle to time-series foundation models, we can take a consecutive group of frozen blocks and allow it to act again on the representation it produced. A damped update can control the size of each step. We can also go beyond layer depth. In Chronos-2 (Ansari et al., 2025), the observed context and the future occupy separate token positions. With a forecast-token-only update, repeated computation can revise the forecast-token representations while the context tokens return to their usual states. The choice of blocks, number and size of steps, and representations written back gives us "compute programs" whose cost, in encoder-block evaluations per forecast, is known in advance. Their effect on forecast error has to be learned from the task.

Thinking of these choices as compute programs gives us something meaningful to test. A program provides us a specific change to the frozen forecaster and its cost. In this way, we can compare several such programs on the same historical windows. The non-trivial step comes after those measurements are available. Suppose widening the repeated block group improves squared error but worsens absolute error, while reducing the update size moves the two errors in the opposite ways. We can try the wider group with the smaller update. But we should also consider whether either gain holds across the historical windows or comes from only a few forecasts. With room to test only a small number of programs, we need to interpret these responses and decide which change is worth attempting next. This resembles neural architecture search (Elsken et al., 2019) over the execution paths of a frozen forecasting model, but with a reasoning model using measured forecasting outcomes to choose which program to test next (Yang et al., 2024a; Liu et al., 2024a; Romera-Paredes et al., 2024). Given a description of the task and the results of earlier trials, it must propose a program with an expected effect and revises its proposals as new results arrive. We call this process *recursive inference*. Later windows determine whether a candidate should be "deployed", and then, one program remains fixed throughout the test period.

This paper. To summarize, the usual forward pass through a pretrained model is one way of using the transformations that the model has learned. In our experiments, useful extra compute does not always belong to the same part of Chronos-2 type model. Many selected programs can revisit the final blocks. On the other hand, depending on the task, other successful programs return to blocks near the input. All of these forecasts come from the same frozen checkpoint, with the execution chosen separately for each forecasting task. Specifically, the paper makes the following **contributions.** We view the execution of a pretrained forecaster into a simple task-level search problem. Compute programs suggest which frozen blocks to revisit and which token representations to update. Recursive inference uses forecasting errors to guide a reasoning model's next proposals. And later windows decide whether a proposed program should be used. Across 56 tasks from 14 datasets, this procedure reduces test MSE by 4.02% on average at 1.28 times the baseline encoder-block evaluations per forecast during deployment. The 51 tasks that depart from standard inference select 37 distinct programs. Not surprisingly, we find that giving the reasoner measured responses also improves the programs it proposes in later rounds.

2 Compute Programs for a Frozen Forecasting Model

Consider a forecasting task τ defined by a dataset and a forecast horizon H. At a forecast origin t, the model receives a history of C series, each of length L, written as $\mathbf{X}_t \in \mathbb{R}^{C \times L}$. Its target is simply the next H values of those series, $\mathbf{Y}_t \in \mathbb{R}^{C \times H}$. Before any encoder block is applied, a Chronos-2

type model turns the observed history and the requested future horizon into a sequence of token representations. Some token positions are for the observed context and others for the future. Then, a stack of encoder blocks processes this sequence. A prediction head then reads the future-token representations to generate the forecast.

Let us write tok_H for the initial token preparation and head for the prediction head. We can express the computation as

$$f_\theta(\cdot; H) = \mathrm{head} \circ B_D \circ \cdots \circ B_1 \circ \mathrm{tok}_H.$$

Here $B_1, \ldots, B_D$ are the encoder blocks whose execution we will vary. But their parameters, together with those of the token preparation and prediction head, will remain fixed. In general, during inference, we apply each block once in order. We will call this execution *Base*. For a model with D blocks, Base simply requires D encoder-block evaluations per forecast window.

2.1 Revisit a window of blocks

Let us say we are offered a choice of where to spend additional compute. We can select a consecutive window $B_a, \ldots, B_b$ and write $F = B_b \circ \cdots \circ B_a$ for the corresponding transformation. Let x be the representation entering this window. In the ordinary forward pass, F acts on x *once*. Its output then continues through the remaining blocks. We will allow this same window to act more than once before the representation moves forward. The blocks before B_a and after B_b still run once.

We can check how the output of one application becomes the input to the next. We will use a damped update. If the window is evaluated k times, its successive representations are

$$z^{(0)} = x, \qquad z^{(j+1)} = z^{(j)} + \alpha\big(F(z^{(j)}) - z^{(j)}\big), \qquad j = 0, \ldots, k-1. \tag{1}$$

Here, $\alpha \in (0, 1]$ simply controls the size of each update. The first evaluation of F is the one already present in the ordinary forward pass. Thus k counts all evaluations of the selected window, including the first one. When $k = 1$ and $\alpha = 1$, the result is $F(x)$ and the forecaster runs exactly as Base. With more evaluations, F acts on a representation based on its earlier output. A smaller α reduces the size of each change.



Figure 2: Revisiting a selected block window. After $B_1{:}B_9$ run once, the window $B_{10}{:}B_{12}$ is evaluated twice ($k = 2$). This raises the cost from 12 block evaluations (Base) to 15.

There are existing ideas suggesting the value of reusing transformations to refine a representation over several steps (Dehghani et al., 2019). Work on pretrained language models has also evaluated executions that revisit selected layers (Li et al., 2026). Here, the window is chosen for a forecasting task. Its location is relevant because an additional application near the input acts on a different representation from one near the output. The step size is also important. Even with the same window and the same number of evaluations, it can change how far the representation moves at each step. We can therefore compare executions that spend equal compute in different parts of the forecaster or alternatively, make different updates at the same location.

2.2 Controlling what the repeat computation will change

A block window also acts on token positions with different roles. For example, in Chronos-2 (Ansari et al., 2025) and similar models, some positions represent the observed context, while future positions carry the representations from which the forecast is produced. On an additional pass, we can keep the newly computed representations at every token position, or only at the future positions. We express this choice with a binary mask $\mathbf{u}$. A one writes the new representation back, and a zero keeps the representation from the ordinary first pass. An all-token update uses ones everywhere. A forecast-token-only update uses ones at future positions and zeros at context positions.

First pass and the context. The first pass through the selected window is important. It evaluates $F(x)$ on the full sequence, similar to the ordinary forecaster. Specifically, the context tokens receive their usual update. Denote that output by $c = F(x)$. The forecast-token-only mask is used from the second pass onward. Before each of those passes, the context positions are set to a chosen reference. The reference can be their values in x, before the window, or their values in c, after its ordinary pass.

The future positions retain their current values. The resulting *full* sequence is passed through F. We will write back the updated future positions and return the context positions to their ordinary values in c before execution leaves the window.

Observe that on each additional pass, F still processes the context and future tokens together. The reference determines which context representation enters that pass, and the mask determines which resulting token states are kept. This is relevant because context tokens can influence the future tokens inside F (even when their newly computed states are discarded). So, if we change the reference or the mask, it can change the forecast without changing the number of block evaluations. We measure the effect of these choices on historical windows.

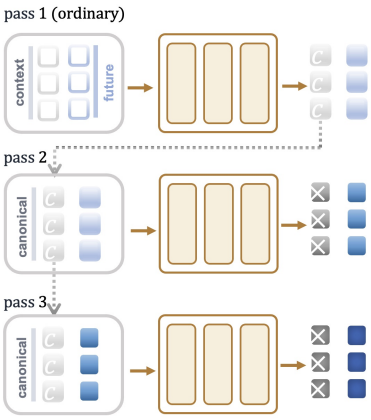


Figure 3: Forecast-token-only write-back with $k = 3$. After pass 1 gives $c = F(x)$, each later pass resets the context to the canonical reference $r = c$ and writes back only the future tokens; context outputs ($\times$) are discarded.

Specifying the program. We call one such specification a *compute program* p. It records the window $[a, b]$, the number of steps k, the step size α, and which token positions are written back. A forecast-token-only program also records its context reference. Executing p replaces the ordinary sequence of blocks with an execution Φ_p, giving

$$f_\theta^p(\cdot\,; H) = \mathrm{head} \circ \Phi_p \circ \mathrm{tok}_H. \tag{2}$$

It is important to point out that all of these programs use the same parameters θ. Base is included as the program that runs every block *once* without changing its output. Notice that the block-evaluation cost of the damped programs is known before any forecast is evaluated. If the chosen window has width $w = b - a + 1$, its first evaluation is already included in the D evaluations of Base. Each additional step costs another w evaluations. So,

$$\kappa(p) = D + (k - 1)(b - a + 1). \tag{3}$$

For example, Chronos-2 has $D = 12$ encoder blocks. Revisiting blocks B_{10} through B_{12} for two steps costs 15 block evaluations per forecast window, compared with 12 for Base. Revisiting a three-block window much earlier in the model has the same count. Changing α or the token update at either location also leaves this count unchanged. Those changes can still produce different forecasts.

The search space of programs can additionally express skipped blocks and methods that evaluate a window several times within a step. We explain these extensions in Appendix E. The programs discussed here use the damped update above and do not skip blocks. So, their costs follow (3). We will limit admissible programs by a maximum number of block evaluations per forecast.

3 Recursive Inference: Choosing a Compute Program

The cost of a compute program can be calculated before we run it, but its effect on a forecasting task cannot. A wider block window might improve one horizon and worsen another. A smaller step might correct an occasional large error while making the usual forecast slightly worse. The possible programs also exceed the trials we can afford for each task. So, we use completed forecasting windows to decide what to test next. After those trials, later windows help check if any of the resulting programs should be used (Figure 1).

3.1 Choosing the first programs

For each task, we split the series into training, verification and test segments in chronological order. The task summaries and measured responses shown to the reasoner come from the training segment.

Model guidance and task evidence. The first proposals have to be made before we know how this particular task responds to repeated blocks. We give the reasoner **two** sources of guidance. The first is a fixed text about the frozen forecaster, which we call the *skill* and denote by $\mathcal{K}$. It explains the meaning and cost of the program controls introduced in §2. It also gives conditional advice about

useful comparisons. For Chronos-2, this includes trying a limited update in the later encoder blocks, while testing at least one alternative elsewhere in the network. It advises the reasoner to compare window widths, step sizes and token updates through trials whose differences can be identified. The skill is the same for every task and every round. Appendix J describes more details.

The second source is a short summary e_τ of the task's training data. It describes temporal dependence, changes in level or volatility, and structure shared across channels. For instance, a change in recent volatility may make a shallower or narrower repeated window worth testing. Neither this summary nor the skill assigns a program to the task. They help the reasoner choose initial comparisons whose outcomes can be measured. This type of a small evaluation budget is also used in sequential model selection (Snoek et al., 2012). And naturally, searching over specified network executions also has a connection to architecture search (Elsken et al., 2019).

We use three rounds, with at most six proposals per task in each round. So, no task requires more than 18 proposed program evaluations. The first round includes contrasts such as a narrow versus a wider block window or two step sizes at the same location. At least one proposal tests a region outside the skill's favored late blocks. A mechanical check rejects programs that violate the allowed controls or the block-evaluation budget.

Using four horizons in one session. One reasoner session serves the four horizons of a dataset. It can see their task summaries and, in later rounds, the measured responses for all four horizons. A proposed program is still evaluated for its specified dataset and horizon. Each horizon will also receive its own deployment decision. In our code, the shared session is resumed across rounds, giving three reasoner calls per dataset. Appendix F details the inputs and outputs of those calls.

3.2 Revising programs from measured forecasts

Every proposal contains a program p and a rationale h_p. The rationale states the expected change in forecasting error, the observations motivating it, and an observation that would count against it. In later rounds, the reasoner should name the earlier measured program it revises. It compares that parent with relevant alternatives and explains what the next trial will resolve. Where possible, it changes one program control and keeps the others fixed, so its response is easier to interpret.

Returns from each trial. For each proposed program, we run the frozen forecaster on up to 16 non-overlapping training windows at evenly spaced origins. Their future values are already known. Base is run once on those same windows for each task, and its forecasts are reused across proposals. A pooled MSE improvement by itself can be less informative. The gain may hold across the training period, or it may come largely from one window with an unusually large error. MSE and MAE may also move in different directions. We therefore return a response with four groups of measurements:

$$\rho_p = \big(\rho_p^{\mathrm{eff}}, \rho_p^{\mathrm{agr}}, \rho_p^{\mathrm{sup}}, \rho_p^{\mathrm{con}}\big). \tag{4}$$

The first group gives the pooled effect on MSE and MAE. The second shows how the two errors move on each window. The third describes support across windows, including the fraction improved, the worst degradation, and the results in the earlier and later parts of the training segment. The fourth shows how much of the total gain comes from a few windows, including the result after removing the window with the largest gain. The exact statistics are listed in Appendix F. The reasoner also receives a summary of the candidate pool, including how many programs lower each error and how similar their window-level responses are.

The reasoner uses these measurements to prepare another set of trials. Let d denote a dataset, and let $\mathcal{T}_d$ contain its four forecasting tasks, one for each horizon. In round r, the shared reasoner session proposes a batch $\mathcal{P}_{r,d}$ of task-labeled programs (τ, p). The history $\mathcal{H}_{r-1,d}$ contains earlier proposals and their measured training responses across all four horizons. We write the next proposal step as

$$\mathcal{P}_{r,d} = \mathcal{A}\big(\mathcal{K}, \{e_\tau\}_{\tau\in\mathcal{T}_d}, \mathcal{H}_{r-1,d}\big). \tag{5}$$

Here $\mathcal{A}$ is the reasoning model's proposal procedure. After the programs in round r have been evaluated, we add each (τ, p), its rationale $h_{\tau,p}$, and its measured response $\rho_{\tau,p}$ to $\mathcal{H}_{r,d}$. For a single fixed task, these are the h_p and ρ_p used above. The next round can use results from all four horizons, while each program is evaluated for its own task. Pairing language-model proposals with external evaluation is also used in program search (Romera-Paredes et al., 2024).

Figure 7 in Appendix F shows what a revision looks like on one task. For ZafNoo at horizon $H = 336$, repeating blocks B_{11} through B_{12} with $\alpha = 0.5$ gives training MSE and MAE ratios of 0.943 and 0.986. One later trial keeps the blocks and reduces α to 0.35. Its MAE ratio improves to 0.981, while its MSE ratio rises to 0.953. Another trial retains $\alpha = 0.5$ and widens the window to B_{10} through B_{12}. Its MSE ratio falls to 0.941, but its MAE ratio rises above Base to 1.003. The next proposal combines the wider window with the smaller step. That combination has to be run, since neither earlier measurement tells us how the two changes will interact. It gives an MSE ratio of 0.941 and an MAE ratio of 0.980 on the training windows. The example shows the role of the reasoner: it uses the measured response to one change to decide which combination is worth testing next. The training result alone does not decide deployment.

3.3 Verification and deployment

The proposed programs have been chosen and revised using the training windows. Selecting the one with the lowest training error would use those windows a second time to judge proposals tailored to them. We close the candidate pool after the third round and check candidates on later verification windows. Their target intervals lie after the training segment and before the test segment. Verification results are never returned to the reasoner.

Checking candidates on later windows. Verification uses up to 12 non-overlapping windows. When a shorter verification segment supplies fewer than four such windows, we use four evenly spaced overlapping windows. The admission conditions are fixed in advance. Under the strict conditions, a candidate must have a pooled MSE ratio below 1 and an MAE ratio no greater than 1.01. It must lower MSE on at least 60% of the verification windows, and its worst per-window MSE ratio must be no greater than 1.10. At least four windows are required. If no program meets these conditions, a relaxed check retains the pooled MSE, MAE and minimum-window requirements while dropping the support and worst-window limits. Appendix E gives the full verification rule.

Candidates are considered in batches of four, ordered by their training MSE. Within each tier, verification stops at the first batch containing an admissible candidate. We select the admitted program in that batch with the lowest verification MSE, using MAE and then the program key to break ties. If neither tier admits a candidate, the task keeps Base. The resulting choice $\hat{p}_\tau$ is fixed for the entire test period of task τ. The reasoner makes no further proposals from test outcomes.

Compute for selection and deployment. Testing programs also consumes forecaster compute. Let C^F_{alloc} count the block evaluations used on training and verification windows, including evaluations of Base. Once $\hat{p}_\tau$ has been selected, each subsequent forecast costs $\kappa(\hat{p}_\tau)$ block evaluations. If the program serves N forecasts, the forecaster cost per forecast, including its share of allocation, is

$$\bar{C}^F(N) = \kappa(\hat{p}_\tau) + \frac{C^F_{\text{alloc}}}{N}. \tag{6}$$

The first term is paid on every forecast. The second depends on how many forecasts reuse the task's chosen program. Reasoner calls and tokens are counted separately because they have different units from encoder-block evaluations. We do not assume a particular deployment length N, so the experiments report the cost of finding a program and the cost of running it separately.

4 Test-Time Computation as Allocation

Our experiments ask whether test-time computation can be treated as an allocation problem rather than a fixed property of a frozen forecaster: what the method gains and costs, where it spends compute (Section 4.1), what shapes deployment (Section 4.2) and what revisits change (Section 4.3).

Setup. We evaluate 56 tasks from 14 multivariate datasets with 5 to 2,000 channels. Ten use horizons $H \in \{96, 192, 336, 720\}$ with lookback $L = 512$, and four shorter ones $H \in \{24, 36, 48, 60\}$ with $L \in \{36, 104\}$. Each series is split chronologically into training, verification and test segments (Appendix C). The forecaster is a frozen Chronos-2 with $D = 12$ encoder blocks, and the reasoner is GPT-5.6 Sol with one skill for all tasks. Each task runs $R = 3$ rounds of at most $m = 6$ proposals on up to 16 training windows, with up to 12 verification windows (Appendices E and F). We report the relative change in test MSE and MAE against Base (negative is better), averaged uniformly

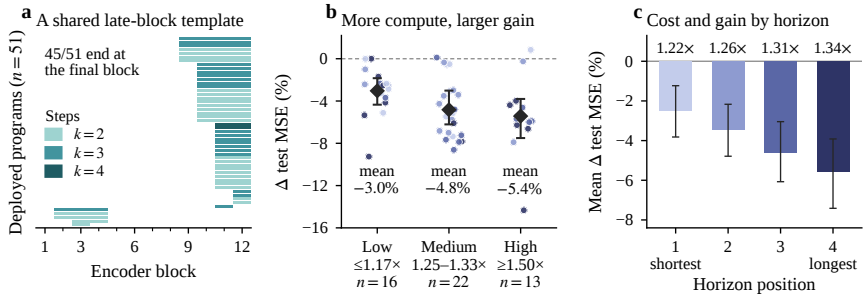


Figure 4: Allocation of deployed compute. (a) Windows of the 51 deployed programs, shaded by k. (b) Test-MSE change by cost level, colored by horizon; diamonds: means. (c) Mean change by horizon within datasets (Base as zero); labels: mean cost. Bars: 95% bootstrap CIs over datasets.

over tasks, counting tasks that keep Base as zero, and 95% intervals from a bootstrap over datasets (Appendix D). Deployment cost is $\kappa(\hat{p}_\tau)/D$, the block evaluations per forecast relative to Base.

Main results. Recursive inference lowers error broadly at 1.28 times Base compute. Averaged over the 56 tasks, it reduces test MSE by 4.02% relative to Base (95% CI 2.83 to 5.22) and test MAE by 0.98% (95% CI 0.50 to 1.54; Table 1). The gains are broad: MSE decreases on 48 of the 56 tasks and MAE on 45. Removing any dataset moves the mean MSE reduction by about 0.3 percentage points. Every task runs the same frozen forecaster, so the gaps across datasets raise a question for Section 4.1: where, and in what form, do the deployed programs spend extra compute?

Allocation cost. Allocation is a one-time cost that amortizes over deployment. It costs a median of 5,570 block evaluations per task (C^F_{alloc}, Section 3.3), the equivalent of 464 Base forecasts. Spread over the test period, it adds a median of 0.12 times Base per forecast to the mean deployment cost of 1.28 times Base. Tasks with short test periods carry a larger share. Over three rounds, the reasoner's inputs per task total about 5×10^4 tokens and its responses about 4×10^3 tokens (Appendix I).

4.1 How is computation allocated across tasks?

Structure without a fixed rule. The deployed allocations are structured but do not collapse to a single program (Figure 4a). Of the 51 programs selected in place of Base, 45 end at the final encoder block, consistent with the skill's late-region prior (Section 3.1). Yet five tasks deploy a window within the first four blocks, among them NYSE at $H = 36$ with a 7.9% reduction in MSE. Within the shared structure, the 51 tasks use 37 distinct programs, and no program serves more than six of them. The programs differ in window placement and width, in the number and size of steps, and in the scope of the write-back. The skill thus supplies a reusable structural prior, and recursive reasoning uses task-specific evidence to instantiate and refine it.

Table 1: Test error and allocation of recursive inference (RI) on Chronos-2, averaged over each dataset's four horizons. Δ is the mean horizon-level relative change against Base, not the ratio of the averaged errors; gains in color. NB: non-Base horizons; Cost: mean κ/D (Base $= 1$).

	MSE ↓			MAE ↓			Alloc.			MSE ↓			MAE ↓			Alloc.	
Dataset	Base	RI	Δ%	Base	RI	Δ%	NB	Cost	Dataset	Base	RI	Δ%	Base	RI	Δ%	NB	Cost
AQShunyi	0.783	0.746	−4.72	0.514	0.510	−0.77	4/4	1.35	ETTm2	0.280	0.268	−4.11	0.311	0.307	−1.00	4/4	1.42
AQWan	0.872	0.839	−3.79	0.496	0.495	−0.29	4/4	1.44	NN5	0.817	0.764	−6.60	0.610	0.590	−3.32	4/4	1.29
Covid-19	1.382	1.382	+0.00	0.040	0.040	+0.03	1/4	1.04	NYSE	0.571	0.536	−5.10	0.475	0.461	−2.49	4/4	1.46
CzeLan	0.222	0.218	−1.65	0.225	0.225	−0.09	3/4	1.08	Weather	0.258	0.235	−7.83	0.252	0.250	−0.89	4/4	1.31
ETTh1	0.418	0.410	−1.85	0.412	0.413	+0.16	3/4	1.10	Wike2000	496.650	470.066	−5.37	1.031	1.029	−0.12	4/4	1.38
ETTh2	0.352	0.349	−0.60	0.377	0.376	−0.31	4/4	1.31	Wind	1.253	1.206	−3.37	0.765	0.755	−1.13	4/4	1.38
ETTm1	0.387	0.371	−4.04	0.372	0.367	−1.34	4/4	1.19	ZafNoo	0.607	0.562	−7.31	0.444	0.434	−2.22	4/4	1.21
All 56 tasks: MSE **−4.02%** (48/56 improved) MAE **−0.98%** (45/56 improved) NB **51/56** Cost **1.28×**																	

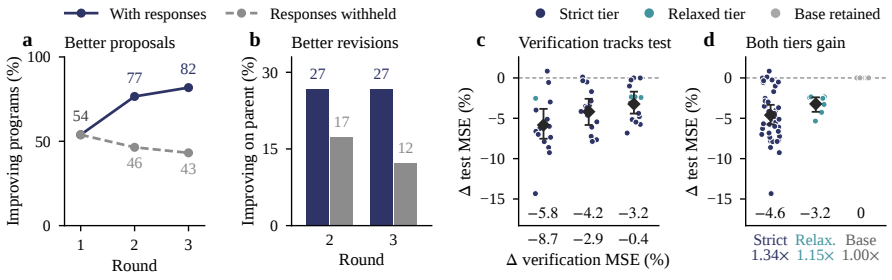


Figure 5: Measured responses improve the candidate pool, and verification gains track test gains. (a, b) Two runs share round 1; only one then shows measured responses. Task-averaged share of proposals lowering both training MSE and MAE relative to (a) Base and (b) their parent (named, or the closest earlier). (c, d) Change in test MSE per task; diamonds: means; bars: 95% CIs. (c) Deployed programs by verification-MSE tercile. (d) All tasks by deployment decision.

Compute without a fixed multiplier. Nor do the allocations collapse to a single compute level. Deployed costs range from 1.08 to 1.67 times Base, and programs that spend more tend to reduce MSE more (Spearman $\rho = -0.37$, 95% CI -0.65 to -0.06). The mean reduction rises from 3.0% for programs costing at most 1.17 times Base to 5.4% for those costing at least 1.5 times Base (Figure 4b). A within-dataset comparison across horizon positions shows the same pattern (Figure 4c). From each dataset's shortest horizon to its longest, the mean cost rises from 1.22 to 1.34 times Base. Over the same range, the mean reduction in MSE more than doubles, from 2.5% to 5.6%. The resulting allocations thus follow neither a single recipe nor a single compute level: they vary task by task in both where computation is placed and how much is spent. Section 4.2 asks what measured responses contribute to the candidates from which these programs are chosen.

4.2 How do responses and verification shape deployment?

Measured responses improve the next round. We compare two runs on all 56 tasks that share the same first round, including its proposals and their measured responses. From the second round on, one run shows the reasoner these responses. The other withholds them and lists the earlier proposals with their rationales but no measured result. Skill, task evidence, budget and verification are the same, and the runs also differ in whether the reasoner session continues from the first round (Appendix F). The runs diverge once the responses are shown (Figure 5a). Over rounds 2 and 3, the share of programs that lower both training MSE and MAE is 34.3 percentage points higher with responses (95% CI 26.1 to 42.9). It is higher on 49 of the 56 tasks and equal on 6.

Measured responses make revisions more effective. Beyond tying later proposals more explicitly to earlier programs (Appendix F), measured responses improve the quality of these revisions. With responses, a larger fraction of later proposals lower both training MSE and MAE relative to the program they revise (Figure 5b). The paired difference is 12.0 points (95% CI 7.2 to 16.3). The pattern remains when the comparison is restricted to proposals that explicitly name their parent (24% against 16%). Measured responses therefore make revisions more likely to improve on their measured parents on the training windows, as recursive reasoning intends (Section 3.2).

Verification gains reflect test gains. Verification outcomes never return to the reasoner (Section 3.3). Among deployed programs, larger verification-MSE drops go with larger test-MSE drops (Spearman $\rho = 0.39$, 95% CI 0.04 to 0.67). The third with the largest verification drops reduces test MSE by 5.8% on average, against 3.2% for the third with the smallest drops (Figure 5c). This evidence often overrides the training ranking: the program ranked first in training is deployed on only 8 of the 56 tasks, and 5 tasks retain Base. Of the 51 deployed programs, 48 lower test MSE and 45 lower test MAE, and the three that raise test MSE do so by at most 0.8%. Strict-tier deployments on 44 tasks reduce test MSE by 4.6% on average at 1.34 times Base, and the 7 relaxed-tier deployments all lower it, by 3.2% at 1.15 times Base, although 2 of them raise MAE (Figure 5d).

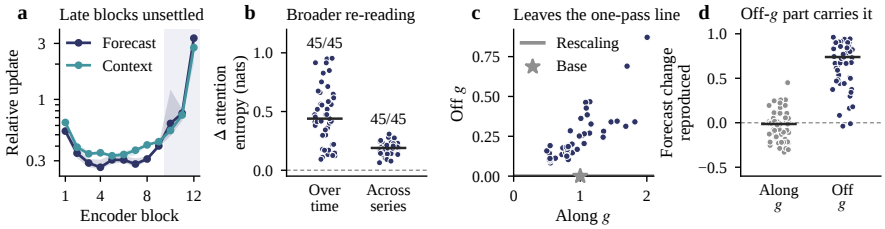


Figure 6: What a revisit changes inside Chronos-2 (training windows). (a) Relative update per block in one pass (median, IQR over tasks); shading: where 45 of 51 deployed windows end. (b) Forecast-token attention-entropy change on the revisit (45 late-window tasks; bars: medians). (c) Forecast-token displacement along and off the one-pass update $g = F(x) - x$, relative to $\|g\|$; rescaling g stays on the x-axis. (d) Forecast change reproduced by each part alone.

4.3 What revisits can change inside the forecaster?

Looping reaches computation that one pass cannot express. If a revisit only rescaled the step a window already takes, the flexibility that recursive inference allocates could be obtained without computing more. Comparing deployed programs with Base shows it does not (Figure 6; Appendix H). A single pass leaves the last blocks the least settled, and this is where most deployed windows lie (Figure 6a). On the revisit, forecast tokens query the history with their updated states and attend more broadly over time and across series (Figure 6b). The forecast state therefore leaves the line of the one-pass update (Figure 6c), and this off-line component, which no rescaling can produce, carries most of the forecast change (Figure 6d). Re-evaluating the frozen blocks on a revised state reaches a direction that one pass does not: the flexibility that recursive inference allocates.

5 Related work

Changing how a frozen model executes. Time-series foundation models such as Chronos-2 forecast zero-shot (Ansari et al., 2025). Methods serve a task by fine-tuning or reprogramming pretrained models (Zhou et al., 2023; Jin et al., 2024), adapting forecasters during deployment (Kim et al., 2025; Im & Kwon, 2026), or changing their context or surroundings (Faw et al., 2025; Gao et al., 2026), altering their parameters, inputs or outputs. Models trained to iterate reuse shared blocks (Graves, 2017; Dehghani et al., 2019; Yang et al., 2024b; Geiping et al., 2025), and pretrained language-model layers can be skipped, repeated or looped without training (Sun et al., 2025; Li et al., 2026; Kapl et al., 2026). Recursive inference instead changes how a frozen forecaster executes its own blocks, with damped, token-scoped revisits of known cost chosen per task (Appendix B).

Allocating computation from measured responses. Language-model test-time computation can be allocated by input difficulty (Snell et al., 2025; Damani et al., 2025); for forecasting, it has been scaled by iterative refinement of language-model forecasters (Le et al., 2026) or by specialist portfolios (Kayaalp et al., 2026), not by re-executing the blocks of one frozen forecaster. Language models can also propose candidates from earlier scores (Yang et al., 2024a; Liu et al., 2024a; Romera-Paredes et al., 2024). Here, candidates are interpretable execution programs with rationales, the reasoner reads multi-component responses, not a single score, and later windows held out from selection (Cawley & Talbot, 2010) decide whether any is deployed.

6 Conclusion

A single forward pass is only one way of using what a pretrained forecaster has learned. Recursive inference allocates this execution per task: a reasoning model proposes and revises compute programs from their measured responses, and later windows decide whether any is deployed. On Chronos-2, this lowers test MSE by 4.02% on average at 1.28 times Base computation, and its revisits reach directions that one pass cannot. Test-time computation can thus be allocated inside a frozen model, at a cost known in advance, by reasoning about how it computes, not only what it predicts.

AI USE STATEMENT

In this work, we used generative AI tools to assist in implementing methods and experiments, to provide feedback on experimental design, and to support the analysis and interpretation of results. We have not used generative AI tools to propose or refine hypotheses, to develop the conceptual framework, or to clean or reformat datasets, and generating synthetic data sets, formulating mathematical claims or proofs, and qualitative or thematic analysis are not applicable to this work. Additionally, we used generative AI tools to draft and edit parts of the paper, create figures, write analysis code, and identify and format references. We have reviewed all AI-assisted work, checking code and analyses against the experimental outputs and revising all AI-assisted text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI. Separately, a large language model is a component of the proposed method, and its role is described in Section 3.

ETHICS STATEMENT

This work does not involve human subjects. It uses only publicly available benchmark datasets of aggregate measurements, which contain no personal information. We therefore see no specific concerns under the ICLR Code of Ethics beyond those common to forecasting applications in general.

REPRODUCIBILITY STATEMENT

We have made the following efforts to ensure reproducibility. The method is fully specified in Section 2 and Section 3, and the experimental protocol in Section 4. The appendices describe the datasets and their splits (Appendix C), the evaluation and statistical procedures (Appendix D), the implementation and computing environment (Appendix E), and the inputs and outputs of the reasoner (Appendix F). The source code will be released upon publication.

REFERENCES

Abdul Fatir Ansari, Lorenzo Stella, Ali Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Bernie Wang. Chronos: Learning the language of time series. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=gerNCVqqtR.

Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, Mononito Goswami, Shubham Kapoor, Danielle C. Maddix, Pablo Guerron, Tony Hu, Junming Yin, Nick Erickson, Prateek Mutalik Desai, Hao Wang, Huzefa Rangwala, George Karypis, Yuyang Wang, and Michael Bohlke-Schneider. Chronos-2: From univariate to universal forecasting, 2025. URL https://arxiv.org/abs/2510.15821.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 96572–96617, 2025. doi: 10.52202/085713-3229.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20232–20242, 2022. doi: 10.52202/068431-1471.

Souhaib Ben Taieb, Gianluca Bontempi, Amir F. Atiya, and Antti Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting

competition. *Expert Systems with Applications*, 39(8):7067–7083, 2012. doi: 10.1016/j.eswa.2012.01.039.

Vianney Bruned, Abdul Fatir Ansari, Belkiss Saidi, Carlos Ramirez, Filippo Giruzzi, Ioan Catana, and Oleksandr Shchur. How Decathlon runs demand forecasting at scale with Chronos-2. AWS Blogs, Artificial Intelligence, August 2026. URL https://aws.amazon.com/blogs/machine-learning/how-decathlon-runs-demand-forecasting-at-scale-with-chronos-2/. Accessed 2026-09-25.

Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(70):2079–2107, 2010. URL http://jmlr.org/papers/v11/cawley10a.html.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In *International Conference on Learning Representations*, volume 2025, pp. 102783–102802, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ff414825df833edb8b1839e3d5d495e9-Paper-Conference.pdf.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10148–10167. PMLR, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. URL http://jmlr.org/papers/v20/18-598.html.

Matthew Faw, Rajat Sen, Yichen Zhou, and Abhimanyu Das. In-context fine-tuning for time-series foundation models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 16355–16374. PMLR, 2025.

Fuli Feng, Xiangnan He, Xiang Wang, Cheng Luo, Yiqun Liu, and Tat-Seng Chua. Temporal relational ranking for stock prediction. *ACM Transactions on Information Systems*, 37(2):1–30, 2019. doi: 10.1145/3309547.

Raphael Fischer and Amal Saadallah. AutoXPCR: Automated multi-objective model selection for time series forecasting. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 806–815, 2024. doi: 10.1145/3637528.3672057.

Wentao Gao, Jiuyong Li, Lin Liu, Thuc Duy Le, Jixue Liu, Yanchang Zhao, and Yun Chen. Lightweight wrappers for adapting time series foundation models to regional drought forecasting. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 10957–10968, 2026. doi: 10.1145/3770855.3818835.

Jan Gasthaus, Konstantinos Benidis, Yuyang Wang, Syama Sundar Rangapuram, David Salinas, Valentin Flunkert, and Tim Januschowski. Probabilistic forecasting with spline quantile function RNNs. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pp. 1901–1910, 2019.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 41340–41391, 2025. doi: 10.52202/085713-1380.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. MOMENT: A family of open time-series foundation models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16115–16152. PMLR, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983.

Klaus Greff, Rupesh K. Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skn9Shcxe.

Jeonghwan Im and Hyuk-Yoon Kwon. COSA: Context-aware output-space adapter for test-time adaptation in time series forecasting. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=L7Z5wBMPrW.

Stanisław Jastrzebski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. Residual connections encourage iterative inference. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SJa9iHgAZ.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. Time-LLM: Time series forecasting by reprogramming large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Unb5CVPtae.

Benedikt Kaas, Manuel Treutlein, Hannes Benedikt Gerber, Oliver Neumann, Cheewan Phatthanakhuha, Oliver Resch, Ralf Mikut, and Veit Hagenmeyer. Probabilistic low-voltage peak load forecasting with time series foundation models evaluated on application-oriented metrics, 2026. URL https://arxiv.org/abs/2607.01966.

Ferdinand Kapl, Emmanouil Angelis, Kaitlin Maile, Johannes von Oswald, and Stefan Bauer. From growing to looping: A unified view of iterative computation in LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=FrXKVuK0yS.

Mert Kayaalp, Ali Caner Turkmen, Oleksandr Shchur, Pedro Mercado, Abdul Fatir Ansari, Michael Bohlke-Schneider, and Bernie Wang. Test-time efficient pretrained model portfolios for time series forecasting. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=iqUMjxfDNH.

HyunGi Kim, Siwon Kim, Jisoo Mok, and Sungroh Yoon. Battling the non-stationarity in time series forecasting via test-time adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 17868–17876, 2025. doi: 10.1609/aaai.v39i17.33965.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. Remarkable robustness of LLMs: Stages of inference? In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 130050–130083, 2025. doi: 10.52202/085713-4332.

Xuan-May Le, Minh-Tuan Tran, Ling Luo, Uwe Aickelin, Dinh Phung, and Trung Le. Efficient test-time scaling for LLM-based time series forecasting. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 2437–2448, 2026. doi: 10.1145/3770855.3818201.

Yan Li, Xinjiang Lu, Yaqing Wang, and Dejing Dou. Generative time series forecasting with diffusion, denoise, and disentanglement. In *Advances in Neural Information Processing Systems*, volume 35, pp. 23009–23022, 2022.

Zhe Li, Xiangfei Qiu, Peng Chen, Yihang Wang, Hanyin Cheng, Yang Shu, Jilin Hu, Chenjuan Guo, Aoying Zhou, Christian S. Jensen, and Bin Yang. TSFM-Bench: A comprehensive and unified benchmark of foundation models for time series forecasting. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 5595–5606, 2025. doi: 10.1145/3711896.3737442.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? Learning program-of-layers in LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=pl10b6EQAN.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance Bayesian optimization. In *International Conference on Learning Representations*, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/84b8d9fcb4e262fcd429544697e1e720-Paper-Conference.pdf.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32369–32399. PMLR, 2024b.

Yong Liu, Guo Qin, Zhiyuan Shi, Zhi Chen, Caiyin Yang, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Sundial: A family of highly capable time series foundation models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 39295–39317. PMLR, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019.

George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer graph neural networks for pandemic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 4838–4845, 2021. doi: 10.1609/aaai.v35i6.16616.

R. Poyatos, V. Granda, V. Flo, M. A. Adams, et al. Global transpiration data from sap flow measurements: the SAPFLUXNET database. *Earth System Science Data*, 13(6):2607–2649, 2021. doi: 10.5194/essd-13-2607-2021. URL https://essd.copernicus.org/articles/13/2607/2021/.

Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Junyang Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S. Jensen, Zhenli Sheng, and Bin Yang. TFB: Towards comprehensive and fair benchmarking of time series forecasting methods. *Proceedings of the VLDB Endowment*, 17 (9):2363–2377, 2024. doi: 10.14778/3665844.3665863.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17456–17472, 2022. doi: 10.52202/068431-1269.

Xiaoming Shi, Shiyu Wang, Yuqi Nie, Dianqi Li, Zhou Ye, Qingsong Wen, and Ming Jin. Time-MoE: Billion-scale time series foundation models with mixture of experts. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=e1wDDFmlVu.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b623663fd9b874366f3ce019fdfdd44-Paper-Conference.pdf.

Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. Transformer layers as painters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25219–25227, 2025. doi: 10.1609/aaai.v39i24.34708.

Leonard J. Tashman. Out-of-sample tests of forecasting accuracy: an analysis and review. *International Journal of Forecasting*, 16(4):437–450, 2000. doi: 10.1016/S0169-2070(00)00065-0.

Michał Wiliński, Mononito Goswami, Willa Potosnak, Nina Żukowska, and Artur Dubrawski. Exploring representations and interventions in time series foundation models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 66861–66886, 2025.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 53140–53164. PMLR, 2024.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, volume 34, pp. 22419–22430, 2021.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/3339f19c5fcee3ad74502947a32be9e6-Paper-Conference.pdf.

Liu Yang, Kangwook Lee, Robert D Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=HHbRxoDTxE.

Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen. Cautionary tales on air-quality improvement in Beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2205):20170457, 2017. doi: 10.1098/rspa.2017.0457.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11106–11115, 2021. doi: 10.1609/aaai.v35i12.17325.

Tian Zhou, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. One fits all: Power general time series analysis by pretrained LM. In *Advances in Neural Information Processing Systems 36*, pp. 43322–43355, 2023. doi: 10.52202/075280-1877.

A Notation

Table 2 collects the symbols used in the main text and in the appendix, grouped by the part of the method they belong to.

B Extended related work

Time-series foundation models. Besides Chronos and Chronos-2, pretrained forecasters include TimesFM, Moirai, MOMENT, Timer, Time-MoE and Sundial (Das et al., 2024; Woo et al., 2024; Goswami et al., 2024; Liu et al., 2024b; Shi et al., 2025; Liu et al., 2025). Most of them forecast zero-shot across domains, and the adaptation methods of Section 5 change their parameters, inputs or outputs.

Table 2: Notation.

Symbol	Meaning
Data and forecaster	
τ	A task: one dataset at one horizon
C, L, H	Number of channels, lookback (context length), horizon
t, $\mathbf{X}_t$, $\mathbf{Y}_t$	Forecast origin, its context in $\mathbb{R}^{C\times L}$ and its target in $\mathbb{R}^{C\times H}$
σ, $\mathcal{O}_\sigma$	Segment (training, verification or test) and the origins whose target lies in it
$\widehat{\mathcal{R}}_\sigma(f)$	Mean squared error of f over $\mathcal{O}_\sigma$
f_θ; tok_H, head	Frozen forecaster with parameters θ; its token preparation and prediction head
D; $B_1, \ldots, B_D$	Number of encoder blocks; the encoder blocks
Programs and cost	
p, $\mathcal{P}$; p_0	Compute program and the program space; Base, the standard execution
Φ_p, f_θ^p	Execution defined by p and the resulting forecaster
$[a, b]$, w; F	Window of blocks $B_a, \ldots, B_b$ and its width $w = b - a + 1$; the window map $B_b \circ \cdots \circ B_a$
k, α, s	Number of steps, step size, and window evaluations per step
$\mathbf{u}$; S	Token mask that selects the states a program rewrites; blocks skipped outside the window
$\kappa(p)$; $\kappa_{\max}$	Block evaluations per forecast of p ($\kappa(p_0) = D$); the execution budget
Recursive inference	
$\mathcal{K}$	The skill: model-specific knowledge given to the reasoner
e_τ	Task evidence computed on the training segment
R, m; $M = Rm$	Number of rounds and proposals per round; the proposal budget
$\mathcal{P}_{r,d}$; $\mathcal{H}_{r-1,d}$	Programs proposed in round r for dataset d; the history of earlier programs, rationales and responses
h_p; ρ_p	Rationale of p (hypothesis, evidence, failure condition); its training response (effect, agreement, support, concentration)
$\hat{p}_\tau$	Program deployed on task τ after verification
Evaluation and cost accounting	
δ_τ	Relative change in test error of $\hat{p}_\tau$ with respect to Base
G	Number of datasets, over which confidence intervals are bootstrapped
C^F_{alloc}; N	Block evaluations spent on allocation for a task; the number of forecasts that reuse $\hat{p}_\tau$

Allocating and iterating depth. Residual blocks can be read as iterative refinement of a shared representation (Greff et al., 2017; Jastrzebski et al., 2018). Depth can also be allocated per input or per token by learned mechanisms, such as confidence-based early exits during decoding (Schuster et al., 2022) or routers that give each token its own recursion depth (Bae et al., 2025). Iterating a block raises the question of where the iteration settles and how far it can go. Deep equilibrium models solve directly for the fixed point of a repeated layer (Bai et al., 2019), whereas recurrent networks run for too many iterations can degrade their state unless the architecture recalls its input (Bansal et al., 2022). In recursive inference, the damped update limits each step, and verification keeps Base whenever a revisit does not hold on later windows. Pretrained language models also keep much of their accuracy when adjacent layers are deleted or swapped (Lad et al., 2025), and time-series foundation models show redundancy across layers (Wiliński et al., 2025), which suggests that the blocks of a trained model are not tied to a single order or number of passes.

Revising from feedback. Language models can refine their outputs from self-generated feedback (Madaan et al., 2023) or from verbal reflections on task feedback (Shinn et al., 2023). In these settings, the model's output is the artifact that is improved. In recursive inference, the reasoner's output is a hypothesis about computation, and the frozen forecaster's measured response decides whether it holds.

Choosing configurations under small budgets. Configurations are also chosen under small evaluation budgets by Bayesian optimization (Snoek et al., 2012) and by automated model selection, including for forecasting (Fischer & Saadallah, 2024).

C Datasets and protocol

Table 3 lists the 14 multivariate datasets. We use them as distributed with TSFM-Bench (Li et al., 2025). Following the source attributions of TSFM-Bench and TFB (Qiu et al., 2024), the datasets derive from ETT (Zhou et al., 2021), Weather (Wu et al., 2021), Wind (Li et al., 2022), the Beijing multi-site air-quality data for AQShunyi and AQWan (Zhang et al., 2017), the SAPFLUXNET database for CzeLan and ZafNoo (Poyatos et al., 2021), Covid-19 (Panagopoulos et al., 2021), NYSE (Feng et al., 2019), NN5 (Ben Taieb et al., 2012) and Wike2000 (Gasthaus et al., 2019). Each dataset is split chronologically into training, verification and test segments. The split is 70/10/20 in general and 60/20/20 for the ETT and air-quality series. Every channel is standardized with the mean and standard deviation of its training segment, and all errors are computed on this scale. A forecast origin t uses the L preceding steps as context and predicts the next H steps. An origin belongs to a segment when its whole target interval lies in that segment. Test evaluation uses every rolling origin of the test segment with stride one, so the number of test origins ranges from 100 to 11,425 per task.

Table 3: Datasets. Split: training/verification/test percentages. Test origins: range over the four horizons.

Dataset	Freq.	Channels	Length	L	Horizons H	Split	Test origins
AQShunyi	1h	11	35,064	512	96, 192, 336, 720	60/20/20	6,294–6,918
AQWan	1h	11	35,064	512	96, 192, 336, 720	60/20/20	6,294–6,918
Covid-19	1d	948	1,392	36	24, 36, 48, 60	70/10/20	220–256
CzeLan	30min	11	19,934	512	96, 192, 336, 720	70/10/20	3,268–3,892
ETTh1	1h	7	14,400	512	96, 192, 336, 720	60/20/20	2,161–2,785
ETTh2	1h	7	14,400	512	96, 192, 336, 720	60/20/20	2,161–2,785
ETTm1	15min	7	57,600	512	96, 192, 336, 720	60/20/20	10,801–11,425
ETTm2	15min	7	57,600	512	96, 192, 336, 720	60/20/20	10,801–11,425
NN5	1d	111	791	104	24, 36, 48, 60	70/10/20	100–136
NYSE	1d	5	1,243	36	24, 36, 48, 60	70/10/20	190–226
Weather	10min	21	52,696	512	96, 192, 336, 720	70/10/20	9,821–10,445
Wike2000	1d	2,000	792	36	24, 36, 48, 60	70/10/20	100–136
Wind	15min	7	48,673	512	96, 192, 336, 720	70/10/20	9,016–9,640
ZafNoo	30min	11	19,225	512	96, 192, 336, 720	70/10/20	3,126–3,750

D Evaluation details

Error measures. For a segment $\sigma \in \{\mathrm{tr}, \mathrm{va}, \mathrm{te}\}$, let $\mathcal{O}_\sigma$ be the forecast origins whose target interval lies in σ (Appendix C). The error of a forecaster f on the segment is

$$\widehat{\mathcal{R}}_\sigma(f) = \frac{1}{|\mathcal{O}_\sigma|} \sum_{t \in \mathcal{O}_\sigma} \frac{1}{CH} \big\| f(\mathbf{X}_t) - \mathbf{Y}_t \big\|_F^2,$$

the mean squared error over origins, channels and forecast steps; MAE is defined in the same way with absolute errors.

Aggregate change. For a task τ, let

$$\delta_\tau = \frac{\widehat{\mathcal{R}}_{\mathrm{te}}\big(f_\theta^{\hat{p}_\tau}\big)}{\widehat{\mathcal{R}}_{\mathrm{te}}\big(f_\theta^{p_0}\big)} - 1$$

be the relative change in test error of the deployed program with respect to Base, computed from unrounded errors; MAE is treated in the same way. A task that retains Base has $\hat{p}_\tau = p_0$ and hence $\delta_\tau = 0$. The reported change is the uniform mean of δ_τ over the 56 tasks. Each of the $G = 14$ datasets contributes its four horizons, so this mean equals the mean of the dataset-level averages $\bar{\delta}_d = \frac{1}{4} \sum_{\tau \in \mathcal{T}_d} \delta_\tau$, where $\mathcal{T}_d$ is the set of tasks of dataset d.

Confidence intervals. The four horizons of a dataset share the same series and test period, so their changes are not independent, and we therefore resample datasets rather than tasks. Each of 10,000 bootstrap replicates draws G datasets with replacement and averages their $\bar{\delta}_d$, and the 95% interval spans the 2.5th to the 97.5th percentile of the replicates. The same scheme gives every interval reported in Section 4. Table 4 shows that the interval is insensitive to its construction. A bias-corrected and accelerated (BCa) bootstrap and a Student's t interval on the 14 dataset-level averages yield nearly the same bounds, and none of them includes zero. The intervals describe variation across datasets, with the forecast windows of each task and the proposals of the reasoner held fixed. Two further checks show that the mean is not carried by a few tasks. The median task changes MSE by -4.07%, and leaving out one dataset at a time gives means between -4.33% and -3.73%.

Table 4: Robustness of the confidence interval. Mean relative change in test error over the 56 tasks and its 95% confidence interval under three constructions. Both bootstraps resample the 14 datasets with 10,000 replicates; the Student's t interval uses the 14 dataset-level means. †Interval reported in the main text.

	ΔMSE (%)	ΔMAE (%)
Mean over 56 tasks	-4.02	-0.98
95% confidence interval		
Percentile bootstrap†	-5.22 to -2.83	-1.54 to -0.50
Bias-corrected and accelerated bootstrap	-5.20 to -2.80	-1.64 to -0.54
Student's t, 13 degrees of freedom	-5.40 to -2.64	-1.59 to -0.38

E IMPLEMENTATION DETAILS

Forecaster and environment. We use Chronos-2 (`amazon/chronos-2`, revision `29ec3766`, 119.5M parameters) in float32 with its median forecast and seed 2025, and its parameters are never updated. Each task runs the native multivariate model with cross-learning across series disabled and a context length equal to the lookback L. The model also rescales each input window internally before prediction; this step is part of its standard execution and is separate from the training-segment standardization of Appendix C. The encoder has $D = 12$ blocks, which the paper indexes $B_1, \ldots, B_D$ from the input side. All forecaster evaluations of the main run, from training responses to the test period, ran on NVIDIA A800-SXM4-80GB GPUs with Python 3.11.16 and PyTorch 2.8.0 (CUDA 12.8).

Program space. A program specifies a window $[a, b]$ of contiguous encoder blocks, a solver, a number of steps $1 \le k \le 8$, a step size $\alpha \in (0, 1]$, and a mask and reference. It may also replace a set S of blocks outside the window by the identity. The solvers are the damped update of Equation 1 ($s = 1$ window evaluation per step) and multi-stage updates with $s = 2$ (Heun, midpoint, Ralston), $s = 3$ (RK3) and $s = 4$ (RK4). Three mask–reference pairs are admissible: all states with input reference, forecast tokens with input reference, and forecast tokens with canonical reference. For Chronos-2, the forecast tokens are the $\lceil H/16 \rceil$ output patch tokens. The admissible space contains the programs with $\kappa(p) \le \kappa_{\max} = 4D = 48$ block evaluations. All 1,000 programs proposed in the main run use the damped update, with $2 \le k \le 5$, $0.1 \le \alpha \le 1$, no skipped blocks and $\kappa(p) \le 22$.

Masked update. Let x be the state entering the window, $c = F(x)$ the output of the window's standard execution, $\mathbf{u}$ the mask and $\bar{\mathbf{u}} = \mathbf{1} - \mathbf{u}$. The first step reuses c for every mask and reference, so it adds no computation:

$$z^{(1)} = x + \alpha\, \mathbf{u} \odot (c - x).$$

Each later step evaluates the window once, on the full token sequence, at a probe state whose masked tokens take their current values and whose unmasked tokens are set to a reference r:

$$z^{(j+1)} = z^{(j)} + \alpha\, \mathbf{u} \odot \big(F(\mathbf{u} \odot z^{(j)} + \bar{\mathbf{u}} \odot r) - z^{(j)}\big), \qquad j = 1, \ldots, k-1,$$

with $r = x$ for the input reference and $r = c$ for the canonical reference. Unmasked tokens are not written during the update and are returned to their canonical values afterwards, so the window outputs $\mathbf{u} \odot z^{(k)} + \bar{\mathbf{u}} \odot c$. The update therefore performs $k - 1$ additional window evaluations, as

counted by Equation 3. With the all-state mask ($\mathbf{u} = \mathbf{1}$) it reduces to Equation 1, and with $k = 1$ the two references coincide. Multi-stage solvers follow the same rule: only their first window evaluation reuses c.

Verification. Verification uses up to 12 non-overlapping windows of the verification segment. For the 16 tasks whose verification segment admits fewer than four such windows, it uses four evenly spaced overlapping windows instead. Candidates are verified in batches of four in the order of their training MSE, and verification stops after the first batch that contains an admitted candidate. The strict tier admits a candidate that meets five conditions. Its verification MSE ratio is below 1, and its MAE ratio is at most 1.01. It lowers MSE on at least 60% of the windows, and its worst per-window MSE ratio is at most 1.10. Finally, it is evaluated on at least four windows. When no candidate is admitted, the relaxed tier keeps the MSE, MAE and window-count conditions and drops the other two. Within the first batch that contains an admitted candidate, the one with the lowest verification MSE is deployed, with ties broken by MAE and then by the program key.

F REASONER

Model, inputs and outputs. The reasoner is GPT-5.6 Sol with its reasoning effort set to low. Table 5 lists what it receives in each round and what it returns, following the allocation step of Equation 5. The skill $\mathcal{K}$ and the task evidence e_τ, computed on the training segment, are given in every round. From the second round on, the history $\mathcal{H}_{r-1,d}$ adds every earlier program together with the rationale it was proposed with and the training response it produced. The reasoner returns the next programs $\mathcal{P}_{r,d}$, each with its own rationale. It also returns a decision summary that states what the responses so far have established and what the new round tests.

Table 5: What the reasoner receives and returns. Statistics are named here and defined in Tables 6 and 7.

Given in every round	
Knowledge $\mathcal{K}$	The skill (Appendix J); the output schema; the budget (three rounds of at most six programs); the supported solvers
Task description	Task name; channels; context length; horizon; horizon-to-context ratio; length of the training segment
Task evidence e_τ	*Segment summary*: lag-1 autocorrelation; recent shift in mean and ratio of standard deviations; spectral peak fraction; spread of channel scales. *Context statistics* (mean and standard deviation over eight training contexts): median and 90th-percentile cross-channel correlation; trend correlation; lag-1 autocorrelation; leading shared-energy fraction; normalized effective rank; recent shifts in mean and log volatility
Window design	Origins of the up to 16 training and up to 12 verification windows
Added from round 2: the history $\mathcal{H}_{r-1,d}$, for every earlier program	
Rationale h_p	Parameters; hypothesis; evidence; failure condition
Response ρ_p	*Effect*: MSE and MAE ratios to Base. *Agreement*: per-window MSE and MAE ratios. *Support*: fraction of windows improved; worst-window ratio; ratios over the earlier and later half of the windows; share of Base error on improved windows. *Concentration*: per-window gain contributions; total gain and harm; largest-gain window and the ratio without it; leave-one-window-out ratios
Round summaries	The decision summary of every earlier round
Pool diagnostics	Numbers of programs that lower MSE, lower MSE and MAE, and meet the admission limits on the training windows; correlation of per-window responses for every pair of programs
Returned in each round: the new programs $\mathcal{P}_{r,d}$	
Programs	Up to six, each with its parameters and rationale h_p
Decision summary	What the responses so far establish, what remains open, and what the new round tests

Task evidence and training responses. Table 6 defines the statistics of the task evidence e_τ, grouped by the dependence, non-stationarity and cross-channel structure they describe (Section 3.1). All of them are computed on the standardized training segment. The context statistics use eight context windows whose endpoints are evenly spaced over the segment, and each is reported as its mean and standard deviation over the eight windows. Each proposed program is evaluated on up to 16 non-overlapping training windows at evenly spaced origins (fewer for the ten tasks with a short training segment), paired with Base on the same windows, and Table 7 defines its response ρ_p. Base is evaluated once per task and phase and reused. The first round includes at least one alternative outside the region that the skill favors.

Table 6: Statistics of the task evidence e_τ. They are grouped by the aspect of the task they describe. S: segment summary; C: context statistics, reported as mean and standard deviation over eight contexts. Channel-wise quantities are summarized by their median over channels.

Statistic	Definition	What it indicates
Temporal dependence		
Lag-1 autocorrelation (S, C)	Lag-one normalized inner product of each channel	How persistent each series is from step to step
Trend correlation (C)	Absolute correlation of each channel with a linear time trend	How strongly the context trends
Spectral peak fraction (S)	Share of spectral energy in the dominant frequency of the last context-length block	Whether one periodicity dominates
Non-stationarity near the forecast origin		
Recent mean shift (S, C)	Absolute shift in mean between the last two context-length blocks (S) or quarters of the context (C)	Drift in level
Recent volatility change (S, C)	Ratio (S) or absolute log ratio (C) of standard deviations between the same two parts	Change in variability
Cross-channel structure		
Cross-channel correlation (C)	Median and 90th percentile of the absolute pairwise correlation between channels (at most 256 channels)	How strongly channels move together
Leading shared-energy fraction (C)	Largest squared singular value over the sum of all squared singular values	Dominance of one common component
Normalized effective rank (C)	Exponential of the entropy of the squared singular values, over the smaller of the number of channels and the context length	Number of distinct components, relative to its maximum
Spread of channel scales (S)	Ratio of the 95th to the 5th percentile of the channel standard deviations	How unequal the channel magnitudes are

Table 7: Statistics of the training response ρ_p. They are computed over the program's training windows, relative to Base on the same windows. Each group answers one question about the program's effect.

Group	Statistics	Question it answers
Effect	MSE and MAE ratios to Base, pooled over the windows	Does the program lower error overall?
Agreement	MSE and MAE ratios on each window	Do the two errors move together, window by window?
Support	Fraction of windows with a lower MSE; worst per-window MSE ratio; pooled MSE ratios over the earlier and the later half of the windows; share of Base error on the windows with a lower MSE	Does the gain hold across windows and over time?
Concentration	Contribution of each window to the MSE gain; total positive gain and total harm; the window with the largest gain and the MSE ratio without it; leave-one-window-out MSE ratios	Is the gain carried by a few windows?

An example exchange. The exchange below shows recursive inference at work on one task, Wind at $H = 720$, condensed from the recorded rounds. The reasoner's text is shortened, only the responses it acts on are shown, and all numbers are training ratios to Base unless stated otherwise.

Unless noted, programs use the damped update with $k = 2$ on the forecast tokens with the canonical reference; ratios are to Base on the 16 training windows, and Window 12 is the MSE ratio on the window where losses concentrate.

Round 1 · Reasoner

Reads	The skill: test the last encoder blocks first. The task evidence: a horizon-to-context ratio of 1.41 and a clear shift in recent volatility, which motivate measurement but do not decide the width, the number of steps or a safe step size.
Designs	Six matched contrasts: width (B_{12} against $B_{10\text{–}12}$), step size on the wide window (0.25, 0.5, 0.75), scope and reference, and an early window as a control for placement.
States	For each program an expectation and a refutation condition; for **P1** (step 0.75), a stronger correction at the same cost, refuted if it loses to step 0.25 on MSE without a compensating gain.

Round 1 · Measured responses

Program	MSE	MAE	Window 12
B_{12}, $\alpha = 0.5$	0.986	0.995	1.008
$B_{10\text{–}12}$, $\alpha = 0.25$	0.967	0.989	1.039
$B_{10\text{–}12}$, $\alpha = 0.5$	0.938	0.976	1.039
P1: $B_{10\text{–}12}$, $\alpha = 0.75$	**0.926**	0.972	1.047
$B_{10\text{–}12}$, $\alpha = 0.5$, all states, input ref.	0.947	0.979	1.029
$B_{2\text{–}4}$, $\alpha = 0.5$	0.991	0.996	0.996

P1 lowers MSE on 15 of 16 windows and keeps a ratio of 0.929 without its largest-gain window.

Round 2 · Reasoner

Finds	A wider window and a larger step lower the mean error, and the gain is broad; the loss concentrates in window 12 and grows with the step.
Asks	How far do strength and width help, and can the loss in window 12 be avoided?
Tests	From P1: steps 0.9 and 1.0 (**P2a**); the window one block wider (**P2b**) and one narrower; other scopes and references.
Caveat	Stronger steps may amplify the loss in window 12.

Round 2 · Measured responses (window $B_{10\text{–}12}$ unless noted)

Program	MSE	MAE	Window 12
$\alpha = 0.9$	0.922	0.970	1.050
P2a: $\alpha = 1.0$	**0.920**	0.970	1.051
P2b: $B_{9\text{–}12}$, $\alpha = 0.75$	**0.920**	0.970	1.052
$B_{11\text{–}12}$, $\alpha = 0.75$	0.939	0.975	1.016
$\alpha = 0.75$, all states, input ref.	0.928	0.972	1.044
$\alpha = 0.75$, input ref.	0.957	0.983	1.006

Round 3 · Reasoner

Finds	Along step size and width the mean gain levels off: P2a and P2b reach 0.920 and neither dominates the other, while avoiding the loss in window 12 costs mean error. The number of steps has not been measured at this horizon.
Tests	Combinations of the measured boundaries and strengths, one block of further placement, and a bounded question about the number of steps: **P3**, P1 with a third step, which changes only k and stays within 18 block evaluations.
States	P3 is expected to extend P1's mean benefit and is refuted if its MSE does not improve on P1, or if MAE, support and the worst window all deteriorate.
Caveat	A third step may sharply worsen the loss in window 12.

Round 3 · Measured responses (window $B_{10\text{–}12}$ unless noted)

Program	MSE	MAE	Window 12
$B_{9\text{–}12}$, $\alpha = 1.0$	0.913	0.968	1.062
$B_{11\text{–}12}$, $\alpha = 1.0$	0.932	0.973	1.014
$B_{8\text{–}12}$, $\alpha = 0.75$	0.912	0.967	1.054
$k = 3$, $\alpha = 0.5$	0.896	0.962	1.090
P3: $k = 3$, $\alpha = 0.75$	**0.887**	0.960	1.104
$\alpha = 1.0$, all states, input ref.	0.920	0.970	1.051

P3 is not refuted: its MSE improves on P1 and its MAE and support do not deteriorate, while window 12 worsens as the caveat anticipated.

Verification and test

Verification admits P3 with the lowest verification MSE in its batch (MSE ratio 0.927, MAE ratio 0.974), and it is deployed. On the test period it reduces MSE by 6.6% and MAE by 2.1% relative to Base.

Read against the method, the exchange shows each element of recursive inference. The first round uses the skill and the task evidence to design matched contrasts rather than to pick a program. Every proposal states what it expects and when it counts as refuted (Section 3.2). The reasoner reads each response along several aspects at once: the effect on MSE and MAE, its support across windows, and where its losses are concentrated. This reading lets it see, in round 3, that the controls it had explored were saturating, so it turns to one it had not measured. That change yields the largest gain of the trajectory, and the stated caveat anticipates the cost that comes with it. Each revision changes one control of a program named in the history, so that its response can be attributed to that change. Verification then decides deployment on later windows, and the program it admits is the one the reasoner constructed from the measured responses.

Further patterns of revision. The exchange above turns to a control that had not been measured. Other tasks revise their programs in other ways, and Figure 8 condenses three of them to the lineage of the deployed program. On NN5 at $H = 60$ (a), the round-1 program at the root of the lineage barely lowers MSE and raises MAE, and the reasoner repairs this trade-off by reducing the step size. On NYSE at $H = 36$ (b), four of the five late-window programs raise MSE in the first round. The reasoner therefore leaves the skill's late-region prior for an early window and strengthens it round by round. On ZafNoo at $H = 192$ (c), narrowing the window and reducing the step each lower MAE only partly, and combining the two changes brings MAE below Base. The deployed programs reduce test MSE by 7.8%, 7.9% and 7.0%, respectively.

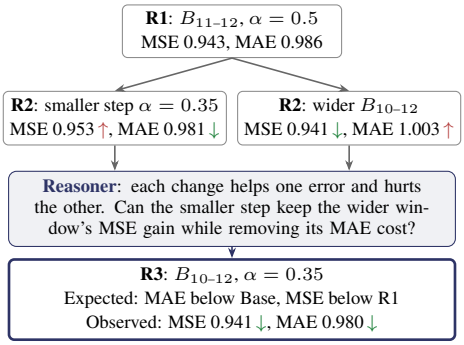


Figure 7: Evidence-guided revision (ZafNoo, $H = 336$). Numbers are training MSE and MAE ratios to Base; arrows compare with R1 (↓ better, ↑ worse). All programs use $k = 2$ on the forecast tokens.

Feedback comparison. The comparison of Section 4.2 uses two runs on all 56 tasks, separate from the main run. Both start from the same first round: its six proposals per task, their rationales and their measured training responses are shared exactly. From the second round on, the run with responses receives the input packet of Table 5 unchanged. The run without responses receives the same packet with every measured quantity removed: the training responses ρ_p, the pool diagnostics

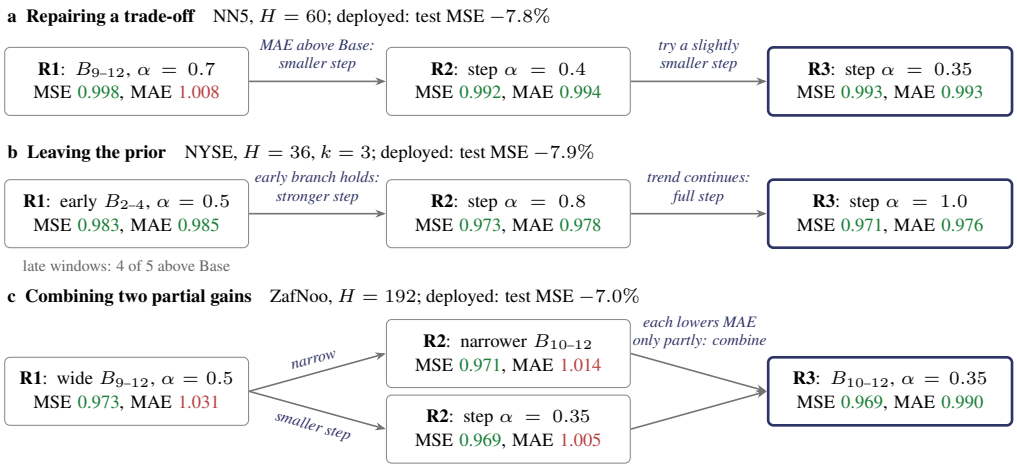


Figure 8: Three further patterns of revision. Each row follows the lineage of a deployed program. Boxes give the change to the preceding program and its training MSE and MAE ratios to Base (green below Base, red above; blue box: deployed); arrows give the reasoner's stated reason. All programs use the damped update on the forecast tokens, with $k = 2$ unless noted.

and the decision summaries. Its history still lists each earlier program with its parameters and rationale. Both runs use the same skill, task evidence, budget, training windows and verification rule. The run without responses continues the reasoner session of its first round, whereas the run with responses starts a new session at the second round, and each run was generated once.

Revision statistics. In the main run, 99% of the 664 programs proposed in rounds 2 and 3 identify by name the earlier program they revise, and 97% change a single control of it. The control changed is most often the step size (55%), followed by the window (26%), the number of steps (12%) and the scope (5%). In the feedback comparison, 67% of the later proposals with responses identify the program they revise and 96% change a single control of it, against 56% and 86% without responses. Without responses, the reasoner also proposes fewer programs in rounds 2 and 3 (532 against 593). The share of its proposals that raise training MSE above Base grows from 15% in the first round to 20% in the third, whereas with responses it falls to 2%.

G Additional results

Per-task results. Table 8 lists, for every task, the program selected by verification and its cost. It also gives the relative change in test error with respect to Base, the forecaster compute spent on allocation, and the number of test origins. Figure 9 shows the same changes task by task: MSE and MAE decrease together on 44 of the 56 tasks, and MSE decreases on 48.

Table 8: Per-task results. Tier: S (strict) or R (relaxed); Blocks: window $[a, b]$ in 1-based block indices; Scope: F (forecast tokens) or A (all states); Cost: κ/D; C^F_{alloc} in thousands of block evaluations; N: test origins.

Dataset	H	Tier	Blocks	k	α	Scope	Cost	ΔMSE (%)	ΔMAE (%)	C^F_{alloc}	N
AQShunyi	96	R	11–12	2	0.65	F	1.17	−3.29	−1.13	7.8	6,918
	192	S	10–12	2	0.35	F	1.25	−3.41	−1.13	5.3	6,822
	336	S	10–12	3	0.5	F	1.50	−5.90	−0.48	5.3	6,678
	720	S	10–12	3	0.5	F	1.50	−6.28	−0.36	5.1	6,294
AQWan	96	S	10–12	2	0.5	F	1.25	−3.03	−0.72	8.3	6,918
	192	S	9–12	2	0.65	F	1.33	−3.51	−0.53	5.6	6,822
	336	S	10–12	3	0.5	F	1.50	−4.62	−0.22	5.5	6,678
	720	S	9–12	3	0.5	F	1.67	−3.98	+0.32	5.7	6,294
Covid-19	24		Base				1.00	0.00	0.00	6.3	256

Dataset	H	Tier	Blocks	k	α	Scope	Cost	ΔMSE (%)	ΔMAE (%)	C^F_{alloc}	N
	36		Base				1.00	0.00	0.00	5.6	244
	48		Base				1.00	0.00	0.00	5.6	232
	60	S	3–4	2	0.5	F	1.17	+0.00	+0.12	5.3	220
CzeLan	96	S	3	2	0.5	F	1.08	−0.01	−0.01	7.1	3,892
	192	R	12	2	0.5	F	1.08	−2.42	−0.20	7.2	3,796
	336		Base				1.00	0.00	0.00	5.7	3,652
	720	S	12	3	0.35	F	1.17	−4.15	−0.14	4.7	3,268
ETTh1	96		Base				1.00	0.00	0.00	7.3	2,785
	192	S	11–12	2	0.5	A	1.17	−2.52	−0.67	5.6	2,689
	336	R	11–12	2	0.45	A	1.17	−2.34	−0.68	6.2	2,545
	720	R	12	2	0.85	F	1.08	−2.53	+1.99	3.8	2,161
ETTh2	96	S	10–12	2	0.35	A	1.25	−0.56	−0.20	7.8	2,785
	192	S	11–12	4	0.45	F	1.50	−0.25	−0.26	5.4	2,689
	336	S	11–12	3	0.4	F	1.33	+0.11	−0.14	5.0	2,545
	720	S	11–12	2	0.5	A	1.17	−1.69	−0.65	4.1	2,161
ETTm1	96	S	11–12	2	0.5	A	1.17	−2.86	−0.98	8.0	11,425
	192	R	10–12	2	0.5	A	1.25	−4.26	−1.26	8.1	11,329
	336	S	11–12	2	0.5	A	1.17	−3.69	−1.35	8.1	11,185
	720	R	11–12	2	0.5	A	1.17	−5.35	−1.79	8.4	10,801
ETTm2	96	R	12	3	0.5	F	1.17	−2.35	+0.25	8.2	11,425
	192	S	11–12	3	0.25	F	1.33	−2.98	−0.39	6.6	11,329
	336	S	9–12	3	0.25	F	1.67	−5.68	−1.80	6.0	11,185
	720	S	10–12	3	0.25	F	1.50	−5.44	−2.04	6.2	10,801
NN5	24	S	10–12	2	0.4	F	1.25	−5.16	−2.61	4.8	136
	36	S	10–12	2	0.4	F	1.25	−5.76	−2.89	3.7	124
	48	S	9–12	2	0.35	F	1.33	−7.66	−3.81	2.8	112
	60	S	9–12	2	0.35	F	1.33	−7.83	−3.98	2.3	100
NYSE	24	S	9–12	3	0.5	F	1.67	+0.84	+2.20	5.3	226
	36	S	2–4	3	1	F	1.50	−7.90	−5.38	5.3	214
	48	S	11–12	3	0.7	F	1.33	−7.24	−3.53	5.5	202
	60	S	11–12	3	0.5	F	1.33	−6.10	−3.25	4.8	190
Weather	96	S	11–12	3	0.35	F	1.33	−7.36	−0.59	5.9	10,445
	192	S	11	3	0.35	F	1.17	−1.01	−0.12	7.7	10,349
	336	S	10–12	2	0.5	F	1.25	−8.62	−1.63	6.4	10,205
	720	S	10–12	3	0.45	F	1.50	−14.33	−1.21	5.3	9,821
Wike2000	24	S	9–12	2	0.5	F	1.33	−5.47	+0.95	4.4	136
	36	S	11–12	3	0.7	F	1.33	−6.83	−0.81	3.7	124
	48	S	11–12	3	0.5	A	1.33	−4.80	−0.10	2.7	112
	60	S	11–12	4	0.5	F	1.50	−4.39	−0.53	2.3	100
Wind	96	S	2–4	2	0.65	F	1.25	−0.51	−0.14	8.1	9,640
	192	S	2–4	2	0.6	F	1.25	−0.35	−0.07	8.4	9,544
	336	S	10–12	3	0.4	F	1.50	−6.00	−2.17	6.6	9,400
	720	S	10–12	3	0.75	F	1.50	−6.62	−2.13	5.1	9,016
ZafNoo	96	S	11–12	2	0.5	A	1.17	−5.12	−1.01	5.0	3,750
	192	S	10–12	2	0.35	F	1.25	−7.00	−2.44	5.2	3,654
	336	S	10–12	2	0.35	F	1.25	−7.87	−2.82	4.6	3,510
	720	S	11–12	2	0.5	F	1.17	−9.26	−2.60	4.5	3,126

Horizons and deployed programs. Table 9 breaks the results down by the position of the horizon within each dataset. The mean reduction in MSE grows from the shortest to the longest horizon, together with the cost, and every dataset deploys a program at its longest horizon. Table 10 summarizes the structure of the 51 programs deployed in place of Base (Section 4.1). Most end at the final block, span two or three blocks and use a step size below 1, yet the 51 tasks use 37 distinct programs.

Tasks without a test gain. For completeness, Table 11 lists the eight tasks on which the deployed decision does not lower test MSE. Five of them keep Base because no candidate meets the verification limits, although the best program in each pool lowers training MSE by 4% to 12%; on these tasks the policy spends no extra computation, as intended (Section 3.3). On the other three tasks, the admitted program changes test MSE by at most 0.84%.

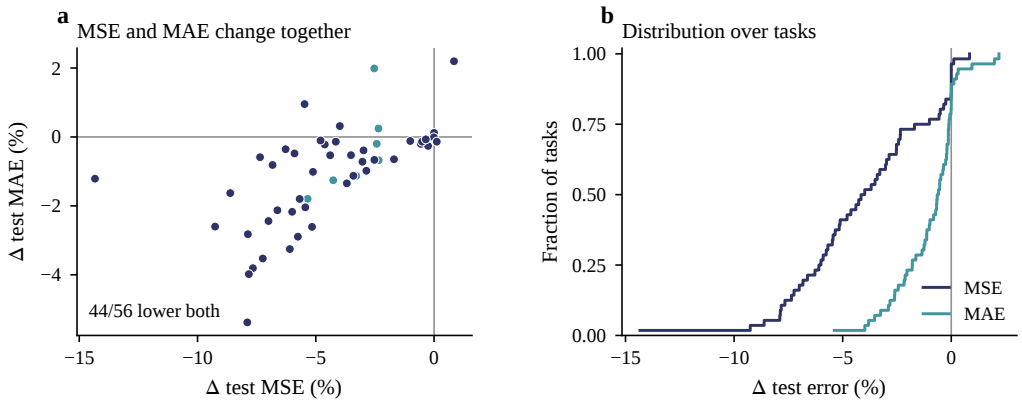


Figure 9: Task-level test outcomes. (a) Relative change in test MSE against test MAE for the 56 tasks, colored by the deployment decision as in Figure 5d. (b) Empirical distribution of the relative changes over tasks.

Table 9: Results by horizon position within each dataset (1: shortest; 4: longest), over the 14 datasets.

Position	ΔMSE (%)	ΔMAE (%)	MSE improved	Non-Base	Cost
1	−2.49	−0.29	11/14	12/14	1.22
2	−3.44	−1.15	13/14	13/14	1.26
3	−4.59	−1.34	11/14	12/14	1.31
4	−5.57	−1.16	13/14	14/14	1.34

Reasoning effort. Table 12 compares GPT-5.6 Sol with its reasoning effort set to low, as in the main run, and to high. All other settings, including the forecaster, program space, proposal budget and verification rule, match the main run. The comparison covers the 52 tasks of the 13 datasets that both settings cover, under the strict verification tier. The two settings reach nearly the same accuracy through different programs. Where both deploy a program, they share the block window on 20 of 39 tasks and the full program on only 4. Yet on the 35 tasks with different programs, their per-task changes in test MSE agree closely (Spearman $\rho = 0.84$). Recursive inference therefore does not converge on one policy for a task; most tasks admit several programs of similar value, and either setting finds one of them.

H Mechanism analysis

Setup. For each of the 51 deployed programs (Figure 6b–d), Base and the program run on the same training-segment windows of their task, 1,320 windows in total and 7 to 32 per task, through the unmodified execution code. Forward hooks record the state after every encoder block, the iterates $z^{(j)}$ of the window and the attention weights, which are recomputed from the queries and keys of each attention layer. The hooks leave every prediction unchanged, loop and Base states agree exactly before the window, and the recomputed attention reproduces the layer outputs to within 1.4×10^{-5}.

Relative update (Figure 6a). For block B_i with input state h, the relative update of a token is $\|B_i(h) - h\|/\|h\|$. It is averaged over the forecast or the context tokens and over the windows of a task, in the ordinary forward pass. This panel uses the ordinary pass of all tasks, including those that keep Base; the figure gives its median and interquartile range over the 52 tasks of 13 of the 14 datasets, with 7 to 16 training windows per task.

Attention on the revisit (Figure 6b). Queries are the forecast tokens. For each block of the window, we compare the attention of its first pass with that of its last revisit on the same window, for time attention over the token sequence and for group attention across the series of a task. The

Table 10: Structure of the 51 deployed programs.

Property	Programs
Window ends at the final block	45/51
Window starts within the first four blocks	5/51
Window width 1 block	6/51
Window width 2–3 blocks	38/51
Window width 4 blocks	7/51
Steps $k = 2$	28/51
Steps $k = 3$	21/51
Steps $k = 4$	2/51
Step size $\alpha < 1$	50/51
Forecast-only write-back	41/51
Skipped blocks	0/51
Distinct programs	37
Distinct programs, step size ignored	18
Distinct block windows	8
Largest number of tasks sharing a program	6

Table 11: Tasks without a test gain. Training: lowest training MSE ratio in the pool. Verification: MSE ratio of the deployed program or, for Base, the lowest among the verified programs.

Dataset	H	Decision	Training	Verification	ΔMSE (%)	ΔMAE (%)
Covid-19	24	Base	0.881	1.001	0.00	0.00
Covid-19	36	Base	0.876	1.001	0.00	0.00
Covid-19	48	Base	0.914	1.000	0.00	0.00
CzeLan	336	Base	0.938	0.997	0.00	0.00
ETTh1	96	Base	0.960	1.002	0.00	0.00
Covid-19	60	Strict	0.922	0.999	+0.00	+0.12
ETTh2	336	Strict	0.982	0.973	+0.11	−0.14
NYSE	24	Strict	0.924	0.899	+0.84	+2.20

entropy of each query's attention distribution is averaged over queries, heads and windows. Both entropies rise in all 45 tasks whose window ends at the final block, by a median of 0.43 nats over time and 0.19 nats across series. Over time, the share of attention on the most recent context patch falls in all 45 tasks, and across series, the attention of a series to itself falls in all 45.

Geometry of the displacement (Figure 6c). Let x be the state entering the window, $g = F(x) - x$ the update of one ordinary pass and e the loop's displacement of a forecast token from x. Its along-g part is γg with $\gamma = \langle e, g\rangle / \|g\|^2$, and its off-$g$ part is $r = e - \gamma g$, which is orthogonal to g. Rescaling the one-pass update produces only states $x + \gamma g$, on the line through x and $F(x)$. Averaged over the forecast tokens and windows of a task, γ follows the damped schedule $1 - (1 - \alpha)^k$ with Spearman $\rho = 0.95$, and $\|r\|/\|g\|$ is positive in all 51 tasks, with a median of 0.19 and an interquartile range of 0.14 to 0.30.

Reproducing the forecast change (Figure 6d). Each part is written back in place of the window's output in the same way as the program, and the remaining blocks and the output head then run unchanged. The along-g part gives the state $x + \gamma g$, and the off-g part gives Base plus r. For a variant with forecast $\hat{y}_v$, the share of the loop's forecast change it reproduces is $R = 1 - \|\hat{y}_{\text{loop}} - \hat{y}_v\| / \|\hat{y}_{\text{loop}} - \hat{y}_{\text{Base}}\|$, with norms taken over all windows of a task. $R = 1$ reproduces the loop's forecast exactly, and $R = 0$ is no closer than Base. The median of R is -0.01 for the along-g part and 0.74 for the off-g part. Rescaling with the single γ that minimizes training MSE on a grid from 0 to 2 reproduces a median of only 2%, so no rescaling of the one-pass update recovers the loop's effect on the forecast.

Table 12: Low versus high reasoning effort under the strict tier.

	Low effort	High effort
Tasks compared	52	
Mean ΔMSE (%)	−3.30	−3.35
Mean ΔMAE (%)	−0.94	−0.98
Tasks deploying a program	40	40
Same deploy-or-retain decision	50/52	
Same block window, both deploying	20/39	
Same program, both deploying	4/39	
ΔMSE agreement, different programs	$\rho = 0.84$ ($n = 35$)	

I ALLOCATION COST

Allocation compute. With n_{tr} training and n_{va} verification windows, the forecaster compute spent on allocation for a task is

$$C^F_{\mathrm{alloc}} = n_{\mathrm{tr}}\Big(D + \sum_{p\in\mathcal{P}_{\mathrm{pool}}} \kappa(p)\Big) + n_{\mathrm{va}}\Big(D + \sum_{p\in\mathcal{P}_{\mathrm{ver}}} \kappa(p)\Big),$$

where $\mathcal{P}_{\mathrm{pool}}$ is the candidate pool and $\mathcal{P}_{\mathrm{ver}}$ the candidates evaluated during verification; each program and Base are evaluated once per phase. On the reasoner's side, the input packet of Table 5 grows with the history, from about 7×10^3 tokens per task in the first round to about 1.7×10^4 and 3.1×10^4 in the next two. The response stays at about 1.3×10^3 tokens per round, so a task needs about 5×10^4 input and 4×10^3 response tokens in total.

Amortization. Allocation is paid once per task and then shared by every forecast that reuses the selected program. Over N forecasts, the cost per forecast relative to Base is $\big(\kappa(\hat{p}_\tau) + C^F_{\mathrm{alloc}}/N\big)/D$ (Equation 6). Figure 10 shows this cost as a function of N for all 56 tasks, together with its value at each task's test length. Allocation costs a median of 5,570 block evaluations per task. Its share per forecast falls below the deployment cost itself once a program serves about 350 forecasts, in the median task. Over the test periods, the median task adds 0.12 times Base per forecast. Tasks with short test periods, such as the four daily datasets with 100 to 256 test origins, carry a larger share. They raise the mean over tasks to 1.98 times Base, and the per-task values are listed in Table 8.

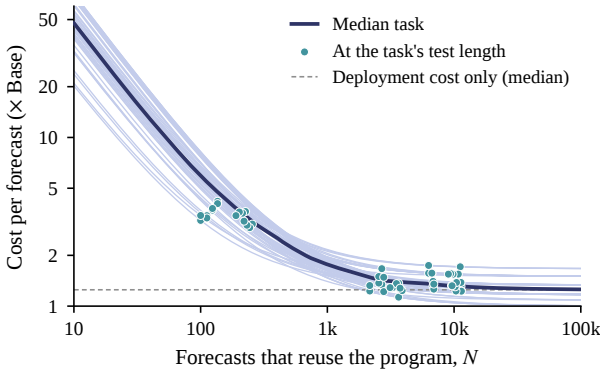


Figure 10: Amortization of the allocation cost. Cost per forecast as a function of the number of forecasts N that reuse the selected program. Thin lines: individual tasks; thick line: median task; points: each task's test length; dashed line: median deployment cost without allocation.

J THE CHRONOS-2 SKILL

The skill is a fixed text that the reasoner receives in every round (Table 5). It follows the allocation questions of the program space (Section 2) and states, for each of them, what an intervention means, which effects are plausible and which observations are informative. It does not assign programs to tasks, and Table 13 summarizes its components and the guidance each provides.

Table 13: Components of the Chronos-2 skill and the guidance each provides.

Component	Guidance
Operator semantics	What each control does and costs: the window input as the starting state, what k counts, the setting that reproduces Base, and why a small step size is not automatically safe
Placement prior	Test a bounded late-region repair first, always paired with an alternative outside that region; the prior does not rank exact windows
Conditional association	When recent volatility changes strongly across training contexts, include a shallower or narrower alternative before extrapolating to deeper or wider recurrence
Objective distinction	Read MSE and MAE, and each part of the training period, separately; compare step sizes only at matched windows, depths and scopes
First-round design	Pose a few separable contrasts within the budget: narrow against wide, preserving against rewriting context, and a matched step-size bracket
Later-round design	For each proposal, name the unresolved question, the measured parent, the changed control, the expected response, a competing explanation and the outcome that would refute it
Reading principles	Compare with the measured parent rather than Base; scope and strength interact; win counts, magnitudes and objectives can disagree; unsuccessful directions are informative; past concentration of proposals is not evidence of optimality
Output format	Admissible parameters and the fields of each proposal and decision summary

Throughout, the skill frames its content as hypotheses and checks rather than rules. It tells the reasoner which questions are worth asking and how to read the answers, while the task's measured responses decide which hypotheses hold and verification on later windows decides whether a resulting program is deployed (Section 3.3).