KOMPAS: KOOPMAN DYNAMICS MODELING AND BLACK-BOX PREDICTIVE CONTROL OF MULTI-TURN LLM REPLY ATTRIBUTES

Anonymous authors
Paper under double-blind review

ABSTRACT

In tasks such as multi-turn jailbreak defense or format-constrained generation, a language model must hold a reply attribute at a target on every turn. However, the attribute does not reset between turns: what one reply does on it carries into the next, so its readings form a trajectory with inertia. Existing remedies react to the current reading alone and ignore this inertia, so their corrections arrive a turn late or push past the target. We fill this gap with **KOMPAS**, **KO**opman dyna**M**ics modeling and **P**redictive control of **A**ttributes at the prompt **S**tep, which treats a prompt–response exchange as one step of a controlled system. It fits a linear dynamics model to past trajectories of the attribute, with the instruction as the input. Before sending an instruction, KOMPAS predicts how each candidate would move the attribute and sends the one predicted to reach the target, without querying the language model. On three public instruction-following benchmarks, KOMPAS raises constraint satisfaction by 14.5 and 14.8 points over two uncontrolled models. It also beats the stronger of two baselines that steer the model's internal activations, by 5.3 and 5.7 points.

1 INTRODUCTION

Multi-turn interaction turns a reply attribute of a frozen language model into a sequence with inertia. The attribute may be a word count, a sentence count, a CEFR readability grade, a sentiment polarity, a safety score under jailbreak attempts, or whether a formatting constraint is met. What one reply does on such an attribute carries into the next reply, instead of being drawn fresh at every turn. Instruction following in particular is measured to decay over a dialogue, so a constraint met early is not met for free later (Li et al., 2024). Deployments need to hold such an attribute at a target, such as a length budget or a readability grade, across every turn, and inertia complicates that. The same current reading can sit on a rising path or on a falling path, and the next reply differs between the two. A controller that reads one reply at a time cannot tell those two futures apart.

Prior work approaches this problem along three lines, reviewed in Section 5. Latent-space controllers steer activations or logits at every generated token, so they need white-box access, while a deployment acts only between turns through a black-box API (Nguyen et al., 2026; Miyaoka & Inoue, 2026a; Tran et al., 2026; Hu et al., 2026; Cheng & Alonso, 2026; Skifstad et al., 2026; Kong et al., 2024). Prompt-level remedies reinstate an instruction on a fixed schedule or after a per-turn check, but they are stateless: none predicts what the next instruction will do (Li et al., 2024; Dongre et al., 2025; Wang, 2026). Control-theoretic views write prompting as a control problem, yet their results are reachability statements, bounds, or token-level detectors (Bhargava et al., 2024; Cheng et al., 2026; Fan et al., 2026; Akrout & Wilson, 2026). What is missing is a dynamics model at the interaction level that uses only what a black-box caller can observe and carries the instruction as an explicit input.

Our Perspective: An Interaction-Level Koopman Perspective. We treat one prompt–response exchange as one step of a controlled dynamical system, whose only observable is the verifier's score of the completed reply. The intuition is that the next reading decomposes into three parts: what recent history carries forward, what the current instruction pushes, and a constant offset. The state is the window of the last few readings, so a reading on a rising path and the same reading on a

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.

falling path become two different states. Linear predictors with inputs, fit on observables and paired with model predictive control, are an established route for nonlinear systems that must be planned over in real time (Korda & Mezić, 2018). The same route fits here: a window of verifier readings is the feature set the linear predictor acts on, and the instruction, encoded as a scalar command, is the input. In its linear form the resulting model is a delay regression with an exogenous input. The operator view adds the freedom to swap that feature set, a closed-form multi-step prediction, and an unchanged control toolchain. The controller rolls every candidate command forward through this model, scores the predicted trajectory against the target, executes the best command, and replans after the next reply. Scoring a candidate takes one matrix product and a readout, with no extra query to the language model. **The key insight is that scoring on the predicted trajectory lets the controller send a command that differs from the target while the attribute is still moving.** A dynamics model that predicts well also selects well: the cost of the chosen command exceeds the best available cost by at most a term set by the prediction error. This yields **KO**opman dyna**M**ics modeling and **P**redictive control of **A**ttributes at the prompt **S**tep, or **KOMPAS**.

Our contributions are as follows.

1. We model the multi-turn dynamics of a reply attribute from verifier readings alone. History beyond the current turn improves prediction on four of seven tracked attributes, and most of that gain is a constant offset per prompt rather than turn-to-turn dynamics.
2. We fit, by least squares, a linear model that predicts the next few readings from the last few readings and the command. It ranks first on four of seven tracked attributes, and a learned nonlinear feature map beats it on none.
3. We build a black-box predictive controller with no extra call to the target model and bound its selection error by the prediction error. On three public instruction-following benchmarks it raises constraint satisfaction by 14.5 and 14.8 points over uncontrolled Qwen3-4B and Qwen3-8B, and exceeds the stronger of two white-box latent controllers by 5.3 and 5.7 points, as point estimates with no interval.

2 Problem Formulation

Two objectives share one interaction model: predicting the trajectory of a measured attribute over several exchanges, and driving that attribute to a target through prompt-level commands.

Modeling. At interaction step t, a frozen large language model receives a task input, the accumulated dialogue history, and a textual control instruction, and produces the next response. The resulting input–output interaction is $o_{t+1} \sim p_\theta(\cdot \mid x, \mathcal{H}_t, u_t)$ with reading $y_{t+1} = V(o_{t+1})$. Here x denotes the task input, p_θ the frozen LLM, $\mathcal{H}_t$ the dialogue history available at step t, and u_t the textual instruction supplied by the controller. The response o_{t+1} is text, while V is a task-specific verifier that maps the completed response to a scalar attribute y_{t+1}, such as sentence length or sentiment polarity. We track one such attribute at a time, and a step corresponds to one complete prompt–response exchange. The verifier output is the only signal the controller observes, so the measured attribute y_t is used both to fit the model and to state the objective.

Control decisions depend on the consequences of an instruction over several future turns. The dynamics model is therefore evaluated by its multi-step prediction error $\ell_h = \left(y_{t+h} - \widehat{y}_{t+h|t}\right)^2$ for $h = 1, \ldots, H$. Here $\widehat{y}_{t+h|t}$ is the prediction of the attribute at step $t + h$ from information available at step t, and y_{t+h} is the realized reading. The error should remain small throughout the horizon H, rather than only at the one-step prediction $h = 1$.

Control. At each interaction step, the controller selects a scalar command c_t, which determines the textual instruction u_t applied to the LLM. Over a finite horizon of H steps, the control objective is $J_t = \sum_{h=1}^{H} \left(r - y_{t+h}\right)^2$. Here $r \in \mathbb{R}$ denotes the desired value of the tracked attribute, and $e_t = r - y_t$ is the corresponding tracking error. Predictive control replaces the future readings y_{t+h}, which are unavailable at decision time, by their predictions $\widehat{y}_{t+h|t}$ when selecting the commands.

The controlled quantity is continuous, whereas the experiments report constraint satisfaction. They meet in the task-specific feasible set $\mathcal{Y}$: a response is feasible for $y_t \in \mathcal{Y}$ and the reference satisfies $r \in \mathcal{Y}$, so driving y_t toward r raises the feasible rate.

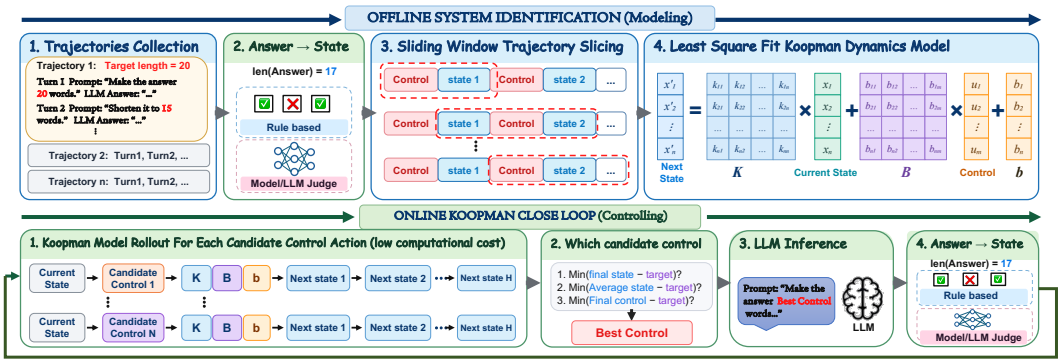


Figure 1: Overview of KOMPAS. Top, offline fitting: recorded trajectories are mapped by the verifier V to readings, a sliding window forms the state s_t, and the affine Koopman model is fit by least squares. Bottom, online closed loop: candidate commands c_t are rolled forward through the dynamics model, scored against the reference r, and the best one is sent through the prompting protocol as u_t; the next reading is shifted into the state and the selection repeats.

The reference r is the desired output, whereas the command c_t is what the controller chooses, and it reaches the LLM as the instruction u_t. A direct target-following controller sets $c_t = r$; predictive control allows $c_t \neq r$.

3 Method

KOMPAS consists of an offline dynamics model, fit on recorded attribute readings, and an online controller that plans in that model and replans after every reply (receding horizon). The frozen LLM p_θ is never updated, and candidate evaluation requires no additional query to it. Figure 1 summarizes the pipeline; the first three subsections follow it and §3.4 gives the guarantee.

3.1 Observable-State Koopman Dynamics

The controller does not model text. Each completed response o_t is mapped by the task-specific verifier V from Section 2 to a scalar attribute $y_t = V(o_t)$. The particular definition of V depends on the task and is given in the experimental setup; the dynamics model uses only the resulting scalar readings.

A single reading generally does not retain enough temporal context to predict the next one. The same value y_t, for example, may occur along an increasing or decreasing trajectory and therefore lead to different future responses. We represent this short-term history with delay coordinates $s_t = [\, y_t,\, y_{t-1},\, \ldots,\, y_{t-L} \,]^\top \in \mathbb{R}^{L+1}$ with readout $y_t = Cs_t$, where $L \geq 1$ is the number of previous readings retained and $C = [1, 0, \ldots, 0]$ reads the current attribute from the state. We use s_t as a finite-memory stand-in for the part of the dialogue history that predicts the attribute, not as a summary of the dialogue text itself.

The controller acts through a scalar command rather than over the space of text strings. A fixed prompting protocol maps a command $c_t \in \mathcal{C} \subseteq \mathbb{R}$ to the textual instruction $u_t = \mathcal{P}(x, \mathcal{H}_t, c_t)$. Thus, after fixing $\mathcal{P}$, the controlled interaction can be viewed at the attribute level as a transition from (s_t, c_t) to s_{t+1}.

This controlled transition induces a command-conditioned Koopman operator on observables. For an observable f, we define $(\mathcal{K}_c f)(s) = \mathbb{E}[f(s_{t+1}) \mid s_t = s, c_t = c]$. The operator is linear in f even when the underlying prompt–response interaction is nonlinear. We use the delay coordinates in s_t as a finite dictionary of observables, that is, the feature set the linear model acts on. The induced dynamics are approximated with the affine model

$$\widehat{s}_{t+1} = Ks_t + Bc_t + b, \qquad \widehat{y}_{t+1} = C\widehat{s}_{t+1}. \tag{1}$$

Here $K \in \mathbb{R}^{(L+1)\times(L+1)}$ propagates the memory state, $B \in \mathbb{R}^{(L+1)\times 1}$ captures the effect of the command, and $b \in \mathbb{R}^{L+1}$ is a bias term. Sharing one transition matrix across commands and letting the command enter through one linear channel are modeling choices, not consequences of Koopman linearity. We do not assume that the frozen LLM admits an exact finite-dimensional Koopman representation.

Koopman dynamics may also be defined on lifted coordinates $E_\phi(s_t)$: a feature map applied to the window before the linear model, called a lifting. Our reported predictor uses the identity lifting, that is, no feature map; a learned lifting is kept as a comparison with its objective in Appendix A. The reported predictor is thus also a delay regression with an exogenous input. The Koopman view places both dictionaries in one framework and gives closed-form multi-step prediction.

3.2 Offline Fitting (System Identification)

We fit the dynamics model from recorded interaction trajectories. Sliding a window of $L+1$ readings over a trajectory produces transition tuples (s_t, c_t, s_{t+1}), where c_t is the command applied before observing the successor state.

Because K, B, and b enter the one-step prediction linearly, the identity-lifting model is estimated by least squares, $(\widehat{K}, \widehat{B}, \widehat{b}) = \arg\min_{K,B,b} \sum_t \|s_{t+1} - (Ks_t + Bc_t + b)\|_2^2$. The fit is performed offline and does not modify the parameters θ of the target LLM. The learned dynamics can be trusted only on the state–command combinations represented in the recorded trajectories.

3.3 Receding-Horizon Prompt Control

At inference time, candidate commands are evaluated in the learned dynamics rather than by querying the LLM. Let $\mathcal{C}_t \subseteq \mathcal{C}$ be the finite set of candidate commands at step t; each $c \in \mathcal{C}_t$ is held fixed inside a hypothetical H-step rollout from the observed state s_t. The predicted state obeys $\widehat{s}_{t|t}(c) = s_t$ and $\widehat{s}_{t+h|t}(c) = K\widehat{s}_{t+h-1|t}(c) + Bc + b$, with $\widehat{y}_{t+h|t}(c) = C\widehat{s}_{t+h|t}(c)$.

Because the transition is affine, the rollout has the closed form $\widehat{s}_{t+h|t}(c) = K^h s_t + \sum_{j=0}^{h-1} K^j(Bc + b)$, so candidate evaluation is linear propagation through the fitted model and adds no call to p_θ.

Each candidate is scored by the deviation of its predicted trajectory from the desired attribute,

$$\widehat{J}_t(c) = \sum_{h=1}^{H} w_h \left|\widehat{y}_{t+h|t}(c) - r\right|^2, \qquad c_t^\star \in \arg\min_{c\in\mathcal{C}_t} \widehat{J}_t(c), \quad w_h \geq 0. \tag{2}$$

Here $\widehat{J}_t(c)$ is the predicted cost of holding candidate c, w_h weights the horizon steps, and $c_t^\star$ is the selected command. Scoring the whole predicted trajectory allows the selected command to differ from the reference while the attribute is still moving toward it.

The controller executes only $c_t^\star$: it sends $u_t = \mathcal{P}(x, \mathcal{H}_t, c_t^\star)$, measures $y_{t+1} = V(o_{t+1})$ from the actual response, shifts it into the memory state, and repeats the selection from s_{t+1}. The held-command assumption applies only inside a candidate's hypothetical rollout; the executed sequence is replanned after every reply and may change command at every step.

3.4 Why Prediction Accuracy Matters

Predictive control is useful only when ranking candidates under the learned dynamics model preserves their ranking under the true interaction, up to a bounded error. The following result states this for the comparison among candidates, each held for H steps, at one control step.

For a candidate c, let y_h^c denote the realized attribute after h further interactions under the same held command, and let $J_t(c) = \sum_{h=1}^{H} w_h |y_h^c - r|^2$ be its realized trajectory cost. Suppose the one-step state residual along candidate rollouts is bounded by $\varepsilon_{\mathrm{dyn}}$, the readout residual is bounded by $\varepsilon_{\mathrm{rec}}$, and the readout is Lipschitz on the region visited by the rollouts. Then the h-step prediction error is bounded by some Δ_h, and the command selected by Eq. (2) satisfies $J_t(c_t^\star) \leq \min_{c\in\mathcal{C}_t} J_t(c) + 2\delta_J$, where δ_J is the corresponding bound on the difference between predicted and realized candidate costs. Appendix E gives the explicit forms of Δ_h and δ_J and the proof.

Table 1: Multi-step prediction skill on seven multi-turn tasks (forecasts from step t only, no reading fed back within the horizon). Higher is better. Best per column in bold, second underlined.

Predictor	Sentence len.	Joint-3	Joint-2	Sentiment	Defense	Constraint	CEFR
Stateless	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Last-turn	0.166	0.514	0.395	0.276	**0.067**	0.285	0.518
Memory w/o command	<u>0.764</u>	<u>0.824</u>	0.615	**0.530**	0.023	0.517	<u>0.561</u>
LSTM	0.750	0.808	<u>0.634</u>	0.423	-0.054	**0.546**	-0.101
KOMPAS (learned lifting)	0.752	0.821	0.627	0.396	0.021	0.471	-0.032
KOMPAS (linear)	**0.769**	**0.826**	**0.640**	<u>0.496</u>	<u>0.035</u>	<u>0.519</u>	**0.567**

Table 2: Constraint satisfaction on IFBench, IFEval and COLLIE. Access states what each controller needs from the model.

Model	Method	Access	IFBench				IFEval				COLLIE			Avg.
			WC	UWC	NS	NP	KF	WC	NS	NP	Word	Sent.	Para.	
Qwen3-4B	Base	–	25.5	7.0	36.7	12.4	20.7	44.6	33.9	28.9	23.0	51.3	11.9	26.9
	TMPC	white-box	35.2	9.9	46.3	19.0	29.1	53.4	43.1	37.5	33.2	59.6	21.3	35.2
	RE-Control	white-box	38.2	10.4	47.1	19.7	29.8	54.2	43.8	38.1	34.0	60.2	21.9	36.1
	KOMPAS	black-box	**44.6**	**12.8**	**53.4**	**24.1**	**35.2**	**59.8**	**49.6**	**43.7**	**39.8**	**65.4**	**26.7**	**41.4**
Qwen3-8B	Base	–	31.8	9.6	42.8	16.7	26.8	50.9	39.7	34.8	29.1	57.4	17.2	32.4
	TMPC	white-box	41.4	12.8	52.1	23.7	35.4	59.4	49.5	43.7	39.3	65.2	27.0	40.9
	RE-Control	white-box	42.6	13.2	52.9	24.2	36.0	60.1	50.2	44.3	40.0	65.9	27.6	41.5
	KOMPAS	black-box	**50.8**	**15.9**	**59.7**	**29.3**	**41.8**	**65.7**	**55.8**	**49.9**	**46.1**	**71.4**	**32.8**	**47.2**

The result is local: it concerns candidate selection at one replanning step and does not assert convergence of the closed loop. Its role is to link the two objectives: more accurate rollouts bring the selected command closer to the best command under the realized costs.

4 Experiments

The experiments answer two questions, one per experiment line. **RQ1 (modeling).** Does the learned operator predict a multi-turn attribute from observable readings alone? Its parts are (1a) history versus the current reading and (1b) the linear case versus a learned lifting or an LSTM (§4.2), and (1c) what the memory state carries (§4.3). **RQ2 (control).** Does predictive control built on that operator raise constraint satisfaction on a frozen model? Its parts are (2a) against no control and (2b) against two white-box latent controllers, both in §4.5. §4.4 displays the mechanisms behind RQ1 on the raw readings, and §4.6 states when a gain on RQ1 can become a gain on RQ2.

4.1 Experimental Setup

Experiments cover two task groups. The first is self-built multi-turn tasks that form the modeling line. Table 1 carries seven of them: sentence length, Joint-3, Joint-2, sentiment, defense, constraint and CEFR. Joint-2 tracks two attributes of one sentence at once, word count and average word length; Joint-3 adds comma count. The other five tasks track a single attribute. Each task is a set of prompts, called items; every item is run for several turns and yields one trajectory. Appendix B carries the four small-sample columns, sentiment and defense again plus average word length and GSM8K-sharded, with their sample-size caveat. The second group is three public instruction-following benchmarks that form the control line: IFBench, IFEval and COLLIE. Together they carry eleven subtasks: WC, UWC, NS and NP for IFBench; KF, WC, NS and NP for IFEval; Word, Sent. and Para. for COLLIE. The modeling line runs on a single LLM, Qwen3-4B-Instruct-2507, which produces all self-built trajectories and also serves as the judge for the LLM-scored attributes, so every number in Table 1 comes from that one LLM. The control line is separate: it reports two frozen target models, Qwen3-4B and Qwen3-8B, produced in a separate pipeline whose protocol Appendix D records.

The modeling line compares six predictors, the six rows of Table 1. Stateless is a memoryless null, Last-turn is a single-reading state, and Memory w/o command is a multi-turn state without the command term. LSTM is a recurrent sequence baseline, and KOMPAS (learned lifting) and

KOMPAS (linear) are the two operator variants. KOMPAS (linear) uses the identity lifting of §3.1, the reported instance throughout this paper. Window length, horizon, row counts, trajectory counts and the bootstrap protocol vary by task and are tabulated in Appendix C; every interval in §4.2–§4.4 is a paired bootstrap over trajectories.

The control line compares four methods per frozen model, the four rows of Table 2: Base, TMPC, RE-Control and KOMPAS. Base applies no control. TMPC and RE-Control are white-box latent controllers that require activation access to the frozen model. KOMPAS needs only the instruction it sends, marked black-box Access, since it never opens the model. TMPC and RE-Control are reproductions under that same protocol rather than the original authors' releases.

A predictor that never beats the best memoryless guess scores zero, and one that halves that guess's error scores one half. We define the prediction skill score as $\mathrm{Skill}_H = 1 - \sum_{h=1}^{H} \ell_h \big/ \sum_{h=1}^{H} \ell_h^{\mathrm{null}}$. Here ℓ_h is the squared forecast error at horizon step h from §2, and ℓ_h^{null} is the same quantity for that memoryless guess. That guess is the Stateless row of Table 1, which therefore scores 0.000 by construction. The score Skill_H is therefore the fraction of that null's horizon error which the operator removes. A score of zero means tied with that null; a negative score means worse than it.

The control-line metric is each benchmark's constraint-satisfaction score, a per-subtask percentage with no interval attached. The modeling line tests whether the operator predicts the trajectory; the control line tests whether acting on that prediction changes the outcome.

4.2 Predictive Accuracy of the Learned Operator

For RQ1a, we ask whether using multi-turn history improves prediction over using only the current reading. For RQ1b, we compare the learned lifting and LSTM with KOMPAS, our linear case. The setup, tasks, and baselines are described in §4.1; Table 1 reports the skill score defined there.

KOMPAS ranks first on four of the seven task columns and second on the remaining three. Its point estimate is close to the top method in each of those three cases, trailing by no more than 0.034. Multi-turn history improves on the current reading in four columns, with paired bootstrap intervals that exclude zero. In the other three, the difference is not distinguishable from zero. The history window captures the attribute's current level and, to a lesser extent, its trend—information unavailable from a single reading.

The learned nonlinear lifting does not outperform the identity lifting with an interval excluding zero on any of the seven columns. Figure 4(c) shows the per-column contrasts; §4.4 discusses a possible explanation. Compared with the LSTM, the results are mixed: the linear case performs better on Joint-3 and CEFR, while the LSTM performs better on Constraint. We do not establish a mechanism for the LSTM's advantage on Constraint.

Overall, history improves prediction over the current reading on four of seven task columns, supporting RQ1a for those tasks. Neither the learned lifting nor the LSTM outperforms KOMPAS consistently, so the results do not support RQ1b.

4.3 What the Memory State Carries

RQ1c, in two parts: how many past readings the state needs, and whether the gain from memory is inter-turn dynamics or a fixed level per prompt item. Tasks, baselines and the skill score follow §4.1; both experiments vary one thing in the KOMPAS (linear) row of Table 1. Both experiments fit every depth on the same transitions, which separates the depth of the state from the size of its training set; Table 1 does not. Under this matched window the Table 1 memory gains shrink, and Appendix C compares the two window conventions.

Figure 2(a) covers the four tasks whose data allow varying the window length. On three of them the whole gain arrives with the second reading, and the third and fourth readings add nothing that separates from zero; on Joint-2 the gain itself does not separate from zero. Constraint is the exception: each added reading keeps paying, with every step separated from zero. Of the three tasks recorded at only two window lengths, only sentiment gains from one extra reading (Figure 2b). A level and a trend take two readings to express, and the attribute tasks carry little structure beyond that pair. No mechanism is established for why constraint keeps gaining, and none is proposed here.

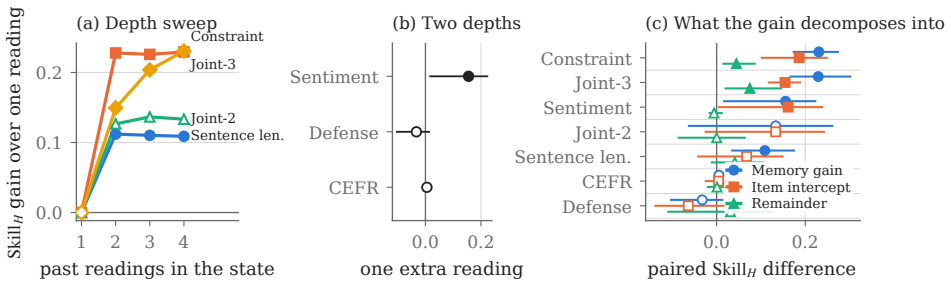


Figure 2: How deep the memory state has to be, and what its gain decomposes into, both under a matched training window. (a) Gain of a state holding d past readings over a state holding one, on the four tasks whose data allow varying the window length. (b) The same gain for one extra reading on the three designs that carry two depths. (c) Per task: the memory gain over Last-turn, the part a per-item intercept recovers, and the remainder; intercepts are estimated on turns outside the scored window. Filled markers: 95% paired bootstrap interval excludes zero.

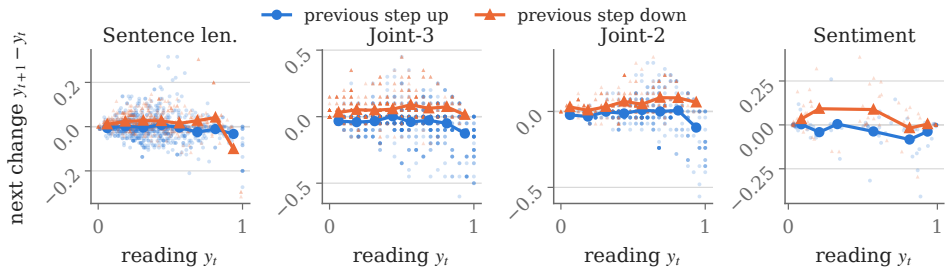


Figure 3: One reading is consistent with two opposite next steps. For each task, the next change against the current reading, split by the direction of the step before it; faint marks are transitions and each line is the mean of its group in eight equal-width bins. Transitions whose previous step was flat carry no direction and are dropped, which removes 7% of transitions on sentiment and up to 66% on Joint-2.

A per-item intercept added to the one-reading state recovers most of the memory gain: 81% on constraint and 67% on Joint-3. Only on those two tasks does the remainder beyond the intercept separate from zero; on sentiment and Joint-2 the intercept accounts for the whole gain (Figure 2c). This happens because each prompt item holds its own level across turns, and several past readings estimate that level where one reading cannot. Together these results answers RQ1c. A remainder attributable to dynamics is measured on two of seven tasks.

4.4 Mechanisms Read Off the Data

This subsection adds no new research question. It examines two mechanisms suggested by the fitted scores in §4.2, as well as a data-collection artifact relevant to §4.6. Figure 3 and Figure 4(a,b) show patterns in the recorded readings, not predictor skill. Their values therefore should not be compared with skill scores. Figure 4(c), by contrast, shows the paired skill contrasts reported in Table 1.

At the same reading, the next change differs depending on whether the preceding step moved up or down, across all four tasks (Figure 3). The size of this separation does not follow the pattern of memory gains: sentence length has the largest gain in Table 1, but the smallest separation here. Alongside the results in §4.3, this suggests that the window's benefit comes more from information at the item level than from turn-to-turn trends. The command plots reveal a data-collection artifact relevant to the caveat in §4.6 and Appendix F. In both tasks, the applied command remains at the target across the full range of tracking errors in the recorded transitions. These data therefore cannot identify an independent command effect.

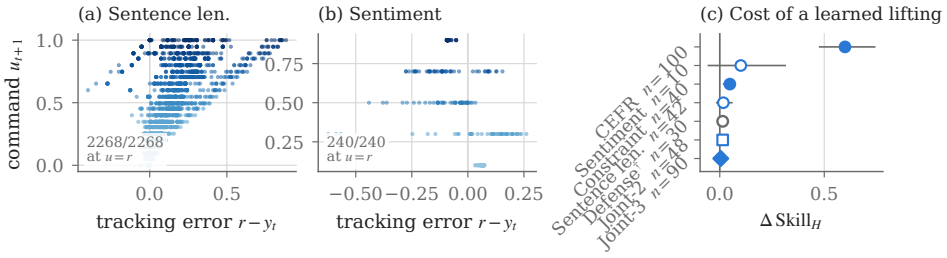


Figure 4: What the command channel carries, and what a learned lifting loses. (a) and (b) Applied command against tracking error, on the two Table 1 tasks that log a per-turn command; color marks the target and repeats the vertical position, which is the point. (c) Paired Skill_H contrast of the linear case over the learned lifting, on all seven tasks, grouped by the number of attributes a task tracks: a circle marks the five single-attribute tasks, a square Joint-2 and a diamond Joint-3. A filled marker means the 95% interval excludes zero; the dagger marks a caveated entry.

The comparison between the linear case and the learned lifting shows a different pattern. The learned lifting's skill disadvantage is concentrated in the five single-attribute tasks and is close to zero on Joint-2 and Joint-3 (Figure 4c). This pattern does not track the amount of training data: the largest gap occurs in a task fit on 100 trajectories, while the smallest occurs in one fit on 90. Across these seven tasks, the results suggest that a single tracked attribute gives the lifting less structure to capture.

Taken together, the window distinguishes opposite next steps at the same reading, while the skill gap associated with a learned lifting is larger on single-attribute tasks than on multi-attribute tasks.

4.5 Constraint Satisfaction under Predictive Control

For RQ2a and RQ2b, we ask whether predictive control using the learned operator improves constraint satisfaction over the same frozen model without control, and how it compares with two white-box latent controllers. The setup, benchmarks, and baselines are described in §4.1; Table 2 reports the constraint-satisfaction score defined there.

Averaged across the eleven subtasks, predictive control improves constraint satisfaction over the uncontrolled baseline by 14.5 points on Qwen3-4B and 14.8 points on Qwen3-8B. These are point estimates, with no seeds or intervals. The controller scores each candidate by its predicted trajectory rather than its immediate effect, allowing it to choose a command different from the target while the attribute is still changing. Predictive control also outperforms the stronger white-box baseline, RE-Control, by 5.3 points on Qwen3-4B and 5.7 points on Qwen3-8B. These are also point estimates without intervals. RE-Control uses model activations, whereas predictive control operates from text alone.

Across the 22 model-by-subtask comparisons, predictive control scores higher than the uncontrolled baseline in every case and higher than both white-box controllers in every case. This count answers RQ2a and RQ2b affirmatively; it is not a claim of statistical significance.

4.6 From Prediction Quality to Control Quality

Prediction accuracy matters for control only when it helps select a better command. The selection bound in §3.4 formalizes this link: if each candidate's predicted cost differs from its realized cost by at most δ_J, the selected command's realized cost is at most $2\delta_J$ above the best realized cost. This is a theoretical guarantee for one comparison at a single step, with each candidate held for H steps; it does not describe the replanned closed loop.

Candidate evaluation requires one linear propagation and a readout using the closed-form rollout in §3.3, so it adds no queries to the target model. Because the transition is affine and the command enters through a scalar column, the h-step lookahead reduces to a matrix power.

We also compared the closed-loop controller with the best fixed schedule under the same instruction budget on Defense, Constraint, and CEFR (Appendix D). The controller tied on two tasks and performed worse on Defense. On these tasks, the instruction itself shifted reading by 0.12–0.14, while rescheduling the same budget changed it by at most 0.03.

Finally, on four of the seven modeling columns the command channel carries no information beyond the memory state. Two of these tasks log a per-turn command, and it stays at the target on every transition. For the other three columns, the estimated command effects all have intervals containing zero (Appendix F). This limitation reflects how the trajectories were collected, rather than a finding about the controller. Together, the bound connects the prediction error to command selection in a single step, and candidate evaluation does not require additional queries to the target-model. The equal-budget and command-effect checks show where the predictive gains studied in RQ1 may not carry over to the control gains studied in RQ2.

5 Related Work

Multi-turn drift and instability. Instruction following is measured to decay over a dialogue, and later work characterizes the turn-level signal: context equilibria, attractor states, and persona drift detectable from text alone (Li et al., 2024; Dongre et al., 2025; Ko & Geiping, 2026; Wang, 2026; Luo & Laban, 2026). One reading of these results is that an instruction given once competes with every later turn. The proposed remedies stop at a decoder-side correction or a reminder; the instruction is not an input to a model that predicts the next reading.

Steering and control in the latent space. Activation steering edits hidden states or logits during generation (Allbert et al., 2025; Yang et al., 2025; Shi et al., 2026); feedback variants close the loop with PID control (Nguyen et al., 2026; Bharadwaj, 2026), barrier functions (Miyaoka & Inoue, 2026a;b; Tran et al., 2026; Hu et al., 2026), or linear optimal and predictive control (Cheng & Alonso, 2026; Skifstad et al., 2026; Kong et al., 2024). These controllers take the token as clock and a hidden state as state, so they need white-box access; a caller who can change only the next prompt cannot run them.

Control-theoretic views of language models. A third line writes prompting as a control problem: reachability of target outputs (Bhargava et al., 2024), a controllability toolkit (Cheng et al., 2026), conditions for steerability (Fan et al., 2026), and linear dynamics fit to token or embedding trajectories (Akrout & Wilson, 2026; Akrout et al., 2026; Nosrati et al., 2026). The object is a token or embedding sequence and the output is a reachability statement, a bound, or a classifier. No model with the instruction as input is identified at the level of one exchange.

Koopman predictors with inputs. The Koopman operator represents nonlinear dynamics as a linear operator on observables. Linear predictors with inputs, fit by least squares, plug into model predictive control (Proctor et al., 2014; Korda & Mezić, 2018); delay coordinates supply observables under partial observation (Arbabi & Mezić, 2017; Brunton et al., 2017); autoencoders learn the dictionary (Lusch et al., 2018; Enyeart & Lin, 2024). There the inputs are actuator signals and the observables are sensor readings. An instruction as input, a verifier score as observable, and a black-box text generator as plant have not been reported.

6 Conclusion

We studied holding a reply attribute of a frozen language model at a target across a dialogue in which the attribute carries inertia. The takeaway is that **a controller should choose its instruction from where the attribute is heading, not from where it is**, which lets it act while the attribute is moving. KOMPAS realizes this view: a linear dynamics model fit to past readings and instructions, and predictive selection that adds no query to the model. On three public instruction-following benchmarks, KOMPAS exceeds the stronger of two white-box latent controllers by 5.3 and 5.7 points while reading only text. KOMPAS applies where a verifier scores the attribute after every reply and the instruction varies along one command axis. Its edge over a fixed schedule needs the instruction's effect to depend on the state; on the self-built tasks that effect varies little, and the loop ties or trails an equal-budget schedule. We plan to extend the command to several axes so that jointly measured attributes can be tracked together.

AI USE STATEMENT

Large language models were used as writing assistants for this paper. Specifically, they helped polishing grammar, wording, and terminology. They were not used to generate the research idea, the hypotheses, or the method; to design or run the experiments. All AI-assisted text was reviewed and edited by the authors, who take full responsibility for the content of this paper.

REFERENCES

Mohamed Akrout and Dan Wilson. Guarantees on dynamical system distinguishability for llm token generation, 2026. URL https://arxiv.org/abs/2607.28667.

Mohamed Akrout, Olivera Kotevska, and Dan Wilson. Enforcing llm safety through dmd-based classification of prompt-response embedding dynamics, 2026. URL https://arxiv.org/abs/2608.19579.

Rumi Allbert, James K. Wiles, and Vlad Grankovsky. Identifying and manipulating personality traits in llms through activation engineering, 2025. URL https://arxiv.org/abs/2412.10427.

Hassan Arbabi and Igor Mezić. Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the koopman operator. *SIAM Journal on Applied Dynamical Systems*, 16(4):2096–2126, January 2017. ISSN 1536-0040. doi: 10.1137/17m1125236. URL http://dx.doi.org/10.1137/17M1125236.

Aryasomayajula Ram Bharadwaj. Adaptive activation steering for efficient llm reasoning via closed-loop pid control, 2026. URL https://arxiv.org/abs/2506.18831.

Aman Bhargava, Cameron Witkowski, Shi-Zhuo Looi, and Matt Thomson. What's the magic word? a control theory of llm prompting, 2024. URL https://arxiv.org/abs/2310.04444.

Steven L. Brunton, Bingni W. Brunton, Joshua L. Proctor, Eurika Kaiser, and J. Nathan Kutz. Chaos as an intermittently forced linear system. *Nature Communications*, 8(1), May 2017. ISSN 2041-1723. doi: 10.1038/s41467-017-00030-8. URL http://dx.doi.org/10.1038/s41467-017-00030-8.

Emily Cheng and Carmen Amo Alonso. Liseco: Linear semantic control for language generation, 2026. URL https://arxiv.org/abs/2405.15454.

Emily Cheng, Carmen Amo Alonso, Federico Danieli, Arno Blaas, Luca Zappella, Pau Rodriguez, and Xavier Suau. Genctrl – a formal controllability toolkit for generative models, 2026. URL https://arxiv.org/abs/2601.05637.

Vardhan Dongre, Ryan A. Rossi, Viet Dac Lai, David Seunghyun Yoon, Dilek Hakkani-Tür, and Trung Bui. Drift no more? context equilibria in multi-turn llm interactions, 2025. URL https://arxiv.org/abs/2510.07777.

Dustin Enyeart and Guang Lin. Loss terms and operator forms of koopman autoencoders, 2024. URL https://arxiv.org/abs/2412.04578.

Chenrui Fan, Yize Cheng, Ming Li, Soheil Feizi, and Tianyi Zhou. When is your llm steerable?, 2026. URL https://arxiv.org/abs/2606.11599.

Hanjiang Hu, Alexander Robey, and Changliu Liu. Steering dialogue dynamics for robustness against multi-turn jailbreaking attacks, 2026. URL https://arxiv.org/abs/2503.00187.

Ting-Wen Ko and Jonas Geiping. Attractor states emerge in multi-turn llm conversations, 2026. URL https://arxiv.org/abs/2606.30571.

Lingkai Kong, Haorui Wang, Wenhao Mu, Yuanqi Du, Yuchen Zhuang, Yifei Zhou, Yue Song, Rongzhi Zhang, Kai Wang, and Chao Zhang. Aligning large language models with representation editing: A control perspective, 2024. URL https://arxiv.org/abs/2406.05954.

Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, July 2018. ISSN 0005-1098. doi: 10.1016/j.automatica.2018.03.046. URL http://dx.doi.org/10.1016/j.automatica.2018.03.046.

Kenneth Li, Tianle Liu, Naomi Bashkansky, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Measuring and controlling instruction (in)stability in language model dialogs, 2024. URL https://arxiv.org/abs/2402.10962.

Han Luo and Guy Laban. Spasm: Stable persona-driven agent simulation for multi-turn dialogue generation, 2026. URL https://arxiv.org/abs/2604.09212.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1), November 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-07210-0. URL http://dx.doi.org/10.1038/s41467-018-07210-0.

Yuya Miyaoka and Masaki Inoue. Control barrier function for aligning large language models. *IEEE Transactions on Control Systems Technology*, 34(4):1821–1830, July 2026a. ISSN 2374-0159. doi: 10.1109/tcst.2026.3675329. URL http://dx.doi.org/10.1109/TCST.2026.3675329.

Yuya Miyaoka and Masaki Inoue. Control barrier function for aligning large language models. *IEEE Transactions on Control Systems Technology*, 34(4):1821–1830, July 2026b. ISSN 2374-0159. doi: 10.1109/tcst.2026.3675329. URL http://dx.doi.org/10.1109/TCST.2026.3675329.

Dung V. Nguyen, Hieu M. Vu, Nhi Y. Pham, Lei Zhang, and Tan M. Nguyen. Activation steering with a feedback controller, 2026. URL https://arxiv.org/abs/2510.04309.

Komeil Nosrati, Aleksei Tepljakov, Juri Belikov, and Eduard Petlenkov. When control meets large language models: From words to dynamics. *Engineering Applications of Artificial Intelligence*, 178:115119, August 2026. ISSN 0952-1976. doi: 10.1016/j.engappai.2026.115119. URL http://dx.doi.org/10.1016/j.engappai.2026.115119.

Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz. Dynamic mode decomposition with control, 2014. URL https://arxiv.org/abs/1409.6358.

Yingdong Shi, Ruiming Zhang, Changming Li, Zhiyu Yang, Kaixing Zhang, Jingyi Yu, and Kan Ren. Unisteer: Text-guided flow matching in activation space for versatile llm steering, 2026. URL https://arxiv.org/abs/2605.30076.

Julian Skifstad, Xinyue Annie Yang, and Glen Chou. Local linearity of llms enables activation steering via model-based linear optimal control, 2026. URL https://arxiv.org/abs/2604.19018.

Thanh Q. Tran, Arun Verma, Kiwan Wong, Bryan Kian Hsiang Low, Daniela Rus, and Wei Xiao. Barriersteer: Llm safety via learning barrier steering, 2026. URL https://arxiv.org/abs/2602.20102.

Chunxiao Wang. Nautilus compass: Black-box persona drift detection for production llm agents, 2026. URL https://arxiv.org/abs/2605.09863.

Jingyuan Yang, Rongjun Li, Weixuan Wang, Ziyu Zhou, Zhiyong Feng, and Wei Peng. Lf-steering: Latent feature activation steering for enhancing semantic consistency in large language models, 2025. URL https://arxiv.org/abs/2501.11036.

A Estimator Details and the Learned Lifting

The reported instance is fit by the least-squares problem of Section 3.2, on the transition tuples $\mathcal{D} = \{(s_t, c_t, s_{t+1})\}$ constructed there. This appendix covers the learned-lifting variant instead, in which the encoder and the decoder are not the identity. What a lifting is, and why the reported instance leaves it off, is said in Section 3.1.

Training transition. Fitting the operator needs a one-step latent prediction from an encoded state and the applied command. Let

$$q_t = KE_\phi(s_t) + Bc_t + b. \tag{3}$$

Here q_t is the predicted encoding of s_{t+1} under the current parameters, computed from the encoder E_ϕ, the transition K, the command column B, and the bias b.

Training objective. Three losses jointly shape the encoder, decoder, and transition. We optimize

$$\begin{aligned} \mathcal{L}_{\text{rec}} &= \mathbb{E}_{\mathcal{D}}\left[\|s_t - D_\psi(E_\phi(s_t))\|_2^2\right], \\ \mathcal{L}_{\text{lin}} &= \mathbb{E}_{\mathcal{D}}\left[\|E_\phi(s_{t+1}) - q_t\|_2^2\right], \\ \mathcal{L}_{\text{pred}} &= \mathbb{E}_{\mathcal{D}}\left[\|s_{t+1} - D_\psi(q_t)\|_2^2\right], \\ \mathcal{L} &= \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{lin}}\mathcal{L}_{\text{lin}} + \lambda_{\text{pred}}\mathcal{L}_{\text{pred}}. \end{aligned} \tag{4}$$

The nonnegative weights λ_{rec}, λ_{lin} and λ_{pred} set how much each term contributes to the total. The reconstruction term $\mathcal{L}_{\text{rec}}$ preserves information present in the memory state s_t. The latent-consistency term $\mathcal{L}_{\text{lin}}$ pulls the encoded successor toward the affine prediction q_t. The prediction term $\mathcal{L}_{\text{pred}}$ ties that latent transition back to accuracy in the observed state space.

These three terms are complementary. Latent consistency alone admits uninformative constant encodings, since a constant encoder can drive $\mathcal{L}_{\text{lin}}$ to zero without capturing any structure. Reconstruction alone does not require the representation to support prediction, since $D_\psi(E_\phi(s_t))$ can recover s_t while carrying no information about s_{t+1}. Training updates (ϕ, ψ, K, B, b), and the frozen language model's parameters θ never change.

B Small-Sample Columns

Table 3: The four small-sample columns, reported for completeness.

Predictor	Sentiment	Word len.	Defense	GSM8K-sharded
Stateless	0.000	**0.000**	0.000	<u>0.000</u>
Last-turn	0.276	<u>-0.018</u>	**0.067**	**0.637**
Memory w/o command	**0.530**	-0.677	0.023	-0.186
LSTM	0.423	-0.071	-0.054	-0.795
KOMPAS (learned lifting)	0.396	-0.399	0.021	-0.130
KOMPAS (linear)	<u>0.496</u>	-0.427	<u>0.035</u>	-0.129

Table 3 reports the four columns that §4.1 sets aside: sentiment and defense, which also appear in Table 1, plus average word length and GSM8K-sharded. The four rest on 10, 14, 30 and 4 trajectories, against 40 to 100 for the other five columns of Table 1. A paired bootstrap over so few trajectories cannot separate the six rows, so the ranking marks are point estimates.

The Defense column. Three properties of that column are not visible in Table 1 and apply to every reading of it. The readings are scored by the model that produced the responses, and a self-judge misses in one direction, so it understates the effect. The column rests on two data seeds, and its intervals resample the thirty attacks rather than the seeds. In rollout error the six rows span 0.0686 to 0.0776 around the Stateless null of 0.0736, a range under 13% of that null. The column reads as a design that cannot separate the six predictors, and the ranking marks it carries are point estimates without intervals.

Table 4: Modeling-line setup per task. State depth is the number of readings in the memory state ($L + 1$); rows are held-out transitions scored after windowing; trajectories are the bootstrap groups. The last two rows are the extra columns of Appendix B.Best trivial null names which of three memoryless predictors has the lowest rollout error on the scored rows: a constant (the training mean), a per-turn mean, or a linear regression on the turn index (stateless). That predictor is the Stateless row of Table 1 and sets the zero of Skill_H.

Task	State depth	Horizon H	Scored rows	Trajectories	Best trivial null
Sentence len.	4	6	252	42	stateless
Joint-3	4	6	1,620	90	stateless
Joint-2	4	6	576	48	stateless
Sentiment	2	3	30	10	stateless
Defense	2	3	180	30	constant
Constraint	4	4	12,376	40	stateless
CEFR	2	4	4,776	100	stateless
Word len.	2	3	42	14	stateless
GSM8K-sharded	4	4	64	4	constant

Table 5: Matched training window. Every depth is fit on the same transitions. Columns 2–5 give Skill_H of KOMPAS (linear) at a state depth of 1 to 4 readings; the two gain columns are paired bootstrap differences with 95% intervals. The default-window gain is the Table 1 contrast of KOMPAS (linear) against Last-turn, whose training set grows as the depth shrinks. Joint-3 uses the deep-anchor fit, the only one with a recorded interval.

Task	Depth 1	Depth 2	Depth 3	Depth 4	Matched gain (deepest − depth 1)	Default gain (Table 1)
Sentence len.	0.660	0.772	0.770	0.769	$+0.109$ $[+0.035, +0.175]$	$+0.603$ $[+0.438, +0.765]$
Joint-3	0.714	0.920	0.918	0.925	$+0.211$ $[+0.115, +0.339]$	$+0.312$ $[+0.245, +0.385]$
Joint-2	0.507	0.634	0.644	0.640	$+0.133$ $[-0.063, +0.261]$	$+0.245$ $[-0.023, +0.415]$
Sentiment	0.341	0.496	–	–	$+0.155$ $[+0.017, +0.223]$	$+0.221$ $[-0.051, +0.329]$
Defense	0.068	0.035	–	–	-0.033 $[-0.103, +0.013]$	-0.033 $[-0.103, +0.013]$
Constraint	0.288	0.438	0.492	0.519	$+0.231$ $[+0.174, +0.274]$	$+0.234$ $[+0.176, +0.279]$
CEFR	0.562	0.567	–	–	$+0.005$ $[-0.004, +0.017]$	$+0.049$ $[+0.007, +0.097]$

C Modeling-Line Setup Details and Matched Training Window

Table 4 lists the per-task settings behind Table 1. State depth ranges from 2 to 4 readings and horizon from 3 to 6 steps. The depth is the deepest state that still leaves a rollout of at least three steps on that task's trajectory length. Scored held-out rows per task range from 30 to 12,376, counted after windowing. The trajectories those rows come from number 10 to 100 for the seven columns of Table 1, and 4 to 30 for the columns of Appendix B. Every interval reported in §4.2 is a paired grouped bootstrap over trajectories, with 2000 resamples.

The two experiment lines report different metrics in different reference frames. The modeling line's skill score is referenced against a best-trivial-null baseline and can go negative, while the control line's percentage carries no reference point.

The matched-window analysis of §4.3 fits every state depth on the same set of transitions; Table 5 lists the per-task numbers behind Figure 2. The window convention matters. On sentence length the memory gain over Last-turn is $+0.603$ under the default window of Table 1 and $+0.109$ under the matched window, so the two conventions answer different questions. The shrinkage appears where the default window lets a shallower state train on more transitions, which is the case on the four core-line tasks and on CEFR. On defense and constraint the default fit is the matched fit, and the two gains coincide. Under either convention the second reading carries most of the gain; the third and fourth readings add $+0.054$ and $+0.027$ on constraint and stay within ± 0.004 elsewhere.

Table 6: Closed loop against the best equal-budget fixed schedule. Positive differences favor KOMPAS on defense and constraint; on CEFR the quantity is an error, so negative favors KOMPAS. Spend is instructions per trajectory (defense), total actions (constraint) and step-down instructions per item (CEFR). The last column is what the instruction itself buys against no control, for scale.

Task	Quantity	KOMPAS − schedule [95% CI]	Spend (KOMPAS / schedule)	Instruction effect
Defense	safety level, turns 3–5	-0.025 $[-0.050, -0.004]$	0.75 / 1.00	$+0.119$
Constraint	retention, turns 15–20	$+0.001$ $[-0.017, +0.019]$	517 / 300	$+0.116$
CEFR	terminal level error	-0.007 $[-0.068, +0.050]$	3.37 / 1.03	-0.136

D Control-Line Provenance and the Equal-Budget Closed-Loop Comparison

Control-line provenance. The benchmark numbers of Table 2 come from a pipeline separate from the modeling line. It runs two frozen target models, Qwen3-4B and Qwen3-8B, on the eleven subtasks of IFBench, IFEval and COLLIE, and reports each benchmark's constraint-satisfaction score as a per-subtask percentage. Base applies no control, and KOMPAS reads only the text it sends. TMPC and RE-Control are reproductions run under the same protocol, not the original authors' releases. Every cell is a single point estimate: no seeds, intervals or raw logs accompany the table, and the paper treats the numbers as reported.

Equal-budget comparison, protocol. On three self-built tasks the closed loop also runs against open-loop fixed schedules under the same instruction budget cap. The comparison arm is the best of those schedules, chosen after the fact on the same data. Defense sends one safety reminder per trajectory and reads the judge's safety level on turns 3 to 5; its best schedule places the reminder at turn 5. Constraint sends whole-paragraph or targeted reminders over twenty turns and reads constraint retention on turns 15 to 20; its best schedule sends whole-paragraph reminders on fixed turns. CEFR sends step-down instructions toward a target readability level and reads the terminal level error. Its fixed ladder measures the level once and steps down the rounded distance to the target. Defense uses 5 seeds and 8 attack families, constraint 3 seeds and 48 items, CEFR 3 seeds and 100 items; intervals are paired bootstraps over attacks, items and items respectively.

Findings. Table 6 gives the per-task numbers behind the cross-check of §4.6. The loop ties the best schedule on constraint and CEFR, with intervals containing zero. On defense it falls below by 0.025 under the per-attack resampling unit; under the per-trajectory unit that interval also contains zero. The tie is not bought with a smaller budget: KOMPAS spends less only on defense and spends more on the other two tasks. The last column sets the scale: the instruction itself moves the reading by 0.12 to 0.14, while rescheduling the same budget moves it by at most 0.03. Two further checks say the gap is a property of the tasks rather than of the controller. On defense a causal oracle fitted on the logged trajectories, with the same observables as KOMPAS, gains $+0.006$ $[-0.017, +0.029]$ over the best schedule. A perfect controller on those readings would also tie. On CEFR the three feedback policies within one instruction family span 0.013 in terminal error, while switching family moves it by 0.040. The room that feedback can buy on that task is smaller than a wording choice. The defense column carries the self-judge caveat of Appendix B; the other two tasks use an independent judge.

E Assumptions and Proof for the Selection Bound of Section 3.4

The selection bound of Section 3.4 is stated on the memory state, which is the form the reported instance uses. The proof is written in the lifted coordinates of Appendix A, so that it covers the learned variant as well. Set $\xi_t = E_\phi(s_t)$ and write $g_\psi = C \circ D_\psi$ for the attribute readout. The identity lifting recovers the main-text statement, with $\xi_t = s_t$ and $g_\psi = C$.

For a candidate c, let s_h^c and y_h^c be the memory state and the attribute after h further interactions that hold c, starting from $s_0^c = s_t$, and set $\xi_h^c = E_\phi(s_h^c)$. The residuals of the fitted model along such a rollout are what the bound is stated in terms of,

$$\begin{aligned} \xi_{h+1}^c &= K\xi_h^c + Bc + b + \eta_h^c, \quad \|\eta_h^c\|_2 \le \varepsilon_{\mathrm{dyn}}, \\ y_h^c &= g_\psi(\xi_h^c) + \nu_h^c, \qquad\qquad |\nu_h^c| \le \varepsilon_{\mathrm{rec}}. \end{aligned} \tag{5}$$

Here η_h^c is the one-step residual of the fitted transition and ν_h^c is the residual left in the readout. Both bounds are assumed to hold over the candidate rollouts, and both absorb imperfect finite-memory closure, model approximation and generation variability. Assume further that $\|K\|_2 \le \kappa$, and that g_ψ is L_g-Lipschitz on the region the rollouts visit. For stochastic generation the statements hold on the event that these conditions are met.

The predicted and the actual rollout start from the same encoding, so the gap between them is the accumulated one-step residual carried through the readout,

$$\left|y_h^c - \widehat{y}_{t+h|t}(c)\right| \le \varepsilon_{\mathrm{rec}} + L_g \varepsilon_{\mathrm{dyn}} \sum_{j=0}^{h-1} \kappa^j \;\triangleq\; \Delta_h. \tag{6}$$

Here Δ_h is the h-step attribute error bound of the selection bound of Section 3.4. It trades against the horizon: a longer horizon covers more distant consequences and asks the dynamics to hold over more steps. A contraction, $\kappa < 1$, would bound the geometric sum by $1/(1-\kappa)$, and no contraction is assumed for the fitted model.

The cost of a candidate is a weighted sum of squared deviations, so a bound on each deviation gives a bound on the cost. Let R bound how far any attribute, predicted or actual, sits from the reference across candidates and rollout steps,

$$\left|J_t(c) - \widehat{J}_t(c)\right| \le 2R \sum_{h=1}^{H} w_h \Delta_h \;\triangleq\; \delta_J. \tag{7}$$

Here δ_J bounds the gap between the predicted and the actual held-command cost of one candidate. The factor $2R$ comes from the identity $a^2 - b^2 = (a+b)(a-b)$ applied to each deviation pair.

Let $c_t^{\mathrm{opt}} \in \arg\min_{c \in \mathcal{C}_t} J_t(c)$. Two applications of the cost bound around the definition of $c_t^\star$ close the argument,

$$\begin{aligned} J_t(c_t^\star) &\le \widehat{J}_t(c_t^\star) + \delta_J \\ &\le \widehat{J}_t(c_t^{\mathrm{opt}}) + \delta_J \\ &\le \min_{c \in \mathcal{C}_t} J_t(c) + 2\delta_J. \end{aligned} \tag{8}$$

The middle line holds because $c_t^\star$ minimizes the predicted cost. This is the selection bound of Section 3.4. It concerns the current held-command comparison, not the cumulative cost of the replanned loop, and it assumes nothing about the candidate set beyond its being finite and nonempty.

F When the Command Effect Cannot Be Estimated from the Logged Trajectories

On four of the seven modeling columns of Table 1 the command channel carries no information beyond the memory state. The collection logs show it. On the two tasks that record a per-turn command, the applied command equals the target on every transition: 2268 of 2268 on sentence length, 240 of 240 on sentiment. The tracking error spans $[-0.40, +0.88]$ on sentence length across those same transitions. The two joint tasks record no per-turn scalar command. The prediction design therefore cannot estimate an independent command effect on those four columns. On the remaining three columns, defense, constraint and CEFR, the paired difference between the reported model and the Memory w/o command row of Table 1 is +0.0121, +0.0015 and +0.0061, with intervals containing zero. This is a property of how those trajectories were collected, not a result about the controller. The control experiments of §4.5 are the test of the command channel.