LEARNING FROM LATE REWARDS: ASYNCHRONOUS DATA SYNTHESIS WITH REALIZED LEARNING UTILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Synthetic data is increasingly central to improving large language models, yet its value is commonly estimated from signals available before learning takes place. We argue that the true value of synthetic data is instead *learner-relative* and *outcome-defined*: it should be measured by the downstream improvement that training on the data actually induces. We call this quantity **Realized Learning Utility**. Grounding synthesis in realized utility, however, fundamentally turns data synthesis into a **late-reward optimization** problem: synchronous optimization preserves faithful feedback but blocks synthesis, whereas naive asynchronous optimization exposes the entire synthesis update to stale feedback from an evolving learner. We introduce **LateSyn**, a **Predict-Early, Correct-Late** framework that estimates future realized utility at generation time, applies a reliability-adaptive early update, and later corrects the original synthesis trajectory using the residual revealed by the actual learner outcome. A learning-utility memory further warm-starts prediction from previously resolved outcomes. Theoretically, LateSyn preserves the realized-utility target while reducing the synthesis signal exposed to delay from the full advantage to only its unresolved residual. Experiments across diverse agentic environments show that LateSyn produces substantially healthier learner trajectories in regimes where delayed feedback becomes harmful.

1 INTRODUCTION

Large language models increasingly improve through synthetic data, using model-generated problems, trajectories, interactions, and demonstrations to extend learning beyond fixed human-curated corpora Wang et al. (2023); Tang et al. (2025); Pahuja et al. (2025); Wang et al. (2026b). Yet a basic question remains surprisingly under-specified: *what makes a synthetic example valuable?* Existing synthesis pipelines often characterize data by generation-time properties such as difficulty, diversity, correctness, uncertainty, or apparent learnability Li et al. (2026); Tu et al. (2025); Liang et al. (2026); Yang et al. (2026). These signals describe the data before learning, whereas synthetic data ultimately matters through the change it induces in the downstream learner. The same example may be informative for a weak learner, redundant for a stronger one, or even harmful under a different learner state. We therefore argue that synthetic-data value is fundamentally learner-relative and outcome-defined: it should be measured by the improvement that actually results from training on the data. We call this quantity **Realized Learning Utility**, the downstream performance change induced by a learner update. This shifts the central question from generating examples that merely *appear* useful to generating examples that *cause useful learning*: how should a synthesizer be optimized when data value is revealed only through the learning process it triggers?

This view exposes a fundamental limitation in how synthetic data is typically optimized. Most synthesis objectives rely on *pre-learning signals*: quantities that can be evaluated before the learner is actually updated Tu et al. (2025); Li et al. (2026); Liang et al. (2026); Yang et al. (2026). Although such signals can be effective surrogates, they create a **Proxy–Outcome Gap**: they estimate whether data *should* be useful rather than whether the learner *actually becomes better* after training on it. Closing this gap suggests a different principle, which we call **Learning-Grounded Data Synthesis**: the synthesizer should be optimized against the downstream learning outcome induced by its generated data, rather than against a score assigned to the data itself. This directly ties synthesis decisions

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

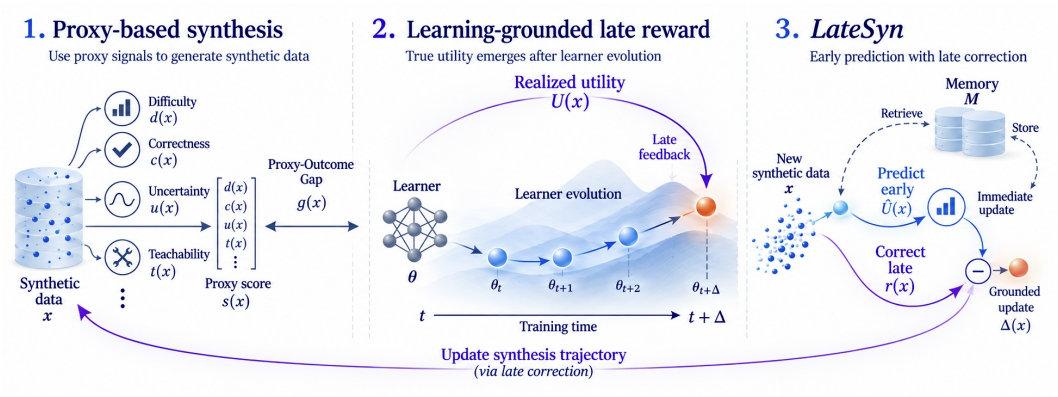

Figure 1: **From proxy-based synthesis to learning-grounded synthesis under late feedback.** Conventional synthesis optimizes generation-time proxy signals, leaving a *Proxy–Outcome Gap* between predicted data value and the learning it actually induces (left). Grounding synthesis in *Realized Learning Utility* closes this gap, but makes the faithful supervision inherently delayed because utility is only observed after the learner evolves (middle). **LateSyn** addresses this late-reward problem through a *Predict-Early, Correct-Late* strategy: it uses predicted future utility for an immediate synthesis update and later reconciles the update with the realized learner outcome, while a learning-utility memory supports future prediction (right).

to their realized consequence on learning, but it also introduces a difficulty absent from proxy-based objectives: the supervision we care about does not yet exist when the synthesis decision is made. The most faithful reward for data synthesis therefore becomes available only after the learner has consumed the data, updated its parameters, and been evaluated again Huang et al. (2025); Wang et al. (2025); Yue et al. (2026). Figure 1 summarizes this paradigm shift.

The resulting optimization problem is inherently one of **late feedback**. To observe the true utility of a synthesized batch, the system must first generate the data, let the learner train on it, and only then evaluate the learner again Huang et al. (2025); Wang et al. (2025). A fully synchronous loop preserves this learning-grounded supervision but forces synthesis to wait behind every learner update and evaluation, whereas a naïve asynchronous loop removes this barrier by continuing generation while previous batches are still being consumed Chen et al. (2026b). The returned utility then becomes stale: by the time the reward for D_t is observed, both the synthesizer and learner may have moved away from the states under which D_t was produced. This creates two coupled mismatches: synthesizer-policy staleness, because delayed feedback is assigned to an older synthesis trajectory, and learner-state staleness, because batch utility depends on the evolving learner that eventually consumes it Yang et al. (2026). More than delayed reward computation, the reward itself is *realized late* by another learning process that continues to evolve while feedback is pending. This leads to the central question of our work: *can a synthesizer act before realized learning utility is known, while remaining grounded in the learning outcome that eventually occurs?*

To address the above challenges, we propose **LateSyn**, an asynchronous learning-grounded data synthesis framework that follows a **Predict-Early, Correct-Late** principle to optimize the synthesizer from realized downstream learning utility without blocking on every learner update. At generation time, a lightweight utility critic estimates the **Future Realized Learning Utility** of each synthesized batch, while a reliability-adaptive mechanism determines how much of this prediction can be trusted for an immediate synthesis update. Once the evolving learner consumes the batch and its actual learning utility is observed, the unresolved residual is routed back to the original synthesis trajectory through delayed off-policy correction. In this way, prediction provides immediacy while realized learner improvement remains the final source of supervision. We show theoretically that Predict-Early, Correct-Late reduces the portion of the synthesis signal exposed to delayed policy drift, and empirically demonstrate its effectiveness under high-feedback-delay synthesis across diverse agentic learning environments.

Our contributions are three-fold:

- We formulate **Learning-Grounded Data Synthesis** through **Realized Learning Utility**, exposing the **Proxy–Outcome Gap** between generation-time signals and realized learning outcomes.
- We propose **LateSyn**, a **Predict-Early, Correct-Late** framework with reliability-adaptive utility prediction and delayed residual correction, together with theoretical guarantees on credit decomposition and delay exposure.
- Experiments across diverse agentic environments show that LateSyn improves learner performance and stabilizes training under harmful feedback delay.

2 Related Work

Synthetic data has become a central component of LLM post-training, spanning instruction generation, reasoning supervision, filtering, and automated data refinement Wang et al. (2023); Xu et al. (2024); Li et al. (2023); Tang et al. (2025); Li et al. (2026); Cao et al. (2025); He et al. (2025); Tu et al. (2025). In verifiable reasoning, methods such as STaR, WizardMath, MetaMath, MAmmoTH, ToRA, DeepSeekMath, and NuminaMath exploit generated problems, rationales, or checkable outcomes to expand supervision Zelikman et al. (2022); Luo et al. (2025); Yu et al. (2023); Yue et al. (2024); Gou et al. (2024); Shao et al. (2024); Li et al. (2024), while learner-aware and self-evolving approaches increasingly use model weaknesses or interaction feedback to guide subsequent synthesis Liang et al. (2026); Huang et al. (2025); Wang et al. (2025); Xia et al. (2025); Yue et al. (2026); Kwan et al. (2026). Related agentic-data work emphasizes executable environments, trajectory grounding, verification, and continual interaction improvement Pahuja et al. (2025); Sun et al. (2025); Ramrakhya et al. (2025); Wang et al. (2026b;a); Chen et al. (2026a; 2025); Yang et al. (2026); Dong et al. (2026); Dai et al. (2026); Zhai et al. (2025); Wu et al. (2025); Zhang et al. (2026). These advances increasingly couple synthesis with learner or environment feedback, but the guiding signal is typically available before training or derived from correctness, difficulty, validity, uncertainty, or related proxies. We provide a more detailed discussion of related synthetic-data and agentic-synthesis methods in Appendix C.

3 Method

In this section, we introduce **LateSyn**, our framework for learning-grounded data synthesis under delayed feedback; an overview is shown in Fig. 2. LateSyn follows a *Predict-Early, Correct-Late* principle: it uses generation-time predictions of future learning utility for immediate synthesis updates, and later reconciles them with the realized learner outcome.

3.1 Problem Formulation: Learning-Grounded Synthesis under Late Feedback

Realized learning utility. Let π_ϕ denote a trainable synthesizer and f_θ the downstream learner. At synthesis step t, the synthesizer generates a batch $D_t \sim \pi_{\phi_t}(\cdot \mid x_t)$, where x_t denotes the information available at generation time. Rather than assigning D_t a proxy score, we measure its value through the improvement it actually induces in the learner. Let $\mathcal{U}$ denote the learner update operator and $V(\theta; \mathcal{Q}_{\text{util}})$ the learner performance on a held-out utility probe. In the synchronous case, $\theta_t^+ = \mathcal{U}(\theta_t, D_t)$, and we define

$$r_t^{\text{LU}} = V(\theta_t^+; \mathcal{Q}_{\text{util}}) - V(\theta_t; \mathcal{Q}_{\text{util}}). \tag{1}$$

We call r_t^{LU} the *Realized Learning Utility* (LU). It is *learner-relative*, the same batch may have different value for different learner states, and *outcome-defined*, since it becomes observable only after the learner has actually trained on D_t.

The corresponding learning-grounded synthesis objective is

$$\max_{\phi} \; \mathbb{E}_{D \sim \pi_\phi(\cdot|x)} \left[V(\mathcal{U}(\theta, D); \mathcal{Q}_{\text{util}}) - V(\theta; \mathcal{Q}_{\text{util}}) \right], \tag{2}$$

which directly rewards synthetic data by the learning it causes rather than by a generation-time proxy.

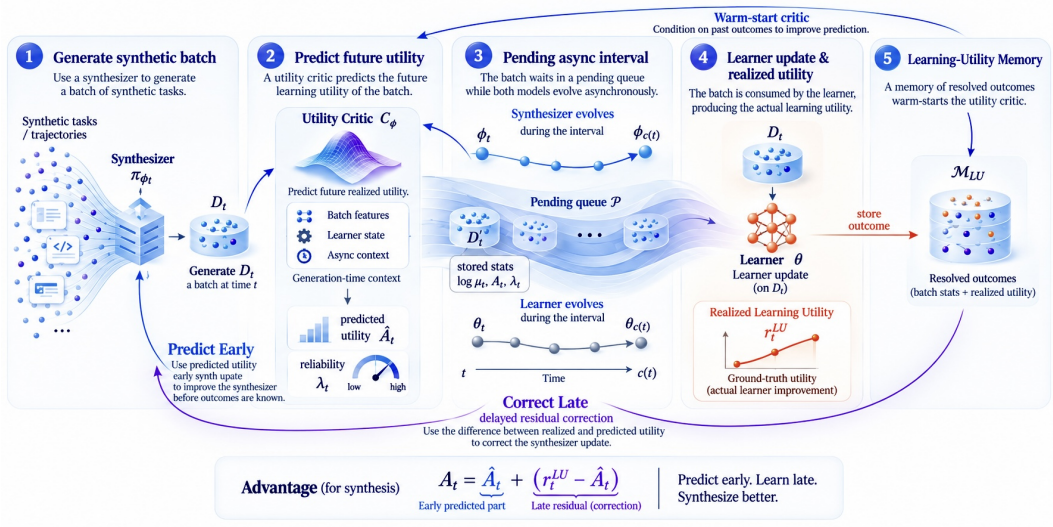

Figure 2: **Overview of LateSyn.** The synthesizer generates a synthetic batch and receives an immediate reliability-adaptive update from predicted future learning utility. After learner consumes batch, the realized learning utility provides a delayed residual correction to the original trajectory.

Late feedback under asynchrony. The difficulty is that the reward in Eq. equation 1 is unavailable when the synthesis decision is made. To decouple synthesis from learner updates, let a batch generated at step t be consumed at $c(t) = t + \tau_t$, where $\tau_t \geq 0$ denotes its feedback delay. Its realized utility is then determined by the learner state at consumption time:

$$\theta_{c(t)+1} = \mathcal{U}\big(\theta_{c(t)}, D_t\big)\,, \qquad r_t^{\mathrm{LU}} = V(\theta_{c(t)+1}; \mathcal{Q}_{\mathrm{util}}) - V(\theta_{c(t)}; \mathcal{Q}_{\mathrm{util}}). \tag{3}$$

While D_t is pending, both sides of the synthesis–learning loop may evolve, $\phi_t \to \phi_{c(t)}$ and $\theta_t \to \theta_{c(t)}$. This induces two coupled mismatches: *synthesizer-policy staleness*, because feedback is assigned to a trajectory generated by an earlier synthesis policy, and *learner-state staleness*, because the batch is ultimately consumed by an evolved learner.

3.2 Predict Early, Correct Late

Realized LU is eventually observable, but it is unavailable *when the synthesis decision is made*. LateSyn therefore predicts the component of future learning utility that can already be inferred at generation time, applies it immediately, and uses the eventual learner outcome to correct the remaining error.

Future realized-utility prediction. For each synthesized batch D_t, we introduce a lightweight utility critic C_ψ using only generation-time information x_t:

$$\hat{r}_t = C_\psi(D_t, x_t) \approx \mathbb{E}\left[r_t^{\mathrm{LU}} \mid D_t, x_t\right]. \tag{4}$$

Importantly, C_ψ predicts *future realized learning utility*, rather than an intrinsic notion of data quality. The expectation implicitly marginalizes over the learner evolution between generation and consumption, avoiding the need to explicitly predict the future learner parameters.

Let b_t be the generation-time reward baseline, and define $A_t = r_t^{\mathrm{LU}} - b_t$ and $\hat{A}_t = \hat{r}_t - b_t$. Instead of replacing the realized advantage with its prediction, LateSyn decomposes it as

$$A_t = \underbrace{\lambda_t \hat{A}_t}_{\text{predict early}} + \underbrace{\left(A_t - \lambda_t \hat{A}_t\right)}_{\text{correct late}}, \tag{5}$$

where $\lambda_t \in [0, 1]$ determines how much of the predicted signal is trusted. Prediction therefore controls *when* part of the update is applied, while the realized learner outcome still determines the complete supervision signal.

Predict early. Immediately after generating D_t, LateSyn applies the fresh update

$$g_t^{\text{early}} = \lambda_t \hat{A}_t \nabla_\phi \log \pi_{\phi_t}(D_t \mid x_t). \tag{6}$$

Because this update is evaluated under the policy that generated D_t, it is not yet exposed to feedback staleness. The synthesizer can then continue producing subsequent batches while D_t waits to be consumed.

Correct late. Once D_t is consumed and r_t^{LU} becomes available, LateSyn computes the unresolved residual $A_t^{\text{late}} = A_t - \lambda_t \hat{A}_t$ and applies

$$g_t^{\text{late}} = A_t^{\text{late}} \nabla_\phi \log \pi_\phi(D_t \mid x_t), \tag{7}$$

using the behavior-policy information stored when D_t was generated; the corresponding off-policy correction is described in Sec. 3.4.

The decomposition is exact: $A_t^{\text{early}} + A_t^{\text{late}} = A_t$ for any prediction $\hat{A}_t$ and any λ_t. Thus, prediction error does not replace the realized learning signal; a better critic simply moves more of that signal from the delayed correction into the fresh update. In short, prediction provides *immediacy*, while realized learning utility preserves *fidelity*.

3.3 Reliability-Adaptive Utility Prediction

The Predict-Correct decomposition is exact for any predictor, but its practical benefit depends on how much of the predicted advantage can be trusted before the realized outcome arrives. LateSyn therefore makes λ_t *reliability-adaptive*: unreliable predictions contribute little to the early update, whereas predictive ones move a larger fraction of the synthesis signal out of the delayed path.

Reliability-adaptive prediction. For a fixed predictor $\hat{A}$, we choose λ to minimize the residual advantage that remains delayed,

$$\lambda^* = \text{clip}\left(\frac{\mathbb{E}[A\hat{A}]}{\mathbb{E}[\hat{A}^2]}, 0, 1\right), \tag{8}$$

which is the minimizer of $\mathbb{E}[(A - \lambda\hat{A})^2]$ over $\lambda \in [0, 1]$. Thus, λ increases only when the predicted advantage aligns with the subsequently realized advantage, and naturally approaches zero when the critic is uninformative.

Because the current A_t is unavailable at generation time, LateSyn estimates Eq. equation 8 online using only previously resolved prediction–outcome pairs:

$$\lambda_t = \text{clip}\left(\frac{\text{EMA}[A_i\hat{A}_i]}{\text{EMA}[\hat{A}_i^2] + \epsilon}, 0, 1\right). \tag{9}$$

Hence, the current synthesis decision never uses its own future outcome to determine its early credit.

Learning-Utility Memory. Realized-LU labels are sparse and delayed, making a purely online critic prone to cold start. We therefore maintain a *Learning-Utility Memory* $\mathcal{M}_{\text{LU}} = \{(x_i, r_i^{\text{LU}})\}_{i=1}^N$ of previously resolved synthesis decisions. The critic is warm-started from $\mathcal{M}_{\text{LU}}$ and subsequently updated online as new learner outcomes are realized, while every new target remains the actual post-update learning utility.

The generation-time context $x_t = [e(D_t); z_t; q_t]$ contains only lightweight information available when D_t is synthesized, allowing the critic to predict future realized LU without an additional learner evaluation or an explicit model of the future learner state. For each batch, LateSyn freezes $(\hat{r}_t, b_t, \hat{A}_t, \lambda_t)$ at generation time for the later residual correction.

3.4 Delayed Residual Optimization

The early component in Eq. equation 5 is applied under the policy that generated D_t, whereas the residual becomes available only after the learner has consumed the batch. By then, the synthesizer

may have evolved from the behavior policy $\mu_t \equiv \pi_{\phi_t}$ to the current policy $\pi_{\phi_{c(t)}}$. LateSyn therefore retains the generation-time behavior log-probabilities and applies the delayed residual with a standard clipped off-policy correction:

$$\rho_t = \frac{\pi_{\phi_{c(t)}}(D_t \mid x_t)}{\mu_t(D_t \mid x_t)}, \qquad \mathcal{L}_{\text{late}}(\phi) = \mathbb{E}_t\left[\min\left(\rho_t A_t^{\text{late}}, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A_t^{\text{late}}\right)\right], \tag{10}$$

where $A_t^{\text{late}} = A_t - \lambda_t \hat{A}_t$. The off-policy correction itself is deliberately conventional; our goal is instead to reduce how much synthesis signal must rely on a stale trajectory.

To decouple synthesis from learner updates, LateSyn maintains a bounded pending queue with at most $M-1$ outstanding batches, where $M = 1$ recovers synchronous execution and larger M induces increasing feedback delay. Each pending batch retains its behavior-policy information and frozen generation-time Predict-Correct quantities. The synthesizer continuously generates batches and applies their early updates, while the learner independently consumes pending data, evaluates realized LU, and triggers the corresponding residual correction.

3.5 Theoretical Properties

Equation equation 5 exactly decomposes the scalar synthesis credit, $A_t^{\text{early}} + A_t^{\text{late}} = A_t$, while the two components may be applied at different policy states under asynchrony. Let $s_t = \nabla_\phi \log \pi_{\phi_t}(D_t \mid x_t)$, $s_{c(t)} = \nabla_\phi \log \pi_{\phi_{c(t)}}(D_t \mid x_t)$, $\rho_t = \pi_{\phi_{c(t)}}(D_t \mid x_t)/\pi_{\phi_t}(D_t \mid x_t)$, and use $g_t^* = A_t s_t$ as the fresh realized-utility reference.

Theorem 1 (Residual-Controlled Delay Exposure)**.** *For the unclipped importance-corrected updates,*

$$\begin{aligned} g_t^{\text{naive}} - g_t^* &= A_t\big(\rho_t s_{c(t)} - s_t\big), \\ g_t^{\text{PC}} - g_t^* &= \big(A_t - \lambda_t \hat{A}_t\big)\big(\rho_t s_{c(t)} - s_t\big). \end{aligned} \tag{11}$$

Hence, under bounded policy drift, naïve asynchronous optimization exposes the full advantage $|A_t|$ to delayed-policy mismatch, whereas Predict-Correct exposes only the unresolved residual $|A_t - \lambda_t \hat{A}_t|$.

Thus, LateSyn does not reduce the physical feedback delay itself; it reduces the amount of synthesis signal exposed to that delay. With the population-optimal reliability coefficient in Eq. equation 8, the expected squared residual is no larger than delaying the full advantage. Full proofs and regularity conditions are provided in Appendix B.

4 Experiments

4.1 RQ1: Does learning-grounded synthesis improve downstream learner performance?

Table 1 compares LateSyn with the base model, published inference-time skill augmentation methods, and a controlled set of synthetic-data learning paradigms. Among the controlled synthesis baselines, LateSyn achieves the strongest downstream performance across all three environments, reaching 45.7 F1 on Search-QA, 34.4% success on ALFWorld, and 72.1 on WebShop. Compared with the strongest alternative synthesis strategy on each task, this corresponds to gains of 6.5, 9.4, and 7.4 points, respectively. More importantly, relative to naive asynchronous realized-LU optimization, LateSyn improves performance by 16.3 points on Search-QA, 9.4 points on ALFWorld, and 10.6 points on WebShop. These results suggest that the benefit does not come merely from introducing synthetic data or realized learning utility alone; effectively optimizing the synthesizer under delayed learner feedback is critical for translating learning-grounded supervision into stronger downstream learners. We additionally report MASA variants as published inference-time references; these results are not part of the controlled synthesis comparison, and the Search-QA values marked with † use success rate rather than F1.

4.2 RQ2: When does delayed learning feedback become harmful?

We next isolate the effect of feedback delay by varying only the maximum number of outstanding synthesis batches while keeping the learner update rule, realized-LU objective, and training bud-

Table 1: Main downstream performance across different data-synthesis and adaptation paradigms. All our learning-guided synthesis experiments use Qwen3-4B as the task learner. MASA results are reported from the original paper and included as inference-time references. Results marked with † report Search-QA success rate rather than F1 and are therefore not directly comparable in that column.

Method	**Search-QA F1 ↑**	**ALFWorld Success ↑**	**WebShop Score ↑**
Base Model			
Qwen3-4B	37.1	0.0	2.7
Published Inference-Time Reference			
MASA (No Skill)	32.9†	17.1	42.2
MASA (Base Skill)	35.5†	20.0	34.8
MASA (DS-Adapter)	36.2†	27.1	25.7
MASA	36.9†	31.4	49.1
Learning-Guided Synthesis			
Proxy Co-training	37.8	23.6	43.5
Static Synthesis	39.2	12.5	36.4
Synchronous Realized-LU	34.3	18.8	64.7
Naive Async-LU ($M = 8$)	29.4	25.0	61.5
LateSyn	**45.7**	**34.4**	**72.1**

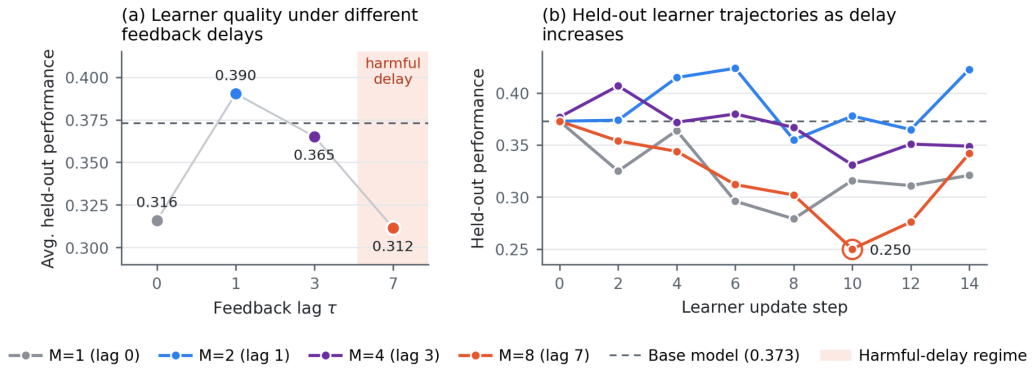

Figure 3: **Effect of delayed learner feedback on learning-grounded synthesis. (a)** Average held-out learner performance under different feedback lags. The effect of delay is non-monotonic: mild asynchrony is well tolerated, whereas sufficiently stale feedback ($\tau = 7$) enters a harmful-delay regime. **(b)** Held-out learner trajectories as feedback delay increases. Large delay causes sustained degradation and a pronounced mid-training collapse, while small and moderate delays maintain substantially healthier trajectories.

get fixed. Figure 3 reveals a non-monotonic relationship between feedback delay and downstream learner quality. Mild asynchrony is well tolerated: a one-step delay ($\tau = 1$) achieves the strongest average held-out performance of 0.390, and a moderate delay ($\tau = 3$) remains competitive at 0.365, both exceeding the synchronous setting ($\tau = 0$, 0.316). In contrast, sufficiently stale feedback enters a clear **harmful-delay regime**. At $\tau = 7$, the average held-out performance drops to 0.312, and the learner trajectory deteriorates substantially below the base-model level, reaching a mid-training low of 0.250. These results show that delay is not inherently detrimental; rather, synthesis optimization becomes unstable once learning feedback is sufficiently stale. This distinction motivates LateSyn to preserve the benefits of asynchronous synthesis while mitigating the harmful effects of delayed realized utility.

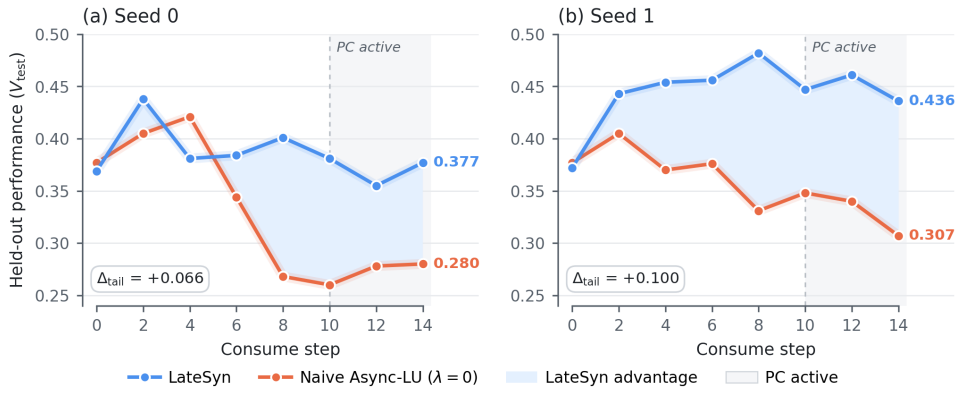

Figure 4: **LateSyn mitigates harmful delayed feedback under high asynchrony.** Held-out learner performance for two matched seeds at $M = 8$ (feedback lag $\tau = 7$). The naive asynchronous realized-LU baseline follows the same code path as LATESYN but fixes $\lambda = 0$, disabling the early predictive update. Across both seeds, LATESYN maintains substantially healthier learner trajectories, achieving tail-performance gains of **+6.6** and **+10.0** points. The performance gap further widens after Predict-Correct becomes active.

4.3 RQ3: CAN LATESYN RECOVER LEARNER QUALITY UNDER HARMFUL FEEDBACK DELAY?

Having identified a harmful high-delay regime in RQ2, we next evaluate whether LATESYN's Predict-Early, Correct-Late optimization can mitigate the resulting degradation. Figure 4 compares LATESYN against a matched naive asynchronous realized-LU baseline at $M = 8$ ($\tau = 7$). The two methods follow the same training and execution path, except that the control fixes $\lambda = 0$, disabling the early predictive update while retaining the delayed realized-LU correction. Across both paired seeds, LATESYN maintains substantially healthier learner trajectories as training proceeds. Its average post-training performance improves over the control by **+6.6** and **+10.0** points for seeds 0 and 1, respectively, while the final-checkpoint gains reach **+9.7** and **+12.9** points. Moreover, the separation becomes larger after Predict-Correct becomes active, indicating that the improvement is concentrated in the regime where delayed feedback begins to affect optimization. Together, these results show that LATESYN can prevent the high-lag degradation observed in naive asynchronous learning and preserve a substantially stronger downstream learner under stale realized-utility feedback.

4.4 RQ4: DOES LEARNING-UTILITY MEMORY ENABLE RELIABLE FUTURE-UTILITY PREDICTION ACROSS DOMAINS?

We finally examine whether the future learning utility required by Predict-Early can be estimated reliably beyond a single environment. Table 2 reports prequential out-of-sample prediction quality after warm-starting the utility critic with previously resolved learning outcomes. Across SearchQA, ALFWorld, and WebShop, the warm-started critic consistently exhibits positive correlation with subsequently realized learner utility, reaching Pearson correlations of 0.347, 0.283, and 0.212, respectively, with sign accuracies up to 0.917. More importantly, the delayed residual ratio falls to 0.998, 0.878, and 0.804 across the three environments, indicating that the predicted component removes a non-trivial fraction of the synthesis signal from the delayed correction path. The positive alignment between early and realized advantages further confirms that the early update points in a useful direction. These results support the central role of **Learning-Utility Memory**: accumulated learner outcomes provide reusable experience for anticipating future learning utility, allowing LateSyn to expose less of the synthesis update to delayed feedback across different agentic domains.

4.5 RQ5: DOES LATESYN REDUCE EXPOSURE TO DELAYED FEEDBACK?

We directly measure the fraction of the realized synthesis advantage that remains in the delayed correction, $\mathcal{R}_{\text{delay}} = \text{RMS}(A^{\text{late}})/\text{RMS}(A^{\text{real}})$, where 1 corresponds to naïve asynchronous optimization that delays the full advantage. As shown in Appendix D Table 3, LateSyn substantially

Table 2: Cross-domain effectiveness of Learning-Utility Memory. We report prequential out-of-sample prediction quality after warm-starting the utility critic with previously resolved learning outcomes.

Domain	Pearson ↑	Sign Acc. ↑	Residual Ratio ↓	Early–Real Cos. ↑
Search-QA	+0.347	0.688	0.998	+0.36–0.53
ALFWorld	+0.283	0.833	0.878	+0.497
WebShop	+0.212	**0.917**	**0.804**	**+0.603**

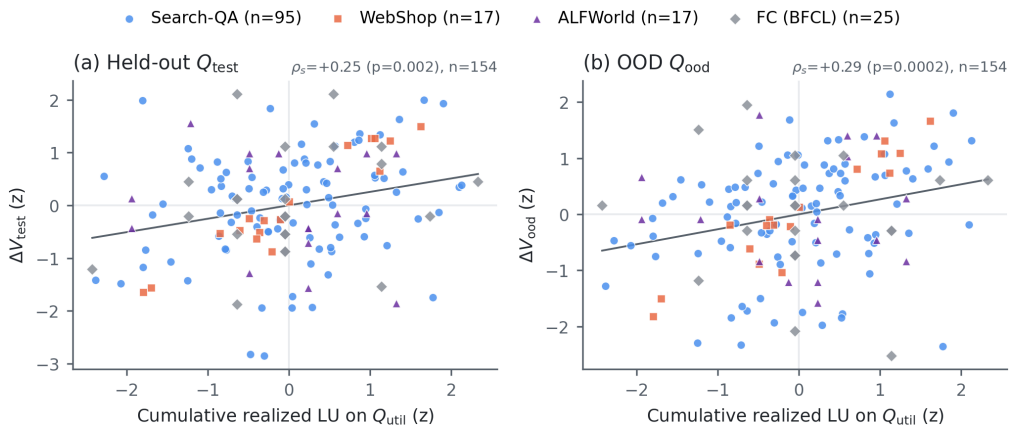

Figure 5: **Realized LU tracks held-out learner improvement.** We align cumulative realized LU on $\mathcal{Q}_{\text{util}}$ with the corresponding changes on held-out $\mathcal{Q}_{\text{test}}$ (left) and OOD $\mathcal{Q}_{\text{ood}}$ (right). Values are standardized within each domain before pooling to account for differences in metric scale. Higher realized LU is positively associated with improvement on both evaluation sets.

reduces the delayed signal on ALFWorld and WebShop, with residual ratios of 0.552 and 0.464, corresponding to 44.8% and 53.6% reductions, respectively. On Search-QA, the ratio remains near the naïve baseline (0.994), indicating that the reliability-adaptive mechanism conservatively moves little credit early when the predicted utility is less informative. These results support the mechanism predicted by Theorem 1: LateSyn does not shorten the physical feedback delay, but can reduce the amount of synthesis signal exposed to it.

4.6 RQ6: Does realized LU reflect held-out learning?

Figure 5 examines whether realized LU measured on $\mathcal{Q}_{\text{util}}$ reflects learning beyond the utility probe. Pooling within-domain standardized intervals, realized LU is positively associated with both held-out ($\rho = 0.25$, $p = 0.002$) and OOD improvement ($\rho = 0.29$, $p < 0.001$). The relationship also extends beyond the utility distribution: on Search-QA, realized LU remains significantly associated with OOD improvement ($\rho = 0.33$, $p = 0.001$). These results suggest that realized LU captures broader learning progress rather than merely probe-specific gains, while remaining a noisy and imperfect learning signal.

5 Conclusion

In this paper, we presented **LateSyn**, a learning-grounded framework for synthetic-data optimization under delayed learner feedback. By defining data value through **Realized Learning Utility**, LateSyn treats synthesis as a late-feedback problem and addresses it with a **Predict-Early, Correct-Late** strategy that applies predictable utility early and corrects the remaining residual once the learner outcome is observed. Across diverse agentic learning settings and feedback regimes, LateSyn improves learner performance and stabilizes optimization when feedback delay becomes harmful. More broadly, our results motivate evaluating synthetic data by the learning it actually causes, rather than only by properties observed before training.

AI USE STATEMENT

Generative AI tools were used in this work in two capacities. (i) As components of the proposed method and experiments: this work studies learning-grounded synthetic data generation with LLM-based synthesizers and downstream learners, and large language models are therefore integral to the experimental setup. In our main experiments, Qwen3-4B is used as both the trainable synthesizer and the downstream task learner, instantiated as separate model copies and optimized independently. The resulting agents generate synthetic training data, interact with retrieval or benchmark environments, and are evaluated in Search-QA, ALFWorld, and WebShop. The complete model, training, and evaluation configurations are described in the experimental setup and appendix.

(ii) For research and manuscript assistance: generative AI tools were used to assist with code implementation and debugging, figure preparation, language polishing, and manuscript editing. All research questions, methodological decisions, experimental protocols, and scientific conclusions were determined by the authors. AI-assisted code and analyses were inspected against the underlying experiment logs and outputs, and AI-assisted text and figures were reviewed and revised by the authors for correctness and consistency with the reported results. Generative AI tools were not used to fabricate or alter empirical measurements. We take full responsibility for the final content of this work, including all text, claims, code, analyses, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies synthetic data generation and LLM-agent training in benchmarked interactive environments. It does not involve human subjects, private user data, or deployment to real users. Experiments are conducted using publicly available datasets, benchmark environments, and simulated or programmatic interactions. As with other synthetic-data methods, generated content may inherit biases or inaccuracies from the underlying language models; our experiments are intended to study learning and optimization behavior rather than to establish the safety of generated data for unrestricted real-world deployment.

REPRODUCIBILITY STATEMENT

We provide the complete formulation of LateSyn in the main paper and additional implementation details in the appendix, including model configurations, learner and synthesizer optimization, asynchronous execution, utility evaluation, training hyperparameters, data splits, and evaluation protocols. Our experiments use fixed benchmark splits and explicitly define the utility, held-out, and auxiliary evaluation sets. We additionally describe the construction of the delayed-feedback controls and the Predict-Early, Correct-Late implementation to facilitate faithful reproduction of the reported comparisons.

REFERENCES

Maosong Cao, Taolin Zhang, Mo Li, Chuyu Zhang, Yunxin Liu, Haodong Duan, Songyang Zhang, and Kai Chen. Condor: Enhance llm alignment with knowledge-driven data synthesis and refinement. *arXiv preprint arXiv:2501.12273*, 2(4), 2025.

Jiaao Chen, Jingyuan Qi, Mingye Gao, Wei-Chen Wang, Hanrui Wang, and Di Jin. Eigendata: A self-evolving multi-agent platform for function-calling data synthesis, auditing, and repair. *arXiv preprint arXiv:2603.05553*, 2026a.

Yiqun Chen, Wei Yang, Erhan Zhang, Shijie Wang, Qi Liu, Zechun Niu, Bin Zhang, Haitao Li, Rui Li, Lingyong Yan, et al. Unitymas-o: A general rl optimization framework for llm-based multi-agent systems. *arXiv preprint arXiv:2605.26646*, 2026b.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, et al. Scaling agent learning via experience synthesis. *arXiv preprint arXiv:2511.03773*, 2025.

Yuqin Dai, Ning Gao, Wei Zhang, Jie Wang, Zichen Luo, Jinpeng Wang, Yujie Wang, Ruiyuan Wu, and Chaozheng Wang. Sead: Self-evolving agent for multi-turn service dialogue. *arXiv preprint arXiv:2602.03548*, 2026.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, volume 2024, pp. 48362–48395, 2024.

Junliang He, Ziyue Fan, Shaohui Kuang, Li Xiaoqing, Kai Song, Yaqian Zhou, and Xipeng Qiu. Fine: Filtering and improving noisy data elaborately with large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8686–8707, 2025.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

Wai-Chung Kwan, Aryo Pradipta Gema, Joshua Ong Jun Leang, and Pasquale Minervini. Scope: Self-play via co-evolving policies for open-ended tasks. *arXiv preprint arXiv:2605.31433*, 2026.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. Self-alignment with instruction backtranslation. In *The twelfth international conference on learning representations*, 2023.

Zhongzhi Li, Xuansheng Wu, Yijiang Li, Lijie Hu, and Ninghao Liu. Less is enough: synthesizing diverse data in feature space of llms. *arXiv e-prints*, pp. arXiv–2602, 2026.

Xiao Liang, Zhong-Zhi Li, Yeyun Gong, Yang Wang, Hengyuan Zhang, Ying Nian Wu, Weizhu Chen, et al. Sws: Self-aware weakness-driven problem synthesis in reinforcement learning for llm reasoning. *Advances in Neural Information Processing Systems*, 38:56801–56839, 2026.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, et al. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. In *International Conference on Learning Representations*, volume 2025, pp. 49573–49609, 2025.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6300–6323, 2025.

Ram Ramrakhya, Andrew Szot, Omar Attia, Yuhao Yang, Anh Nguyen, Bogdan Mazoure, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. *arXiv preprint arXiv:2509.25047*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025.

Shuo Tang, Xianghe Pang, Zexi Liu, Bohan Tang, Rui Ye, Tian Jin, Xiaowen Dong, Yanfeng Wang, and Siheng Chen. Synthesizing post-training data for llms through multi-agent simulation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23306–23335, 2025.

Zeao Tu, Xiangdi Meng, Yu He, Zihan Yao, Tianyu Qi, Jun Liu, and Ming Li. Resofilter: Fine-grained synthetic data filtering for large language models through data-parameter resonance analysis. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5414–5428, 2025.

Rui Wang, Ce Zhang, Jun-Yu Ma, Jianshu Zhang, Hongru Wang, Yi Chen, Boyang Xue, Tianqing Fang, Zhisong Zhang, Hongming Zhang, et al. Webaggregator: Enhancing compositional reasoning capabilities of deep research agent foundation models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24486–24517, 2026a.

Shaobo Wang, Zhengbo Jiao, Zifan Zhang, Yilang Peng, Xu Ze, Boyu Yang, Wei Wang, Hu Wei, and Linfeng Zhang. Socratic-zero: Bootstrapping reasoning via data-free agent co-evolution. *arXiv preprint arXiv:2509.24726*, 2025.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Pei Chen, Ziwei Dong, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Qun Liu, et al. Trajectory2task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44021–44044, 2026b.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, volume 2024, pp. 30745–30766, 2024.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Coevolve: Training llm agents via agent-data mutual evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23015–23036, 2026.

Jianxiang Yu, Jiapeng Zhu, Bochen Lin, Qier Cui, Zichen Ding, and Xiang Li. Skill is not one-size-fits-all: Model-aware skill alignment for llm agents. *arXiv preprint arXiv:2605.30723*, 2026.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. In *International Conference on Learning Representations*, volume 2024, pp. 40320–40341, 2024.

Zhenrui Yue, Kartikeya Upasani, Xianjun Yang, Suyu Ge, Shaoliang Nie, Yuning Mao, Zhe Liu, and Dong Wang. Dr. zero: Self-evolving search agents without training data. *arXiv preprint arXiv:2601.07055*, 2026.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

Shuo Zhang, Chaofa Yuan, Ryan Guo, Xiaomin Yu, Rui Xu, Zhangquan Chen, Zinuo Li, Zhi Yang, Shuhao Guan, Zhenheng Tang, et al. Evofsm: Controllable self-evolution for deep research with finite state machines. *arXiv preprint arXiv:2601.09465*, 2026.

APPENDIX

A METHOD

A.1 UTILITY CRITIC AND RELIABILITY ESTIMATION DETAILS

Online reliability estimation. To approximate Eq. equation 8, we maintain exponential moving averages over previously resolved prediction–outcome pairs:

$$m_t^{\text{cross}} = \beta m_{t-1}^{\text{cross}} + (1-\beta) A_i \hat{A}_i, \tag{12}$$

$$m_t^{\text{pred}} = \beta m_{t-1}^{\text{pred}} + (1-\beta) \hat{A}_i^2, \tag{13}$$

and compute

$$\lambda_t = \text{clip}\left(\frac{m_t^{\text{cross}}}{m_t^{\text{pred}} + \epsilon}, 0, 1 \right). \tag{14}$$

Only resolved samples contribute to these statistics.

Generation-time context. We construct

$$x_t = [e(D_t); z_t; q_t],$$

where $e(D_t)$ summarizes the synthesized batch, z_t captures lightweight behavioral statistics of the current learner, and q_t describes the asynchronous state. All features are available at generation time and require no additional learner evaluation. The critic directly estimates

$$C_\psi(x_t) \approx \mathbb{E}\left[r_t^{\text{LU}} \mid x_t \right],$$

thereby marginalizing over the learner evolution between generation and consumption rather than explicitly predicting $\theta_{c(t)}$.

Frozen generation-time quantities. For each synthesized batch, we store

$$(\hat{r}_t, b_t, \hat{A}_t, \lambda_t)$$

together with its synthesis trajectory. The delayed residual is always computed from these frozen generation-time values rather than from a subsequently updated critic or reliability estimate.

A.2 ASYNCHRONOUS RUNTIME AND DELAYED-FEEDBACK BOOKKEEPING

For each synthesized batch, LateSyn stores the behavior log-probability $\log \mu_t(D_t \mid x_t)$ together with the frozen generation-time Predict-Correct quantities. We maintain a bounded pending queue

$$\mathcal{Q}_t = \{\mathcal{C}_i\}, \qquad |\mathcal{Q}_t| \le M - 1, \tag{15}$$

where each pending chunk contains

$$\mathcal{C}_i = \left(D_i,\, x_i,\, \log \mu_i,\, \hat{r}_i,\, b_i,\, \hat{A}_i,\, \lambda_i,\, A_i^{\text{early}} \right). \tag{16}$$

When D_t is consumed, its delayed residual is computed exclusively from the stored generation-time quantities:

$$A_t^{\text{late}} = \left(r_t^{\text{LU}} - b_t \right) - \lambda_t \left(\hat{r}_t - b_t \right). \tag{17}$$

Hence,

$$A_t^{\text{early}} + A_t^{\text{late}} = A_t \tag{18}$$

for every resolved synthesis decision, even though the learner, critic, and synthesizer may all have evolved in the meantime.

Operationally, the synthesizer acts as a producer that generates batches, applies early updates, and places them into the pending queue. The learner acts as a consumer that trains on pending batches, evaluates their realized learning utility, and triggers the corresponding delayed residual correction. The bounded queue controls the maximum amount of outstanding synthesis work and therefore the degree of asynchrony.

B THEORETICAL ANALYSIS

This section provides the theoretical details underlying Sec. 3.5. We first distinguish the exact decomposition of synthesis credit from gradient updates evaluated at different policy states, and then establish the optimal generation-time predictor, reliability-adaptive mixing rule, and the resulting reduction in delay exposure.

B.1 SETUP

For a synthesis decision D_t generated under the behavior policy $\mu_t \equiv \pi_{\phi_t}$, define the generation-time score

$$s_t \triangleq \nabla_\phi \log \pi_{\phi_t}(D_t \mid x_t). \tag{19}$$

When its realized learning utility becomes available at consumption time $c(t)$, the synthesizer may have evolved to $\pi_{\phi_{c(t)}}$. We write

$$s_{c(t)} \triangleq \nabla_\phi \log \pi_{\phi_{c(t)}}(D_t \mid x_t), \qquad \rho_t \triangleq \frac{\pi_{\phi_{c(t)}}(D_t \mid x_t)}{\pi_{\phi_t}(D_t \mid x_t)}. \tag{20}$$

Recall that

$$A_t = r_t^{\mathrm{LU}} - b_t, \qquad A_t^{\mathrm{early}} = \lambda_t \hat{A}_t, \qquad A_t^{\mathrm{late}} = A_t - \lambda_t \hat{A}_t. \tag{21}$$

All generation-time quantities $(b_t, \hat{A}_t, \lambda_t)$ are treated as frozen when the delayed feedback is resolved.

For the policy-gradient interpretation, we assume that b_t is a generation-time control variate independent of the sampled synthesis action conditional on its synthesis context. More generally, the results below remain valid algebraically if A_t is interpreted simply as the scalar weight used by the synthesis optimizer.

B.2 EXACT CREDIT DECOMPOSITION

We first make explicit what is, and is not, exactly recovered by Predict-Correct.

Proposition 1 (Exact Predict-Correct Credit Decomposition)**.** *For any $\hat{A}_t$ and any $\lambda_t \in [0, 1]$,*

$$A_t^{\mathrm{early}} + A_t^{\mathrm{late}} = A_t. \tag{22}$$

Proof. By definition,

$$A_t^{\mathrm{early}} + A_t^{\mathrm{late}} = \lambda_t \hat{A}_t + \left(A_t - \lambda_t \hat{A}_t\right) = A_t.$$

□

Proposition 1 is an exact decomposition of the *scalar synthesis credit*. It does not by itself imply exact equality between the actual parameter updates, because the early and late components are generally evaluated at different synthesizer parameters. If both components were evaluated using the same generation-time score s_t, then

$$A_t^{\mathrm{early}} s_t + A_t^{\mathrm{late}} s_t = A_t s_t.$$

Under asynchronous execution, however, the delayed term uses an evolved policy, and the discrepancy induced by this policy drift is analyzed below.

B.3 OPTIMAL GENERATION-TIME UTILITY PREDICTION

Let X_t denote all information available to the utility critic at generation time. The critic seeks to predict future realized learning utility before the learner outcome is known.

Proposition 2 (Optimal Generation-Time Utility Predictor)**.** *Among all square-integrable predictors measurable with respect to X_t, the predictor*

$$C^*(X_t) = \mathbb{E}\left[r_t^{\mathrm{LU}} \mid X_t\right] \tag{23}$$

minimizes

$$\mathbb{E}\left[\left(r_t^{\mathrm{LU}} - C(X_t)\right)^2\right].$$

If b_t is measurable with respect to X_t, the corresponding optimal advantage predictor is

$$\hat{A}_t^* = \mathbb{E}\left[A_t \mid X_t\right]. \tag{24}$$

Proof. Let $m(X_t) = \mathbb{E}[r_t^{\mathrm{LU}} \mid X_t]$. For any measurable predictor $C(X_t)$,

$$\begin{aligned}&\mathbb{E}\left[\left(r_t^{\mathrm{LU}} - C(X_t)\right)^2\right]\\ &= \mathbb{E}\left[\left(r_t^{\mathrm{LU}} - m(X_t)\right)^2\right] + \mathbb{E}\left[\left(m(X_t) - C(X_t)\right)^2\right],\end{aligned} \tag{25}$$

because the cross term vanishes by the defining property of conditional expectation. The second term is nonnegative and is minimized by $C(X_t) = m(X_t)$.

If b_t is measurable with respect to X_t, then

$$\mathbb{E}[A_t \mid X_t] = \mathbb{E}[r_t^{\mathrm{LU}} \mid X_t] - b_t,$$

which gives Eq. equation 24. □

The residual

$$\epsilon_t = A_t - \mathbb{E}[A_t \mid X_t] \tag{26}$$

is orthogonal in L_2 to every square-integrable function of X_t. It therefore represents the component of realized learning utility that cannot be predicted from the available generation-time information.

A richer generation-time context cannot increase the Bayes prediction error. Specifically, if X_t' contains all information in X_t, then

$$\min_{C'} \mathbb{E}\left[\left(r_t^{\mathrm{LU}} - C'(X_t')\right)^2\right] \le \min_{C} \mathbb{E}\left[\left(r_t^{\mathrm{LU}} - C(X_t)\right)^2\right], \tag{27}$$

since every predictor measurable with respect to X_t is also a valid candidate under X_t'.

B.4 Optimal Reliability-Adaptive Mixing

Given an imperfect predicted advantage $\hat{A}$, LateSyn determines how much of it should be used before the realized outcome arrives.

Proposition 3 (Optimal Reliability-Adaptive Mixing)**.** *For a fixed predictor $\hat{A}$ with finite second moment and $\mathbb{E}[\hat{A}^2] > 0$, the solution to*

$$\lambda^* = \arg\min_{\lambda \in [0,1]} \mathbb{E}\left[(A - \lambda \hat{A})^2\right] \tag{28}$$

is

$$\lambda^* = \mathrm{clip}\left(\frac{\mathbb{E}[A\hat{A}]}{\mathbb{E}[\hat{A}^2]}, 0, 1\right). \tag{29}$$

If $\mathbb{E}[\hat{A}^2] = 0$, we define $\lambda^* = 0$. Moreover,

$$\mathbb{E}\left[\left(A - \lambda^* \hat{A}\right)^2\right] \le \mathbb{E}[A^2]. \tag{30}$$

Proof. Expanding the objective gives

$$J(\lambda) = \mathbb{E}[A^2] - 2\lambda \mathbb{E}[A\hat{A}] + \lambda^2 \mathbb{E}[\hat{A}^2]. \tag{31}$$

For $\mathbb{E}[\hat{A}^2] > 0$, the unconstrained minimizer satisfies

$$J'(\lambda) = -2\mathbb{E}[A\hat{A}] + 2\lambda\mathbb{E}[\hat{A}^2] = 0,$$

which yields

$$\lambda_{\text{unc}} = \frac{\mathbb{E}[A\hat{A}]}{\mathbb{E}[\hat{A}^2]}.$$

Projecting this value onto the feasible interval $[0, 1]$ gives Eq. equation 29. Finally, $\lambda = 0$ is always feasible, so optimality immediately gives Eq. equation 30. □

We define the delayed residual ratio

$$\mathcal{R}_{\text{delay}}(\lambda) \triangleq \sqrt{\frac{\mathbb{E}\left[(A - \lambda\hat{A})^2\right]}{\mathbb{E}[A^2]}}. \tag{32}$$

For the population-optimal coefficient,

$$\mathcal{R}_{\text{delay}}(\lambda^*) \leq 1. \tag{33}$$

This is a population statement. The online coefficient used by LateSyn is estimated from finitely many previously resolved outcomes, and its empirical residual ratio need not be below one at every point during training.

Bayes-optimal predictor. If $\hat{A} = \mathbb{E}[A \mid X]$, then the orthogonality property of conditional expectation gives

$$\mathbb{E}[A\hat{A}] = \mathbb{E}[\hat{A}^2].$$

Hence, whenever $\mathbb{E}[\hat{A}^2] > 0$, the population-optimal coefficient satisfies $\lambda^* = 1$. Intuitively, an exact conditional-mean predictor should expose all predictable utility early, leaving only the unpredictable innovation for delayed correction. The adaptive coefficient is needed because the learned critic is generally imperfect and may undergo distribution shift.

B.5 Residual-Controlled Delay Exposure

We now analyze the effect of policy drift on the delayed synthesis update.

Define the fresh realized-utility reference direction

$$g_t^* = A_t s_t. \tag{34}$$

For theoretical analysis, we consider the unclipped importance-corrected delayed update. The naïve asynchronous direction is

$$g_t^{\text{naive}} = \rho_t A_t s_{c(t)}, \tag{35}$$

while Predict-Correct uses

$$g_t^{\text{PC}} = \lambda_t \hat{A}_t s_t + \rho_t \left(A_t - \lambda_t \hat{A}_t\right) s_{c(t)}. \tag{36}$$

Proposition 4 (Exact Delay-Error Factorization)**.** *The deviations from the fresh reference direction satisfy*

$$g_t^{\text{naive}} - g_t^* = A_t \left(\rho_t s_{c(t)} - s_t\right), \tag{37}$$

and

$$g_t^{\text{PC}} - g_t^* = \left(A_t - \lambda_t \hat{A}_t\right) \left(\rho_t s_{c(t)} - s_t\right). \tag{38}$$

Proof. For the naïve update,

$$\begin{aligned} g_t^{\text{naive}} - g_t^* &= \rho_t A_t s_{c(t)} - A_t s_t \\ &= A_t \left(\rho_t s_{c(t)} - s_t\right). \end{aligned}$$

For Predict-Correct,

$$\begin{aligned} g_t^{\text{PC}} - g_t^* &= \lambda_t \hat{A}_t s_t + \rho_t (A_t - \lambda_t \hat{A}_t) s_{c(t)} - A_t s_t \\ &= (A_t - \lambda_t \hat{A}_t) \left(\rho_t s_{c(t)} - s_t\right). \end{aligned}$$

□

This proposition directly addresses the distinction between credit and gradient decomposition. Predict-Correct does not claim that gradients evaluated at different policies add exactly to the generation-time gradient. Rather, it shows that the *error caused by evaluating the delayed component under an evolved policy* is multiplied only by the unresolved residual instead of the full realized advantage.

B.6 Bounding Policy-Drift Error

To relate the previous identity to feedback delay, we impose standard local regularity conditions.

[Score regularity] There exist constants $L_s, B_s > 0$ such that, in the local policy region visited between generation and feedback,

$$\|s_\phi(D,x) - s_{\phi'}(D,x)\| \le L_s\|\phi - \phi'\|, \tag{39}$$

$$\|s_\phi(D,x)\| \le B_s. \tag{40}$$

[Local ratio regularity] There exist constants $L_\rho > 0$ and $\bar{\rho} < \infty$ such that

$$|\rho(\phi) - 1| \le L_\rho\|\phi - \phi_t\|, \qquad \rho(\phi) \le \bar{\rho} \tag{41}$$

over the same local region, where $\rho(\phi) = \pi_\phi(D_t \mid x_t)/\pi_{\phi_t}(D_t \mid x_t)$.

[Bounded parameter drift] Each synthesizer update satisfies

$$\|\phi_{k+1} - \phi_k\| \le \eta G. \tag{42}$$

A delay of τ_t updates then implies

$$\|\phi_{c(t)} - \phi_t\| \le \tau_t \eta G. \tag{43}$$

[Effective delayed-score drift] Under Assumptions B.6–B.6,

$$\left\|\rho_t s_{c(t)} - s_t\right\| \le L_{\text{eff}}\eta G\tau_t, \tag{44}$$

where

$$L_{\text{eff}} \triangleq \bar{\rho}L_s + B_s L_\rho. \tag{45}$$

Proof. Using

$$\rho_t s_{c(t)} - s_t = \rho_t\left(s_{c(t)} - s_t\right) + (\rho_t - 1)s_t,$$

the triangle inequality gives

$$\begin{aligned}\|\rho_t s_{c(t)} - s_t\| &\le \rho_t\|s_{c(t)} - s_t\| + |\rho_t - 1|\|s_t\| \\ &\le \left(\bar{\rho}L_s + B_s L_\rho\right)\|\phi_{c(t)} - \phi_t\|.\end{aligned}$$

Applying Eq. equation 43 completes the proof. □

Theorem 2 (Residual-Controlled Delay Exposure)**.** *Under Assumptions B.6–B.6,*

$$\left\|g_t^{\text{naive}} - g_t^*\right\| \le L_{\text{eff}}\eta G\tau_t|A_t|, \tag{46}$$

whereas Predict-Correct satisfies

$$\boxed{\left\|g_t^{\text{PC}} - g_t^*\right\| \le L_{\text{eff}}\eta G\tau_t\left|A_t - \lambda_t\hat{A}_t\right|.} \tag{47}$$

Proof. Combining Proposition 4 with Lemma B.6 yields

$$\begin{aligned}\|g_t^{\text{naive}} - g_t^*\| &= |A_t|\|\rho_t s_{c(t)} - s_t\| \\ &\le L_{\text{eff}}\eta G\tau_t|A_t|,\end{aligned}$$

and similarly

$$\begin{aligned}\|g_t^{\text{PC}} - g_t^*\| &= |A_t - \lambda_t\hat{A}_t|\|\rho_t s_{c(t)} - s_t\| \\ &\le L_{\text{eff}}\eta G\tau_t|A_t - \lambda_t\hat{A}_t|.\end{aligned}$$

□

If the feedback delay is bounded by $\tau_t \le \bar{\tau}$, then Cauchy–Schwarz gives

$$\mathbb{E}\left[\|g_t^{\mathrm{PC}} - g_t^*\|\right] \le L_{\mathrm{eff}}\eta G\bar{\tau}\sqrt{\mathbb{E}\left[(A-\lambda\hat{A})^2\right]}. \tag{48}$$

The corresponding naïve upper bound is

$$L_{\mathrm{eff}}\eta G\bar{\tau}\sqrt{\mathbb{E}[A^2]}. \tag{49}$$

Thus, the ratio between these two *upper bounds* is exactly

$$\mathcal{R}_{\mathrm{delay}}(\lambda) = \sqrt{\frac{\mathbb{E}\left[(A-\lambda\hat{A})^2\right]}{\mathbb{E}[A^2]}}. \tag{50}$$

At the population-optimal λ^*,

$$\mathcal{R}_{\mathrm{delay}}(\lambda^*) \le 1.$$

Accordingly, reliability-adaptive prediction cannot worsen this population upper bound relative to delaying the full realized advantage, although finite-sample online estimates of λ_t do not guarantee this inequality at every training step.

B.7 Relation to the Clipped Practical Objective

The analysis above considers the unclipped importance-corrected update in order to isolate the effect of delayed policy drift. In practice, LateSyn uses the clipped surrogate in Eq. equation 10 to prevent excessively large off-policy updates.

When clipping is inactive, the practical delayed update coincides with the update analyzed above. When clipping is active, the surrogate truncates the delayed update, and the exact factorization in Proposition 4 no longer describes the clipped gradient branch exactly. We therefore treat clipping as an additional optimization-stability mechanism rather than as part of the theoretical reduction claim.

Finally, all results in this section are local, per-decision statements about the synthesis update direction. They do not claim that the entire parameter trajectory produced by asynchronous Predict-Correct training is identical to that of a synchronous optimizer: early updates change subsequent synthesizer states and therefore future generated data. The theoretical claim is narrower and more precise: for a resolved synthesis decision, the component of its update that remains exposed to delayed policy drift is weighted by $A_t - \lambda_t\hat{A}_t$ rather than by the full realized advantage A_t.

C Related Work

C.1 Synthetic Data for Post-Training and Self-Evolution

Synthetic supervision has become an important alternative to manually curated data for post-training large language models. Early approaches expand instruction-following corpora by generating new tasks or responses from seed examples, stronger models, or unlabeled text, as in Self-Instruct Wang et al. (2023), WizardLM Xu et al. (2024), and instruction backtranslation Li et al. (2023). More recent work improves synthetic datasets through richer generation procedures, multi-agent interaction, targeted data construction, filtering, and automated data refinement Tang et al. (2025); Li et al. (2026); Cao et al. (2025); He et al. (2025); Tu et al. (2025). Collectively, these methods demonstrate that model-generated supervision can substantially broaden the scale and coverage of post-training data. A common pattern, however, is to optimize the synthetic corpus before or separately from the learner update, so the generator receives limited information about the learning effect that its data ultimately produces.

Verifiable reasoning tasks have provided a particularly effective testbed for synthetic post-training. STaR Zelikman et al. (2022) iteratively bootstraps rationales from successful reasoning attempts, while WizardMath Luo et al. (2025), MetaMath Yu et al. (2023), MAmmoTH Yue et al. (2024), and ToRA Gou et al. (2024) enlarge mathematical and tool-assisted reasoning supervision through transformed problems, evolved instructions, and generated solution traces. At larger scale, DeepSeek-Math Shao et al. (2024) and NuminaMath Li et al. (2024) further show the effectiveness of combining synthetic or curated reasoning data with verifiable outcomes. Recent methods also make synthesis increasingly learner-aware. For example, SwS Liang et al. (2026) identifies persistent learner

weaknesses and generates additional training problems around the corresponding concepts. These approaches move beyond static data construction, but the signals guiding generation are typically based on correctness, difficulty, diagnosed weaknesses, or other proxies available without directly measuring the learner improvement caused by each synthesized batch.

A complementary direction places data generation and learning in a continually evolving loop. R-Zero Huang et al. (2025) jointly evolves a Challenger and Solver through generated problems and solution feedback, while Socratic-Zero Wang et al. (2025) coordinates Teacher, Solver, and Generator roles to build curricula from learner weaknesses. Agent0 Xia et al. (2025), Dr. Zero Yue et al. (2026), and SCOPE Kwan et al. (2026) extend related self-evolving ideas to tool use, search, and document-grounded reasoning. These systems establish that the learner can serve as an important source of feedback for subsequent synthesis. Our work focuses on a distinct optimization issue that arises when this feedback is grounded in the learner's *actual post-training improvement*: the synthesis reward is only realized after the learner has consumed the generated data, making learning-grounded synthesis inherently a delayed-feedback problem.

C.2 Feedback-Driven Agentic Data Synthesis

Interactive agent training imposes additional requirements on synthetic data beyond semantic plausibility. Generated tasks must be compatible with environment state, executable through valid action or tool sequences, consistent with domain constraints, and associated with outcomes that can be evaluated. Several recent methods address these requirements by grounding synthesis in environment interaction. Explorer Pahuja et al. (2025) obtains diverse web-agent trajectories through broad environment exploration; OS-Genesis Sun et al. (2025) constructs executable GUI tasks from successful interactions; AutoPlay Ramrakhya et al. (2025) explores application states before instantiating grounded tasks; and Trajectory2Task Wang et al. (2026b) converts valid tool trajectories into corresponding user tasks. WebAggregator Wang et al. (2026a) similarly synthesizes search tasks that require aggregating evidence from multiple sources. These methods substantially improve grounding and executability, but their feedback primarily determines whether an interaction is valid, feasible, or sufficiently diverse, rather than directly quantifying the learner improvement caused by training on it.

Another line of work emphasizes verification and refinement of synthetic tool-use data. EigenData Chen et al. (2026a) generates multi-turn tool-use conversations together with executable checkers, while APIGen, APIGen-MT, and TOUCAN-style pipelines rely on structured generation and validation to construct function-calling and tool-use supervision. DreamGym Chen et al. (2025) further scales synthetic interaction experience through a learned experience model. Such methods make verification an integral part of agentic data generation, helping reject malformed trajectories and improve the reliability of synthetic supervision. Yet validity and executability characterize whether an example is suitable for training; they do not necessarily reveal how much training on that example will improve the current learner.

Recent work has also begun to use learner or trajectory feedback to drive continual agent improvement. CoEvolve Yang et al. (2026) uses signals such as forgetting and uncertainty to guide subsequent tool-use task synthesis, while Agent-World Dong et al. (2026), SEAD Dai et al. (2026), AgentEvolver Zhai et al. (2025), EvolveR Wu et al. (2025), and EvoFSM Zhang et al. (2026) study continual evolution of environments, curricula, workflows, memories, or interaction experience. Together, these efforts motivate increasingly tight feedback loops between synthesis and learning. LateSyn complements this direction by asking what supervision should optimize the synthesizer once the objective is defined by the learner's *realized* improvement. Unlike generation-time proxy signals, realized learning utility is only observed after a downstream update and evaluation. This temporal separation creates a tension between faithful synchronous feedback and stale asynchronous optimization, which LateSyn addresses through its *Predict-Early, Correct-Late* formulation.

D Experiments

D.1 Experimental Details

Domains and Evaluation. We evaluate LateSyn across three agentic learning settings: multi-hop Search-QA, ALFWorld, and WebShop. Search-QA trains a ReAct-style retrieval agent that

may issue up to four search actions against a local Wikipedia retriever and is evaluated by answer-level F1. ALFWorld evaluates multi-turn embodied text interaction using binary task success, while WebShop evaluates web-shopping agents using the benchmark-native graded shopping score. For every domain, we construct disjoint utility, held-out test, and auxiliary evaluation splits. Realized Learning Utility is always computed on the utility split $\mathcal{Q}_{\text{util}}$, whereas $\mathcal{Q}_{\text{test}}$ and $\mathcal{Q}_{\text{ood}}$ are never exposed to the synthesizer and are used only for evaluation. All evaluations use greedy decoding.

Models and Infrastructure. All main experiments use **Qwen3-4B** as both the downstream learner and the trainable synthesizer. The two models are initialized from the same base checkpoint but maintained and optimized as independent parameter copies. We use the default Qwen3 chat template with thinking enabled. Inference is served with vLLM in bfloat16, while model updates are performed with full-parameter fine-tuning in float32 with gradient checkpointing.

For Search-QA, retrieval follows the Search-R1 setup using an `intfloat/e5-base-v2` encoder and an exact FAISS index over the `wiki-18` corpus, returning the top three passages for each search query.

Baselines and Published References. For the main comparison in Table 1, we consider several controlled synthetic-data learning baselines under the same Qwen3-4B learner, synthesis budget, and evaluation protocol. **Proxy Co-training** jointly updates the synthesizer using a generation-time teachability reward rather than realized learner improvement. **Static Synthesis** trains the learner on a fixed synthetic dataset without online adaptation of the synthesizer. **Synchronous Realized-LU** optimizes the synthesizer directly from realized learning utility but waits for each learner update and evaluation before generating the next batch. **Naïve Async-LU** removes this synchronization barrier and applies the full realized advantage after delayed feedback, corresponding to the $\lambda = 0$ variant of our asynchronous pipeline. LateSyn uses the same asynchronous execution setting but applies the proposed Predict-Early, Correct-Late optimization.

We additionally report **MASA** Yu et al. (2026) variants as published inference-time references on the corresponding agent benchmarks. These numbers are taken from the original MASA evaluation rather than reproduced within our training pipeline, and therefore are not treated as budget-matched synthesis baselines. In particular, the reported Search-QA MASA numbers use task success rather than F1 and are marked accordingly in Table 1.

Learner Optimization. Across all main domains, the learner is updated from synthetic batches using GRPO. For each synthetic item, we sample $G = 5$ learner rollouts at temperature 1.0, compute group-normalized advantages, and optimize the resulting rollout batch for $K = 2$ epochs. We use AdamW with learning rate 5×10^{-6}, PPO clipping coefficient 0.2, maximum gradient norm 1.0, and no KL penalty. Only learner-generated tokens contribute to the training loss; prompts, retrieved evidence, and environment observations are masked. After each learner update, we evaluate $\mathcal{Q}_{\text{util}}$ greedily and define

$$r_t^{\text{LU}} = V(\theta_{t+1}; \mathcal{Q}_{\text{util}}) - V(\theta_t; \mathcal{Q}_{\text{util}}).$$

The post-update utility score is cached and reused as the pre-update score of the next learner step.

Synthesizer Optimization. The synthesizer is trained with a score-function objective using an exponential-moving-average reward baseline. Unless otherwise specified, we use AdamW with learning rate 1×10^{-6}, two optimization epochs, PPO clipping coefficient 0.2, sampling temperature 1.0, and top-$p = 0.95$. Search-QA uses synthetic batches of 32 examples, while ALFWorld and WebShop use batches of 12. For realized-LU training, the synthesizer reward is the learner improvement defined above. For the proxy co-training baseline, we instead use teachability, measured as the mean absolute GRPO advantage of learner rollouts on the synthesized batch. All other training settings are matched between the corresponding comparison arms.

Asynchronous Execution and Predict-Correct. We control the degree of asynchrony through the maximum number of outstanding synthesis batches M. We consider $M \in \{1, 2, 4, 8\}$, where $M = 1$ corresponds to synchronous execution and the steady-state feedback lag is approximately $M - 1$ synthesizer updates. Our high-delay setting uses $M = 8$, corresponding to a lag of seven updates.

Table 3: Effective exposure to delayed feedback under LateSyn. $\mathcal{R}_{\text{delay}} = 1$ corresponds to naïve asynchronous optimization, where the full realized advantage is delayed.

Domain	$\mathcal{R}_{\text{delay}} \downarrow$	Delayed Signal Reduction $\uparrow$
Search-QA	0.994	0.6%
ALFWorld	0.552	44.8%
WebShop	**0.464**	**53.6%**

For Predict-Early, Correct-Late, behavior log-probabilities are stored when each synthetic batch is generated. At generation time, the early credit is $A_t^{\text{early}} = \lambda_t \hat{A}_t$; after the realized LU becomes available, the delayed correction is $A_t^{\text{late}} = A_t - A_t^{\text{early}}$. The reliability coefficient is estimated from previously resolved prediction–outcome pairs using an exponential moving average with coefficient 0.7, and remains zero until at least three resolved pairs are available. The future-utility critic is a ridge regressor with regularization coefficient 1.0 over standardized generation-time features. Our $\lambda = 0$ control follows the identical asynchronous code path but suppresses the early component, reducing to naïve delayed-LU optimization.

Evaluation Protocol. Held-out evaluation is performed every two learner updates unless otherwise specified, with the base model evaluated before the first update. We report the learner trajectory throughout training and use task-native metrics for each domain: F1 for Search-QA, success rate for ALFWorld, and the standard WebShop score for WebShop. For delayed-feedback analyses, all compared arms use matched learner and synthesizer optimization settings, synthesis budgets, and evaluation splits; only the feedback mechanism or delay is varied.

D.2 Representative Prompts

We provide representative prompt templates for the trainable synthesizer and downstream learner in our three main experimental domains: Search-QA, ALFWorld, and WebShop. Runtime-specific task instances, retrieved passages, environment observations, and admissible actions are injected into the marked slots. For readability, we omit implementation-only scaffolding, decoding limits, and deterministic validity checks while retaining the instructions that determine synthesis and learner behavior. All prompts are rendered through the Qwen3 chat template, with the default thinking mode enabled unless otherwise specified.

Search-QA

Search-QA Synthesizer Prompt

```
System

You are a question designer that creates challenging multi-hop
factual questions for training a search-augmented QA model.  A good
question:
-- requires reasoning over 2--3 distinct facts (multi-hop), not a
single lookup;

-- has ONE short, unambiguous, verifiable answer (an entity, name,
date, or number);

-- is answerable from a general encyclopedic knowledge base (not
the passage alone);

-- does NOT reveal the answer in the question, and does not say
``according to the passage''.

Anchor the question in the given seed passage (use its entity
as one hop), but require at least one additional hop of world
knowledge.
```

```
Respond with ONLY a JSON object:

{ ``question'': ``<the multi-hop question>'', ``answer'':
``<short answer>'', ``hops'': <int 2 or 3>, ``rationale'':
``<one-sentence reasoning chain>'' }

User

Seed passage (title: {title}):

{text}

Design one multi-hop question as specified. Output only the JSON
object.
```

Search-QA Learner Prompt

```
System

You are a careful research assistant that answers multi-hop
questions by searching a knowledge base. Follow this protocol
exactly:
1. Reason step by step inside <think> and </think>.

2. When you need external facts, issue ONE query inside <search>
and </search>. The system returns passages inside <information>
and </information>.

3. You may search multiple times. Base your reasoning only on
retrieved information and the question.

4. When you can answer, output the final short answer inside
<answer> and </answer> (e.g., <answer> Beijing </answer>) with no
extra text.

User

Question: {question}

Environment Response

<information>
Doc 1: {passage_1}
Doc 2: {passage_2}
Doc 3: {passage_3}
</information>
```

ALFWorld

ALFWorld Synthesizer Prompt

```
System

You are a planner that writes SHORT strategic hints for an agent
solving household tasks in a text world. Given the task and
the objects/commands available, write 1--3 concise sentences of
guidance: which objects to focus on and the rough order of actions
(find -> take -> go to -> place/heat/cool/clean as needed). Do NOT
output a full command list; write natural-language guidance only.
```

User

```
Task: {task_desc}

Objects in scene: {obj_1, obj_2, ...}

Example admissible commands:
-- {cmd_1}
-- {cmd_2}
...

Write the guidance now (natural language, no <action> tags).
```

ALFWorld Learner Prompt

System

```
You are an agent solving a household task in a text world. Each
turn you receive the current observation and the list of admissible
commands. Think briefly, then choose EXACTLY ONE command from the
admissible list and output it wrapped in <action>...</action>. Use
commands verbatim.
```

User

```
Task: {task_desc}

Hint from your planner: {guidance}

<obs>
{observation}

Admissible commands:
-- {cmd_1}
-- {cmd_2}
...
</obs>

Choose your next command.
```

WebShop

WebShop Synthesizer Prompt

System

```
You are a shopping strategist that writes SHORT hints for an
agent buying products on a text-based e-commerce site. Given the
shopping instruction, write 1--3 concise sentences of guidance:
which keywords to search, which attributes/options to check before
buying, and to click Buy Now only after all requested attributes
are satisfied. Do NOT output actions or <action> tags; write
natural-language guidance only.
```

User

```
Shopping instruction: {instruction}
```

```
Write the guidance now (natural language, no <action> tags).
```

WebShop Learner Prompt

System

```
You are a shopping agent on a text-based e-commerce site.  Each
turn you receive the page observation and the list of clickable
items.  Think briefly, then take EXACTLY ONE action wrapped in
<action>...</action>.  Valid actions are:
search[keywords]     (only on the search page)

click[button or item text]     (must match one of the listed
clickables)

Find a product matching ALL the requested attributes/options,
select the right options, then click[Buy Now].  Use clickable text
verbatim.
```

User

```
Shopping instruction:  {instruction}

Hint from your planner:  {guidance}

<obs>
{page_observation}

search[...]  available

Clickables:
-- {clickable_1}
-- {clickable_2}
...
</obs>

Choose your next action.
```