ISSN 1816-6075 (Print), 1818-0523 (Online)
Journal of System and Management Sciences
Vol. 16 (2026) No. 1, pp. 37-55
DOI:10.33168/JSMS.2026.0103

Specification-Driven Synthetic Tabular Data Generation for System Validation and Management Decision Support

Mohammed said BOUKHRYSS, Nihad LECHHAB OUADRASSI, Abdellah AZMANI,

Khalid JEBARI HASSANI

Intelligent Automation and BioMedGenomics Laboratory (IABL), Faculty of Sciences and Technologies of Tangier, Abdelmalek Essaâdi University, Tetouan 93000, Morocco

{mohammedsaid.boukhryss, nihad.lechhabouadrassi}@etu.uae.ac.ma,

{a. azmani, k.jebari}@uae.ac.ma

Abstract. Synthetic tabular data are increasingly needed when real datasets are difficult to access, incomplete, or subject to confidentiality constraints, especially when variables must comply with prescribed distributions and domain rules. This work proposes and evaluates a configurable, specification-driven framework for generating synthetic tabular data that respects user-defined distributions and logical exclusion constraints, demonstrated through multiple case studies, including IT infrastructure and smart building data. The framework is designed to support system validation, benchmarking management systems, and decision-making under constraints by enabling engineers to fine-tune marginal distributions, enforce consistency rules, and scale data generation to millions of rows. It includes a parameter file for defining variables, admissible values or intervals, and target percentages, alongside a conditions file encoding forbidden combinations as logical rules. The framework implements two generation strategies: a sequential baseline and a GPU-accelerated parallel method based on CUDA and Numba. Both methods are validated through checks on distribution fidelity, constraint satisfaction, null rows, and generation time. Experiments on datasets across different domains demonstrate that both methods match target distributions with minimal deviations and satisfy all exclusion rules. The parallel implementation significantly reduces generation time, showcasing scalability and performance improvements.

Keywords: data generation Synthetic, tabular data, Rule-based data generation, Specification-driven generation, GPU-acceleration, CUDA

1. Introduction

Modern IT infrastructures generate large volumes of operational data (Bautista et al., 2019), yet high-quality datasets for analysis and predictive maintenance remain difficult to obtain. Failures and rare events are under-represented, and access to real production logs is often restricted for privacy or security reasons. Recorded data may also be incomplete or unbalanced. In this context, synthetic tabular data becomes a practical alternative to support model development, stress testing, and benchmarking of analytics pipelines for virtualized infrastructures and cloud environments.

Despite recent advances in generative models for tabular data (Stoian et al., 2025), many approaches offer limited control over explicit distributions and domain rules. Practitioners often need to enforce precise marginal percentages, numeric intervals, and logical dependencies between variables. When these requirements are not met, synthetic datasets can deviate from operational constraints, making them hard to trust or reuse. This work addresses the generation of synthetic tabular data that accurately reflects user-defined distributions and logical constraints. The framework accepts two inputs: a parameter file defining variables, admissible values, and target distributions, and a conditions file encoding logical exclusion rules. The resulting synthetic datasets satisfy these specifications and support system validation, benchmarking, and decision-making.

More precisely, the framework aims to: (1) define a general schema and validation workflow for parameters and conditions; (2) implement two generation strategies—a sequential baseline and a GPU-accelerated parallel method; and (3) evaluate both strategies using a pipeline that checks distribution fidelity, constraint satisfaction, null rows, and generation time. This framework is not intended to compete with existing generative models such as CTGAN (Xu et al., 2019) or TVAE (Kiran et al., 2025), but rather to complement them by providing a fully specification-driven generator that enforces explicit distributions and logical constraints defined by the user.

These capabilities are particularly valuable in regulated or high-stakes environments, where operational decisions depend on data that is accurate, reliable, and traceable.

The remainder of the paper is structured as follows. Section 2 presents the background and related work. Section 3 describes the proposed methodology, including the parameter and conditions files and the two generation modes. Section 4 details the IT-infrastructure case study. Section 5 reports and discusses the experimental results for both the sequential and parallel implementations. Section 6 concludes the paper and outlines limitations and future perspectives.

2. Background

Data generation for advanced applications in IT infrastructure faces numerous challenges that can hinder the effectiveness of the models and algorithms. These challenges often stem from issues related to data scarcity, dependency complexity, and the limitations of traditional data collection methods.

One of the primary challenges in data generation is the scarcity of high-quality, labeled data. In predictive maintenance, particularly for IT infrastructure, obtaining sufficient training data for machine learning models is often difficult due to the rarity of failures and anomalies in complex systems. These difficulties are further compounded when available data is noisy or imbalanced, complicating the training of reliable models (Bansal et al., 2022).

As predictive maintenance systems increasingly rely on large-scale data from IT and industrial infrastructures, data privacy and security have become critical concerns. Traditional centralized architectures pose significant risks due to the concentration of sensitive operational data in cloud environments, increasing exposure to breaches and non-compliance with regulations such as the GDPR (Chen & Zhao, 2012; Zhang, 2018). This is particularly problematic in IT systems, where proprietary configurations and sensitive operational metrics are involved.

To address these concerns, researchers have proposed federated learning (FL) as a privacy-preserving alternative. FL enables models to be trained locally on edge devices without transmitting raw data, thereby maintaining data sovereignty and reducing communication overhead. Privacy-enhancing techniques such as homomorphic encryption, differential privacy, and secure multi-party computation (SMPC) have been integrated to further protect model updates from inference attacks (Hosni, 2025).

Beyond technical safeguards, broader conceptual challenges persist. Mühlhoff (2023) introduced the notion of predictive privacy, arguing that even anonymized data can cause privacy violations when models infer sensitive traits about individuals who never directly provided such information. This challenges existing regulations and calls for protecting communities as well as individuals. The European Parliament (2020) similarly recognizes that AI systems' reliance on inferred data creates tensions with core GDPR principles such as data minimization and purpose limitation, generating significant ambiguity around derived data and predictive modeling.

3. Related Work

Synthetic tabular data generation is now a mature and active research area, aiming to create artificial datasets that preserve the statistical properties, constraints, and inter-column dependencies of real data. Key approaches include classical probabilistic methods, deep generative models such as GANs, diffusion-based models, and hybrid/LLM-based methods. Earlier probabilistic approaches relied on Bayesian networks (Gogoshin et al., 2021) and copulas (Kamthe et al., 2021) to learn and sample joint distributions over variables (Xu et al., 2019).

Generative Adversarial Networks (GANs) have been adapted to tabular data, but directly applying image-style GANs to mixed-type tabular data leads to problems such as mode collapse and poor handling of rare categories. To address this, Xu et al. (2019) proposed CTGAN (Conditional Tabular GAN), which introduces conditional sampling to account for rare categories. The model also employs mode-specific normalization for continuous variables, enabling it to jointly learn numeric and categorical columns. CTGAN showed strong performance against Bayesian baseline methods and earlier GAN/VAEs in multiple benchmark datasets. Later works have extended or adapted CTGAN for domain-specific uses (Parise et al., 2025), and comparative benchmarking studies frequently place CTGAN as a strong baseline (Pathare et al., 2023). Rigorous experimental analyses also compare CTGAN against VAE-based alternatives (TVAE) in different dataset settings (Yadav et al., 2024).

Diffusion models, which have shown great success in image and signal domains, are now being adapted for tabular generation:

- TabDDPM (Kotelnikov et al., 2022) :A diffusion model for mixed-type tabular data that outperforms many GAN- and VAE-based synthetic generators, particularly in terms of fidelity and diversity. It is shown to better handle structural correlations and reduce privacy risk compared to interpolation methods.
- Diffusion Models for Tabular Data Imputation & Generation (Villaizán-Vallelado et al., 2025): Introduces a model combining transformer-based conditioning and dynamic masking to jointly handle missing data imputation and synthetic data generation, comparing favorably with GANs and VAEs over benchmarks.
- TabDiff (Shi et al., 2024): Proposes a joint diffusion framework to model both continuous and categorical features with feature-wise learnable diffusion processes, improving preservation of pairwise correlations and outperforming numerous baselines across multiple metrics.
- Surveys and recent reviews (Z. Li et al., 2025) have detailed current progress, challenges, and future directions in diffusion approaches tailored to tabular data, which systematically reviews nearly all relevant works from 2015 to 2024, and organizes them by application

(augmentation, imputation, privacy, anomaly detection).

- Hybrid models combining autoencoders and diffusion also demonstrate efficacy in capturing correlations in tabular data (Suh et al., 2023).

Recently, large language models (LLMs) have been explored for tabular synthetic generation and hybrid pipelines:

- The comparative study (Barr et al., 2025) explores zero-shot generation using GPT-4o versus CTGAN, showing that LLMs (without task-specific fine-tuning) can sometimes match or surpass CTGAN in preserving means, correlations, and privacy under some datasets.

- (Khalil et al., 2025) explores blending CTGAN and LLMs to generate educational tabular data, demonstrating how LLMs can provide semantic scaffolding while CTGAN ensures statistical coherence.

While learned generative models such as CTGAN and TVAE are effective for many applications, our specification-driven framework offers several advantages in system and management contexts where strict control over data generation is required. Specifically, it ensures regulatory compliance by allowing users to define and enforce explicit rules that align with legal and ethical standards. Additionally, the explainability and auditability of our approach provide transparency, making it easier to understand and justify the data generation process, which is crucial for accountability in many industries. Furthermore, the framework enables strict rule enforcement, ensuring that logical constraints and operational dependencies are respected throughout the data generation process. These features make our approach particularly suitable for environments where consistency and governance are paramount, whereas learned models may not guarantee the same level of control.

Parallel computing, especially through the use of Graphics Processing Units (GPUs), has become indispensable in scaling synthetic data generation for large datasets and complex models. GPUs provide massively parallel architectures with thousands of cores that can execute matrix and tensor operations simultaneously, significantly accelerating deep generative training and sampling.

For instance, (Cardoso et al., 2021) explored multi-GPU and TPU parallelization of GAN training in a high-energy physics context. They employed data-parallel strategies in TensorFlow and custom GPU loops, achieving near-linear speed-up in training time while maintaining the quality of generated data comparable to Monte Carlo simulations. Their results showed that scaling to multiple GPU nodes allowed substantial reductions in wall-clock time without degrading model performance. Similarly, “Accelerating GAN training using highly parallel hardware on public cloud” confirms that parallelization over multiple devices retains generative fidelity while improving efficiency.

In the domain of synthetic data pipelines for structured data, (Mora-de-León et al., 2025) present Text-Conditioned Diffusion Models for time-series (e.g. turbine engine sensor readings). They leverage GPU-accelerated diffusion architectures to condition generation on textual prompts. Their results show that GPU-enabled diffusion sampling is feasible for real-time synthetic data augmentation and that the synthetic samples maintain high fidelity in downstream tasks. Similarly, in financial time series, (Takahashi & Mizuno, 2025) applied diffusion-based methods to generate synthetic financial data; they report that diffusion models—when accelerated—offer a favorable trade-off between quality and diversity versus other generative methods like GANs and VAEs.

From systems and infrastructure perspective,(J. Li, 2025) discuss GPU-based Distributed Data Parallel (DDP) strategies for parallel task scheduling and data processing. Their experiments demonstrate that employing GPU-accelerated DDP reduces execution times significantly (a proposed DDP-GPU scheduling approach led to 95.3% improvements in efficiency over conventional single-GPU execution, with an average execution time of 25.7 s in tested scenarios). This underlines how GPU parallelism is not just advantageous at the model level, but also in orchestrating data workflows and resource scheduling in generative pipelines.

In a broad survey of GPU applications in data-centric domains, (Siddique & Ashour, 2024) review the role of GPUs as accelerators for tasks such as data mining, logging, machine learning, and deep learning. They emphasize that the massive parallelism of GPUs allows complex computational problems to be decomposed into smaller, more efficient tasks, making them particularly suitable for high-throughput workloads in ML and DL. According to the authors, this capability makes GPUs an ideal choice for handling the growing computational requirements of modern data-driven applications, especially as datasets and model complexity continue to increase.

The limitations identified in these approaches—particularly the lack of explicit distribution control and constraint enforcement—motivate the specification-driven framework described in the following section.

4. Methodology

As described in the Introduction, the framework generates synthetic tabular datasets that remain faithful to user-defined parameters, statistical distributions, and logical constraints. Users provide configuration files specifying variable values, distribution percentages, and exclusion rules; these specifications are enforced during data generation so that the resulting tables preserve target proportions, adhere to defined intervals, and exclude invalid combinations.

A central feature of this study is the comparative evaluation between a sequential generation approach and a parallel generation approach accelerated by GPUs. The sequential method serves as a baseline, providing transparency and ease of verification while demonstrating how parameters and conditions are respected. In contrast, the parallel strategy leverages GPU-based parallel computing to accelerate sampling and constraint checking, enabling the generation of much larger datasets in significantly less time.

To structure this comparison, the work addresses four research questions: (1) To what extent can a rule-driven process preserve distributions and constraints? (2) How does GPU acceleration affect execution time relative to a sequential baseline? (3) Does parallelization alter statistical fidelity or constraint satisfaction? (4) How does dataset size influence performance differences? From these questions, three hypotheses follow: GPU acceleration reduces computation time without affecting constraint adherence; constraint complexity increases execution time more slowly in the parallel strategy; and both approaches maintain comparable statistical fidelity when definitions are well specified.

Evaluating both strategies is essential for two reasons. First, the sequential approach establishes baseline performance and correctness, ensuring that distributions and constraints are accurately reproduced. Second, the GPU-accelerated approach demonstrates scalability and efficiency, making it possible to synthesize millions of records within seconds—an outcome particularly valuable in fields such as IT infrastructures, where large-scale datasets are required for model training, simulation, and benchmarking. This dual perspective highlights the trade-off between simplicity and computational power while underscoring the practical advantages of parallel computing in modern data generation.

4.1. Input Data and Configuration

The synthetic data generation process is driven by two key inputs: the parameter file and the conditions file. The parameter file defines the schema, target distributions, and admissible values for each variable, ensuring the generated data aligns with user-defined statistical properties. The conditions file encodes logical exclusion rules that prevent invalid combinations of values across variables. These inputs guide the framework to generate large-scale datasets that meet the objectives of distribution fidelity, constraint satisfaction, and scalability.

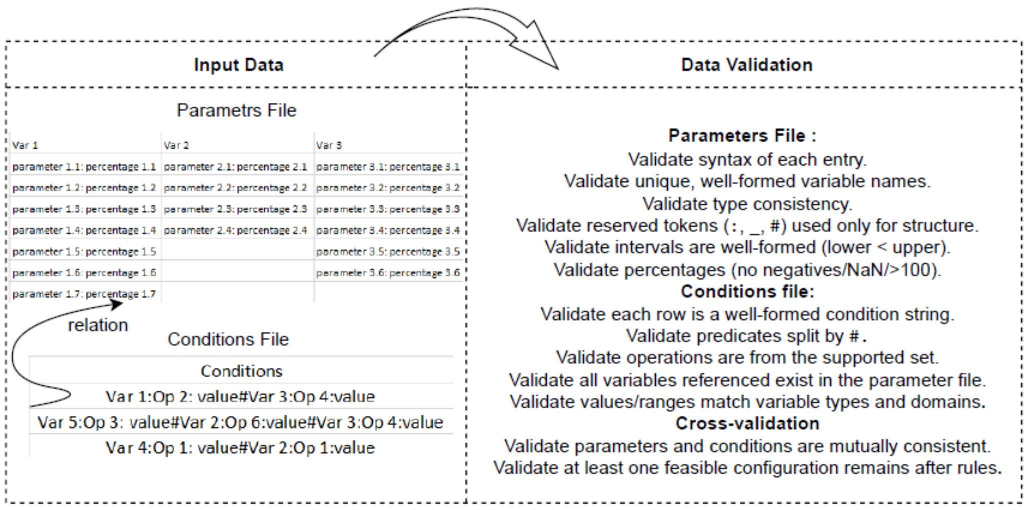
Input Data
Data Validation
Parametrs File
Var 1
Var 2
Var 3
parameter 1.1: percentage 1.1
parameter 2.1: percentage 2.1
parameter 3.1: percentage 3.1
parameter 1.7: percentage 1.7
relation
Conditions File
Conditions
Var 1:Op 2: value#Var 3:Op 4:value
Var 5:Op 3: value#Var 2:Op 6:value#Var 3:Op 4:value
Var 4:Op 1: value#Var 2:Op 1:value
Parameters File :
Validate syntax of each entry.
Validate unique, well-formed variable names.
Validate type consistency.
Validate reserved tokens (:, _, #) used only for structure.
Validate intervals are well-formed (lower < upper).
Validate percentages (no negatives/NaN/>100).
Conditions file:
Validate each row is a well-formed condition string.
Validate predicates split by #.
Validate operations are from the supported set.
Validate all variables referenced exist in the parameter file.
Validate values/ranges match variable types and domains.
Cross-validation
Validate parameters and conditions are mutually consistent.
Validate at least one feasible configuration remains after rules.

Fig. 1: Input Data Structure and Validation Workflow.

- Parameter File

The parameter file defines the tabular schema and specifies the target marginal distributions for each variable to be synthesized. It serves as the main configuration that governs how synthetic values are sampled, as illustrated in Table 1 where each column represents a variable and each row encodes admissible values together with their associated percentages.

Table 1. Structure of the parameter file showing variables, admissible values, and associated percentages.

Var 1	Var 2	Var 3
parameter 1.1: percentage 1.1	parameter 2.1: percentage 2.1	parameter 3.1: percentage 3.1
parameter 1.2: percentage 1.2	parameter 2.2: percentage 2.2	parameter 3.2: percentage 3.2
parameter 1.3: percentage 1.3	parameter 2.3: percentage 2.3	parameter 3.3: percentage 3.3
parameter 1.4: percentage 1.4	parameter 2.4: percentage 2.4	parameter 3.4: percentage 3.4
parameter 1.5: percentage 1.5		parameter 3.5: percentage 3.5
parameter 1.6: percentage 1.6		parameter 3.6: percentage 3.6
parameter 1.7: percentage 1.7		

The file is organized as a spreadsheet (Excel or CSV) with one column per variable. Each column contains the admissible values for the variable together with their associated target percentages. Supported specifications include categorical labels for discrete variables, fixed numeric values, and numeric intervals for continuous ranges.

For each variable, the sum of all percentages is constrained to equal 100%. Intervals expressed with percentage ranges (e.g., 10_15) are resolved during generation to ensure the final distribution is statistically consistent. Certain special characters are reserved for structural encoding and therefore cannot be used in raw labels. This avoids parsing ambiguity and ensures compatibility across processing stages. The reserved tokens are summarized in Table 2.

Table 2. Reserved tokens in the parameter and conditions file.

Token	Function in encoding	Restriction
:	Separates value from percentage	Cannot appear in variable names, labels, or parameter values
-	Represents numeric intervals (lower_upper)	Cannot appear in variable names or categorical labels
#	Logical conjunction in conditions	Cannot appear in variable names or parameter values

The parameter file implicitly constrains admissible ranges by omitting values outside the defined domains. These bounds act as hard constraints during sampling, ensuring that generated data remain consistent with the intended distributions.

- Conditions File

The conditions file specifies row-level exclusion rules that prevent the generation of invalid or undesired combinations of values across variables. It complements the parameter file by ensuring that synthetic tabular data remain not only statistically coherent but also logically consistent with domain-specific constraints, as illustrated in Table 3, where each row encodes one forbidden configuration expressed as a conjunction of predicates. The file is provided as a CSV with a single column named conditions. Each row contains one complete rule expressed as a structured string. When multiple predicates must be combined, they are concatenated using the reserved symbol #, which denotes logical conjunction.

Each predicate in the rule follows the form: Variable:operation:value. Where the operation is drawn from a predefined set, including equality, inequality, order relations, and interval membership. The presence of # indicates that all listed predicates must be simultaneously satisfied for the rule to apply.

During generation, each candidate record is evaluated against the list of conditions. If the values assigned to its variables satisfy all predicates within a condition row, that record is flagged as invalid and excluded from the final dataset. Thus, every line in the conditions file defines one forbidden configuration of values.

To ensure efficiency, the conditions are parsed once at initialization and converted into numeric and categorical checks. Operations involving continuous variables are evaluated with tolerance thresholds, while categorical operations are resolved through encoded lookups. This enables the enforcement of conditions at scale in both sequential and parallel generation modes.

The conditions file operates together with the parameter specifications through a two-step validation process. For every candidate row, the first step is to verify the percentage constraints, ensuring that the selected values are permitted according to the predefined distributions. Once this check is satisfied, the second step is to evaluate the same row against the logical rules defined in the conditions file. Only if both validations are successful is the row accepted into the synthetic dataset; otherwise, it is discarded and the generation process proceeds to the next candidate. This layered mechanism guarantees that every generated record simultaneously respects the statistical distributions and complies with the logical constraints.

Table 3. Structure of the conditions file showing exclusion rules as conjunctions of predicates.

Conditions
Var 1:Op 2: value#Var 3:Op 4:value
Var 5:Op 3: value#Var 2:Op 6:value#Var 3:Op 4:value
Var 4:Op 1: value#Var 2:Op 1:value

- Input Data Validation Step

Before the generation process begins, all input files undergo a validation stage to guarantee correctness and coherence. For the parameter file, validation ensures that the syntax is respected, reserved tokens are used only in their structural roles, and variable definitions are type-consistent.

Percentage feasibility is verified as follows: when only fixed percentages are present, their sum must equal 100%; when intervals are present, the sum of fixed percentages plus the lower bounds of all ranges must be at most 100%, and the sum with upper bounds must be at least 100%, ensuring a consistent allocation is possible. Intervals are also checked to be well-formed, non-overlapping, and non-contiguous, since connected ranges would create ambiguity in sampling.

For the conditions file, each row must follow the required syntax, with predicates separated by the reserved connector, and all referenced variables must exactly match those declared in the parameter file. Operations must belong to the predefined set, values must be consistent with variable types, and categorical or numeric references must lie within admissible domains. This dual validation guarantees that percentage distributions are statistically feasible, intervals are structurally sound, and logical rules are syntactically correct and semantically aligned with the parameter specifications.

4.2. Tools and Development Environment

Programming Language – Python: The framework was implemented in Python, an open-source and object-oriented language that is widely used in data science and high-performance computing. Python offers extensive libraries for data manipulation and numerical analysis while maintaining readability and modularity, which makes it an ideal choice for developing configurable and extensible synthetic data generation systems (Del Gobbo, 2025; Joshi & Tiwari, 2023).

Acceleration Tools – CUDA: CUDA was employed to exploit the parallel architecture of the GPU. By enabling thousands of threads to run concurrently, CUDA significantly accelerates core operations such as random sampling, distribution enforcement, and constraint validation. This parallelism allows the system to scale efficiently from small datasets to millions of tabular records without compromising performance (Garland et al., 2008).

Compilation and Kernel Programming – Numba: Numba complements CUDA by compiling Python functions into highly optimized machine code through just-in-time (JIT) compilation. This makes it possible to write computational kernels in Python while achieving near-native GPU execution speed (Garland et al., 2008; Oden, 2020).

4.3. Sequential Data Generation (Baseline)

The baseline adopts a single-threaded, rejection-sampling workflow that produces one record at a time. It treats the user specifications as two complementary contracts: a statistical contract, defined by per-variable target marginals (percentages and numeric intervals), and a logical contract, defined by row-level exclusion rules. Each candidate row is drawn from the declared marginals and is admitted only if it passes a two-step validation: first, it must be permissible under the percentage specifications; second, it must satisfy all logical constraints (i.e., not match any forbidden pattern). The overall workflow of sequential data generation, including data preparation, percentage verification, and conditions validation, is illustrated in Fig 2.

Before generation starts, the specification is parsed once to establish an internal schema comprising variable names, types (categorical, numeric-point, numeric-interval), admissible domains, and normalized target percentages. Percentage ranges are checked for feasibility and converted into concrete weights so that each variable's mass sums to 100%. For sampling, categorical variables are associated with discrete distributions over their labels. Numeric-point variables are sampled from finite supports. Numeric-interval variables are sampled by first selecting an interval according to its weight, then drawing a value within that interval using a chosen intra-interval distribution. In parallel, the conditions file is compiled from its single "conditions" column into conjunctions of atomic predicates with well-defined operators; categorical terms are mapped to encodings for fast lookup, and numeric comparisons are equipped with tolerances to avoid floating-point artifacts. A fixed random seed and a snapshot of the configuration ensure reproducibility.

Generation proceeds row by row. For each record, values are first proposed independently according to the per-variable target marginals, which guarantees that the candidate is allowed by the parameter file (percentage step). The same candidate is then evaluated against the compiled predicates from the conditions file (conditions step). If any rule is fully satisfied, the candidate is rejected and resampled; otherwise, the record is accepted and written to the output. This loop continues until the requested sample size is reached. When specifications are tight and rejection rates rise, the loop remains correct—no invalid rows are emitted—at the cost of additional proposals; the process logs rejection statistics to make potential conflicts visible.

The design explicitly separates statistical enforcement from logical enforcement. Sampling from normalized marginals drives convergence of the empirical distributions toward the declared percentages as the dataset grows, while interval geometry validated upfront (non-overlap and non-contiguity) ensures unambiguous numeric draws inside ranges. Post-sampling rule checks guarantee compliance with forbidden combinations without distorting the marginals specified by the parameter file. To support verification, the baseline reports marginal gaps (observed minus target percentages) during and after generation, along with rule-hit frequencies identifying which constraints most often trigger rejections. Because the procedure is single-threaded with a fixed seed and a fixed configuration snapshot, it is fully reproducible and serves as a reference implementation against which the GPU-accelerated variant can be compared for both correctness and performance.

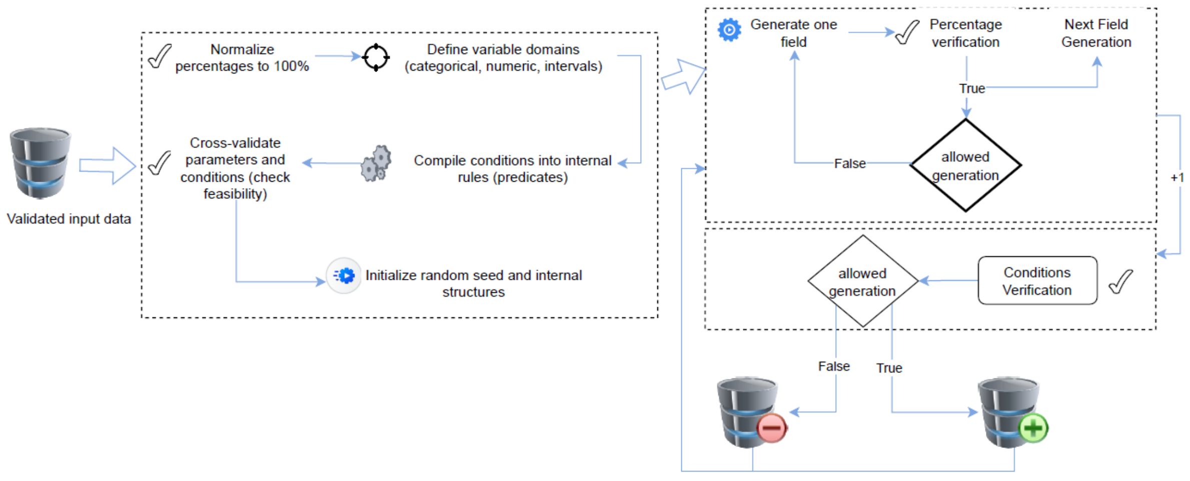
Validated input data
Normalize percentages to 100%
Define variable domains (categorical, numeric, intervals)
Cross-validate parameters and conditions (check feasibility)
Compile conditions into internal rules (predicates)
Initialize random seed and internal structures
Generate one field
Percentage verification
Next Field Generation
True
False
allowed generation
+1
allowed generation
Conditions Verification
False
True

Fig. 2: Workflow of sequential data generation showing data preparation, percentage verification, and conditions validation steps

4.4. Parallel Data Generation: Proposed Acceleration

The proposed acceleration extends the sequential baseline by embedding the entire generation process into a GPU kernel executed in parallel. This approach leverages CUDA through the Numba library to distribute the generation workload across thousands of threads, where each thread is responsible for generating a fixed batch of rows (1,000 rows per thread, depending on the configuration). By parallelizing both the sampling of values and the enforcement of constraints, the method achieves massive speedups while preserving correctness as shown in Figure 3.

The process begins with preparing the input specifications as a nested list structure adapted for GPU processing, ensuring that every thread can access variables and their parameters in a uniform way. The overall structure is organized as a list of lists of lists: each outer list corresponds to a variable, each sub-list contains all of that variable's parameters, and each parameter is itself represented as a small record of five fields. These fields store the variable name (as a coded identifier), the value of the parameter, the upper bound if the parameter is an interval (or an epsilon placeholder if not), a flag indicating whether the parameter is treated as a floating-point value (1) or not (0), and the target percentage weight assigned to it. To satisfy GPU array alignment requirements, all parameter sub-lists are padded with zero-records so that each variable has the same number of parameters, making the structure regular and directly translatable into a device array. This preparation guarantees that the GPU kernel can index parameters consistently across all threads, without irregular memory access.

In parallel, the conditions file is transformed into GPU-compatible encodings by converting each predicate into a numerical representation consisting of the variable identifier, the operator code, and one or two associated values. Strings are mapped into integer codes, operators are replaced with symbolic identifiers, and interval checks are represented by paired bounds. To allow parallel evaluation, each rule is padded to a uniform size with placeholder values, producing regular arrays that can be broadcast to all threads. GPU kernel functions do not support string manipulation, and most high-level Python methods are unavailable in Numba's kernel environment. By relying only on numerical representations, the system ensures that both sampling and rule checking can be executed entirely on the GPU, eliminating costly data transfers and enabling efficient large-scale parallel generation.

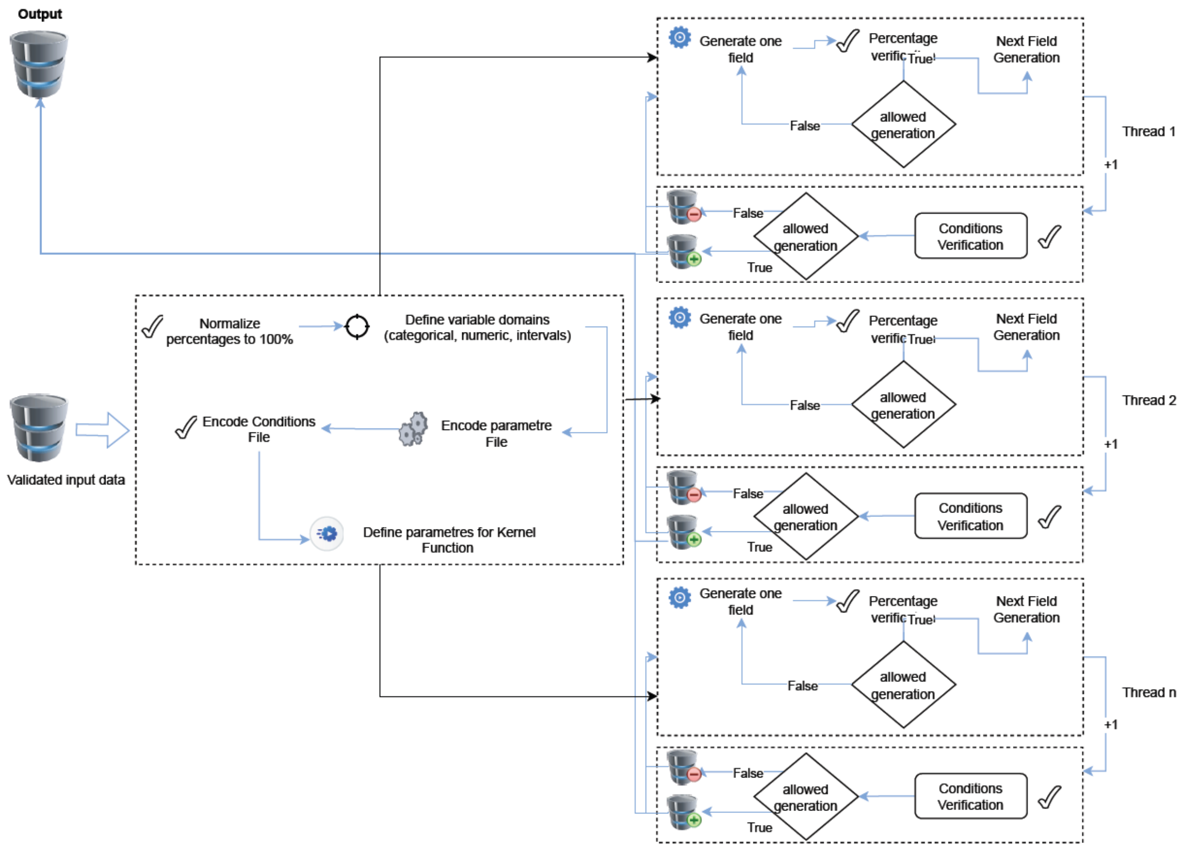

Fig. 3: Workflow of parallel data generation showing data preparation, thread-level field generation, percentage verification, and conditions validation executed concurrently across GPU threads.

During generation, the GPU kernel executes with a grid of blocks and threads that is dynamically configured according to the number of rows to generate. The total number of threads is computed as n/1000, meaning that each thread is responsible for generating 1,000 rows. Threads are grouped into blocks of 10, and the number of blocks is determined by the following formula:

$$blocksPerGrid = \frac{numThreads + threadsPerBlock - 1}{threadsPerBlock}$$

This adaptive configuration ensures that the workload is evenly distributed across the GPU and scales automatically with the dataset size. Random values are produced using the XOROSHIRO128+ generator (Haramoto et al., 2022) from Numba's CUDA random library, ensuring statistical independence between threads. For each variable, the thread selects either a categorical label, a numeric point, or a numeric interval, and assigns a value consistent with the target percentages. A local check guarantees that the percentage allocation for each variable does not exceed the declared limits, preventing drift in the marginals.

After values are proposed, the conditions are evaluated directly inside the GPU kernel. Each candidate row is tested against all compiled rules. If any rule is violated, the row is rejected immediately within the GPU, avoiding the need for post-processing on the CPU. This design ensures efficiency by minimizing host–device communication and guaranteeing that only valid rows are written to the GPU result buffer. To avoid infinite rejection loops when constraints are tight, the kernel includes an iteration cap, after which rows are reset if no feasible combination is found.

Once generation is complete, the results are decoded back to their original representations. Numerical codes are mapped back to variable names and categorical labels, while numeric values are formatted according to their original schema. The final dataset is returned as a tabular structure that mirrors the input configuration. This decoding ensures interpretability for users while preserving the computational advantages of numerical encoding during GPU execution.

The following section demonstrates this framework through a concrete IT infrastructure case study.

4.5. Application Example

The IT infrastructure parameters file used for the application examples (Table 4) was designed to emulate a realistic cloud and data-center environment, capturing both operational metrics and system configurations relevant to resource management and workload distribution. The parameters file includes five primary variables—cpu_usage, region, status, memory_usage, and workload—each chosen to represent a key dimension of virtualized computing systems. The variable cpu_usage models the distribution of virtual CPU consumption (vCPUs) across active instances, serving as a continuous indicator of computational load. region denotes the deployment zone or data-center location, introducing categorical diversity and reflecting geographical heterogeneity that influences latency and capacity allocation. The status variable encodes the lifecycle state of each instance—such as active or paused—capturing dynamic behavior and resource transitions that are essential for operational analyses. Similarly, memory_usage represents the distribution of RAM consumption and complements cpu_usage to characterize workload intensity and efficiency. Finally, workload identifies the type of application or service running on the virtual machine, introducing semantic variability and allowing logical constraints to be defined between workload types, performance metrics, and operational states.

Table 4. Structure of the parameter file showing the five variables— cpu_usage, Status memory_usage, Region, and workload —along with their admissible values or intervals and the associated target percentages that define the statistical distributions used during data generation.

cpu-usage	region	status	memory-usage	workload
0.1_2.0:20.5	EU-West:25	Active:50	0.1_4.0:15	dev:20.5
2.1_50:49.5	US-East:10.5	Paused:50	4.1_16.0:15_50	devops:69.5
50.1_80:10	APAC:34.5		16.1_24.0:5_50	sre:10
81_100:20	EU-Central:30		24.1_32.0:15	
			32.1_64.0:15	
			64.1_128.0:0_50	

This dataset (Table 4) was chosen to demonstrate the completeness of the developed generation framework. It incorporates categorical, numeric, and interval-based variables, allowing the system to utilize all available generation mechanisms: discrete sampling, continuous interval generation, and mixed-type processing. The structure also supports multiple logical dependencies, making it ideal for testing constraint enforcement. The parameter file defines the target distributions for each variable, while the associated conditions file (Table 5) encodes the logical rules that ensure consistency between attributes.

Table 5. Structure of the conditions file illustrating the logical exclusion rules expressed as conjunctions of predicates. Each row represents one forbidden configuration combining variables, operators, and values to ensure logical and operational consistency in the generated IT infrastructure data.

conditions
memory-usage:include:0.1 4.0#status:equal:Active#region:equal:EU-Central
status:equal:Active#cpu-usage:inf:2
cpu-usage:inf:15#workload:equal:dev

As described in Section 4.3, each row in the conditions file defines a logical rule specifying value combinations to exclude, using the format Variable : operation : value with predicates connected by the # symbol.

For example, the first rule excludes records with memory usage between 0.1 GB and 4.0 GB, where the status is Active, and the region is EU-Central. This ensures that low-memory active instances aren't assigned to high-performance regions. The second rule filters out records where the status is Active and CPU usage is below 2%, ensuring active instances always have some CPU activity. The third rule excludes records where CPU usage is below 15% and the workload type is development, ensuring sufficient resources for development tasks.

During the experiment, 1,000,000 rows were generated using both sequential and parallel methods. The sequential method took 8990 seconds, while the parallel method completed in 20 seconds, achieving a 449.5× speedup. This demonstrates the effectiveness of GPU parallelization and the scalability of the approach.

To ensure data quality, we applied distribution verification and condition verification. The first ensures parameter frequencies match the target specifications, while the latter checks that all exclusion rules and dependencies are respected, guaranteeing the dataset’s validity and consistency regardless of the generation method.

4.6. Conceptual Contributions of the Framework

This paper introduces three main conceptual contributions. First, the specification-driven generation mechanism enables users to precisely define and enforce statistical distributions and logical constraints, providing a level of control that learned generative models do not guarantee. Second, the formalization of domain rules as explicit exclusion constraints ensures logical consistency and reproducibility in generated datasets. Third, the framework supports reproducible system stress testing under controlled conditions, enabling large-scale benchmarking and validation of system performance. Together, these contributions make the framework a valuable tool for domains requiring governed, auditable synthetic data generation.

5. Results and Discussion

The validation of the parallel generation method shows that the produced dataset accurately reproduces the target distributions defined in the parameter file, as shown in Figure 4. For region, status, workload, and cpu_usage, the empirical proportions match the specified percentages, confirming that the parallel generator preserves both the categorical and interval-based distributions of these variables. For memory_usage, all intervals satisfy the prescribed constraints: fixed-percentage bins are generated at the expected 15%, while flexible intervals each fall within their admissible percentage ranges (see Figure 4 for details). In combination with the condition checker, which confirms that no forbidden configurations appear in the one-million-row dataset, these results demonstrate that the parallel method respects both the distributional specifications and the logical constraints encoded in the generation schema.

For the sequential generation method, the empirical distributions of region, status, workload, cpu_usage, and memory_usage are consistent with the target specifications, as illustrated in Figure 5. In particular, the memory-usage intervals that have admissible ranges (4.1–16.0 GB, 16.1–24.0 GB, and 64.1–128.0 GB) fall within their prescribed percentage bounds, confirming that the sequential sampler correctly handles flexible intervals. As in the parallel setting, all exclusion rules are satisfied, and no forbidden configurations are observed, indicating that the sequential generator also preserves both the statistical distributions and the logical constraints encoded in the schema. Table 6 summarizes these results, comparing the two approaches in terms of distribution fidelity, condition satisfaction, generation time, and the number of generated null rows.

Table 6. Comparison between the sequential and parallel generation methods

Aspect	Sequential method	Parallel method
Distribution fidelity	Matches target distributions for all variables; flexible memory-usage intervals fall within admissible ranges	Matches target distributions for all variables; flexible memory-usage intervals fall within admissible ranges
Conditions satisfaction	All exclusion rules satisfied; no forbidden configurations detected	All exclusion rules satisfied; no forbidden configurations detected
Generation time (1,000,000 rows)	8990 s	20 s
Generated null rows	119 rows	77 rows

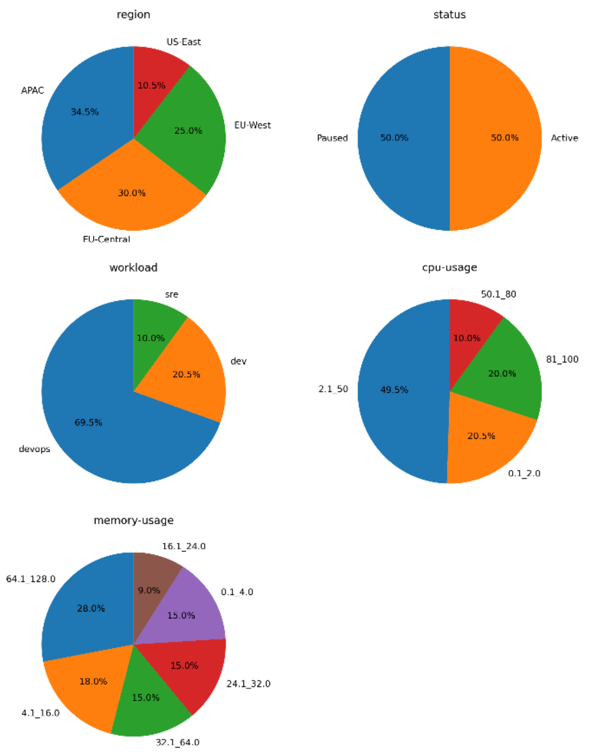

Fig. 4. Distribution of the five parameters generated by the parallel method

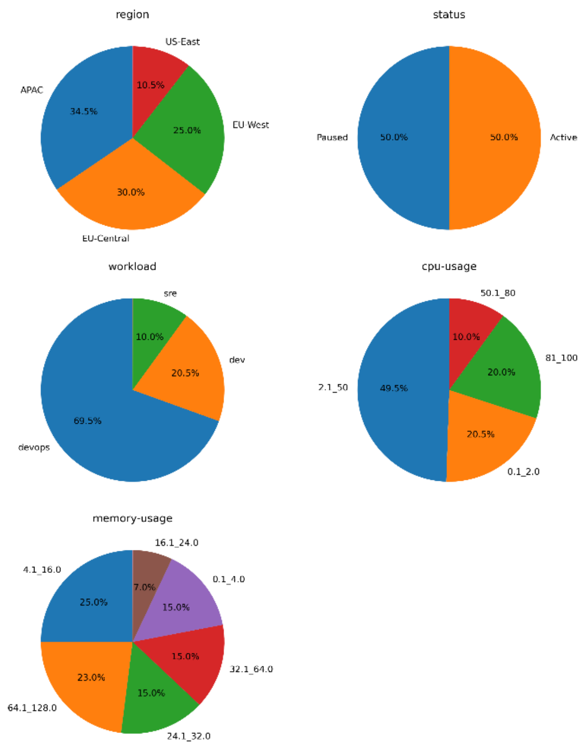

Fig. 5. Distribution of the five parameters generated by the sequential method

The results obtained on the IT infrastructure case study show that the proposed framework can generate large tabular datasets while remaining consistent with the specification encoded in the parameter and conditions files. Both the sequential and parallel methods reproduce the intended distributions and satisfy all exclusion rules, which indicates that the combination of distribution verification and condition checking is effective for controlling the behavior of the generator without manual post-filtering.

Beyond this correctness aspect, the main difference between the two methods lies in performance and robustness. As reported in Table 6, the GPU-based parallel method reduces the generation time by several orders of magnitude while maintaining the same level of distribution fidelity and constraint satisfaction as the sequential baseline. The number of null rows remains very low for both approaches and slightly lower for the parallel method, suggesting that the acceleration does not introduce additional artifacts in the generated data.

These observations suggest that the framework can be used in settings where large volumes of synthetic IT infrastructure data are needed, such as stress testing, benchmarking, or machine learning experiments. At the same time, the current evaluation is limited to one parameter configuration and a relatively simple set of exclusion rules. Future work should investigate more complex dependency patterns, additional application domains, and the impact of the generated data on downstream tasks, such as prediction accuracy or anomaly detection performance.

To further evaluate the scalability and applicability of the framework, we conducted additional case studies across different domains. As described in Table 7, the following summarizes the number of features, conditions, data generated, and the corresponding generation times for both sequential and parallel methods. The results are summarized in Table 7.

Table 7. Performance Comparison Across Different Case Studies

Case Study	**Number of Features**	**Number of Conditions**	**Number of Data Generated**	**Sequential Method Time (s)**	**Parallel Method Time (s)**
IT Infrastructure	5	3	1,000,000	8990	20
Smart Building	31	14	1,000,000	20101	42
Smart Manufacturing	25	10	1,000,000	15230	38
Healthcare System	20	8	1,000,000	15040	30
Financial Services (Fraud Detection)	15	5	1,000,000	9800	22

The development of the parallel generation method raised several practical difficulties related to debugging on the GPU. Unlike standard Python code, CUDA kernels do not support interactive, step-by-step inspection, which limits the possibility of observing intermediate states in real time and slows down error localization. As a result, some defects, such as infinite loops or incorrect termination conditions, were harder to detect and reproduce. In addition, type handling inside the kernel function required special care: small inconsistencies in the representation of indices, counters, or probability values could lead to silent failures or incorrect behavior that were not immediately visible from the host side. These limitations increased the effort needed to stabilize the parallel implementation compared with the sequential baseline.

The framework, while robust, may encounter potential failure modes and system-level risks, including infeasible specifications where user-defined constraints conflict, leading to the inability to generate valid data. Contradictory constraints can cause rejection loops or prevent data generation altogether. Additionally, biased distributions may arise if parameters don't accurately represent real-world data or domain-specific characteristics. User misconfiguration is another risk, potentially resulting in invalid data or failure to enforce intended constraints. While the framework includes validation checks to detect these issues, further work is needed to address and mitigate these risks, particularly as system complexity and dataset size increase.

6. Conclusions and future work

This work presented a configurable framework for synthetic tabular data generation driven by user-defined parameters and logical conditions. The framework was implemented in two variants: a sequential baseline and a GPU-accelerated parallel method. Through a series of case studies, including an IT infrastructure use case, as well as smart building data, the experiments demonstrated that both methods successfully reproduced the target marginal distributions for all variables and adhered to all exclusion rules defined in the conditions file. The validation pipeline, based on distribution checking and condition verification, effectively controlled the generator and ensured that invalid configurations were avoided. Notably, the parallel implementation significantly reduced the generation time for one million rows, from 8990 s to just 20 s, while maintaining a low number of null rows that were comparable to the sequential baseline. These results, summarized in Table 7, confirm the framework's scalability and efficiency across domains and complexity levels.

These results have direct implications for IT infrastructure applications. The framework can support data-driven model training when real operational logs are scarce or sensitive, enable large-scale stress testing with controlled distributions and dependencies, and provide a transparent mechanism for encoding domain rules that aligns synthetic data with operational policies. Future work may extend the approach to richer dependency structures and additional variable types, and include downstream evaluations of impact on predictive maintenance and decision support.

The current framework has several limitations. First, the management of logical conditions remains partly manual. The system does not yet detect contradictory or mutually exclusive rules in a systematic way. As a result, a set of conditions may block the generator or drastically reduce the effective sampling space when two constraints cannot be satisfied at the same time. Identifying such situations still requires manual inspection and trial-and-error runs. Second, the evaluation was carried out on a single IT infrastructure case study with a moderate number of variables and a relatively simple rule set. More complex dependency structures, larger condition sets, and heterogeneous data types may expose additional weaknesses in scalability or expressiveness.

Future work will address these limitations in several directions. A first step will be to integrate constraint programming (Echeverria et al., 2025; Krings et al., 2020) to model the conditions more formally. This will allow the system to analyse the constraints before generation, detect unsatisfiable or contradictory rules, and report whether a given configuration of conditions makes data generation impossible or severely restricted. Finally, applying the approach to multiple domains and datasets will help assess its generality and guide further improvements in performance, usability, and validation procedures.

Acknowledgements

This work was supported in part by the Ministry of Higher Education, Scientific Research, and Innovation, and in part by the National Center for Scientific and Technical Research (CNRST) of Morocco under the PhD-Associate Scholarship (PASS) Program.

Data Availability

The reproduction package for the study containing datasets and scripts used to perform the analyses (with the exclusion of confidential data) is publicly available. Reproduction package – https://doi.org/10.5281/zenodo.17631796.

Appendix A: GPU Configuration

All experiments were executed on an NVIDIA Quadro M4000 equipped with 8 GB of GDDR5 memory. The system ran with driver version 570.86.10 and CUDA version 12.8. During the experiments, the GPU operated in performance state P0 and reached a temperature of 58 °C. Power usage peaked at 73 W out of a 120 W capacity. The workload used approximately 118 MiB of device memory, and GPU utilization reached 100% during execution.

References

Bansal, Ms. A., Sharma, Dr. R., & Kathuria, Dr. M. (2022). A Systematic Review on Data Scarcity Problem in Deep Learning: Solution and Applications. *ACM Computing Surveys*, *54*(10s), 1–29. https://doi.org/10.1145/3502287

Barr, A. A., Rozman, R., & Guo, E. (2025). *Generative adversarial networks vs large language models: A comparative study on synthetic tabular data generation*. arXiv. https://doi.org/10.48550/ARXIV.2502.14523

Bautista, E., Romanus, M., Davis, T., Whitney, C., & Kubaska, T. (2019). Collecting, Monitoring, and Analyzing Facility and Systems Data at the National Energy Research Scientific Computing Center. *Workshop Proceedings of the 48th International Conference on Parallel Processing*, 1–9. https://doi.org/10.1145/3339186.3339213

Cardoso, R., Golubovic, D., Lozada, I. P., Rocha, R., Fernandes, J., & Vallecorsa, S. (2021). *Accelerating GAN training using highly parallel hardware on public cloud*. arXiv.

https://doi.org/10.48550/ARXIV.2111.04628

Chen, D., & Zhao, H. (2012). Data Security and Privacy Protection Issues in Cloud Computing. *2012 International Conference on Computer Science and Electronics Engineering*, 647–651. https://doi.org/10.1109/ICCSEE.2012.193

Del Gobbo, C. (2025). *A Comparative Study of Open-Source Libraries for Synthetic Tabular Data Generation: SDV vs. SynthCity*. arXiv. https://doi.org/10.48550/ARXIV.2506.17847

Echeverria, I., Murua, M., & Santana, R. (2025). Leveraging constraint programming in a deep learning approach for dynamically solving the flexible job-shop scheduling problem. *Expert Systems with Applications*, *265*, 125895. https://doi.org/10.1016/j.eswa.2024.125895

European Parliament. Directorate General for Parliamentary Research Services. (2020). *The impact of the general data protection regulation on artificial intelligence.* Publications Office. https://data.europa.eu/doi/10.2861/293

Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., & Volkov, V. (2008). Parallel Computing Experiences with CUDA. *IEEE Micro*, *28*(4), 13–27. https://doi.org/10.1109/MM.2008.57

Gogoshin, G., Branciamore, S., Rodin, A. S., & Department of Computational and Quantitative Medicine, Beckman Research Institute, and Diabetes and Metabolism Research Institute, City of Hope National Medical Center, 1500 East Duarte Road, Duarte, CA 91010 USA. (2021). Synthetic data generation with probabilistic Bayesian Networks. *Mathematical Biosciences and Engineering*, *18*(6), 8603–8621. https://doi.org/10.3934/mbe.2021426

Haramoto, H., Matsumoto, M., & Saito, M. (2022). Unveiling patterns in xorshift128+ pseudorandom number generators. *Journal of Computational and Applied Mathematics*, *402*, 113791. https://doi.org/10.1016/j.cam.2021.113791

Hosni, H. (2025). SECURE PREDICTIVE MAINTENANCE FOR INDUSTRIAL SYSTEMS USING FEDERATED LEARNING. *Journal of Innovations in Business and Industry*, *4*(1), 69–78. https://doi.org/10.61552/JIBI.2026.01.007

Joshi, A., & Tiwari, H. (2023). An Overview of Python Libraries for Data Science. *Journal of Engineering Technology and Applied Physics*, *5*(2), 85–90. https://doi.org/10.33093/jetap.2023.5.2.10

Kamthe, S., Assefa, S., & Deisenroth, M. (2021). *Copula Flows for Synthetic Data Generation*. arXiv. https://doi.org/10.48550/ARXIV.2101.00598

Khalil, M., Vadiee, F., Shakya, R., & Liu, Q. (2025). Creating Artificial Students that Never Existed: Leveraging Large Language Models and CTGANs for Synthetic Data Generation. *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, 439–450. https://doi.org/10.1145/3706468.3706523

Kiran, A., Rubini, P., & Kumar, S. S. (2025). Challenges and Limitations of TVAE Tabular Synthetic Data Generator. In D. Garg, V. Pendyala, S. K. Gupta, & M. Najafzadeh (Eds.), *Advanced Computing* (Vol. 2434, pp. 243–254). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-84602-1_17

Kotelnikov, A., Baranchuk, D., Rubachev, I., & Babenko, A. (2022). *TabDDPM: Modelling Tabular Data with Diffusion Models*. https://doi.org/10.48550/ARXIV.2209.15421

Krings, S., Schmidt, J., Skowronek, P., Dunkelau, J., & Ehmke, D. (2020). Towards Constraint Logic Programming over Strings for Test Data Generation. In P. Hofstedt, S. Abreu, U. John, H. Kuchen, & D. Seipel (Eds.), *Declarative Programming and Knowledge Management* (Vol. 12057, pp. 139–159). Springer International Publishing. https://doi.org/10.1007/978-3-030-46714-2_10

Li, J. (2025). Distributed data processing and task scheduling based on GPU parallel computing. *Neural Computing and Applications*, *37*(4), 1757–1769. https://doi.org/10.1007/s00521-024-10489-4

Li, Z., Huang, Q., Yang, L., Shi, J., Yang, Z., van Stein, N., Bäck, T., & van Leeuwen, M. (2025). *Diffusion Models for Tabular Data: Challenges, Current Progress, and Future Directions*. arXiv. https://doi.org/10.48550/ARXIV.2502.17119

Mora-de-León, L. P., Solís-Martín, D., Galán-Páez, J., & Borrego-Díaz, J. (2025). Text-Conditioned Diffusion-Based Synthetic Data Generation for Turbine Engine Sensor Analysis and RUL Estimation. *Machines*, *13*(5), 374. https://doi.org/10.3390/machines13050374

Mühlhoff, R. (2023). Predictive privacy: Collective data protection in the context of artificial intelligence and big data. *Big Data & Society, 10*(1), 20539517231166886. https://doi.org/10.1177/20539517231166886

Oden, L. (2020). Lessons learned from comparing C-CUDA and Python-Numba for GPU-Computing. *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP),* 216–223. https://doi.org/10.1109/PDP50117.2020.00041

Parise, O., Kronenberger, R., Parise, G., De Asmundis, C., Gelsomino, S., & La Meir, M. (2025). CTGAN-driven synthetic data generation: A multidisciplinary, expert-guided approach (TIMA). *Computer Methods and Programs in Biomedicine*, *259,* 108523. https://doi.org/10.1016/j.cmpb.2024.108523

Pathare, A., Mangrulkar, R., Suvarna, K., Parekh, A., Thakur, G., & Gawade, A. (2023). Comparison of tabular synthetic data generation techniques using propensity and cluster log metric. *International Journal of Information Management Data Insights*, *3*(2), 100177. https://doi.org/10.1016/j.jjimei.2023.100177

Shi, J., Xu, M., Hua, H., Zhang, H., Ermon, S., & Leskovec, J. (2024). *TabDiff: A Mixed-type Diffusion Model for Tabular Data Generation*. https://doi.org/10.48550/ARXIV.2410.20626

Siddique, M., & Ashour, M. W. (2024). Parallel Computing with GPU: An accelerator for Data-Centric High Performance Computing. *2024 1st International Conference on Innovative Engineering Sciences and Technological Research (ICIESTR),* 1–6. https://doi.org/10.1109/ICIESTR60916.2024.10798202

Stoian, M. C., Giunchiglia, E., & Lukasiewicz, T. (2025). *A Survey on Tabular Data Generation: Utility, Alignment, Fidelity, Privacy, and Beyond*. arXiv. https://doi.org/10.48550/ARXIV.2503.05954

Suh, N., Lin, X., Hsieh, D.-Y., Honarkhah, M., & Cheng, G. (2023). *AutoDiff: Combining Auto-encoder and Diffusion model for tabular data synthesizing*. arXiv. https://doi.org/10.48550/ARXIV.2310.15479

Takahashi, T., & Mizuno, T. (2025). Generation of synthetic financial time series by diffusion models. *Quantitative Finance*, *25*(10), 1507–1516. https://doi.org/10.1080/14697688.2025.2528697

Villaizán-Vallelado, M., Salvatori, M., Segura, C., & Arapakis, I. (2025). Diffusion Models for Tabular Data Imputation and Synthetic Data Generation. *ACM Transactions on Knowledge Discovery from Data*, *19*(6), 1–32. https://doi.org/10.1145/3742435

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). *Modeling Tabular data using Conditional GAN.* arXiv. https://doi.org/10.48550/ARXIV.1907.00503

Yadav, P., Gaur, M., Madhukar, R. K., Verma, G., Kumar, P., Fatima, N., Sarwar, S., & Dwivedi, Y. R. (2024). Rigorous Experimental Analysis of Tabular Data Generated using TVAE and CTGAN. *International Journal of Advanced Computer Science and Applications*, *15*(4). https://doi.org/10.14569/IJACSA.2024.01504125

Zhang, D. (2018). Big Data Security and Privacy Protection. *Proceedings of the 8th International Conference on Management and Computer Science (ICMCS 2018)*. 8th International Conference on Management and Computer Science (ICMCS 2018), Shenyang, China. https://doi.org/10.2991/icmcs-18.2018.56