SPECIALISTS LEARN FASTER: KNOWLEDGE ORGANIZATION IN PRE-TRAINING SHAPES KNOWLEDGE UPDATE EFFICIENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

An ideal (language) model must continually update its knowledge. Existing work largely focuses on making these updates efficient after training. We investigate how the organization of time-varying knowledge during pre-training shapes the efficiency of subsequent updates. Models trained on chronologically ordered knowledge specialize in the most recently encountered knowledge; we call them *specialists*. In contrast, models trained on the same knowledge in randomly shuffled order perform well across all represented time periods; we call them *generalists*. Surprisingly, specialists require far fewer update steps than generalists to acquire new knowledge. To explain this advantage, we derive a law from first principles that relates update cost to the model's optimization dynamics and closely predicts empirical results. Guided by these findings, we design a strategy for organizing pre-training data that preserves stable historical knowledge while enabling efficient updates to current knowledge. We also show that the benefits of temporal knowledge organization extend to real-world pre-trained models and existing knowledge update tools, and generalize to other knowledge orderings.

1 INTRODUCTION

Both generalists and specialists have evolved in nature (Sachdeva et al., 2020). Generalists are fit across a range of environments although they are not particularly optimal for a particular environment. Specialists are well adapted to a particular environment but lack fitness outside their niche.

We show that generalists and specialists can emerge in language models, influenced by how knowledge is ordered and presented during training (Section 4). Without losing generality, we focus on biographical knowledge, which has a natural order — chronological order (Section 3). We find that models trained on chronologically ordered knowledge tend to become specialists: they perform best on the most recently encountered knowledge but poorly on knowledge from other periods. In contrast, models trained on the same knowledge in randomly shuffled order tend to become generalists: they perform reasonably well across all periods represented in training.

Perhaps surprisingly, specialists acquire new knowledge more efficiently during post-training updates (Section 5). To explain this advantage, we derive a law from first principles that describes knowledge-update dynamics in the model's optimization landscape. The law relates update cost to two quantities: the pre-trained model's initial gap in acquiring the new knowledge (the knowledge gap) and how quickly the model closes that gap (the update speed). Together with a gradient analysis, the law shows that specialists' advantage lies primarily in their larger initial gradient magnitudes, which allow them to close the (sometimes larger) knowledge gap much faster than generalists.

Our observations have practical values: an ideal (language) model must continually update its knowledge because, for instance, people change jobs over time or software APIs update across versions. After deployment, a model is usually updated through model editing (Meng et al., 2022; 2023; De Cao et al., 2021; Yao et al., 2023) or continued pre-training (Gururangan et al., 2020; Ibrahim et al., 2024). Most prior work focuses on making these updates cheaper and less harmful *once the model has already been trained* by, for example, meta-learning an editor (Mitchell et al., 2022a), storing updates in external memory (Mitchell et al., 2022b; Hartvigsen et al., 2023), or constraining updates to spare other knowledge (Fang et al., 2025).

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.

These post-training approaches apply after a model has learned its knowledge. Our work studies the role of the original training data in update efficiency. Building on our findings, we design a strategy for organizing pre-training data that preserves rarely changing historical knowledge while enabling efficient updates to recent knowledge (Section 6.1). We also show that the benefits of temporal knowledge organization extends to real-world pre-trained models (Section 6.2) and existing knowledge update tools (Section 6.3), and generalize to other knowledge orderings (Section 6.4).

In summary, we make three main contributions.

- We show that models trained on chronologically ordered knowledge require roughly half as many steps to learn new knowledge as those trained on temporally shuffled knowledge.
- We derive a law governing the update cost. The law shows that the temporal order of knowledge reduces update cost primarily by substantially increasing the initial gradient of the model.
- Using our findings, we show how to design temporal knowledge organizations to train models that are both generalists and specialists. The benefits extend to pre-trained (real-world) models and existing knowledge update tools, and generalize to other knowledge orderings.

2 Related Work

Time-Varying Knowledge. Temporal benchmarks (Lazaridou et al., 2021; Jang et al., 2022a;b; Loureiro et al., 2022; Vu et al., 2024), studies of outdated knowledge (Zhao et al., 2024), date conditioning (Dhingra et al., 2022) and probes of temporal representations (Gurnee & Tegmark, 2024; Park et al., 2025) ask what a model knows about the past and the present. We instead ask how the history of a changing fact is represented and what it costs to change the fact again.

Temporal Order in Training. Data order has been studied as curriculum learning (Bengio et al., 2009; Wu et al., 2021) and, for growing corpora, as chronological or continual pre-training (Jin et al., 2022; Li et al., 2025; Pilchen et al., 2026). We hold the facts fixed and vary only their order, which isolates its effect on representation and update cost.

Knowledge Update. Prior work on knowledge updating differs in when it intervenes and what it changes (see Table 2). After pre-training, editors rewrite facts in place (Meng et al., 2022; 2023; Fang et al., 2025), learn an editor (Mitchell et al., 2022a; Zweiger et al., 2025) or modify the update data (Ovadia et al., 2024; Yang et al., 2025). We ask whether data organization eases later updates.

3 Experimental Setup

Temporal biographies. Our dataset extends the synthetic biographies of Allen-Zhu & Li (2023) with a time axis. It generates 25,000 individuals (at 110M; more at larger scales), each with a unique name and six attributes: birthday, birth city, university, field of study, employer and employer city; each attribute has a corresponding value. Each individual appears 1,000 times between 2000 and 2024. In each year, individuals hold a value for each attribute, and some of these values change over time: their birthday and birth city remain unchanged, university and field change a few times, and their employer and employer's city change more frequently. Each sentence is drawn from about fifty templates for its attribute. An example is shown below (see Appendix A for more details):

> In 2010, John Ivy Bi was born on December 17, 1979 in Moreno Valley, CA. He studied at Bowdoin College and majored in Photography. He worked at Under Armour in San Jose, CA.

In this example, in year $Y = 2010$, the person "John Ivy Bi" and the attribute "birthday" form a fact identity I, which has a value $V =$ "December 17, 1979"; there are six such identity-value pairs for each person in each year (see Figure 9 for more details).

Pre-training organization and models. From the same records we build six pre-training corpora by varying two factors: record order and timestamp inclusion. A chronological order (C) presents each person's records in the chronological order, sorted by the timestamps, which are the "In {year}, " prefix of the example above. Records of different people are unordered. A shuffled order (S) shuffles all records in an random order. For each order, we build three variants, one that does not carry the timestamp (i.e., no "In {year}, " prefix), one carries the timestamp (D), and a third that carries timestamp on randomly chosen half of the records ($D^{1/2}$). This yields six training corpora (C, S, CD,

SD, CD$^{1/2}$, and SD$^{1/2}$). We pre-train GPT-2 style decoders with rotary position embeddings and the GPT-2 tokenizer at three scales; 110M, 350M, and 682M (see Table 4 for model details). The pre-training loss is the standard cross-entropy over the whole record.

Updating. After pre-training, we consider two types of knowledge updates: an *overwrite* update corrects an existing record, while an *injection* update adds a record for a new person. Each update is performed by training a LoRA adapter (Hu et al., 2022). Figure 1 shows an overwrite update: the company John Ivy Bi worked at in 2020 is corrected from Google to PepsiCo. This is done by training the adapter on "In 2020, John Ivy Bi worked at PepsiCo." Before LoRA, $P_{\text{old}}(\theta_0)$, the probability of the old value Google, is the highest, whereas the probability of the new/updated value PepsiCo, $P_{\text{new}}(\theta_0)$, is very low. After N LoRA steps when model parameters are updated from θ_0 to θ_N, $P_{\text{new}}(\theta_N)$ becomes the highest. The fine-tuning uses the cross-entropy loss computed only on the new value, i.e., $-\log P_{new}(\theta)$, and stops when $P_{\text{new}}(\theta)$ passes the competitor's probability $P_{\text{old}}(\theta)$. [1]

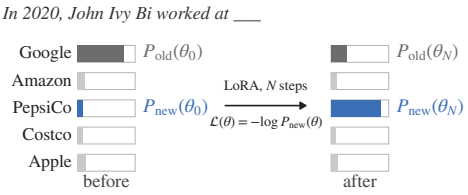

Figure 1: Example of an *overwrite* update. Bars represent the model's predicted probabilities over an attribute's values. Starting from initial parameters θ_0, a LoRA adapter is trained for N steps on the loss $\mathcal{L}(\theta) = -\log P_{\text{new}}(\theta)$, until the new value's probability passes its competitor's, P_{old} (the strongest value model held before updating).

Knowledge Query. A query asks for the value of one attribute of a person. We experiment with two forms of knowledge querying. A *dated query* specifies a year along with a person's name and an attribute and counts the response as correct if the answer is any value held during that year (since a value can change within a year). An *undated query* omits the year and expects the latest value.

4 The Emergence of Generalists and Specialists

This section studies how the temporal organization of knowledge exposed during training shapes what a model learns. We extend the results to other ordering in Section 6.4.

4.1 Organizing Time-Varying Knowledge Influences Generalists vs. Specialists

To demonstrate how the organization of time-varying knowledge during training can influence the model, Figure 2 compares the training dynamics across three settings:

- On the left, the model is trained on the CD corpus, and constantly being queried at different batches and epochs during training (x-axis) using four dated queries (y-axis) — Year 2005, 2010, 2015, and 2020 — as well as the "current" query, which queries using the knowledge trained in the immediately previous batch. The heatmap represents the query accuracy, averaged over all the queries in the corresponding query category.
- The middle figure shows the same training dynamics but using the SD corpus for training.
- Finally, the last figure shows the dynamics when the model is trained on the SD corpus, but is queried using different batches of the training corpus. Importantly, the querying batch order (y-axis) matches that of training.

Comparing the first two figures, we see that the CD-trained model tends to learn only the knowledge that it is just exposed to, and quickly forgets the knowledge in the past. As a result, there are distinct phases during training where each dated query category has a high accuracy; this pattern repeats at every epoch. The model essentially transitions across different specialists, each specializing to knowledge in a given time span. In contrast, the model trained using the SD corpus tends to yield generalists, which perform well on all dated queries *even in the first epoch* when the model has not experienced all the training knowledge yet.

Figure 3 (a) shows intuitively how shuffled training yields generalists. Each year (two here, C_1 and C_2) has an optimization landscape with multiple specialists, each of which learns the knowledge in

[1] We mark an update successful once P_{new} exceeds the competitor, similar to prior model-editing metrics (Meng et al., 2022); Table 5 confirms the same conclusions under stricter criteria empirically.

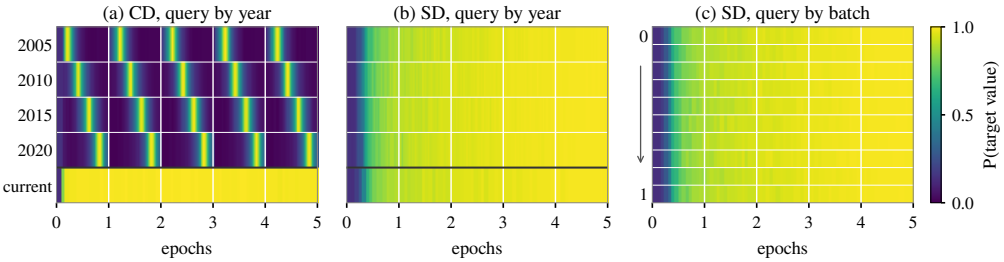

Figure 2: Query accuracy of the 110M CD and SD models over five training epochs; color represents accuracy. (a, b) Query by year (for CD and SD). Rows 2005–2020 query knowledge of a year (e.g., "In 2005, . . . "), and the "current" row queries knowledge from the most recent batch the model is trained on. (c) Query by batch (for SD). Each row queries knowledge from one of the ten batches in the training corpus, from the first batch (0) to the last (1). See Appendix A for details.

that year well and corresponds to a valley. Importantly, the specialists of different years could overlap. The overlapped region contains the "jacks-of-all-trades": they are generalists that are well-optimized for knowledge across years. Given that specialists generally far outweigh generalists in number, a random initialization will much more likely end up in a specialist under chronological training.

Why do models trained under the shuffled corpora end up in the generalist region? When trained under SD, each batch yields a specialist model of that particular batch, i.e., the "shuffled-batch specialists" in Figure 3 (a). Different shuffled batches can be thought of as samples of the same underlying corpus; as a result, different batch's specialists are very close to each other in the landscape and all fall in the generalists region. This intuition is confirmed in Figure 2 (c). Apart from the initial stage in the first epoch, the SD-trained model performs well for all batches, even for unseen batches (in the first epoch). This suggests that shuffled-batch specialists are similar generalists.

4.2 The Role of Time in Chronological Training

A key assumption in Figure 3 (a) is that specialists and generalists, trained under CD, perform equally well on the year for which the specialist is optimized. Yet, a specialist clearly has not learned the year information, as shown by its poor performance under dated queries of other years (see Figure 2 (a)). Why, then, can a model that has not learned the year information perform as well as one that has?

Intuitively, this is because under chronological training, nearby records of the person usually share the same value, because a person's attributes (e.g., job) do not change rapidly in a short period of time; Appendix A provides detailed statistics on this. As a result, the loss can be reduced without relying strongly on time and, accordingly, the gradients would unlikely point toward regions in the landscape that relies on time. Under shuffling, however, different values of the same fact can appear close together, so the model must use the year to tell them apart.

To formalize this, we represent each fact by its identity I, year Y, and value V (see Section 3 for notation convention and Figure 9 for an annotated example). For an empirical distribution $Q_{t,W}$ over W consecutive fact records starting at t, the value-prediction part of the training loss is:

$$L_{t,W}(p) = \mathbb{E}_{(I,Y,V)\sim Q_{t,W}}[-\log p(V \mid I, Y)]. \tag{1}$$

We quantify the predictive benefit of the year by comparing a year-independent predictor $r(V \mid I)$ with a year-conditioned predictor $p(V \mid I, Y)$. The difference between their minimum losses is

$$R_{t,W} := \underbrace{\inf_r L_{t,W}(r)}_{\text{without year}} - \underbrace{\inf_p L_{t,W}(p)}_{\text{with year}} = \mathrm{MI}_{Q_{t,W}}(V; Y \mid I), \tag{2}$$

where $\inf(\cdot)$ is the infimum operator, $R_{t,W}$ is how much the loss would reduce with the additional year Y information, and MI denotes conditional mutual information, representing how much the year Y tells us about the value V once the fact identity I is known. For the detailed derivation of Equations 1–2, see Appendix B.1.

Empirically, $R_{t,W}$ is close to 0 under CD training (see Appendix B.1 for details). Intuitively, this means that predicting V does not depend much on knowing the year information Y when the person's

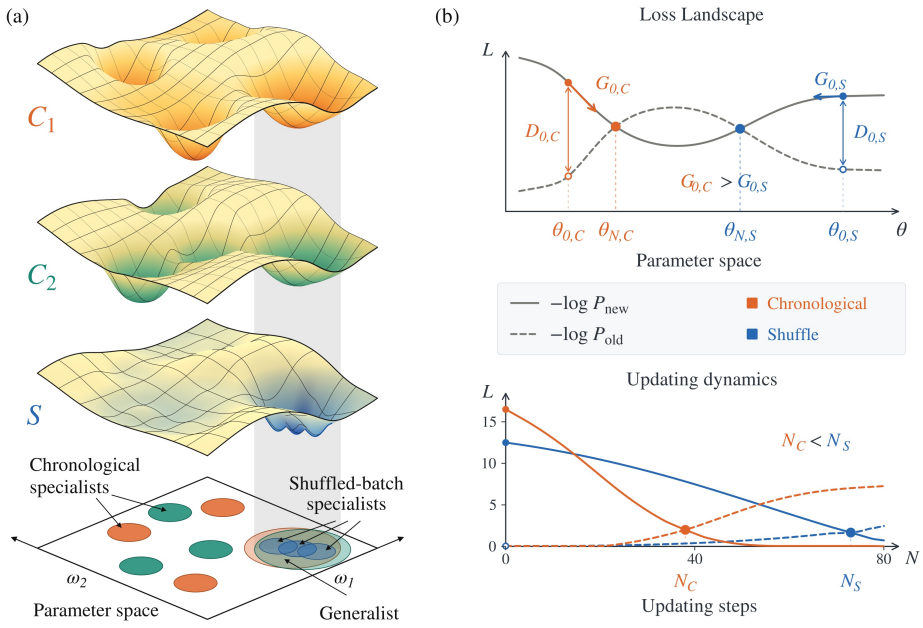

Figure 3: Intuition for generalists vs. specialists and for why specialists update knowledge faster. (a) Loss landscapes of knowledge in two years (C_1 and C_2) and of all knowledge (S) over the same model parameter space (bottom plane). Each valley in C_1 and C_2 is a chronological specialist that learns the knowledge of that year well. Every valley in S is a shuffled-batch specialist for a batch during shuffled training. The specialists of C_1 (orange) and C_2 (green) overlap in a region, which is where generalists lie. Different shuffled-batch specialists (blue) lie close to each other and all fall inside the generalist region. (b) Top: the loss of the new value ($-\log P_{\text{new}}$, solid) and the old value ($-\log P_{\text{old}}$, dashed) across the parameter space θ. An update starts at $\theta_{0,C}$ and $\theta_{0,S}$ for a specialist and a generalist, respectively; $D_{0,C}$ and $D_{0,S}$ are the initial learning gap, respectively, and $G_{0,C}$ and $G_{0,S}$ are the corresponding initial gradients. $\theta_{N,C}$ and $\theta_{N,S}$ are the points where the two losses cross, i.e., where each update completes, in the two cases, respectively. Bottom: the actual change of loss over update steps, comparing a 110M specialist and a 110M generalist; an update ends where they cross ($P_{\text{new}} = P_{\text{old}}$), and the specialist crosses first ($N_C < N_S$).

identity I is known. Therefore, year-agnostic specialists and year-aware generalist have similar query performance. Appendix B.2 further shows that the year information in a specialist model is *not* simply dropped; it does participate in model inference, but the information is treated as an ordinary token and does not contribute much to the prediction.

5 Efficiency of Updating Knowledge

With generalists and specialists naturally emerging from training under different temporal organizations of knowledge, we now turn our attention to how easy it is for them to learn new knowledge.

5.1 Specialists Acquire New Knowledge Faster

How the knowledge is exposed to a model at the pre-training time affects how the model acquires new knowledge later. Table 1 compares the number of updates required by specialists and that by generalists across three model sizes and six training corpora. Specialists require 27%-58% fewer steps to both overwrite a fact or inject a new fact than generalists (Appendix C.1 for more results). The saving increases with the model size, suggesting that more powerful generalists are even harder to update knowledge for.

These trends also hold when the training corpus omits timestamps (C vs. S) or includes them in only half of the records ($\text{CD}^{1/2}$ vs. $\text{SD}^{1/2}$). Updates are undated for C and S, and evenly split between

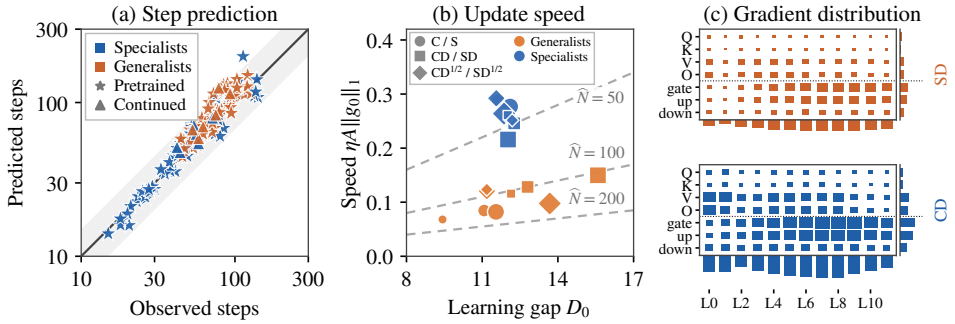

Figure 4: Update cost under chronologically ordered and shuffled training. (a) Predicted updating steps against observed steps; Each of the 18 models (3 model sizes in Table 4 $\times$ 6 training corpora) is shown in two variants: pre-trained from scratch (stars) and continued from a pre-trained model (triangles). (b) We plot these 18 pre-trained models using their initial distance D_0 and the learning speed $\eta A\|g_0\|_1$; each point is a median over 160 updates, and the dashed lines mark a constant number of steps (Eq. 3). Marker sizes are proportional to the model sizes. (c) Contribution to the initial gradient $\|g_0\|_1$ from each layer and each block within a layer, for a generalist (above; trained on SD) and a specialist CD (below; trained on CD); the area of a square is the contribution of that block, the histograms sum the contributions by layer and by block (see details in Appendix C.3).

Table 1: Update cost comparison between different model sizes and training strategies. We use three model sizes, and each model is trained under the six training corpora. An *overwrite* update changes a known person's latest value and an *inject* update adds a fact about an unseen person. The "saved" column is the percentage of update steps that a chronological model saves over the corresponding shuffled model (all results have a paired Wilcoxon $p < 0.05$). We also show the D_0 ratio and the $\|g_0\|_1$ ratio between a corresponding specialist-generalist pair.

		Overwrite				Inject			
Scale	Pair	steps	saved	D_0 ratio	$\|g_0\|_1$ ratio	steps	saved	D_0 ratio	$\|g_0\|_1$ ratio
110M	C / S	51.0 / 88.0	+**42%**	1.27×	2.82×	37.0 / 59.0	+**37%**	1.18×	2.68×
	CD / SD	53.0 / 73.0	+**27%**	1.03×	2.02×	37.0 / 52.0	+**29%**	1.05×	1.90×
	$CD^{1/2}$ / $SD^{1/2}$	53.0 / 75.0	+**29%**	1.00×	1.61×	33.0 / 55.0	+**40%**	0.96×	2.02×
350M	C / S	50.0 / 116	+**57%**	1.00×	2.71×	31.5 / 65.0	+**52%**	0.77×	2.44×
	CD / SD	55.0 / 90.0	+**39%**	0.83×	1.63×	32.0 / 57.0	+**44%**	0.79×	1.45×
	$CD^{1/2}$ / $SD^{1/2}$	50.5 / 94.0	+**46%**	0.87×	1.76×	31.5 / 61.0	+**48%**	0.80×	1.72×
682M	C / S	49.0 / 118	+**58%**	1.10×	2.40×	29.0 / 63.5	+**54%**	0.68×	2.11×
	CD / SD	54.0 / 87.5	+**38%**	0.72×	1.39×	31.0 / 54.5	+**43%**	0.59×	1.22×
	$CD^{1/2}$ / $SD^{1/2}$	50.0 / 100	+**50%**	0.85×	1.91×	29.0 / 60.0	+**52%**	0.64×	1.60×

dated and undated for $CD^{1/2}$ and $SD^{1/2}$. Appendix C.2 shows that the same pattern holds across optimizers and learning rates, and when full fine-tuning replaces LoRA.

5.2 A Step Law for Updating Efficiency

We now aim to understand what determines the update cost. To that end, we derive an empirical law by viewing learning an update as a trajectory through the optimization landscape. Recall from Section 3 that an update, be it an overwrite or an injection, stops at step N when $D_N = \log P_{old}(\theta_N) - \log P_{new}(\theta_N) < 0$. Therefore, N depends on two factors: the initial distance D_0 in the pre-trained model (i.e., the knowledge gap) and how fast the distance is closed per fine-tuning step (i.e., the update speed).

We omit the derivation and refer readers to Appendix C.3. In short, the relationship is given by:

$$\widehat{N} = \frac{D}{\eta A \|g_0\|_1}, \tag{3}$$

where η is the learning rate in the optimizer (Adam), $g_0 = \nabla_\theta[-\log P_{\text{new}}](\theta_0)$ is the initial gradient before the first fine-tuning step and represents the initial learning speed; A is a single calibration

constant that is applied across all training settings. A is calibrated such that $A\|g_0\|_1$ represents the *average* update speed, i.e., how much each fine-tuning step closes the initial gap D_0.

The derivation of Eq. 3 does not depend on whether, and how often, timestamps are part of the training knowledge. Therefore, the law applies to all 18 variants in Table 1. Figure 4(a) compares the observed N and the predicted N using Eq. 3. The model accurately captures the update cost across model scales, timestamp settings, and training organizations; the overall Pearson correlation coefficient $r = 0.91$. Without refitting A, the same model also predicts update cost after continued pre-training changes the model's organization (Appendix D.5), and predicts well across other learning rates and with another optimizer (Appendix C.2).

Since A and η are constant in Eq. 3, differences in update cost between generalists and specialists arise from D_0 and g_0. Figure 4 (b) plots each pre-trained model by its D_0 (x-axis) and $A\|g_0\|_1$ (y-axis). The three lines correspond to three different update-step counts. We make two main observations.

First, update speed contributes more than the initial knowledge gap to specialists' update advantage. Figure 4 (b) shows that generalists have a wide range of the initial learning gap D_0 than that of the specialists. Regardless of the initial learning gap, however, specialists consistently have a much higher speed of learning than that of generalists (2.5 $\times$ higher on average). As an example, even though the 110M specialist trained under CD has actually a slightly (1.03 $\times$) larger knowledge gap than the generalist trained under SD, the former updates more than 2 $\times$ faster and ends up acquiring the knowledge much sooner. Such dynamics are illustrated in Figure 3 (b).

Second, Figure 4 (b) also shows that larger models tend to have larger initial learning gaps (model size is proportional to the marker sizes). This is because larger generalists are trained better, so they are naturally more confident in the old knowledge before the update, leaving a larger knowledge gap to close during update. This explains the greater savings observed for larger models in Table 1.

Figure 4 (c) further visualizes the contribution to the initial gradient from each layer and each block within a layer between a specialist and a generalist. We see that the absolute amplitude of initial gradient increases significantly from generalists to specialists. In particular, the middle layers contribute the most to the specialists' larger gradient, which is where the faster update comes from.

6 Applications of Ordering Knowledge in Pre-Training

Our analyses so far demonstrate a trade-off between generalists and specialists — influenced by the temporal ordering of knowledge. This section shows four applications of this observation to improve language models in practice.

6.1 Mixing Generalists and Specialists

Setup. Consider a deployed model whose more recent knowledge changes frequently, while its historical knowledge remains fixed. Ideally, the pre-trained model should preserve its historical knowledge accurately while efficiently acquiring new knowledge.

Our idea is to train a generalist for the history and a specialist for the present. Specifically, we select a recent time window near the present and organize each person's record within this window chronologically, while shuffling all records from the earlier history beyond the window. This forms the corpus Mix-C. As a comparison, Mix-S contains the same knowledge as Mix-C but shuffles the records randomly. Both models are trained at the 110M scale; full construction and training details are provided in Appendix D.1.

Performance. We find that Mix-C is able to preserve historical knowledge while acquiring new knowledge efficiently. Figure 5 (a) show the knowledge query, both dated and undated, accuracy of Mix-C. Among all training organizations, Mix-C is the only model that achieves high query accuracy on both historical knowledge (with dated queries) and current knowledge (with undated queries). Figure 5 (b) shows the training dynamics of Mix-C. The model behaves as a generalist on the historical knowledge, while exhibiting specialist-like dynamics on recent knowledge, consistent with the observations in Section 4.

At the same time, the specialist-like organization of recent knowledge also enables faster acquisition of new knowledge. Figure 5 (c) compares the *overwrite* update cost of Mix-C and Mix-S under single,

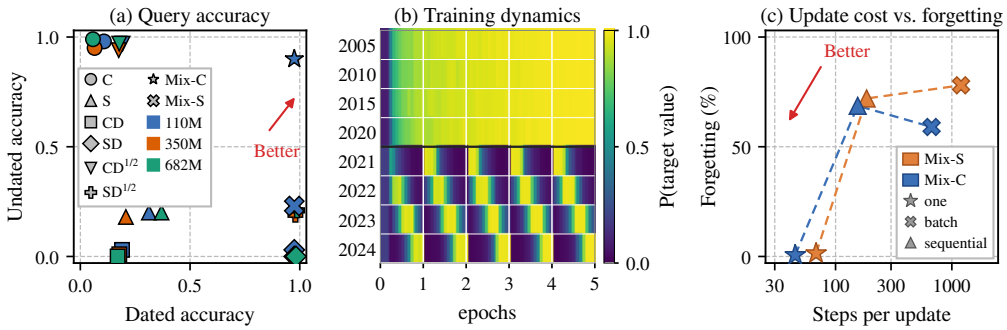

Figure 5: Query accuracy and update cost of the Mix corpora (Section 6.1). (a) Dated query accuracy (2005–2020 average) against undated query accuracy of different pre-trained models. (b) Dated query accuracy of Mix-C through five epochs of training, for historical years (2005–2020) whose knowledge is shuffled and for recent records (2021–2024) whose knowledge is ordered. (c) Median steps per update (log scale) against forgetting (percentage of other people's facts that are incorrectly changed), for single updates (star), batch updates (cross) and sequential updates (triangle).

batch, and sequential updates; we also show the percentage of knowledge forgetting on the y-axis. A single update changes one fact, a batch update changes multiple facts jointly, and sequential updates apply a series of batches. All updates are made to the most recent year; see Appendix D.1 for details. Forgetting is the percentage of facts about other people whose answers are incorrectly changed by an update. Mix-C consistently requires 16%-44% fewer update steps than Mix-S across all three settings, while having 5%-57% less forgetting on the historical records.

6.2 Reshaping a Real-World Pre-trained Model

Setup. Pre-training from scratch is computationally expensive. We therefore ask whether a short stage of continual pre-training can improve a model's update efficiency while preserving its historical knowledge. We start from Gemma-2-9B-it (Team, 2024), an instruction-tuned model that was trained on real-world, general data including time-varying knowledge.

For continual pre-training, we sample historical records with timestamps and keep them temporally shuffled, while sampling a long recent time window and arranging each person's records chronologically. We mix the two parts while preserving the within-person chronological order of the records in the window, forming the corpus C-C. As comparisons, C-S contains exactly the same records but is randomly shuffled, and we also include the original pre-trained model (Base) as a baseline. Further details are provided in Appendix D.2, and Appendix D.5 presents results on our synthetic data.

Performance. Figure 6 (a) shows that after continual pre-training using C-C, the model improves its undated query accuracy of current knowledge from 42% to 98%, while requiring 1.3× fewer steps than Base and 1.8× fewer steps than C-S during *overwrite* updates. Meanwhile, its general ability is preserved; Appendix D.2 reports the forgetting results.

6.3 Improving Published-Knowledge Editors

Setup. So far, we have updated knowledge using LoRA. Another common way of knowledge update is through dedicated knowledge-editing methods. We next ask whether these methods could benefit from chronological pre-training. We run ROME (Meng et al., 2022), MEMIT (Meng et al., 2023), AlphaEdit (Fang et al., 2025) and ENCORE (Gupta et al., 2025), on Mix-C and Mix-S, applying the *overwrite* updates as in Section 6.1. Appendix D.3 contains more experimental details.

Performance. We find that specialists induced by temporal knowledge organization can further improve existing knowledge-editing methods over their shuffled counterparts. Figure 6 (b) compares the update cost (TFLOPs per update) and forgetting of these editors when applied to Mix-C, which contains a specialist for present knowledge (Section 6.1), and to its fully shuffled counterpart Mix-S. For the same editing method, the specialist organization in Mix-C provides an additional reduction in both update cost and forgetting over Mix-S. The gain reaches up to 46% lower update cost with MEMIT (Meng et al., 2023) and 63% less forgetting with AlphaEdit (Fang et al., 2025).

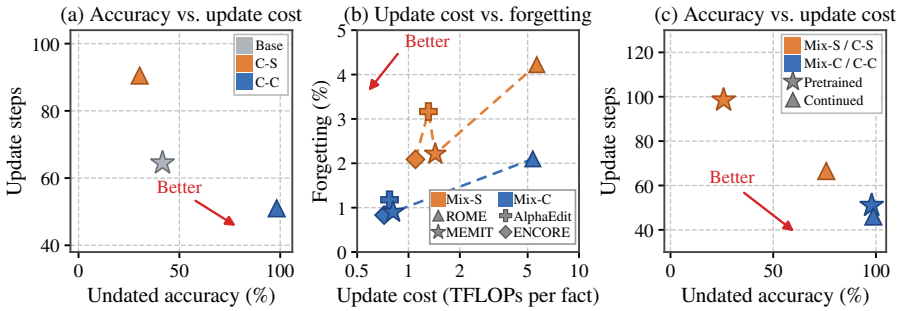

Figure 6: (a) Undated query accuracy of the continual pre-training stage vs. the median update steps on *overwrite* updates, for Gemma-2-9B-it (Base) and its two continually pre-trained variants. (b) Update cost vs. percentage of forgetting for the four knowledge editors applied to the 110M Mix-C and Mix-S models; each model's four editors are joined by a dashed line. (c) Undated query accuracy against the median update steps in the versioned-software dataset; results are for a 110M model.

6.4 Generalizing to Another Knowledge Ordering: Versioned Software Change

Setup. So far, biographical knowledge is ordered chronologically, but do the same observations hold under other domains and orderings? We study a practical case: software tools, where the version of a tool serves as the ordering key. For example, when an API changes its URL across versions, a model should quickly learn the API's latest endpoint without losing its knowledge of how to call other tools. We build the tool-call dataset in a similar way to the biography dataset in Section 3. The dataset contains multiple tools, each described by six fields (for example, host, authentication header, etc.), each of which has a value. An example is (see Appendix D.4 for more examples):

> For v8, the Flint Payments API carries host api.flintpay.com. Its input parameter is refund_id.

We follow the pre-training and continual pre-training setups in Section 6.1. Specifically, for pre-training from scratch, we shuffle the older versions and present the recent versions in version order, as in Mix-C. For continual pre-training, we start from a shuffled base model, present records within recent version window in version order, and replay its older versions in shuffled order, as in C-C. We ask whether the same setup can learn a specialist for recent versions while keeping a generalist over older versions. Appendix D.4 shows more details about the training setup.

Performance. The same pattern appears in the tool-use domain. Figure 6 (c) shows that both Mix-C and C-C answer 98–99% of queries about the latest version, while cutting update cost by up to 48% and 33% relative to Mix-S and C-S, respectively. Both also keep their knowledge of older versions.

7 Concluding Remarks

Pre-training knowledge ordering shapes the cost of future knowledge updates. Models trained on temporally ordered knowledge, which we call specialists, require roughly half as many update steps as generalists, models trained on shuffled knowledge. We derive a step law to explain this advantage. We use this insight to design new knowledge organization to preserve historical knowledge while enabling efficient updates. The benefit of pre-training knowledge organization extends to already-trained models and existing post-training update tools.

Apart from time, Section 6.4 explores another ordering, where knowledge about software documentation is ordered by software version. Regardless of the specific order Y, what matters is the additional information that Y provides to a knowledge query given all other available information (see the mutual information Eq. 2). An interesting future investigation is to quantify how this mutual information influences the emergence and behaviors of generalists and specialists.

While our work focuses on language models, evolutionary biologists have long studied how generalists and specialists can evolve in time-varying environments (Kassen, 2002). Many also have looked at various advantage of specialists such as faster decision making (Bernays & Funk, 1999). Another interesting path is to investigate whether the learning benefit of specialists extend to evolution, which obtains knowledge through gradient-free methods.

AI Assistance Statement

AI tools were used while writing Appendix A and Appendix D to improve grammar and correct typographical errors. The technical content, analyses, and results were determined by authors.

Reproducibility Statement

We provide the details needed to reproduce the experiments and analyses in the paper and appendices. Appendix A describes the data construction, model training, query protocol, and update procedure. Appendix B gives the derivations and measurements behind Section 4. Appendix C provides the full update-cost results, the derivation of the step law, and additional robustness analyses. Appendix D gives the experimental setup and implementation details for the applications in Section 6. We will release the code for data generation, model training, and evaluation upon acceptance.

References

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Elizabeth A Bernays and Daniel J Funk. Specialists make faster decisions than generalists: experiments with aphids. *Proceedings of the Royal Society B: Biological Sciences*, 266(1415):151, 1999.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In *Proceedings of EMNLP*, 2021.

Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. Time-aware language models as temporal knowledge bases. *Transactions of the Association for Computational Linguistics*, 10:257–273, 2022.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632: 768–774, 2024.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. AlphaEdit: Null-space constrained knowledge editing for language models. In *International Conference on Learning Representations*, 2025.

Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. Does fine-tuning LLMs on new knowledge encourage hallucinations? In *Proceedings of EMNLP*, pp. 7765–7784, 2024.

Akshat Gupta, Phudish Prateepamornkul, Maochuan Lu, Ahmed Alaa, Thomas Hartvigsen, and Gopala Anumanchipalli. Lifelong knowledge editing requires better regularization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025.

Wes Gurnee and Max Tegmark. Language models represent space and time. In *International Conference on Learning Representations*, 2024.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 8342–8360, 2020.

Thomas Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. Aging with GRACE: Lifelong model editing with discrete key-value adaptors. In *Advances in Neural Information Processing Systems*, 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. Simple and scalable strategies to continually pre-train large language models. *Transactions on Machine Learning Research*, 2024.

Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. TemporalWiki: A lifelong benchmark for training and evaluating ever-evolving language models. In *Proceedings of EMNLP*, 2022a.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. Towards continual knowledge learning of language models. In *International Conference on Learning Representations*, 2022b.

Xisen Jin, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew Arnold, and Xiang Ren. Lifelong pretraining: Continually adapting language models to emerging corpora. In *Proceedings of NAACL*, pp. 4764–4780, 2022.

Rees Kassen. The experimental evolution of specialists, generalists, and the maintenance of diversity. *Journal of evolutionary biology*, 15(2):173–190, 2002.

Zixuan Ke, Yijia Shao, Haowei Lin, Tatsuya Konishi, Gyuhak Kim, and Bing Liu. Continual pre-training of language models. In *International Conference on Learning Representations*, 2023.

Jiyeon Kim, Hyunji Lee, Hyowon Cho, Joel Jang, Hyeonbin Hwang, Seungpil Won, Youbin Ahn, Dohaeng Lee, and Minjoon Seo. Knowledge entropy decay during language model pretraining hinders new knowledge acquisition. In *International Conference on Learning Representations*, 2025.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomas Kocisky, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom. Mind the gap: Assessing temporal generalization in neural language models. In *Advances in Neural Information Processing Systems*, 2021.

Jeffrey Li, Mohammadreza Armandpour, Seyed Iman Mirzadeh, Sachin Mehta, Vaishaal Shankar, Raviteja Vemulapalli, Samy Bengio, Oncel Tuzel, Mehrdad Farajtabar, Hadi Pouransari, et al. Tic-lm: A web-scale benchmark for time-continual llm pretraining. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32231–32273, 2025.

Houjun Liu, John Bauer, and Christopher D. Manning. Drop dropout on single-epoch language model pretraining. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-Collados. TimeLMs: Diachronic language models from Twitter. In *Proceedings of ACL, System Demonstrations*, 2022.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, 2022.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *International Conference on Learning Representations*, 2023.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. In *International Conference on Learning Representations*, 2022a.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, pp. 15817–15831, 2022b.

Oded Ovadia, Menachem Brief, Moshik Mishaeli, and Oren Elisha. Fine-tuning or retrieval? comparing knowledge injection in LLMs. In *Proceedings of EMNLP*, pp. 237–250, 2024.

Yein Park, Chanwoong Yoon, Jungwoo Park, Minbyul Jeong, and Jaewoo Kang. Does time have its place? temporal heads: Where language models recall time-specific information. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16616–16643, 2025.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Hippolyte Pilchen, Romain Fabre, Franck Signe Talla, Patrick Perez, and Edouard Grave. Understanding data temporality impact on large language models pre-training. *arXiv preprint arXiv:2605.22769*, 2026.

Vedant Sachdeva, Kabir Husain, Jiming Sheng, Shenshen Wang, and Arvind Murugan. Tuning environmental timescales to evolve and maintain generalists. *Proceedings of the National Academy of Sciences*, 117(23):12693–12699, 2020.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. Fine-tuned language models are continual learners. In *Proceedings of EMNLP*, pp. 6107–6122, 2022.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Kushal Tirumala, Aram H. Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. In *Advances in Neural Information Processing Systems*, 2022.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019.

Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, and Thang Luong. FreshLLMs: Refreshing large language models with search engine augmentation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13697–13720, 2024.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. WISE: Rethinking the knowledge memory for lifelong model editing of large language models. In *Advances in Neural Information Processing Systems*, 2024.

Xiaoxia Wu, Ethan Dyer, and Behnam Neyshabur. When do curricula work? In *International Conference on Learning Representations*, 2021.

Zitong Yang, Neil Band, Shuangping Li, Emmanuel Candès, and Tatsunori Hashimoto. Synthetic continued pretraining. In *International Conference on Learning Representations*, 2025.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. In *Proceedings of EMNLP*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Bowen Zhao, Zander Brumbaugh, Yizhong Wang, Hannaneh Hajishirzi, and Noah A. Smith. Set the clock: Temporal alignment of pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15015–15040, 2024.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *arXiv preprint arXiv:2506.10943*, 2025.

Table 3: Pre-training datasets. C/S: chronological/shuffled; D: timestamped; $^{1/2}$: half timestamped.

Timestamps	Chronological	Shuffled
0%	C	S
50%	CD$^{1/2}$	SD$^{1/2}$
100%	CD	SD

Table 4: Model scales, all with a context of 512 tokens. Records: per epoch; LR: peak learning rate. Other settings in Appendix A.

Scale	Layers	Heads	Hidden	Records	Tokens	LR (10^{-4})
110M	12	12	768	24M	10.88B	6
350M	24	16	1024	68M	30.80B	3
682M	32	20	1280	134M	60.68B	2.5

A Data, Training and Query Protocol

This appendix gives the detailed settings behind Section 3 and Section 4: how the biographies are generated, how the models are trained, how a query is scored, and how an update is run and measured.

Table 2: Prior work by stage and task. Red: through data organization and training dynamics; Blue: through architecture or hyperparameters. Ours is the only data organization-level work on efficient editing at pre-training stage.

	Efficient updating & editing	Catastrophic forgetting
Pre-training	**Ours** Liu et al. (2025), Kim et al. (2025)	Toneva et al. (2019), Tirumala et al. (2022), Ibrahim et al. (2024) Ke et al. (2023), Dohare et al. (2024)
Post-training	Ovadia et al. (2024), EntiGraph (Yang et al., 2025), SEAL (Zweiger et al., 2025) ROME (Meng et al., 2022), MEMIT (Meng et al., 2023), MEND (Mitchell et al., 2022a)	Scialom et al. (2022), Luo et al. (2023), Gekhman et al. (2024) EWC (Kirkpatrick et al., 2017), GRACE (Hartvigsen et al., 2023), WISE (Wang et al., 2024)

Biographies. Names and attribute values come from the vocabulary lists of Allen-Zhu & Li (2023). Each person appears 1,000 times between 2000 and 2024, and each appearance is one record that states the six attributes. Birthday and birth city never change. University and field change 3 times over a lifetime (median; mean 2.8), employer and employer city 10 times (mean 9.9), at random points of the 1,000 appearances.

In a chronological stream, consecutive records of the same person usually contain the same value. In CD, the values are unchanged in about 99.0% of consecutive records. After shuffling, consecutive records can come from very different years, so these numbers drop to around 20.0%. So in a chronological stream the year says almost nothing about a person's current value; Section 4.2 relies on this, and Appendix B.1 turns it into a bound. 1,000 individuals in the corpus are held out from training and used only for injection updates; A timestamped record starts with "In {year},"; the half-timestamped corpora drop this prefix from a random half of the records, the same half in CD$^{1/2}$ and SD$^{1/2}$, so that the two differ only in order (Table 3).

Pre-training. The three scales share the GPT-2 architecture with rotary position embeddings (Su et al., 2024) and the GPT-2 tokenizer (Table 4). Every model trains on batches of 2,560 records, with a 5% linear warmup, a cosine decay from the peak learning rate of Table 4 to a tenth of it, weight decay 0.1 and gradient clipping at 1.0. For the main results in Section 4, we train the models for five epochs, by which point query accuracy has converged already.

Queries. A query prompts the model with a person's name and an attribute, cut just before the value, and reads the probability of every value in the attribute's vocabulary. A dated query starts with "In {year}," and is correct if the answer is any value the person held during that year. In contrast, an undated query gives no year, and is correct if the answer is the latest value (See Figure 7 about "latest"). The year rows (2005–2020) queries 200 people per time-varying attribute about a fixed year, every 125 steps. The queries keep unchanged for different settings. The current row asks, at every checkpoint (every 625 steps), about 200 records of the batch the model has just trained on, in the form each record

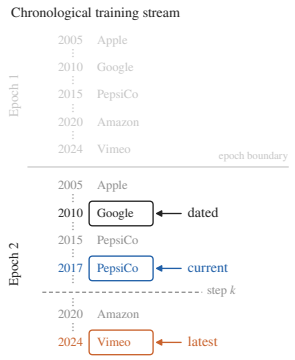

Figure 7: Targets of the three kinds of query in Figure 8: a value in the queried year, the value in the batch trained at step k, and the final value.

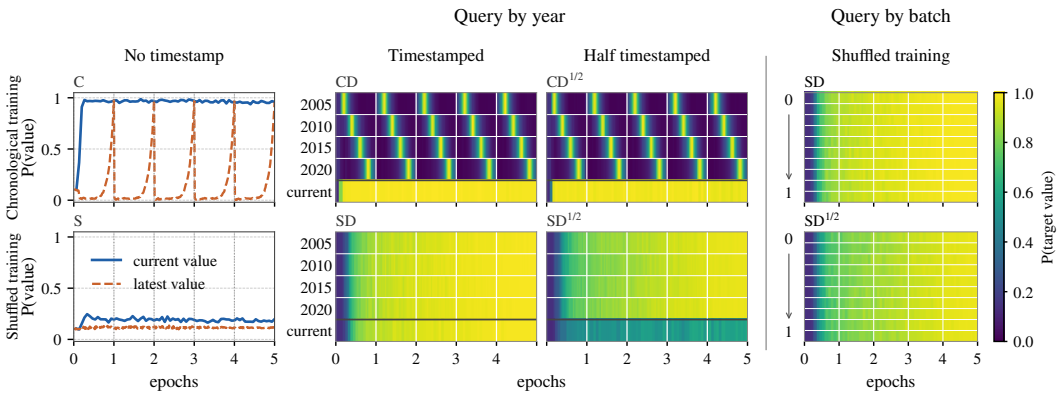

Figure 8: Query accuracy of the six 110M models over five epochs of training, the full version of Figure 2. Rows of panels compare chronological and shuffled training, columns compare timestamp settings, and the right column queries SD and $SD^{1/2}$ by batch as in Figure 2c. Rows 2005–2020 ask with a year ("In 2005, . . . ") and count any value the person held in that year; the *current* row asks about records of the batch the model has just trained on, in the form each was trained in. In the two panels on the left, the solid curve is this current score for the undated models and the dashed curve the probability of the final value, held in 2024.

was trained in. The query-by-batch rows of Figure 2c ask in the same way about 200 records from each of ten batches spaced evenly over the SD stream. Figure 8 extends Figure 2 to all six corpora.

Updates. An update trains a rank-16 LoRA adapter (Hu et al., 2022), with $\alpha = 16$, on every attention and MLP projection of every layer (84 adapters at 110M), using Adam (Kingma & Ba, 2015) without weight decay at a fixed learning rate to the pre-training peak (1/60, e.g. 10^{-5} at 110M), no weight decay. The training sentence states the new value in one of the attribute's sentence templates; for timestamped models it starts with "In 2024,", and for half-timestamped models a random half of the updates do. An *overwrite* update targets a person the model was trained on and an *injection* update targets the held-out person during training. Either succeeds when the new value's probability passes that of its competitor (i.e. the strongest value the model held before updating).

The learning gap D is the log-probability gap to that competitor before the update. Each model receives 480 updates, 40 *overwrite* updates and 40 *injection* updates per attribute. For each attribute we sample the individuals so that different people are equally represented. Table 1 reports medians over all overwrite updates or all injection updates of a model. Forgetting is measured on facts of other people, as the percentage whose answer the update changes.

Updating criteria. Our results do not depend on the stopping criterion used for an update. On 120 updates from each 110M pair in Table 1, stopping when the new value passes the model's strongest previous value gives the same conclusion as requiring it to become the top-1 value over the full attribute vocabulary (Table 5). The same conclusion also holds with two stricter thresholds: a 3-nat log-probability margin over the competitor or $P_{\text{new}} \geq 0.5$.

Table 5: Update cost of the 110M pairs under three stricter success criteria, which are the new value is the top-1 over the attribute's whole vocabulary, its log-probability exceeds the competitor's by 3 and its probability P_{new} reaches 0.5. Cells represent median steps, chronological / shuffled, and the share of steps the chronological model saves (bold: paired Wilcoxon $p < 0.05$). 60 *overwrites* and 60 *injections* updates per pair.

		Overwrite						Inject					
		Vocabulary top-1		Margin ≥ 3		$P_{\text{new}} \geq 0.5$		Vocabulary top-1		Margin ≥ 3		$P_{\text{new}} \geq 0.5$	
Scale	Pair	steps	saved	steps	saved	steps	saved	steps	saved	steps	saved	steps	saved
110M	C / S	53.0 / 89.0	**+40%**	63.5 / 108	**+41%**	59.5 / 99.5	**+40%**	38.0 / 57.0	**+33%**	46.5 / 78.5	**+41%**	47.0 / 71.5	**+34%**
	CD / SD	51.5 / 75.0	**+31%**	63.0 / 88.5	**+29%**	57.5 / 80.5	**+29%**	40.0 / 52.0	**+23%**	53.0 / 66.5	**+20%**	50.0 / 62.5	**+20%**
	$CD^{1/2}$ / $SD^{1/2}$	54.5 / 88.0	**+38%**	67.0 / 103	**+35%**	61.0 / 96.5	**+37%**	31.5 / 61.5	**+49%**	40.5 / 84.5	**+52%**	39.5 / 80.0	**+51%**

B GENERALISTS AND SPECIALISTS: DERIVATIONS AND MEASUREMENTS

B.1 THE PREDICTIVE VALUE OF TIMESTAMPS

This appendix derives Eq. 1 and Eq. 2 in Section 4.2. It shows how much the year helps prediction within a training window. Chronological and shuffled training have the same global objective, but the year is much less useful in a chronological window than in a shuffled one.

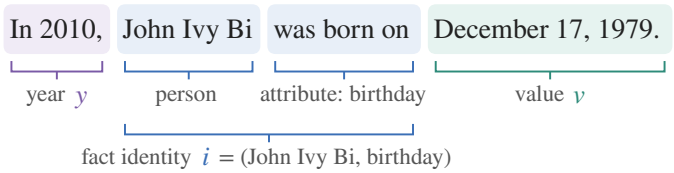

Figure 9: Mapping the example in Section 3 to the fact-level variables. The person and attribute jointly identify the fact i; $y = 2010$ is the record's timestamp, and $v =$ December 17, 1979 is its value. Uppercase I, Y, V denote the variables and lowercase i, y, v their observed values. Here the attribute is time-invariant; for a changing attribute such as employer, the same i can have different values v in different years y.

Step 1: the global objective is the same for both orders. Pretraining minimizes the negative log-likelihood of whole records, whereas a query tests the value of one fact. For a tokenized record $x = (x_1, \ldots, x_T)$ whose value tokens sit at positions $\mathcal{V}(x)$, the record loss splits exactly into a value part and a remainder,

$$\ell_{\text{pre}}(\theta; x) = -\sum_{j=1}^{T} \log p_\theta(x_j \mid x_{<j}) = \underbrace{-\sum_{j \in \mathcal{V}(x)} \log p_\theta(x_j \mid x_{<j})}_{\ell_{\text{value}}(\theta;x)} + \underbrace{-\sum_{j \notin \mathcal{V}(x)} \log p_\theta(x_j \mid x_{<j})}_{\ell_{\text{other}}(\theta;x)}, \tag{4}$$

and for a value span v after context c the value part equals $-\log p_\theta(v \mid c)$ by the chain rule. We study the value part, since it trains the associations a query measures, and represent its context by the fact identity I and the year Y (Figure 9); if further context C is kept, every quantity below is also conditioned on C and nothing else changes. Let $\mathcal{D} = \{(i_m, y_m, v_m)\}_{m=1}^{N}$ be the fact-level records. Their empirical distribution and value-prediction loss are

$$P(i, y, v) = \frac{1}{N} \sum_{m=1}^{N} \mathbf{1}[(i_m, y_m, v_m) = (i, y, v)], \qquad \bar{L}(\theta) = \mathbb{E}_P[-\log p_\theta(V \mid I, Y)]. \tag{5}$$

Permuting the records changes neither P nor $\bar{L}(\theta)$ at any θ, so chronological and shuffled training don't differ in their global objective.

Step 2: the local objective depends on the order. For an ordering $Z_1, \ldots, Z_N$ with $Z_n = (I_n, Y_n, V_n)$, the empirical distribution of the W consecutive records from position t is

$$Q_{t,W}(i, y, v) = \frac{1}{W} \sum_{n=t}^{t+W-1} \mathbf{1}[Z_n = (i, y, v)], \qquad 1 \le t \le N - W + 1. \tag{6}$$

The horizon W is an analysis scale, not necessarily a batch size. Two orderings share $Q_{1,N} = P$ but can have different $Q_{t,W}$ for $W < N$. We fix a window and write $Q = Q_{t,W}$, with every conditional quantity evaluated on its support.

Step 3: the best predictor with and without the year. A year-independent predictor $r(V \mid I)$ has loss $-\sum_{i,v} Q(i, v) \log r(v \mid i)$. Writing $-\log r(v \mid i) = -\log Q(v \mid i) + \log[Q(v \mid i)/r(v \mid i)]$

and using $Q(i, v) = Q(i)Q(v \mid i)$,

$$\begin{aligned} L_Q^{\text{blind}}(r) &= -\sum_{i,y,v} Q(i,y,v) \log r(v \mid i) = -\sum_{i,v} Q(i,v) \log r(v \mid i) \\ &= -\sum_i Q(i) \sum_v Q(v \mid i) \log Q(v \mid i) \\ &\quad + \sum_i Q(i) \sum_v Q(v \mid i) \log \frac{Q(v \mid i)}{r(v \mid i)} \\ &= H_Q(V \mid I) + \mathbb{E}_{I \sim Q} D_{\text{KL}}(Q(V \mid I) \,\|\, r(V \mid I)) \,. \end{aligned} \tag{7}$$

Here H is the entropy in bits, $H_Q(V \mid I) = \sum_i Q(i) H(Q(V \mid i))$ is the uncertainty in the value once the fact is known, and the second term is a Kullback–Leibler (KL) divergence, nonnegative and zero at $r^*(v \mid i) = Q(v \mid i)$, the window's mixture of the fact's values across years. The same steps for a year-conditioned predictor $p(V \mid I, Y)$, with $Q(i, y, v) = Q(i, y)Q(v \mid i, y)$, give

$$\begin{aligned} L_Q^{\text{dated}}(p) &= -\sum_{i,y} Q(i,y) \sum_v Q(v \mid i,y) \log p(v \mid i,y) \\ &= -\sum_{i,y} Q(i,y) \sum_v Q(v \mid i,y) \log Q(v \mid i,y) \\ &\quad + \sum_{i,y} Q(i,y) \sum_v Q(v \mid i,y) \log \frac{Q(v \mid i,y)}{p(v \mid i,y)} \\ &= H_Q(V \mid I,Y) + \mathbb{E}_{I,Y \sim Q} D_{\text{KL}}(Q(V \mid I,Y) \,\|\, p(V \mid I,Y)) \,, \end{aligned} \tag{8}$$

where $H_Q(V \mid I, Y) = \sum_{i,y} Q(i, y) H(Q(V \mid i, y))$ is the uncertainty once fact and year are known, and the divergence is zero at $p^*(v \mid i, y) = Q(v \mid i, y)$. Over unrestricted predictors, therefore,

$$\inf_r L_Q^{\text{blind}}(r) = H_Q(V \mid I), \qquad \inf_p L_Q^{\text{dated}}(p) = H_Q(V \mid I, Y). \tag{9}$$

These are the minimum possible losses for the two predictors within the window. The dated loss is zero only when the fact and year uniquely determine the value.

Step 4: the gap is a conditional mutual information. Subtracting the two optima,

$$\begin{aligned} R_{t,W} &:= \inf_r L_Q^{\text{blind}}(r) - \inf_p L_Q^{\text{dated}}(p) \\ &= H_Q(V \mid I) - H_Q(V \mid I, Y) \\ &= \sum_{i,y,v} Q(i,y,v) \log \frac{Q(v \mid i,y)}{Q(v \mid i)} = \text{MI}_Q(V; Y \mid I), \end{aligned} \tag{10}$$

where the last step uses $Q(v, y \mid i) = Q(y \mid i)Q(v \mid i, y)$. The identity holds for every ordering and every window: $R_{t,W}$ is the minimum loss a predictor pays for ignoring the year, and it is positive exactly when the value distribution of a fact depends on the year.

Step 5: an upper bound on the gap. In a chronological window a fact's value is almost constant even though its year changes. For each fact i with $Q(i) > 0$, let $v_i^\star \in \arg\max_v Q(v \mid i)$ be a most probable value, $\varepsilon_i = 1 - Q(v_i^\star \mid i)$ the probability of any other value, and $K_i = |\operatorname{supp} Q(V \mid i)|$ the number of values. For $\varepsilon_i > 0$, normalize the remaining mass as $\widetilde{Q}_i(v) = Q(v \mid i)/\varepsilon_i$ for $v \neq v_i^\star$. Separating the modal value from this tail gives

$$\begin{aligned} H_Q(V \mid I = i) &= -(1 - \varepsilon_i) \log(1 - \varepsilon_i) - \sum_{v \neq v_i^\star} \varepsilon_i \widetilde{Q}_i(v) \log\Big(\varepsilon_i \widetilde{Q}_i(v)\Big) \\ &= h_2(\varepsilon_i) + \varepsilon_i H(\widetilde{Q}_i) \leq h_2(\varepsilon_i) + \varepsilon_i \log(K_i - 1), \end{aligned} \tag{11}$$

where $h_2(\varepsilon) = -(1 - \varepsilon) \log(1 - \varepsilon) - \varepsilon \log \varepsilon$ is the binary entropy and the inequality uses the tail's support of at most $K_i - 1$ values; a fact with $K_i = 1$ has $\varepsilon_i = 0$ and contributes zero. Since $H_Q(V \mid I, Y) \geq 0$, averaging over facts gives

$$0 \leq R_{t,W} \leq H_Q(V \mid I) \leq B_{t,W} := \sum_{i : K_i \geq 2} Q(i) \left[h_2(\varepsilon_i) + \varepsilon_i \log(K_i - 1) \right]. \tag{12}$$

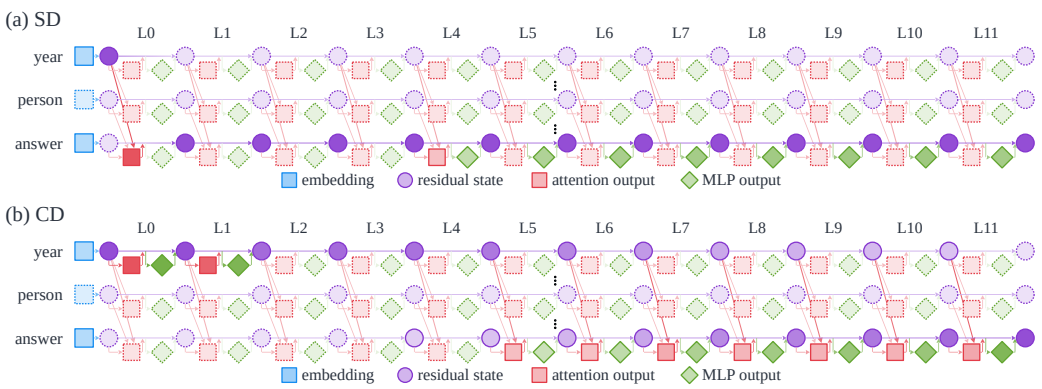

Figure 10: Activation patching of the timestamp in the 110M SD (a) and CD (b). Rows are three positions of a dated question (the year, the person's name and the answer position) and columns the twelve layers; each symbol is one activation: embedding, residual state, attention output or MLP output. A darker symbol recovers more of the model's own year effect when that activation is copied from the same question asked about 2024; colors compare locations within one model. SD recovers the year at the answer position in the first attention block, whereas in CD the year effect builds up gradually through depth.

The bound holds for either order. Under chronological training, values are highly persistent within a window, so ε_i is close to zero and the bound becomes small.

Empirically, for the 110M CD model, $R_{t,W} \leq B_{t,W} = 0.01$ bits per record, while the corresponding shuffled window has $R_{t,W} = 0.69$ bits, at least $69\times$ larger. Thus, the year provides almost no predictive value in a chronological window but much more in a shuffled one.

B.2 Tracing Timestamp Use with Activation Patching

Section 4.2 states that a specialist reads the year but does not rely on it. To test how each model uses the year during inference, we use 115 dated questions whose historical answer differs from the 2024 answer. For each question, we run the model twice: once with the historical year and once with 2024. We then patch one activation from the 2024 run into the historical run and ask whether this makes the model answer more like it was queried about 2024.

We patch activations at the year token, the person's name, or the answer position, across different layers and modules. The patching score measures how much of the original shift from the historical answer to the 2024 answer is recovered by a single patch. It therefore traces how year information flows through the model during inference. Figure 10 reports the median over the questions on which the year moves the answer by more than 1.4 bits.

Both models process the year during inference, but they use it very differently. SD uses the year strongly and early, while CD picks it up only gradually and gives it much less weight. If CD relied on the year to predict the value, it would behave like a generalist and answer dated queries across years accurately. Instead, chronological training places the model in a parameter region where the year is processed but is not needed much for value prediction.

C Update Cost: Full Results and the Step Law

C.1 Update Cost by Rate of Change and Across Seeds

Table 6 extends Table 1 by splitting time-varying facts into slow- and fast-changing groups and by adding the Mix-C/Mix-S pair from Section 6.1. Table 7 repeats the 110M C/S comparison with three additional random initializations. Across the four runs, chronological training saves 44–52% of the update steps, showing that the result is robust to initialization.

Table 6: Update cost by how often a fact changes: the time-varying facts of Table 1 split into slow and fast facts, with the Mix pair (Section 6.1) added. Each "steps" cell gives the median steps of the chronological and shuffled models. The "saved" column is the percentage of update steps that a chronological model saves over the corresponding shuffled model; bold values have a paired Wilcoxon $p < 0.05$.

			Slow		Fast	
	Scale	Pair	steps	saved	steps	saved
Overwrite	110M	C / S	49.0 / 91.5	+**46%**	39.0 / 86.0	+**55%**
		CD / SD	52.5 / 69.0	+**24%**	38.0 / 71.0	+**46%**
		$CD^{1/2}$ / $SD^{1/2}$	51.0 / 74.5	+**32%**	36.0 / 72.5	+**50%**
		Mix-C / Mix-S	46.5 / 70.5	+**34%**	45.0 / 64.5	+**30%**
	350M	C / S	47.0 / 118	+**60%**	36.0 / 118	+**70%**
		CD / SD	54.0 / 81.0	+**33%**	41.0 / 86.5	+**53%**
		$CD^{1/2}$ / $SD^{1/2}$	49.5 / 94.5	+**48%**	35.0 / 92.5	+**62%**
	682M	C / S	45.5 / 114	+**60%**	36.5 / 121	+**70%**
		CD / SD	50.0 / 80.0	+**38%**	44.0 / 84.0	+**48%**
		$CD^{1/2}$ / $SD^{1/2}$	48.0 / 98.0	+**51%**	37.5 / 103	+**64%**
Inject	110M	C / S	37.0 / 62.5	+**41%**	24.5 / 54.0	+**55%**
		CD / SD	38.0 / 49.0	+**22%**	23.0 / 46.5	+**51%**
		$CD^{1/2}$ / $SD^{1/2}$	34.0 / 55.0	+**38%**	21.0 / 51.0	+**59%**
		Mix-C / Mix-S	36.0 / 50.0	+**28%**	33.0 / 44.0	+**25%**
	350M	C / S	32.0 / 67.5	+**53%**	19.0 / 59.0	+**68%**
		CD / SD	31.5 / 50.0	+**37%**	20.0 / 53.5	+**63%**
		$CD^{1/2}$ / $SD^{1/2}$	29.0 / 57.0	+**49%**	17.0 / 55.5	+**69%**
	682M	C / S	27.0 / 71.0	+**62%**	18.0 / 60.0	+**70%**
		CD / SD	29.5 / 52.0	+**43%**	21.0 / 50.5	+**58%**
		$CD^{1/2}$ / $SD^{1/2}$	28.0 / 62.0	+**55%**	18.0 / 55.0	+**67%**

Table 7: Temporal order saves about half of the steps on time-varying facts in each of four independently trained 110M C / S pairs, each with its own population and initialization. Cells read as in Table 1: median steps on time-varying facts, chronological / shuffled, and the share of steps temporal order saves.

	Overwrite		Inject	
Seed	steps	saved	steps	saved
0 (main)	44 / 89	+**51%**	29 / 57.5	+**50%**
1	46 / 89.5	+**49%**	29.5 / 61	+**52%**
2	48 / 89	+**46%**	32 / 57	+**44%**
3	44 / 89	+**51%**	28 / 55.5	+**50%**

C.2 Other Optimizers, Learning Rates and Full Fine-Tuning

The update advantage of specialists on time-varying facts is robust to how the update is performed. Table 8 shows the same pattern across optimizers, learning rates, LoRA ranks, and full fine-tuning: chronological models consistently require fewer update steps than shuffled models. The saving is about 50% under Adam (Kingma & Ba, 2015) and Lion (Chen et al., 2023) , and is even larger under SGD. The step law in Section 5.2 continues to predict Adam, Lion, etc. well. (Appendix C.3.1).

C.3 Derivation of the Step Law

This appendix derives Eq. 3 step by step (Appendix C.3.1), checks each approximation (Appendix C.3.2), and derives the adapter-gradient factorization used in Figure 4c (Appendix C.3.3).

C.3.1 From the Trajectory to the Law

Setting. For each update, we fix the prompt, the new value, and the competitor, defined as the strongest value the model predicts for that fact before the update (Section 3). The adapter parameters

Table 8: The saving of temporal order does not depend on how the update is made. Each block changes one part of the update and keeps the rest as in the main experiments (LoRA of rank 16, Adam, learning rate 10^{-5}). Cells read as in Table 1: median steps, chronological / shuffled, and the share of steps temporal order saves on time-varying facts.

		Overwrite		Inject	
Varied	Setting	steps	saved	steps	saved
Optimizer	Adam	47.0 / 92.5	+**49%**	32.0 / 54.0	+**41%**
	Lion	51.0 / 108	+**53%**	32.5 / 58.5	+**44%**
	SGD	23.5 / 214	+**89%**	18.0 / 118	+**85%**
	SGD + momentum	7.5 / 33.5	+**78%**	6.0 / 21.5	+**72%**
Learning rate	Adam, 3×10^{-5}	18.0 / 32.5	+**45%**	10.0 / 20.0	+**50%**
	Adam, 3×10^{-6}	148 / 266	+**45%**	87.0 / 169	+**49%**
	Adam, 10^{-6}	418 / 781	+**46%**	248 / 492	+**50%**
	Lion, 3×10^{-6}	168 / 357	+**53%**	106 / 194	+**45%**
	Lion, 10^{-6}	497 / 1061	+**53%**	318 / 580	+**45%**
	SGD, 10^{-2}	3.0 / 22.0	+**86%**	2.0 / 12.5	+**84%**
	SGD, 3×10^{-3}	8.0 / 71.5	+**89%**	6.0 / 40.0	+**85%**
	SGD, 3×10^{-4}	76.5 / 708	+**89%**	57.5 / 394	+**85%**
LoRA rank	$r = 16$	49.5 / 87.0	+**43%**	28.0 / 54.5	+**49%**
	$r = 4$	113 / 184	+**38%**	72.0 / 125	+**42%**
	$r = 1$	234 / 368	+**36%**	152 / 258	+**41%**
Full fine-tuning	Adam, 3×10^{-6}	4.0 / 9.0	+**56%**	2.0 / 4.0	+**50%**
	Adam, 10^{-6}	10.0 / 24.0	+**58%**	5.0 / 10.5	+**52%**

θ start from θ_0, with a fresh optimizer state and no weight decay. We define

$$L(\theta) = -\log P_{\text{new}}(\theta), \quad C(\theta) = \log P_{\text{new}}(\theta) - \log P_{\text{old}}(\theta), \quad D = -C(\theta_0) > 0, \quad g_0 = \nabla L(\theta_0), \tag{13}$$

where L is the update loss, C is the log-probability margin of the new value over the competitor, D is the initial gap that the update must close, and g_0 is the gradient before the first update step.

The update succeeds once the new value becomes more likely than the competitor:

$$N = \min\{k : C(\theta_k) > 0\}.$$

We write $\Delta\theta_k := \theta_{k+1} - \theta_k$ for the parameter change made by Adam at step k. All probabilities are computed over the full target value. In practice, we minimize the mean loss over its tokens.

The margin travelled is a path integral. Let $\theta(t)$, $t \in [0, T]$, denote the parameter path, with $\theta(k) = \theta_k$ at each integer step, and let $\dot\theta(t)$ be its velocity. By the chain rule, the total change in the margin from the start to step T is

$$C(\theta_T) - C(\theta_0) = \int_0^T \frac{\mathrm{d}\, C(\theta(t))}{\mathrm{d}t}\,\mathrm{d}t = \int_0^T \nabla_\theta C\big(\theta(t)\big)^\top \dot\theta(t)\,\mathrm{d}t. \tag{14}$$

The integrand has two parts. The first is the margin gradient, $\nabla_\theta C = \nabla_\theta \log P_{\text{new}} - \nabla_\theta \log P_{\text{old}}$, which we can compute at any parameter value, including θ_0 before the update. The second is the parameter velocity $\dot\theta(t)$. Unlike the gradient, this velocity has no closed form because Adam generates each iterate from the history of previous gradients through its running moments.

The five steps below simplify this expression until the update cost can be written using quantities available before the update. In the following, we write ∇ for ∇_θ.

Step 1: break the path into optimizer steps. The optimizer gives us the discrete iterates $\theta_0, \theta_1, \ldots$, so we first split the path at these points. We connect each pair of consecutive iterates with a straight segment:

$$\theta(t) = \theta_k + (t - k)\,\Delta\theta_k, \qquad \dot\theta(t) = \Delta\theta_k, \qquad k \le t \le k + 1. \tag{15}$$

With the substitution $s = t - k$, every segment maps onto the unit interval:

$$C(\theta_T) - C(\theta_0) = \sum_{k=0}^{T-1} \int_0^1 \nabla C(\theta_k + s\,\Delta\theta_k)^\top \Delta\theta_k\,\mathrm{d}s. \tag{16}$$

Because ∇C is a gradient field, the integral between two iterates depends only on their endpoints. Thus, the k-th term is exactly $C(\theta_{k+1}) - C(\theta_k)$, regardless of the path used to connect them.

Step 2: linearize the margin within each step. We next simplify what happens between two consecutive iterates. A segment is short, η per coordinate under Step 4, so we approximate the margin gradient along the segment by its value at the start, $\nabla C(\theta_k)$. This is the first-order Taylor approximation of C around θ_k. The integral then becomes

$$\Delta C_k := C(\theta_{k+1}) - C(\theta_k) \approx v_k := \nabla C(\theta_k)^\top \Delta\theta_k, \qquad \text{so} \qquad C(\theta_T) \approx -D + \sum_{k<T} v_k. \quad (17)$$

We call v_k the update speed at step k: it is how much that step closes the margin. The update starts with a gap D and succeeds once the accumulated gain reaches this gap, i.e., when $\sum_{k<N} v_k \geq D$. This approximation drops only the curvature of C within each optimizer step.

Step 3: use the first-step speed to estimate the average speed. The expression above still requires every v_k along the update trajectory. To predict the update cost before running the update, we instead use v_0, which is available at the start. The speed does not remain exactly equal to v_0 (for example, momentum and the gradient change as P_{new} increases). We therefore assume that the average speed up to success is a fixed multiple A of the first-step speed,

$$\frac{1}{N}\sum_{k<N} v_k \approx A\, v_0 \qquad \Longrightarrow \qquad \widehat{N} = \frac{D}{A\, v_0}, \qquad v_0 > 0. \quad (18)$$

This introduces a fitted parameter A. Appendix C.3.2 measures how well Av_0 approximates the average speed. The remaining two steps express v_0 using the initial loss gradient g_0.

Step 4: Adam's first update follows the gradient sign. We now write the first parameter update explicitly. At iterate k, Adam (Kingma & Ba, 2015) reads the gradient $g_k = \nabla L(\theta_k)$, updates its two running moments, and applies

$$\mu_k = \beta_1 \mu_{k-1} + (1-\beta_1)\, g_k, \qquad \nu_k = \beta_2 \nu_{k-1} + (1-\beta_2)\, g_k^{\odot 2}, \quad (19)$$

$$\Delta\theta_k = -\eta\, \frac{\hat{\mu}_k}{\sqrt{\hat{\nu}_k} + \epsilon}, \qquad \hat{\mu}_k = \frac{\mu_k}{1-\beta_1^{k+1}}, \quad \hat{\nu}_k = \frac{\nu_k}{1-\beta_2^{k+1}}, \quad (20)$$

with $\mu_{-1} = \nu_{-1} = 0$. At the first step, the optimizer state is still zero. Thus, $\mu_0 = (1-\beta_1)g_0$ and $\nu_0 = (1-\beta_2)g_0^{\odot 2}$, and the bias corrections give $\hat{\mu}_0 = g_0$ and $\hat{\nu}_0 = g_0^{\odot 2}$. The first update is therefore

$$\Delta\theta_0 = -\eta\, \frac{g_0}{|g_0| + \epsilon} \;\approx\; -\eta\, \operatorname{sign}(g_0), \qquad v_0 = \nabla C(\theta_0)^\top \Delta\theta_0 \approx -\eta\, \nabla C(\theta_0)^\top \operatorname{sign}(g_0). \quad (21)$$

Ignoring ϵ, each coordinate moves by approximately η, and the gradient determines its direction rather than its magnitude.

Step 5: express the first-step speed using the loss gradient. The expression for v_0 still contains the margin gradient $\nabla C(\theta_0) = \nabla \log P_{\text{new}} - \nabla \log P_{\text{old}}$. The optimizer, however, directly computes only $g_0 = -\nabla \log P_{\text{new}}$. We therefore rewrite ∇C in terms of g_0.

Let $S = P_{\text{new}} + P_{\text{old}}$ be the total probability assigned to the new value and its competitor, and let $q_{\text{old}} = P_{\text{old}}/S$ be the competitor's share. Since $P_{\text{new}}/S = e^C/(1+e^C) = \sigma(C)$, we can write $\log P_{\text{new}} = \log S + \log \sigma(C)$. Differentiating, with $\sigma'/\sigma = 1 - \sigma = q_{\text{old}}$, gives

$$g_0 = -\nabla \log S(\theta_0) - q_{\text{old}}\, \nabla C(\theta_0), \quad (22)$$

The first term changes the total probability of the two values, while the second changes their relative probability.

Solving Eq. 22 for $\nabla C(\theta_0)$ and substituting it into Eq. 21, with $g_0^\top \operatorname{sign}(g_0) = \|g_0\|_1$, gives

$$v_0 \approx \frac{\eta}{q_{\text{old}}}\Big[\|g_0\|_1 + \nabla \log S(\theta_0)^\top \operatorname{sign}(g_0)\Big] \;\approx\; \frac{\eta\, \|g_0\|_1}{q_{\text{old}}}. \quad (23)$$

The last approximation drops the change in the total probability S along the sign step. This term is small in our updates, only one to two percent of $\|g_0\|_1$.

Putting the steps together. Substituting Eq. 23 into Eq. 18 gives

$$\widehat{N} = \frac{q_{\text{old}}\, D}{\eta A\, \|g_0\|_1}, \quad (24)$$

where D is the initial gap and $\|g_0\|_1$ determines the initial update speed. Since $q_{\text{old}} = (1+e^{-D})^{-1}$ is close to one ($q_{\text{old}} \approx 1$), the main text drops this factor and writes Eq. 3. Here, $\|g_0\|_1$ is the L_1 norm of the initial gradient of $\log P_{\text{new}}$ over all adapter parameters, measured directly for every update.

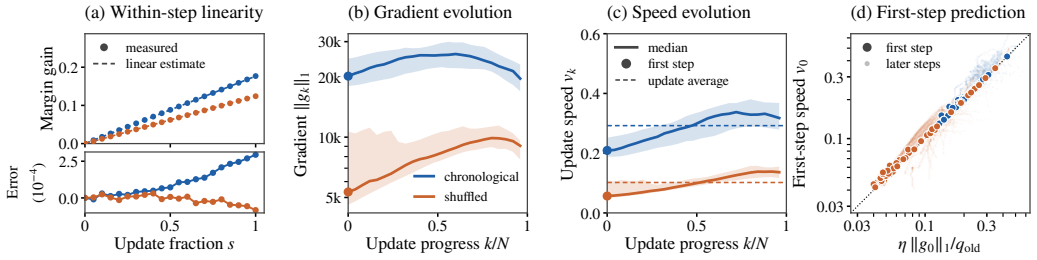

Figure 11: The step-law approximations along update trajectories, for time-varying overwrite updates in the 110M C (blue) and S (orange) under Adam. (a) Measured margin gain against its linear estimate along the first update; the lower plot shows their difference in log-probability, in units of 10^{-4}. Each model is represented by the update with the median first-step error. (b) Gradient norm over update progress k/N. (c) Update speed over the same progress: dots mark the first step and dashed lines the median of per-update average speeds. In (b, c), solid curves are medians and shaded bands interquartile ranges. (d) First-step speed against its gradient-based estimate $\eta\|g_0\|_1/q_{\text{old}}$; the diagonal is exact agreement, and faint points show later steps.

Fitting the constant. We fit A once, as the median ratio of $q_{\text{old}}D/(\eta\|g_0\|_1)$ to the observed steps on half of the updates of the undated C at the three scales, which gives $A = 1.27$; every prediction in Section 5.2 uses the other half. Fitted separately on each model of Figure 4a, A lies between 1.19 and 1.41 for the chronological models and between 1.42 and 1.67 for the shuffled ones, whatever their scale or timestamps.

C.3.2 How Good Each Approximation Is

Figure 11 shows the approximations at work on the 110M C / S pair. Step 2 drops the curvature within one update, which is 0.2% of the margin change at the first step and under 1% at any later one, so the first-order sum recovers 99.4% of the margin travelled (Figure 11 (a)). Step 3 lets the first speed stand for the average; the average is 1.3 times the first speed in C and 1.8 times in S, whose gradient keeps growing as the update proceeds (Figure 11 (b) and Figure 11 (c)). Steps 4 and 5 replace the first speed by $\eta\|g_0\|_1/q_{\text{old}}$, which matches the measured v_0 to within 3% (Figure 11 (d)).

C.3.3 Factorizing the Adapter Gradient

Figure 4 (c) draws the initial gradient module by module.

A LoRA adapter sits on a linear module m with frozen weight W_m, and the training sentence passes through the module once per token p. Let $x_{m,p} \in \mathbb{R}^{d_{\text{in}}}$ be the module's input at token p and $y_{m,p} \in \mathbb{R}^{d_{\text{out}}}$ its output. The adapter adds a low-rank term,

$$y_{m,p} = W_m x_{m,p} + \frac{\alpha}{r} B_m A_m x_{m,p} = W_m x_{m,p} + \frac{\alpha}{r} B_m z_{m,p}, \qquad z_{m,p} := A_m x_{m,p} \in \mathbb{R}^r, \quad (25)$$

with $A_m \in \mathbb{R}^{r \times d_{\text{in}}}$ drawn at random and $B_m \in \mathbb{R}^{d_{\text{out}} \times r}$ zero at the start, so that the adapter changes nothing until it is trained; $\alpha = r$ in all our updates. The objective $\log P_{\text{new}}$ depends on B_m only through the outputs $y_{m,p}$, so with $\delta_{m,p} := \partial \log P_{\text{new}}/\partial y_{m,p} \in \mathbb{R}^{d_{\text{out}}}$, the backward sensitivity that backpropagation delivers at the module's output, the chain rule over tokens gives

$$\frac{\partial \log P_{\text{new}}}{\partial B_m} = \frac{\alpha}{r} \sum_p \delta_{m,p}\, z_{m,p}^\top, \qquad \frac{\partial \log P_{\text{new}}}{\partial A_m} = \frac{\alpha}{r} \sum_p B_m^\top \delta_{m,p}\, x_{m,p}^\top. \quad (26)$$

The gradient of A_m vanishes at the start of the update because $B_m = 0$: the only path from A_m to the output runs through B_m, and it carries nothing back. With $B_m = 0$ and $\alpha = r$, the whole gradient the adapter at module m receives is therefore

$$g_m = \sum_p \delta_{m,p}\, z_{m,p}^\top. \quad (27)$$

The loss gradient $g_0 = -\nabla \log P_{\text{new}}$ is the adapters' gradients laid end to end, with the A-parts all zero; Figure 4 (c) draws each adapter's share of it as a cell area, both models on the chronological model's scale.

D Details on Four Applications

D.1 Mixing Generalists and Specialists

Corpora. Mix-C in Section 6.1 combines a generalist for the history with a specialist for the recent window by splitting each person's records at 2021. Records before 2021 keep their timestamps and are shuffled. Records from 2021 onward lose half of their timestamps and keep each person's chronological order. Mix-S contains the same records but shuffles them all. Both models are trained at 110M using the recipe in Appendix A and updated as in Section 3.

Batch and sequential updates. A batch update writes many facts into one adapter by training on all of their sentences together. A sequential update applies a series of such batches, merging each batch into the model before the next. In Figure 5(c), a batch contains 2,400 facts, while a sequence contains ten batches of fifty. We measure forgetting after the batch, or after the final batch in the sequence, on facts about other people as in Appendix A. Across all three settings, Mix-C needs 16–44% fewer steps than Mix-S and forgets 5–57% less.

D.2 Reshaping a Real-World Pre-trained Model

Data. We continue pre-training Gemma-2-9B-it (Team, 2024) on KairosQA (Pilchen et al., 2026), a set of real facts whose answers change over time. Each fact is a question with a different answer in different years: for example, the head coach of Tottenham Hotspur is Mauricio Pochettino in 2014–2019, José Mourinho in 2020 and Antonio Conte in 2021–2023. This is like the employer of a person in our biographies. We use 5,324 such facts whose answer changes between 2001 and 2023, from eight kinds of question (for example, the head coach of a team, the team of a player and the head of a government), and drop the facts about subjects in the CounterFact evaluation set. Each record asks for one fact in Gemma's chat format, for example "Who was the head coach of Tottenham Hotspur F.C. in 2019? Answer with the name only.". An undated record drops the year.

Stage. The stage trains on three kinds of text. The first asks each fact without a year over its recent years, 2016–2023 (52M tokens); this part makes the model a specialist in the latest answer. The second asks each fact with the year over its whole history, 2001–2023 (35M tokens); this part keeps the model a generalist about the past. The third is ordinary web text from FineWeb-edu (Penedo et al., 2024) (53M tokens), which keeps the model's general ability. C-C shows the recent years of each fact in time order and mixes in the rest at random; C-S shuffles everything. The stage runs 5,174 steps, 140M tokens in total, at a constant learning rate of 3×10^{-6}.

Updates. We adopted 170 *overwrite* updates that are not in the continual pre-training stage. Each update writes the new answer using the LoRA setup of Section 3 (rank 16, learning rate 10^{-6}), with either an undated prompt or a dated prompt beginning with "In 2023".

Forgetting. The replay preserves historical knowledge. On questions about past years, C-C improves accuracy from 40% in Base to 99%. This shows that the stage strengthens the model's generalist knowledge. But the replay only protects the facts it covers. KairosQA facts that never appear in the replay lose about 25% of the past-year answers that Base originally gets right, in both C-C and C-S, while facts from other sources, such as the CounterFact facts below, do not drop. This suggests that historical knowledge should be included in the replay if we want the model to keep it.

Finally, the stage causes little loss in general ability (Table 9). On ARC-Easy (Clark et al., 2018), PIQA (Bisk et al., 2020), and HellaSwag (Zellers et al., 2019), C-C is at most 1.1 below Base. On CounterFact (Meng et al., 2022), C-C matches Base.

D.3 Improving Published-Knowledge Editors

Editors. We use the official implementations and hyperparameters of ROME (Meng et al., 2022), MEMIT (Meng et al., 2023), AlphaEdit (Fang et al., 2025) and ENCORE (Gupta et al., 2025),

Table 9: General ability of Gemma-2-9B-it before (Base) and after the continual pre-training stage (C-C, C-S). Accuracy (%) on each benchmark and on CounterFact facts, higher is better.

Model	ARC-Easy (Clark et al., 2018)	PIQA (Bisk et al., 2020)	HellaSwag (Zellers et al., 2019)	CounterFact (Meng et al., 2022)
Base	85.8	81.1	52.8	89.0
C-C	84.7	81.3	52.8	88.9
C-S	84.4	80.9	52.8	89.1

with ROME on layer 2 and the others on layers 1–4 of our 12-layer model; AlphaEdit's null-space threshold is raised from 0.02 to 0.1, since at 0.02 layer 4 has no null space. Each editor makes the same 60 overwrite updates of time-varying facts in Mix-C and Mix-S. Cost is the FLOPs spent per fact written, where a fact counts as written once its new value's probability passes that of the strongest value the model held before updating.

Results. Every editor forgets 50–63% less on Mix-C than on Mix-S (Table 10).

Table 10: The four editors on Mix-C and Mix-S, over the same 60 overwrite updates. Each cell is Mix-C / Mix-S, followed by the share Mix-C saves; lower is better. Updating cost: 10^{12} FLOPs per fact written. Forgetting, other people: percentage of other people's facts whose answer the update changes. Forgetting, own history: percentage of the updated person's dated history that the update changes. Bold: $p < 0.05$.

			Forgetting (%)			
Editor	Updating cost	saved	Other people	reduced	Own history	reduced
ROME (Meng et al., 2022)	5.36 / 5.65	+**5%**	2.1 / 4.2	+**50%**	81.8 / 80.2	−2%
MEMIT (Meng et al., 2023)	0.78 / 1.44	+**46%**	0.9 / 2.2	+**59%**	14.8 / 22.1	+33%
AlphaEdit (Fang et al., 2025)	0.77 / 1.31	+**41%**	1.2 / 3.2	+**63%**	16.0 / 24.7	+**35%**
ENCORE (Gupta et al., 2025)	0.75 / 1.10	+**32%**	0.8 / 2.1	+**60%**	14.5 / 18.8	+23%

D.4 Versioned Software Change

Tool-call dataset. We build the tool-call dataset in the same way as the biography dataset in Section 3: a tool corresponds to a person, one of its six fields to an attribute, and the software version to the year. The dataset contains 25,000 tools and 25 versions, from v1 to v25; each tool appears 1,000 times, and a fixed set of tools is held out for injection updates.

The six fields follow the same change rates as the biography attributes: package and function never change, host and authentication header change a few times, and path and input parameter change often. Each record gives the six fields of one tool at one version:

> In version v8, the Flint Payments API uses package flintpay_sdk and function revoke_refund. It carries host api.flintpay.com and authentication header X-Api-Key. It is called at path /v1/refunds with input parameter refund_id.

Training and updates. For pre-training from scratch, we follow Section 6.1: Mix-C shuffles the history (v1–v21) with version labels, while keeping each tool's recent records (v22–v25) in version order with half of the version labels removed. Mix-S contains the same records but shuffles them all.

For continual pre-training, we follow Appendix D.5: starting from a Base model trained on shuffled records with version labels, C-C presents a window of new versions in order and mixes them with a shuffled replay of the history, while C-S shuffles the same records. Every model receives single updates both with and without the version in the prompt.

Results. We find that temporal order brings the same benefits to tool calls as to biographies. All models keep the old versions well: with the version in the prompt, query accuracy stays between 0.999 and 1.00. Without the version, Mix-C reaches 0.98 accuracy on the latest version, compared with 0.26 for Mix-S. After continual pre-training, C-C reaches 0.99, compared with 0.76 for C-S. The chronological models also update faster (Table 11). Without the version in the prompt, they need

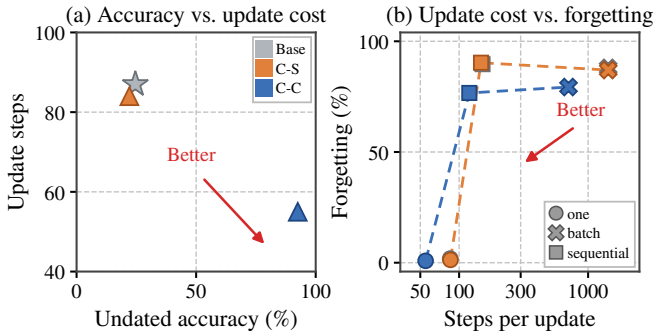

Figure 12: Reshaping the 110M SD$^{1/2}$ model. (a) Undated query accuracy against the median update steps for Base, C-S and C-C. (b) Steps per update against forgetting for single, batch and sequential updates of Base (grey), C-S (orange) and C-C (blue); lower left is better.

14–48% fewer steps to update a changing field. With the version in the prompt, the saving becomes smaller, at 7–29%.

Table 11: Median steps to update a field in the tool-call world, for slow and fast fields. Rows: pre-training from scratch (Mix-C / Mix-S) and continual pre-training (C-C / C-S). *saved* is the share of steps the chronological model saves; bold: $p < 0.05$.

	No version in prompt				Version in prompt			
	Overwrite		Inject		Overwrite		Inject	
Fact	steps	saved	steps	saved	steps	saved	steps	saved
Pre-training from scratch: Mix-C / Mix-S								
slow	53.0 / 102	**+48%**	38.5 / 64.0	**+40%**	55.0 / 77.0	**+29%**	40.5 / 57.0	**+29%**
fast	47.0 / 84.0	**+44%**	33.5 / 47.5	**+29%**	48.5 / 68.5	**+29%**	32.0 / 42.0	**+24%**
Continual pre-training: C-C / C-S								
slow	49.5 / 70.5	**+30%**	37.5 / 45.0	**+17%**	55.5 / 61.5	**+10%**	40.0 / 46.5	**+14%**
fast	43.5 / 64.5	**+33%**	34.5 / 40.0	**+14%**	46.0 / 49.5	**+7%**	32.0 / 37.0	**+14%**

D.5 Reshaping a Pre-trained Model in Synthetic World

The continual pre-training stage of Section 6.2, applied to the 110M SD$^{1/2}$ model of Section 4; Section 6.4 follows the same stage.

Dataset setup. Base is the 110M SD$^{1/2}$ model of Section 4. It is pre-trained on shuffled records, and only a random half of them carry a timestamp. Continual pre-training shows Base only records it has already seen, in two parts. The historical part has 100 timestamped records per person in shuffled order, and states every value the person has held. The recent part has 200 undated records per person in chronological order, and includes every change of value. C-C mixes the two parts and keeps the order within each person; C-S holds exactly the same records, shuffled. Each stage is one pass of about 2800 steps at batch 2,560 (about 0.6B tokens) at a constant learning rate of 6×10^{-5}, the final rate of pre-training.

Results. We find that a short chronological stage gives Base a specialist for the present while keeping its past. C-C raises Base's undated query accuracy from 25% to 93% and keeps its dated accuracy at 98%. C-S, trained on the same records, reaches only 22% (Figure 12 (a)). C-C also updates faster: it needs up to 1.6× fewer steps than Base for single updates, 2.1× fewer in a batch and 1.3× fewer in a sequence. It also forgets 10–15% less (Figure 12 (b)).