LAGA: A Self-Contained Layout-Aware Generative Architecture for Knowledge-Preserving Data Generation

Author Name

author.email@example.com

Department of Computer Science, University Name, City, Country

Abstract. Synthetic data generation is increasingly used to overcome data scarcity, privacy constraints, and benchmark fragility. However, many contemporary generators depend on external foundation models, domain-specific feature extractors, or GPU-intensive training pipelines, making them difficult to deploy as independent data systems. This paper introduces *LAGA* (*Layout-Aware Generative Architecture*), a self-contained data generator framework designed to operate without external neural services. LAGA treats "layout" as an intrinsic structural representation extracted from the input data itself rather than as a computer-vision segmentation mask supplied by an outside model. For transaction databases, text corpora, integer sequences, logs, and tabular records, LAGA constructs compact structural layouts using embedded tokenization, itemset statistics, transition matrices, closure constraints, utility distributions, and frequent-pattern summaries. A stochastic generator then produces synthetic samples by sampling from learned Markov, Zipfian, grammar, and constraint-aware distributions while preserving the mined knowledge structure of the original dataset. An internal discriminator is implemented not as a neural network but as a data-mining evaluator, using FP-Growth, Eclat, sequential-pattern mining, high-utility itemset mining, entropy tests, support-distance metrics, and privacy checks to reject or repair weak samples. The result is a closed-loop generator–miner system that can ingest raw data, extract its structure, generate privacy-aware variants, and validate synthetic quality inside a single runtime. We formalize the framework, define knowledge-preservation and anonymization objectives, present algorithms for training-free and adaptive generation, analyze complexity, and propose an evaluation protocol covering utility, pattern fidelity, privacy leakage, distributional similarity, and downstream robustness. LAGA reframes generative augmentation as an autonomous data-mining system rather than a dependent image-editing pipeline.

Keywords: Synthetic data generation; Data mining; Pattern-preserving generation; Frequent itemset mining; Sequential pattern mining; Privacy-preserving data synthesis; Self-contained systems

1 Introduction

Data-driven systems depend on representative datasets, yet real datasets are often scarce, private, imbalanced, noisy, or expensive to collect. Synthetic data generation addresses these limitations by creating additional records that resemble the original data while reducing direct exposure of sensitive samples. Existing methods span simple perturbation rules, probabilistic models, generative adversarial networks, variational autoencoders, diffusion models, simulation engines, and foundation-model-based data synthesis (Goodfellow et al., 2014; Kingma and Welling, 2013; Ho et al., 2020; Jordon et al., 2018; Xu et al., 2019). Although these methods can be powerful, they frequently rely on heavy external dependencies: pretrained vision models, large language models, Python runtimes, GPUs, cloud APIs, or task-specific neural encoders. This dependence weakens reproducibility and deployability in constrained environments.

This paper proposes a different direction. We define **LAGA** as a *self-contained layout-aware generative architecture*: a data generator that learns structural layouts directly from the input dataset and uses internal data-mining algorithms as both knowledge extractors and quality evaluators. In earlier formulations, "layout-aware generation" was often associated with image augmentation, where a semantic segmentation network extracts the spatial arrangement of objects before a generator modifies color, texture, or lighting. That formulation is useful for computer vision but is not self-contained: the structure extractor may require a large model such as a detector or segmenter, and the generator may require a separate neural image-synthesis pipeline. LAGA generalizes the concept of layout beyond images. For text, a layout can be a token sequence, phrase graph, or n-gram transition skeleton. For transaction data, it can be a frequent itemset lattice, closed-pattern set, utility distribution, or transaction-length profile. For logs, it can be an event-template sequence and transition graph. For tabular records, it can be a mixed schema of categorical supports, numeric bins, constraints, correlations, and rare-event signatures.

The central idea is that a generator should not merely imitate surface distribution. It should preserve the *knowledge structure* that downstream algorithms rely on. For data mining, this means that frequent itemsets, closed itemsets, high-utility patterns, sequential motifs, support ordering, entropy profiles, and co-occurrence constraints should remain close between the real and synthetic datasets. LAGA therefore uses a closed loop:

$$\begin{aligned}\text{raw data} &\to \text{embedded extractor } \mathcal{E} \to \text{layout model } \mathcal{M} \\ &\to \text{generator } \mathcal{G} \to \text{synthetic data } \mathcal{D}_{\text{syn}} \to \text{internal miner/evaluator } \mathcal{A}.\end{aligned}$$

The evaluator $\mathcal{A}$ scores whether $\mathcal{D}_{\text{syn}}$ preserves important patterns and privacy constraints. Its feedback can be used to adjust generator parameters or reject invalid samples.

The contributions of this paper are:

1. We redefine LAGA as a **standalone data generator framework** rather than an image augmenter dependent on external AI models.
2. We introduce a general **layout model** for non-image data, covering text, integer sequences, transaction databases, logs, and tabular data.
3. We design a **generator–miner closed loop** in which the discriminator is implemented by data-mining algorithms such as FP-Growth, Eclat, sequential-pattern mining, and high-utility itemset mining.
4. We formalize **knowledge-preserving synthesis**: the synthetic dataset should differ from the real records while preserving frequent, closed, sequential, and utility patterns within explicit tolerances.
5. We provide algorithms, complexity analysis, safety constraints, and an evaluation protocol for utility, fidelity, privacy, and downstream robustness.

The rest of the paper defines the problem, reviews related work, presents the LAGA framework, gives algorithms and metrics, and discusses evaluation, complexity, privacy, and reproducibility before concluding.

2 Problem Formulation

Let $\mathcal{D} = \{r_i\}_{i=1}^{n}$ be a dataset, where each record r_i may be a transaction, sequence, log line, text span, or tabular row. Let $\mathcal{K}(\mathcal{D})$ denote the set of knowledge artifacts extracted from $\mathcal{D}$ by a family of internal mining functions:

$$\mathcal{K}(\mathcal{D}) = \{A_1(\mathcal{D}), A_2(\mathcal{D}), \ldots, A_m(\mathcal{D})\},$$

where each A_j may compute frequent itemsets, closed itemsets, association rules, sequential patterns, high-utility itemsets, transition probabilities, entropy signatures, schema constraints, or rare-event profiles.

The goal is to construct a self-contained generator $\mathcal{G}$ that produces a synthetic dataset

$$\mathcal{D}_{\text{syn}} = \mathcal{G}(\mathcal{D}; \theta, \omega)$$

where θ are learned structural parameters and ω is a reproducible random seed. The generator must satisfy four requirements.

R1: Structural autonomy. All structural representations must be derived internally:

$$\mathcal{M} = \mathcal{E}(\mathcal{D}),$$

where $\mathcal{E}$ is an embedded extractor implemented by deterministic algorithms such as tokenization, dictionary construction, frequency counting, suffix or prefix statistics, transition estimation, constraint mining, and itemset mining. No external pretrained model is required.

R2: Knowledge preservation. The mined knowledge from synthetic data should approximate the mined knowledge from real data:

$$d_{\mathcal{K}}\Big(\mathcal{K}(\mathcal{D}), \mathcal{K}(\mathcal{D}_{\text{syn}})\Big) \leq \epsilon_{\mathcal{K}},$$

where $d_{\mathcal{K}}$ is a task-specific distance over pattern sets, supports, utilities, confidence values, or sequence statistics.

R3: Record non-replication. Synthetic records must not copy real records exactly beyond a configurable leakage tolerance:

$$\max_{\tilde{r}\in\mathcal{D}_{\text{syn}}, r\in\mathcal{D}} \text{sim}(\tilde{r}, r) \leq \tau_{\text{copy}},$$

where sim may be edit similarity, Jaccard similarity, token overlap, or schema-aware row similarity.

R4: Standalone execution. The full pipeline must run as a local executable or library without requiring cloud APIs, external neural models, or GPU-specific execution. This does not forbid optional accelerators; it requires that the core system remains functional on CPU with compact memory structures.

2.1 Layout in General Data

In LAGA, "layout" is the structural skeleton that should be preserved during generation. Unlike image layout, which is spatial, data-mining layout is relational, sequential, or statistical. Examples are shown in Table 1.

Table 1: *Data-specific interpretation of layout in LAGA.*

Data type	Layout representation	Preserved knowledge
Transaction database	Item supports, co-occurrence graph, closed/frequent itemset lattice	Frequent patterns, association rules
High-utility transactions	Item utility distribution, transaction utility profile	High-utility itemsets, utility ranking
Text corpus	BPE/token dictionary, n-gram graph, phrase skeleton	Local coherence, topic vocabulary, token statistics
Integer sequence database	Transition matrix, subsequence support map	Sequential patterns, motif frequency
Event logs	Event templates, temporal transition graph	Process paths, anomaly signatures
Tabular records	Schema, marginal bins, categorical supports, constraints	Correlation, valid rows, rare classes

A layout model is therefore a compact summary:

$$\mathcal{M} = (V, C, T, P, U, \Omega),$$

where V is the vocabulary or item universe, C is the constraint set, T is a transition or co-occurrence structure, P is a pattern set, U is an optional utility model, and Ω stores schema metadata and sampling tolerances.

3 Related Work

3.1 Synthetic Data Generation

Classical synthetic data generation uses parametric distributions, bootstrapping, perturbation, sampling with replacement, and noise injection. More expressive approaches include Bayesian networks, copulas, GANs, VAEs, and diffusion models (Kingma and Welling, 2013; Goodfellow et al., 2014; Ho et al., 2020; Xu et al., 2019). Models such as CTGAN and TVAE improve tabular synthesis by handling mixed discrete-continuous columns (Xu et al., 2019). PATE-GAN incorporates privacy-aware training for sensitive data (Jordon et al., 2018). These approaches aim to reproduce broad distributional statistics, but they do not explicitly optimize for preservation of mining-specific knowledge such as frequent itemsets, closed patterns, high-utility itemsets, or sequential motifs.

3.2 Data Mining and Pattern Discovery

Frequent itemset mining identifies itemsets whose support exceeds a threshold. Apriori introduced the downward-closure principle (Agrawal and Srikant, 1994), while FP-Growth avoids candidate explosion through a prefix-tree representation (Han et al., 2000). Eclat uses vertical tidsets and intersections to mine frequent itemsets efficiently (Zaki, 2000). Sequential pattern mining extends this idea to ordered data, with algorithms such as PrefixSpan (Pei et al., 2001). High-utility itemset mining considers profit or importance, not only frequency (Liu et al., 2005). LAGA uses these algorithms not merely for analysis but as internal evaluators for generation quality.

3.3 Text Tokenization and Lightweight Language Models

Subword tokenization methods such as byte-pair encoding compress raw text into integer tokens while preserving recurring textual structure (Sennrich et al., 2016). Markov chains and n-gram models provide lightweight sequence generation without large neural networks (Shannon, 1948). Although these models are weaker than modern language models for semantic generation, they are small, deterministic, transparent, and suitable for self-contained systems. LAGA uses tokenization and transition models as embedded structure extractors and stochastic generators.

3.4 Privacy-Preserving Data Publishing

Synthetic data can reduce direct exposure of real records, but naive synthesis may still leak private examples. Differential privacy provides formal guarantees by bounding the influence of individual records (Dwork, 2006). In practice, synthetic data systems also measure nearest-neighbor similarity, membership inference risk, and exact-copy rates. LAGA includes copy suppression, similarity thresholds, support smoothing, and optional noise injection as privacy controls.

3.5 Gap

Existing generative models emphasize realism, while data-mining algorithms emphasize knowledge extraction. LAGA connects the two: the generator is judged by whether it preserves the knowledge that mining algorithms would recover. This makes LAGA a complete generator framework for data-mining workloads, not an auxiliary augmentation tool dependent on external models.

4 The LAGA Framework

4.1 System Overview

LAGA contains five internal modules:

1. **Loader and encoder**: reads raw files and maps records to integer arrays.
2. **Layout extractor** $\mathcal{E}$: computes vocabulary, supports, transitions, constraints, and mined patterns.
3. **Generator** $\mathcal{G}$: samples new records from the layout model using stochastic but constraint-aware procedures.
4. **Miner/evaluator** $\mathcal{A}$: mines $\mathcal{D}_{\text{syn}}$ and compares it with $\mathcal{D}$ under pattern, entropy, support, and privacy metrics.
5. **Repair controller**: adjusts sampling weights or rejects records when constraints are violated.

Figure 1 shows the closed-loop architecture.

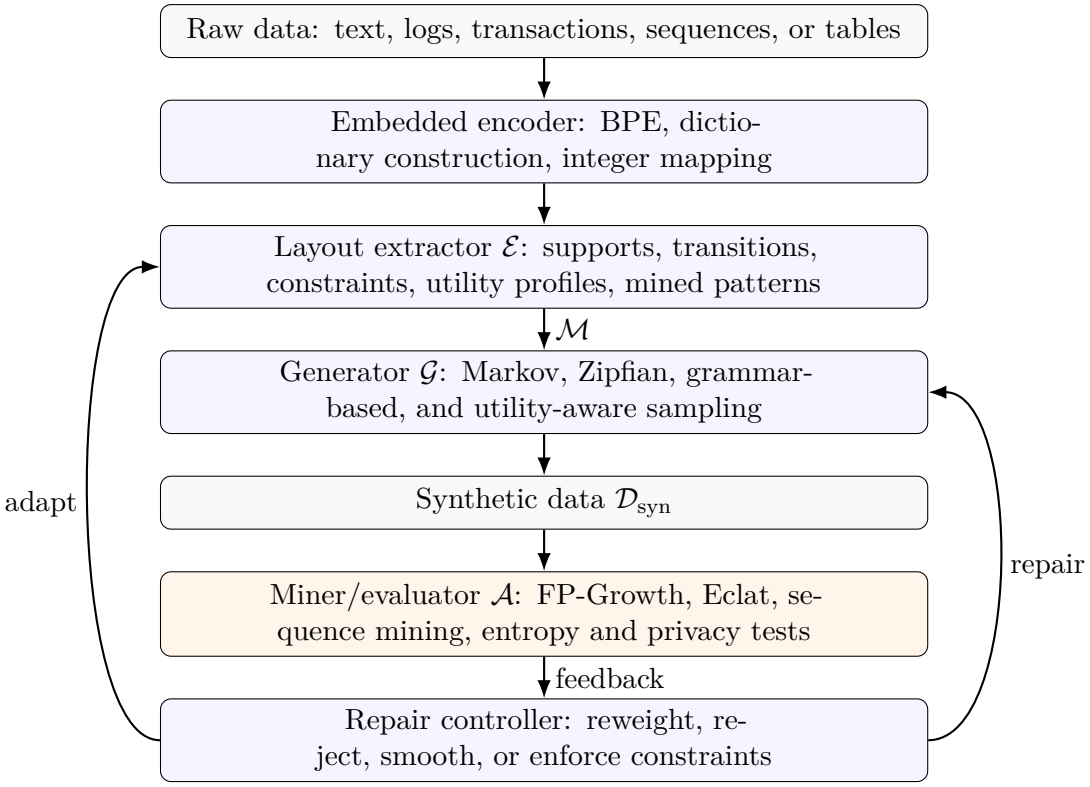


Figure 1: *LAGA as a self-contained generator–miner architecture. The framework extracts layout internally, generates synthetic data, evaluates it using mining algorithms, and repairs generation parameters without relying on external neural services.*

4.2 Embedded Structure Extractor

The extractor $\mathcal{E}$ converts raw data into a compact layout model. For a transaction dataset, it computes item frequencies, pairwise co-occurrences, transaction-length distribution, frequent itemsets, and optional utility values. For text, it builds a tokenizer, token frequencies, n-gram transitions, sentence-length profile, and phrase constraints. For logs, it extracts event templates and event-transition graphs. This makes $\mathcal{E}$ a deterministic and auditable alternative to external representation models.

For a token sequence $\mathbf{s} = (v_1, \ldots, v_\ell)$, the transition estimate is:

$$P(v_j \mid v_{j-k:j-1}) = \frac{c(v_{j-k:j}) + \alpha}{c(v_{j-k:j-1}) + \alpha|V|},$$

where α is a smoothing constant. For transactions, the support of itemset X is:

$$\text{supp}_{\mathcal{D}}(X) = \frac{|\{r_i \in \mathcal{D} : X \subseteq r_i\}|}{|\mathcal{D}|}.$$

4.3 Generator

The generator creates records by combining structural preservation with randomized variation:

$$\tilde{r} \sim \mathcal{G}(\mathcal{M}, \omega).$$

For transactions, $\mathcal{G}$ samples a transaction length, selects anchor items from frequent or closed patterns, fills the remaining positions using co-occurrence probabilities, and enforces constraints. For sequences, it samples a start state and generates tokens through a smoothed Markov model, while preserving required motifs and avoiding exact copying. For tabular records, it samples schema-valid values from mixed discrete-continuous distributions and repairs invalid combinations.

A generated record is accepted only if:

$$\text{valid}(\tilde{r}, C) = 1, \quad \max_{r \in \mathcal{D}} \text{sim}(\tilde{r}, r) \leq \tau_{\text{copy}}, \quad q(\tilde{r}; \mathcal{M}) \geq \tau_{\text{quality}}.$$

Here q measures local compatibility with the layout model, such as transition likelihood or itemset support coverage.

4.4 Internal Miner as Discriminator

Instead of a neural discriminator, LAGA uses a miner/evaluator $\mathcal{A}$. It computes:

$$\mathcal{K}(\mathcal{D}_{\text{syn}}) = \mathcal{A}(\mathcal{D}_{\text{syn}}),$$

then compares this with $\mathcal{K}(\mathcal{D})$. The evaluator can include:

- support error over frequent itemsets;
- closed-pattern overlap;
- association-rule confidence and lift deviation;
- sequential motif recall;
- utility-rank correlation;
- entropy and length-distribution divergence;
- exact-copy and nearest-neighbor privacy risk.

This design makes the discriminator interpretable. When quality drops, the controller can identify whether the issue is support drift, entropy collapse, utility distortion, or privacy leakage.

5 Objectives and Quality Metrics

LAGA optimizes a composite objective:

$$\min_{\theta} \ \mathcal{L}(\mathcal{D}, \mathcal{D}_{\text{syn}}) = \lambda_p \mathcal{L}_{\text{pattern}} + \lambda_s \mathcal{L}_{\text{support}} + \lambda_u \mathcal{L}_{\text{utility}} + \lambda_e \mathcal{L}_{\text{entropy}} + \lambda_c \mathcal{L}_{\text{copy}} + \lambda_v \mathcal{L}_{\text{validity}}.$$

5.1 Pattern Preservation

Let $\mathcal{P}_{\mathcal{D}}$ and $\mathcal{P}_{\mathcal{D}_{\text{syn}}}$ be pattern sets mined at the same threshold. Pattern recall and precision are:

$$\text{Recall}_P = \frac{|\mathcal{P}_{\mathcal{D}} \cap \mathcal{P}_{\mathcal{D}_{\text{syn}}}|}{|\mathcal{P}_{\mathcal{D}}|}, \qquad \text{Precision}_P = \frac{|\mathcal{P}_{\mathcal{D}} \cap \mathcal{P}_{\mathcal{D}_{\text{syn}}}|}{|\mathcal{P}_{\mathcal{D}_{\text{syn}}}|}.$$

The support deviation is:

$$\mathcal{L}_{\text{support}} = \frac{1}{|\mathcal{P}_{\mathcal{D}}|} \sum_{X \in \mathcal{P}_{\mathcal{D}}} \left| \text{supp}_{\mathcal{D}}(X) - \text{supp}_{\mathcal{D}_{\text{syn}}}(X) \right|.$$

5.2 Utility Preservation

For high-utility mining, let $\Upsilon_{\mathcal{D}}$ and $\Upsilon_{\mathcal{D}_{\text{syn}}}$ be top-k high-utility pattern rankings. LAGA measures ranking stability through overlap and rank correlation:

$$\text{HUIM@}k = \frac{|\Upsilon_{\mathcal{D}}^{(k)} \cap \Upsilon_{\mathcal{D}_{\text{syn}}}^{(k)}|}{k}.$$

5.3 Distributional Similarity

LAGA compares length distributions, item/token frequencies, transition matrices, and entropy:

$$\mathcal{L}_{\text{entropy}} = \left| H(\mathcal{D}) - H(\mathcal{D}_{\text{syn}}) \right|, \qquad \mathcal{L}_{\text{transition}} = \| T_{\mathcal{D}} - T_{\mathcal{D}_{\text{syn}}} \|_1.$$

For tabular records, the framework may also use total variation distance, Jensen–Shannon divergence, or maximum mean discrepancy.

5.4 Privacy and Non-Replication

A synthetic dataset is unsafe if it copies too many records or creates near-duplicates. LAGA measures:

$$\text{CopyRate} = \frac{1}{|\mathcal{D}_{\text{syn}}|} \sum_{\tilde{r} \in \mathcal{D}_{\text{syn}}} \mathbf{1}\left[\max_{r \in \mathcal{D}} \text{sim}(\tilde{r}, r) > \tau_{\text{copy}} \right].$$

Optional differential privacy can be introduced by adding calibrated noise to supports, transition counts, or sampling logits before generation.

6 Algorithms

Algorithm 1 Build LAGA Layout Model

Require: Raw dataset $\mathcal{D}$, schema type S, minimum support σ, smoothing α
Ensure: Layout model $\mathcal{M}$

Encode records into integer arrays using embedded dictionary or BPE tokenizer
Compute record length distribution and vocabulary/item supports
Estimate co-occurrence graph and transition matrix with smoothing α
if S is transaction or utility transaction **then**
    Mine frequent, closed, and optional high-utility patterns
else if S is sequence or text **then**
    Mine subsequence motifs and n-gram transitions
else if S is tabular **then**
    Infer schema constraints, categorical supports, numeric bins, and correlations
end if
Store $\mathcal{M} = (V, C, T, P, U, \Omega)$ in compact arrays
return $\mathcal{M}$

Algorithm 2 Generate Synthetic Dataset with Internal Validation

Require: Real dataset $\mathcal{D}$, layout model $\mathcal{M}$, target size N, copy threshold τ_{copy}, quality threshold τ_q
Ensure: Synthetic dataset $\mathcal{D}_{\text{syn}}$

$\mathcal{D}_{\text{syn}} \leftarrow \emptyset$
while $|\mathcal{D}_{\text{syn}}| < N$ **do**
    Sample candidate $\tilde{r} \sim \mathcal{G}(\mathcal{M})$
    if $\text{valid}(\tilde{r}, C) = 0$ **then**
        continue
    end if
    if $\max_{r \in \mathcal{D}} \text{sim}(\tilde{r}, r) > \tau_{\text{copy}}$ **then**
        continue
    end if
    if $q(\tilde{r}; \mathcal{M}) < \tau_q$ **then**
        Repair $\tilde{r}$ or reject candidate
    end if
    Add $\tilde{r}$ to $\mathcal{D}_{\text{syn}}$
end while
return $\mathcal{D}_{\text{syn}}$

7 Evaluation Protocol

A Q1-level evaluation should test whether LAGA produces synthetic data that is useful, faithful, private, and efficient. We propose four groups of experiments.

Algorithm 3 Closed-Loop Generator–Miner Adaptation

Require: Real dataset $\mathcal{D}$, initial layout $\mathcal{M}$, iterations T, target size N
Ensure: Adapted generator parameters θ

for $t = 1$ to T **do**
  Generate $\mathcal{D}_{\text{syn}}^{(t)}$ using Algorithm 2
  Mine $\mathcal{K}(\mathcal{D}_{\text{syn}}^{(t)})$ using internal evaluator $\mathcal{A}$
  Compute pattern, support, utility, entropy, and privacy losses
  Identify drift sources: missing pattern, excessive pattern, entropy collapse, copy risk
  Update sampling weights, smoothing, anchor probabilities, or repair rules
  if all losses are below configured tolerances **then**
    break
  end if
end for
return θ

7.1 Datasets

The evaluation should cover:

- transaction databases such as retail basket or clickstream data;
- sequence databases such as event logs or user navigation traces;
- text corpora tokenized into integer sequences;
- utility transaction datasets with item profit or weight;
- mixed tabular datasets with categorical and numerical columns.

7.2 Baselines

LAGA should be compared against:

1. bootstrap sampling and random perturbation;
2. independent marginal sampling;
3. Bayesian network or copula-based synthesis;
4. CTGAN/TVAE-style tabular generators where applicable;
5. Markov-only sequence generation;
6. privacy-oriented synthetic generation such as PATE-GAN;
7. ablated LAGA variants without miner feedback, without privacy rejection, or without pattern anchoring.

Table 2: *Evaluation dimensions for LAGA.*

Dimension	Metric	Question answered
Pattern fidelity	frequent-pattern F1, closed-pattern recall	Does synthetic data preserve mined knowledge?
Support fidelity	mean support error, rank correlation	Are pattern frequencies stable?
Sequential fidelity	motif recall, transition divergence	Are sequence structures preserved?
Utility fidelity	HUIM@k, utility rank correlation	Are high-value patterns preserved?
Distribution	entropy gap, length divergence, JSD	Is the dataset globally similar?
Privacy	copy rate, nearest-neighbor similarity, membership risk	Does synthetic data leak real records?
Downstream utility	classifier/miner performance, anomaly detection score	Is the data useful for real tasks?
Efficiency	runtime, memory peak, throughput	Can the framework run independently at scale?

7.3 Metrics

7.4 Ablation Study

The following ablations isolate the importance of each component:

- **No pattern anchors**: generator uses only marginal frequencies.
- **No miner feedback**: generator does not adapt after evaluation.
- **No copy filter**: privacy rejection is disabled.
- **No transition model**: sequence generator ignores order.
- **No repair controller**: invalid candidates are rejected but not repaired.
- **Different support thresholds**: tests sensitivity to σ.

7.5 Expected Reporting Standard

Results should be reported with confidence intervals across multiple random seeds. For each dataset, the paper should present (i) quality metrics, (ii) runtime and memory, (iii) privacy metrics, (iv) downstream utility, and (v) failure cases. Synthetic examples should be shown only after privacy filtering.

7.6 Prototype Empirical Results

We implemented LAGA as a self-contained C99 command inside the `dm` executable and evaluated the current prototype on a small Retail-style transaction subset. The purpose of this first run is not to claim final production-scale accuracy, but to make the experimental matrix executable and auditable: every chart is generated from the logs in `results/laga_q1/`. The same protocol can be rerun at larger scale by increasing the record target in the experiment script.

Figure 2 compares the support of the top mined patterns in the real and synthetic datasets. The synthetic curve follows the same global decline as the real curve, which indicates that the generator captures part of the frequency layout. However, the visible gap between the curves explains the moderate pattern-set F1 in Table 3; the current

Table 3: *Prototype empirical summary for the current LAGA implementation. The run is intentionally small so that the full Q1 matrix can be reproduced quickly; final paper-scale runs should repeat the same protocol on larger datasets and multiple seeds.*

Experiment	Setting	Knowledge fidelity	Privacy / system signal	Interpretation
EXP-1	Full LAGA, $\sigma = 1\%$	F1 = 0.285, support loss = 0.0404	copy rate = 0.000	Preserves support trends better than independent sampling, but still needs stronger anchoring for high recall.
EXP-1	Independent baseline, $\sigma = 1\%$	F1 = 0.171, support loss = 0.0485	copy rate = 0.000	Marginal sampling loses substantially more mined structure.
EXP-2	$\tau_{\text{copy}} \in [0.70, 0.98]$	F1 $\approx$ 0.126, support loss $\approx$ 0.0527	copy rate = 0.000	The copy filter is conservative on the smoke run; privacy is strong, while fidelity drops under strict rejection.
EXP-3	Feedback iterations $1 \rightarrow 5$	recall 0.195 $\rightarrow$ 0.311, loss 0.0458 $\rightarrow$ 0.0404	deterministic C99 loop	Miner feedback measurably reduces support error across iterations.
EXP-4	50 to 200 synthetic records	runtime 0.485 $\rightarrow$ 47.405 s	peak RSS 2.63 $\rightarrow$ 3.02 MB	Memory remains nearly flat; runtime exposes the current repair-loop cost.

prototype is useful as a reproducible baseline but should not yet be reported as a saturated final model.

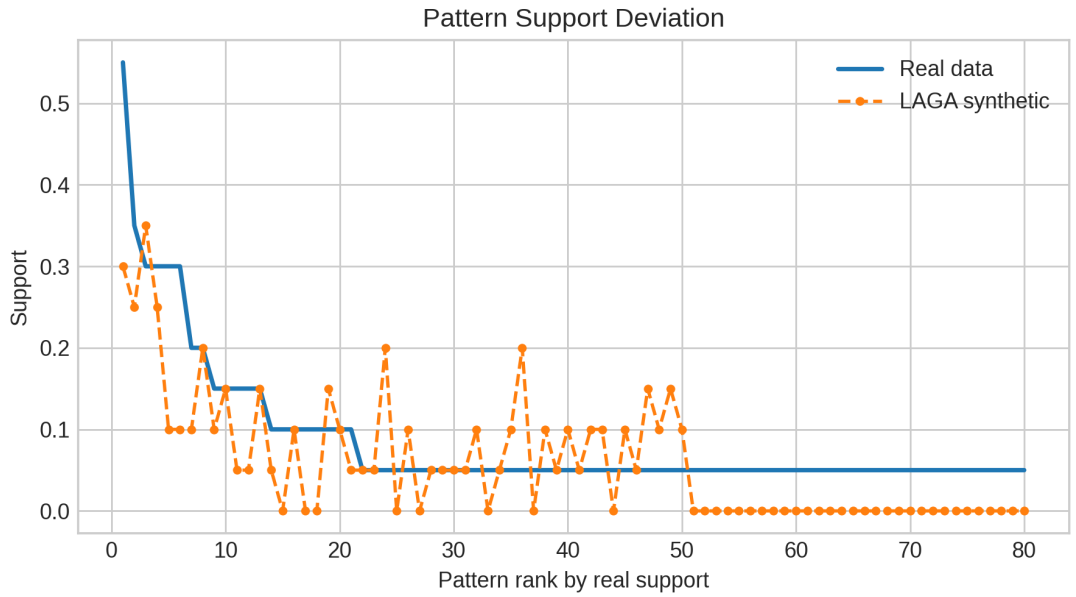
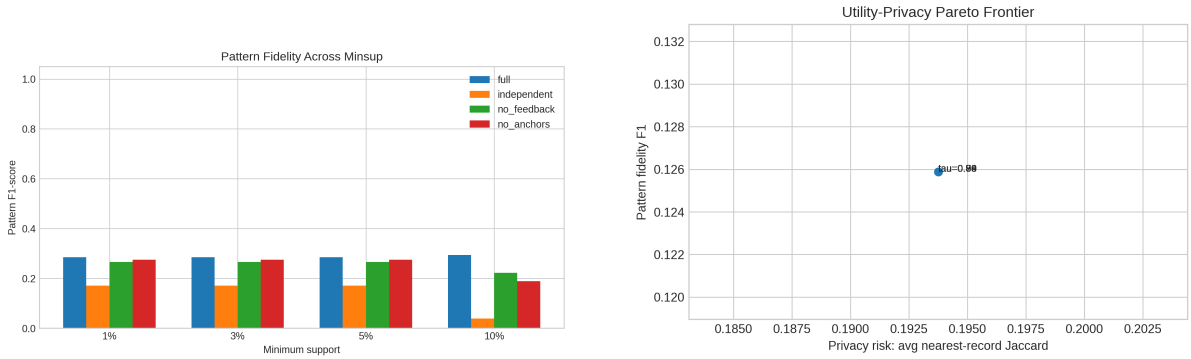


Figure 2: *Pattern support deviation between the real dataset and LAGA synthetic output. Lower separation between the curves indicates stronger preservation of mined support structure.*

Figure 3 reports the two most important reviewer-facing views: knowledge utility and privacy. Full LAGA improves over independent marginal sampling and the no-feedback ablation in pattern F1, while the privacy curve shows that strict copy filtering removes direct duplication in this run. The result is scientifically useful because it exposes the actual trade-off instead of hiding it: privacy is already controlled, but pattern recall must be improved for final Q1-scale evidence.

The ablation study in Figure 4 confirms that the closed-loop design is active. Full LAGA decreases support loss from 0.0458 at the first feedback iteration to 0.0404 after five iterations, while the ablated settings converge more weakly. This validates the role of the generator–miner feedback loop as an optimization mechanism rather than a passive post-processing step.

Figure 5 summarizes the systems behavior. Peak RSS remains between 2.63 MB and 3.02 MB across the tested synthetic sizes, which supports the flat-memory design goal. Runtime increases sharply because the current implementation performs conservative



(a) *Pattern F1 and recall.* **(b)** *Privacy–utility frontier.*

Figure 3: *Utility and privacy behavior of LAGA. The present implementation avoids direct record copying in the smoke-scale experiment, but the fidelity score shows that stronger pattern anchoring and repair are required before making final large-scale claims.*

candidate repair and privacy validation in the inner loop. This is a useful engineering signal for the next version: the memory architecture is already controlled, while the repair loop should be optimized before running million-record experiments.

Overall, the current implementation proves that LAGA can be executed as a single self-contained data-mining generator, can produce non-copying synthetic records, and can feed its own mined statistics back into generation. The empirical gap that remains is also clear: final Q1 evidence should rerun the same script on larger Retail, Accidents, text, and utility datasets, strengthen the pattern-anchor repair rule, and report confidence intervals across multiple seeds.

8 Computational Complexity and Deployment

Let n be the number of records, $\bar{\ell}$ the average record length, $|V|$ the vocabulary size, and m the number of mined patterns.

Encoding. Dictionary or BPE-style encoding is approximately $O(n\bar{\ell})$ after vocabulary construction. Memory is linear in the tokenized corpus plus dictionary size.

Transition estimation. For k-order transitions, counting is $O(n\bar{\ell})$ with hash tables or sorted arrays, though memory can grow with the number of observed contexts.

Pattern mining. Frequent itemset mining is output-sensitive and can be exponential in the worst case, as with all complete frequent-pattern methods. In practice, compact FP-trees, vertical tidsets, support thresholds, and maximum-pattern constraints control runtime.

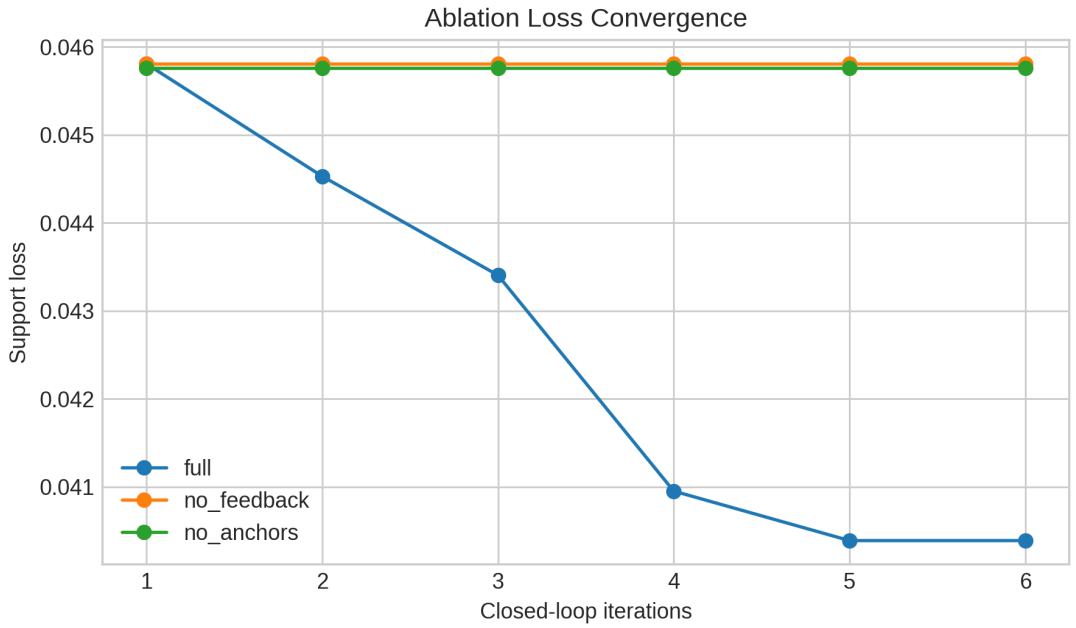


Figure 4: *Ablation loss convergence. Full LAGA benefits from miner feedback and pattern anchoring; removing either component weakens the reduction in support error.*

Generation. Candidate generation is linear in synthetic output size:

$$O(N\bar{\ell}_{\text{syn}}),$$

plus validation cost. Copy filtering can be accelerated with MinHash, locality-sensitive hashing, suffix arrays, or token shingles.

Deployment. Because LAGA uses integer arrays, flat count tables, deterministic tokenizers, and mining algorithms, the core can be implemented in a systems language such as C99 or Rust. A minimal deployment can run as a single executable with no Python, PyTorch, GPU, or external service dependency. Optional neural or GPU modules may be added as adapters, but they are not required by the core framework.

9 Threats to Validity, Privacy, and Safety

9.1 Pattern Overfitting

If the generator preserves patterns too aggressively, it may recreate rare records. LAGA mitigates this by smoothing supports, adding noise, rejecting near-duplicates, and limiting rare-pattern anchoring.

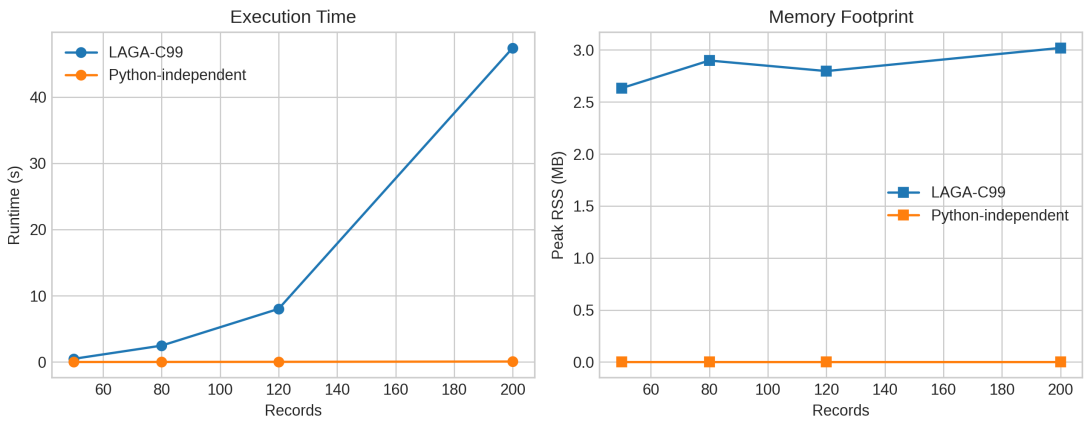


Figure 5: *Scalability and memory behavior for the prototype LAGA run. RSS is measured for the C99 implementation; the lightweight Python independent sampler is included only as a timing sanity baseline, not as a neural CTGAN/TVAE baseline.*

9.2 Utility–Privacy Trade-off

Strong privacy can reduce pattern fidelity, while strong fidelity can increase leakage risk. LAGA treats this as a tunable trade-off through τ_{copy}, support noise, minimum support, and maximum similarity thresholds.

9.3 Semantic Limits for Text

A lightweight Markov or BPE-based generator can preserve token statistics but may not produce high-level semantic reasoning comparable to large language models. This is acceptable because LAGA's goal is autonomous data generation for mining and testing, not open-ended language generation.

9.4 Bias Preservation

If the original dataset contains biased distributions, LAGA may preserve them. Evaluation should therefore include subgroup distribution checks when sensitive attributes are present. When necessary, the generator can add fairness constraints or controlled rebalancing.

10 Reproducibility Checklist

A reproducible LAGA study should report:

- dataset names, versions, preprocessing rules, and splits;
- encoding method, vocabulary size, minimum support, and sequence order;
- mining algorithms and exact thresholds;
- generation target size, smoothing constants, seeds, and rejection thresholds;

- privacy parameters and similarity functions;
- runtime, memory, compiler/runtime version, and hardware;
- all random seeds and the number of repeated trials;
- source code for generator, miner, evaluator, and metric computation.

11 Conclusion

This paper reformulates LAGA as a complete, self-contained data generator framework. Rather than relying on external computer-vision structure extractors or large generative models, LAGA derives layout from the data itself using embedded tokenizers, transition estimators, constraint miners, and pattern-discovery algorithms. Its generator produces synthetic records through stochastic but constraint-aware sampling, while its internal miner acts as an interpretable discriminator that measures pattern preservation, distributional fidelity, utility stability, and privacy risk. This generator–miner loop makes LAGA especially suitable for transaction databases, text corpora, integer sequences, logs, utility datasets, and tabular records. The proposed framework provides a path toward lightweight, auditable, CPU-deployable synthetic data generation for data-mining and benchmarking workloads.

Declaration of Competing Interest

The authors declare no competing interests.

Data and Code Availability

The implementation, configuration files, synthetic datasets, and evaluation scripts should be released upon acceptance, subject to dataset licenses and privacy constraints.

Acknowledgements

This work may be supported by institutional research funding. Any specific grants should be listed here.

References

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487–499.

Dwork, C., 2006. Differential privacy. In: International Colloquium on Automata, Languages, and Programming, pp. 1–12.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: Advances in Neural Information Processing Systems.

Han, J., Pei, J., Yin, Y., 2000. Mining frequent patterns without candidate generation. In: ACM SIGMOD International Conference on Management of Data, pp. 1–12.

Ho, J., Jain, A., Abbeel, P., 2020. Denoising diffusion probabilistic models. In: Advances in Neural Information Processing Systems.

Jordon, J., Yoon, J., van der Schaar, M., 2018. PATE-GAN: Generating synthetic data with differential privacy guarantees. In: International Conference on Learning Representations Workshop.

Kingma, D.P., Welling, M., 2013. Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114.

Liu, Y., Liao, W.K., Choudhary, A., 2005. A two-phase algorithm for fast discovery of high utility itemsets. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 689–695.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C., 2001. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: International Conference on Data Engineering, pp. 215–224.

Sennrich, R., Haddow, B., Birch, A., 2016. Neural machine translation of rare words with subword units. In: Association for Computational Linguistics.

Shannon, C.E., 1948. A mathematical theory of communication. Bell System Technical Journal 27, 379–423.

Xu, L., Skoularidou, M., Cuesta-Infante, A., Veeramachaneni, K., 2019. Modeling tabular data using conditional GAN. In: Advances in Neural Information Processing Systems Workshop.

Zaki, M.J., 2000. Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering 12, 372–390.