UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

APPLYING DIFFERENTIAL PRIVACY TO GENERATE PRIVATE HPC WORKLOAD TRACES

DANIEL LÓPEZ GARCÍA

Thesis supervisor
MANUEL GIMÉNEZ DE CASTRO MARCIANI (BARCELONA SUPERCOMPUTING CENTER-CENTRO NACIONAL DE SUPERCOMPUTACION)

Tutor: GLADYS MIRIAM UTRERA IGLESIAS (Department of Computer Architecture)

Degree
Bachelor's Degree in Artificial Intelligence

Bachelor's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Thesis defense date: 30 of June of 2026

Abstract (English)

The increasing reliance on High-Performance Computing (HPC) systems for critical research raises concerns about protecting operational data. Workload logs are irreplaceable for optimizing and improving HPC systems, but their public release is severely restricted due to privacy risks. To address these issues, this thesis presents a differentially private (DP) data synthesis pipeline tailored for HPC job logs, developed during a research internship at the Barcelona Supercomputing Center (BSC) [2]. This pipeline combines several existing DP solutions, implements them, in some cases modifies them, and evaluates them in a real-world scenario. Said pipeline mainly consists in the integration of a modified implementation of PrivTree [49] for DP density-adaptive discretization (to better preserve the data's correlations) together with a DP Synthetic Data Generator (Maximum Spanning Trees [28]) for private data synthesis. Beyond these two main algorithms, the pipeline incorporates additional DP mechanisms, including an implementation of a DP pseudonymization to protect User and Group IDs, and a logarithmic transformation to allow the modified PrivTree to better discretize the skewed, long-tailed attributes typically found in HPC workloads. Thanks to this, the proposed pipeline proficiently generates synthetic traces, which are evaluated across a wide range of utility and privacy metrics. More importantly, practical applicability is validated by feeding the synthetic traces into a SLURM simulator to reproduce the results obtained from the real data in a different study (Autosubmit [25]). The results indicate that, while the generated traces preserves sufficiently well distributions and correlations to suggest potential applicability for machine learning, deep learning and exploratory data analysis (EDA) (as shown with the ML utility metrics and well-synthesized distribution plots), they lack substantial outliers for the simulation of reality-faithful traces. This is because, although the generated private traces capture central tendencies and typical patterns of a HPC workload, they fail to capture some of the nuances and extreme values resulting in a simulation of a trace without those extremes, without worst-case scenarios, while at the same time making mid case extremes (jobs bigger than usual but not extreme offshoots) more common. This shortcomings are attributable to the privacy preserving influence of DP, which suppresses outliers, smooths them, making them both less extreme and more common.

Keywords: differential privacy, HPC logs, data synthesis, privacy-preserving machine learning, BSC internship, MST, SLURM.

Abstract (Català)

La creixent dependència dels sistemes de computació d'altes prestacions (HPC, en el seu acrònim en anglès) per a investigacions planteja preocupacions sobre la protecció de les dades operatives. Els *workloads* són crucials per optimitzar i millorar els sistemes HPC, però la seva publicació pública està restringida a causa dels riscos de privacitat. Per solucionar aquests problemes, aquesta tesi presenta una *pipeline* de síntesi de dades amb privacitat diferencial (DP, en el seu acrònim en anglès) adaptat als registres de treballs d'HPC, desenvolupat durant les pràctiques al *Barcelona Supercomputing Center* (BSC) [2]. Aquesta *pipeline* combina de manera única solucions DP existents i les aplica de manera pràctica per avaluar-les. Principalment, la *pipeline* integra una implementació modificada de PrivTree [49] per a la discretització adaptativa per densitat (per preservar millor les correlacions de les dades) juntament amb un Generador de Dades Sintètiques amb DP basat en *Maximum Spanning Trees* [28] per a la síntesi de dades privades. Més enllà d'aquests dos algoritmes principals, el procés incorpora mecanismes DP addicionals, incloent una implementació de pseudonimització amb DP per protegir els ID d'Usuari i Grup, i una transformació logarítmica. Aquesta última permet que el PrivTree modificat discretitzi millor els atributs amb distribucions esbiaixades i de llargues cues típiques dels workloads d'HPC. Gràcies a això, la pipeline proposada genera traces sintètiques realistes. Aquestes dades sintètiques s'avaluen mitjançant un ampli ventall de mètriques d'utilitat i privacitat. De manera crucial, l'aplicabilitat pràctica es valida introduint els registres sintètics a un simulador de SLURM per reproduir els resultats obtinguts a partir de les dades reals en un estudi diferent (Autosubmit [25]). Els resultats indiquen que, tot i que els registres generats preserven bé les distribucions i correlacions com per suggerir que funcionen adequadament per a l'aprenentatge automàtic, l'aprenentatge profund i l'anàlisi exploratòria de dades (EDA, en el seu acrònim en anglès) (també mostrat per les mètriques de *Machine Learning utility* i els plots de distribucions ben sintetitzats), manquen de *outliers* substancials per a la simulació de registres fidels a la realitat. Els registres capturen les tendències centrals i els patrons típics d'una càrrega de treball d'HPC, però no aconsegueix capturar alguns dels matisos i valors extrems, donant com a resultat una simulació sense aquests moments extrems, aquests escenaris de pitjor cas (que són el focus de molts estudis), mentre que els casos intermedis (treballs més grans de l'habitual, però no desviacions extremes) es tornen més comuns. Aquestes mancances són atribuïbles a la influència de la privacitat diferencial, la qual suprimeix els valors atípics, els suavitza, fent que siguin tant menys extrems com més comuns.

Abstract (Castellano)

La creciente dependencia en sistemas de computación de altas prestaciones (HPC, en su acrónimo en inglés) para el desarrollo de investigaciones críticas plantea preocupaciones sobre la protección de los datos operativos. Los *workload logs* son insustituibles para optimizar y mejorar los sistemas HPC, pero su publicación está severamente restringida debido a los riesgos de privacidad. Para abordar estos problemas esta tesis presenta un proceso de síntesis de datos con privacidad diferencial (DP, en su acrónimo en inglés) adaptado a los registros de trabajos de HPC, desarrollado durante las prácticas en el *Barcelona Supercomputing Center* (BSC)[2]. Esta *pipeline* combina de manera única soluciones DP existentes, en determinados casos las modifica y las aplica en una circunstancia de uso real. Esta *pipeline* consiste, principalmente, en la integración de una implementación modificada de PrivTree [49] para la discretización adaptativa por densidad (para preservar mejor las correlaciones de los datos) junto con un Generador de Datos Sintéticos con DP (el algoritmo *Maximum Spanning Trees* [28]) para la síntesis de datos privados. Más allá de estos dos algoritmos principales, el proceso incorpora mecanismos DP adicionales, incluyendo una implementación de pseudonimización con DP para proteger los ID de Usuario y Grupo, y una transformación logarítmica. Esta última permite que el PrivTree modificado discretice mejor los atributos con distribuciones sesgadas y de colas largas típicas de las cargas de trabajo de HPC. Gracias a ello, el proceso propuesto genera registros sintéticos de manera adecuada. Los conjuntos de datos sintéticos se evalúan mediante una amplia gama de métricas de utilidad y privacidad. De manera crucial, la aplicabilidad práctica se valida alimentando con los registros sintéticos un simulador de SLURM para reproducir los resultados obtenidos a partir de los datos reales en un estudio diferente (Autosubmit [25]). Los resultados indican que, aunque los registros preservan bien las distribuciones y correlaciones como para indicar que funcionan adecuadamente para el aprendizaje automático, el aprendizaje profundo y el análisis exploratorio de datos (EDA, en su acrónimo en inglés) (también mostrado por las métricas de *Machine Learning utility* y los plots de distribuciones ben sintetizadas), carecen de valores atípicos sustanciales para la simulación de registros fieles a la realidad. Como consecuencia, aunque dichos registros capturan adecuadamente las tendencias centrales y los patrones típicos de un workload de HPC, no logran capturar algunos de los matices y valores extremos, dando como resultado una simulación sin esos extremos, sin esos escenarios de peor caso (que son el foco de muchos estudios), mientras que los casos intermedios (trabajos más grandes de lo habitual, pero no desviaciones extremas) se vuelven más comunes. Estas carencias son atribuibles a la influencia de la preservación de la privacidad diferencial, que suprime los valores atípicos, los suaviza, haciendo que sean tanto menos extremos como más comunes.

Contents

List of Figures

Acknowledgments

I would like to thank my supervisor Manuel Giménez de Castro Marciani and tutor Gladys Míriam Utrera Iglesias for their guidance and support throughout the internship, as well as the entire Earth Sciences Department at BSC.

1 Introduction

The increasingly growing reliance on High Performance Computing (HPC) systems, more widely known as "supercomputers", for critical research makes protecting their operational data an urgent concern. Supercomputers process workloads from many different fields, from climate modeling and genomics to national security, in consequence, the workload traces (the logs) of their usage contain detailed information about job submissions, resource usage and user activity, all crucial data for optimizing the HPC system's algorithms and processes.

However, publishing these logs without proper safeguards risks a critical security breach, a catastrophic privacy leak which would allow malicious actors to identify user identities, usage patterns and would have legal consequences [19].

As a result, there is a scarcity of public, real-world HPC datasets (from modern systems) which severely limits data-driven research in system optimization, job scheduling and failure prediction. This effectively slows down these systems and, ultimately, due to their widespread usage, all research. The central motivation of this work is, thus, to bridge the gap between data utility and privacy protection.

This challenge directly impacted recent research at BSC, for example, during the research in Autosubmit [25], a scheduling mechanism that groups jobs to reduce queue times. Researchers required realistic workload traces from BSC's flagship supercomputer, MareNostrum [2]. Lacking a privacy-preserving release mechanism, they had to rely on public logs from a different system, limiting the local applicability of their results.

Born from this need, this work explores whether Differential Privacy (DP) can enable not only the BSC, but more broadly any other HPC center, to safely share sensitive data from their systems. With this objective in mind, this thesis developed and implemented a pipeline combining, and modifying (whenever necessary) existing DP solutions. Furthermore, the resulting pipeline was tested in a real-life scenario via the replication of the results of the Autosubmit study [25] thus providing a more robust evaluation of the synthetic data generated. Among the DP solutions applied, the two main algorithms used were Maximum Spanning Trees [28] and a modified version of PrivTree [49] (this combination improved upon other implementations of MST shown in the literature were equal-binning is instead used) a binning technique which is unfit for skewed, long tailed data and erases most correlations.

In addition, the pipeline is composed of other auxiliary DP solutions to aid mainly in the preprocessing stage, such as an implementation of a DP

pseudonymization and a logarithmic transformation to allow our implementation of PrivTree to better discretize the skewed data of HPC logs.

The results obtained indicated that the pipeline is able to provide private synthetic data useful for Machine Learning, Deep Learning and Exploratory Data Analysis purposes. However, due to DP's nature, outliers are removed or smoothed. This impacts the traces simulated in SLURM from the synthetic data, causing anomalous behaviors to arise, ultimately differentiating them from the ones obtained from the original data.

The remainder of this report is organized as follows:

- Chapter 3 introduces the fundamental concepts of differential privacy and HPC log structures.
- Chapters 4 introduces the empirical experimental development process through which the final pipeline was developed. Inside this section, the datasets used, the validation metrics and the experimentation are explained.
- Chapter 5 describes our proposed anonymization pipeline and evaluation metrics.
- Chapter 6 evaluates the results obtained.
- Finally, Chapters 7 and 8 summarize our findings and outline avenues of future work.

In addition, the appendices contain a section detailing how to use the proposed pipeline (Appendix A), the project's work plan (Appendix B), a brief cost analysis (Appendix C), a brief summary of other non-research related activities done at BSC (Appendix D), a short analysis of potential gender biases (Appendix E), a discussion of ethical considerations (Chapter F), an assessment of environmental and sustainability aspects (Appendix G), as well as additional experimental results (Appendix H) and the full pipeline and evaluation code (Appendix I).

2 Goals

The primary objective of this thesis is to develop and evaluate a Differentially Private generative pipeline tailored for HPC job logs. This pipeline aims to synthesize private datasets by injecting noise in order to provide formal privacy guarantees so as to allow BSC to have plausible deniability, the ability to deny responsibility for an action because evidence is indistinguishable from alternative explanations, while allowing institutions to safely share data. By establishing a trade-off between privacy loss and data fidelity, this work seeks to offer BSC, and more broadly HPC centers, a practical avenue toward open and private dataset publication.

To achieve this main objective, the following secondary goals were established:

- **Methodological Analysis:** Test and analyze various differential privacy techniques and synthetic data generation methods to ascertain the best approach experimentally.
- **Tool Discovery:** Identify and utilize efficient libraries and tools to seamlessly implement these privacy methods.
- **Utility Evaluation:** Analyze the utility of the resulting DP data to ensure it retains its value (its statistical properties).
- **Practical Applicability:** Evaluate the synthesized HPC traces by using them to replicate research studies, specifically the Autosubmit scheduling mechanism [25], to confirm that conclusions drawn from synthetic data do align with those drawn from real data.

3 Background and State of the Art

In this section the main methods researched will be explained, as well as any related necessary information about differential privacy, so as to better understand the ultimately implemented pipeline.

Firstly, the basic concept of differential privacy (DP) must be properly addressed, as well as why and what it was developed for as first proposed by Cynthia Dwork [8] at 2006 with her paper "Differential Privacy".

To know why it was proposed, why it was necessary, we must expose the shortcomings of the solutions proposed previous to the emergence of DP.

To this end, we review these earlier approaches in Subsection 3.1 and introduce the basics of DP in Subsection 3.2. Later, we analyze DP-based synthetic data generators and explain the used algorithm in Subsection 3.3. Finally, we discuss the importance of discretization and introduce an algorithm to do so (Subsection 3.4), and outline the theoretical background of Slurm (Subsection 3.5).

3.1 Former solutions

As stated, currently most supercomputers rarely publish their data. One of the plethora of reasons is that the research on data protection techniques applied on HPC logs has been limited, and part of the motive has been the failings of the solutions available. According to the work of Solorzano et al. (2025) [47], the (main) traditional privatization techniques where:

- **Encryption**: Converts data into a coded format that requires a key to decode. It protects confidentiality but prevents data analysis and processing. Furthermore, when used as a deterministic pseudonym it still creates persistent identifiers that can be linked across datasets. For example, Culnane, Rubinstein & Teague [6] re-identified individuals in the Australian Medicare dataset by linking deterministic encrypted patient IDs across

years. In addition, Naveed et al. [34] also showed that plaintext values can be recovered from property-preserving encrypted databases using frequency analysis.

- **Hashing of Identifiers**: Replaces original identifiers with strings of a fixed length produced by a hash function. This technique remains vulnerable to frequency analysis and can expose Personally Identifiable Information (PII), in other words, information that can be used for identification purposes. The reason is that a cryptographic hash is deterministic, making it easily reversible given that the input domain is small enough. For example, Pandurangan [38] brute-forced the MD5-hashed taxi medallion numbers in the New York City Taxi dataset, re-identifying every driver. Similarly, the AOL search data release [1] demonstrated that user IDs replaced by random pseudonyms (which effectively is akin to a hash) does not prevent re-identification when the content itself reveals the individual.

- **Obfuscation**: Adds noise to the data to mask original values. This technique often introduces significant computational overhead and reduces data utility. Furthermore, even when the data is noised, re-identification remains possible. de Montjoye et al. [32] showed that only four spatio-temporal points are enough to uniquely re-identify most individuals in a heavily obfuscated mobility dataset. In addition, Kargupta et al. [22] also demonstrated that the noise can be removed and the original private values reconstructed.

- **Basic Anonymization**: Removes direct identifiers (such as names, complete ZIP codes...) from the dataset. It nevertheless remains highly vulnerable to linkage and re-identification attacks. Sweeney [44] linked the "anonymized" medical records of Massachusetts state employees to the public voter registration list and uniquely re-identified the Governor's health data using only ZIP code, birth date and sex. Moreover, Narayanan & Shmatikov [33] also was able to de-anonymize the Netflix Prize dataset by linking it to public IMDb reviews, uncovering the viewing histories of individual users.

The vulnerabilities in these more traditional methods leaves datasets exposed to multiple sorts of privacy breaches, of attacks. Said attacks primarily come in two forms:

- A **Linkage Attack** occurs when an attacker attempts to re-identify an individual by combining the released "anonymized" data with auxiliary information (in the form of public databases for instance). By matching combinations of sensitive attributes (PIIs), the attacker tries to link a synthetic record back to a specific individual.

- In a **Membership Inference Attack (MIA)**, an adversary exploits a trained machine learning model to determine whether a specific individual was included in the private training dataset. Because models frequently

overfit and exhibit higher confidence on data they have memorized, the attacker can construct an inference function. If the function evaluates to 1 (true or positive), the attacker successfully deduces membership, thereby identifying the individual.

Up to this point every method discussed shares a critical vulnerability: none offers mathematically provable privacy guarantees, and each of them has been proved fallible, in other words, susceptible to privacy breaches. Differential Privacy was devised to solve all of these shortcomings. This is because, unlike those methods, DP provides a mathematically provable guarantee of privacy that holds against any kind of attack, regardless of the attacker's auxiliary information, as formally established by Dwork [8]. In fact, and to elaborate, Differential Privacy impresses by the long list of assumptions it does not require: it is not necessary to know what information the attacker has, nor whether attackers are colluding nor what they are looking for in particular.

3.2 Differential Privacy

Privacy-preserving data analysis has long been a challenge that researchers have been keen to overcome through different methodologies, such as the ones shown in the previous section 3.1. As stated, all of those techniques were vulnerable to privacy breaches, and thus was Differential Privacy proposed.

However, prior to defining differential privacy, we first must define what privacy actually is.

In 1977, Dalenius stated a desideratum for what is to be considered privacy in statistical databases: nothing about an individual should be learnable from the database that cannot be learned without access to the database. This definition shapes our objective: to learn characteristics and patterns of the overall population as a whole, while protecting individual data. Or, to paraphrase, to learn nothing from no individual while learning of them overall.

Clearly, previous techniques were not enough to ensure this desideratum. In fact, work by Dwork [8] demonstrated Dalenius's strict formalization of privacy to be a general impossibility. This impossibility mainly comes down to two major issues: the existence of auxiliary information and the concept of utility.

Auxiliary information is any external data that an adversary (a malicious researcher attempting to deduce personal information) has at their disposal. This cannot be predicted, stopped nor known, ergo, it cannot be avoided.

As for the utility constraint, the usefulness of the data is inversely proportional to its privacy: a completely private dataset is useless, since it would consist of random noise or hold no information at all. This forces us to find a robust middle ground.

To clearly expose the impossibility of Dalenius's Desideratum we simply need to observe how auxiliary information breaks his definition. An example was provided by Dwork [8]. Suppose an adversary possesses external auxiliary knowledge stating that *"Terry Gross is two inches shorter than the average Lithuanian woman."*

If a statistical database allows the adversary to accurately query the average height of a Lithuanian woman, the adversary immediately learns Terry Gross's exact height. More importantly, this privacy breach occurs whether or not Terry Gross's personal data was included in the database in the first place.

Since the database successfully revealed a valid, global statistical property of a population, it inadvertently compromised an individual's privacy when combined with outside knowledge, even if this individual wasn't part of the database. The specific mathematical demonstration of this impossibility can be found in Dwork's study [8].

Given this impossibility, an entirely different approach, a new privacy philosophy, was proposed. Instead of promising that an adversary will learn nothing new about an individual, the new aim is to guarantee that an individual will incur no additional risk or harm by choosing to participate in the dataset/analysis.

To elaborate: the outcome of any data analysis mechanism should be virtually indistinguishable regardless of whether a specific individual opts in or opts out of a given study/dataset. It does not, however, guarantee that said individual will be unaffected by the analysis outcome.

This new paradigm is what the formal framework of Differential Privacy aims to mathematically guarantee, to mathematically bound.

Another good example of this aim was provided by Dwork: if an analysis finds that smoking is related to higher death rates, insurance companies might increase your insurance fee. Differential Privacy ensures that your participation in the study will neither increase nor decrease your risk of being affected (here, the effect is a higher insurance fee) by more than a bounded amount. In other words, your risk cannot change beyond a certain threshold regardless of your participation.

To formalize this mathematically, the concept of neighboring datasets must first be established. Following the conventions stated by Dwork [10], two datasets, denoted D_1 and D_2, are neighboring databases if they differ by at most one individual's record.

Given this foundation, we now can introduce the primary definition of Differential Privacy, originally formalized by Dwork, McSherry, Nissim, and Smith [8]:

Definition 1 (ϵ-*Differential Privacy*)**.** *A randomized function $\mathcal{K}$ gives ϵ-differential privacy if for all pairs of neighboring datasets D_1 and D_2 differing on at most one element, and for all sets of potential outcomes $\mathcal{S} \subseteq Range(\mathcal{K})$:*

$$\Pr[\mathcal{K}(D_1) \in \mathcal{S}] \leq e^{\epsilon} \times \Pr[\mathcal{K}(D_2) \in \mathcal{S}] \tag{1}$$

To properly understand why this definition complies with our privacy concerns, we break down its core components:

- **The Necessity of Randomization:** This is where the core mechanism relies on stochasticity. The random function $\mathcal{K}$ must be non-deterministic. If $\mathcal{K}$ were a deterministic function, the probability of an outcome given

a specific dataset would evaluate strictly to 0 or 1. This would make it mathematically impossible to satisfy the multiplicative bound (e^ϵ) across different datasets unless the output ignored the data entirely.

- **The Privacy Parameter (ϵ):** This parameter, widely known as the privacy budget, measures the strictness of the privacy guarantee: it is the threshold limiting the risk. Its name captures its purpose: to control how much privacy loss is allowed, to bound how much the presence of an individual impacts the results. If ϵ is set near 0, the outputs on neighboring datasets D_1 and D_2 become indistinguishable, maximizing privacy at the expense of utility. To paraphrase, an ϵ near 0 means that whether an individual opts in or out makes no difference: the increase in their risk of harm becomes virtually nonexistent.

- **Quantification over all $\mathcal{S}$:** The inequality must hold for any subset of outcomes $\mathcal{S}$. This means that no matter what conclusion or observation an adversary attempts to draw from the output, the probability of reaching that conclusion differs by at most the multiplicative factor of e^ϵ depending on whether any single individual's data was included in the calculation or not.

To summarize, this definition of differential privacy ensures that observing an output gives an adversary no mathematical leverage to deduce whether the underlying database was D_1 or D_2 (if an individual was present or not in a database). Thus, individuals are now incentivized to contribute their data to scientific and statistical studies, with the guarantee that any potential harms or benefits resulting from the study are bounded and would occur regardless of their participation.

Having said this, we now need to define how we can inject the noise necessary in a way that it ensures differential privacy. To do this properly the noise to inject must be gauged. As seen before, said noise has to be enough so as to bound to e^ϵ any influence a single individual has on the results, and for that we need calculate the maximum impact that a single individual's record can have on $\mathcal{K}$'s outcome. This is the foundational concept known as global sensitivity (or ℓ_1-sensitivity). Following the framework established by Dwork and Roth [10], sensitivity is the calculation that will determine how much noise is required to be injected to a query to bound any single individual's contribution to e^ϵ.

Definition 2 (ℓ_1-*Sensitivity*)**.** *For a query function $f : D \to \mathbb{R}^d$, the ℓ_1-sensitivity of f, denoted as Δf, is defined as:*

$$\Delta f = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|_1 \tag{2}$$

for all neighboring datasets D_1, D_2.

Understanding sensitivity is crucial because it's one of the main factors controlling the amount of noise to be introduced. Functions with higher sensitivity

require larger amounts of noise to maintain the same level of privacy protection (it is important to remember that epsilon is not the amount of noise itself) whereas low-sensitivity operations (such as simple counting queries where $\Delta f = 1$) require significantly less noise to ensure that same protection.

In other words, two operations calculated in a differentially private manner with the same epsilon, the same privacy guarantees, can have different amounts of noise injected depending on their respective sensitivity.

Given this, we now have to define mechanisms, functions, that inject said noise in a differentially private way. The most classic of this functions, the first one to be found to ensure this definition of differential privacy, is the Laplace Mechanism first introduced by Dwork [10].

Definition 3 (The Laplace Mechanism)**.** *Let $f\colon D \to \mathbb{R}^d$ be a deterministic function whose ℓ_1-sensitivity is Δf (see Definition 2). Given an input $x \in D$, the Laplace mechanism $\mathcal{M}_L$ outputs*

$$\mathcal{M}_L(x) = f(x) + (Y_1, \ldots, Y_d),$$

where $Y_1, \ldots, Y_d$ are i.i.d. random variables drawn from a symmetric Laplace distribution centered at 0 with scale parameter $b = \frac{\Delta f}{\epsilon}$, denoted $Y_i \sim Lap(0, b)$. The probability density function of each Y_i is

$$p(y) = \frac{1}{2b} e^{-|y|/b} = \frac{\epsilon}{2\Delta f} e^{-\epsilon|y|/\Delta f}.$$

This mechanism's main idea is simple: to mask the true numeric answer to a given query (to a given random function $\mathcal{K}$) by adding random values, noise, drawn from the Laplace probability density function 3.

The main advantage of using this mechanism is that the scale (the noise) b is calculated with a straightforward ratio involving our global sensitivity and our privacy budget: $\frac{\Delta f}{\epsilon}$. Hence, for a given a query with a high sensitivity (Δf), the mechanism injects a larger dose of noise to keep individual records safe.

Conversely, if the privacy budget (ϵ) is larger, our privacy guarantees are weaker, thus needing less noise to ensure that level of privacy, thereby allowing for the answer to be more accurate.

Additionally, this mechanism guarantees the strict ϵ-differential privacy definition and it works proficiently for standard numeric database calculations such as averages, sums, or frequency counting.

Nevertheless this mechanism is not perfect. While this continuous real-valued noise addition is effective for numeric or frequency-based metrics, it only works for numeric values. It fails when dealing with discrete, non-numeric categorical attributes, where adding continuous noise is conceptually nonsensical. This types of attributes require a different approach, a different mechanism, to ensure differentially private noise addition. One technique that ensures this is the k-ary Randomized Response mechanism, also defined by Dwork [10].

This mechanism's main idea is to provide the participant plausible deniability. In a generalized k-ary domain $\mathcal{V}$ where an attribute can take one of k distinct

categorical values (meaning $|\mathcal{V}| = k$), the mechanism $\mathcal{M}_{RR}$ randomizes the output for an individual's true value $v \in \mathcal{V}$. That is, to satisfy ϵ-differential privacy, the mechanism reports the true value with a higher probability, while uniformly splitting the remaining probability across all alternative options, meaning that although we are probably assigning the real category, this might not be so and won't be so for a certain percentage of the individuals (we flip the true category with a different value selected from the attribute's domain). Formally, for any output value $v' \in \mathcal{V}$:

$$\begin{array}{l} \Pr[\mathcal{M}_{RR}(v) = v'] = \frac{e^\epsilon}{e^\epsilon + k - 1} \\ \Pr[\mathcal{M}_{RR}(v) \neq v'] = \frac{1}{e^\epsilon + k - 1} \end{array} . \tag{3}$$

Moreover, we must ensure that this complies with DP's privacy constraints. This is easily proven by looking at the ratio of the maximum and minimum probabilities for any two possible true states, which evaluates to:

$$\frac{\frac{e^\epsilon}{e^\epsilon + k - 1}}{\frac{1}{e^\epsilon + k - 1}} = e^\epsilon. \tag{4}$$

And this satisfies strict ϵ-differential privacy. This mechanism now allows for categorical logs, status types, or other discrete events to be privatized properly, serving as a vital complement to the Laplace mechanism when processing diverse structural traces with different attribute types, such as HPC logs.

Yet, additional mechanisms must be considered. For instance, when the goal is not to randomize a single individual's categorical response but rather to select the best option from a discrete set of candidates according to a specific metric, a different approach is required.

This scenario is particularly relevant for differentially private synthetic data generation, specifically for Maximum Spanning Trees [28], where we need to select which statistical relationships between columns are most worth measuring, in a differentially private manner. The Exponential Mechanism is such a tool, as defined at McKenna et al. [28].

Definition 4 (*The Exponential Mechanism*)**.** *Let $q : D \times \mathcal{R} \to \mathbb{R}$ be a quality score function and ϵ a privacy parameter. The exponential mechanism $\mathcal{M}_E$ outputs a candidate item $r \in \mathcal{R}$ with probability proportional to* $\exp(\epsilon \cdot q(D, r))$*:*

$$\Pr[\mathcal{M}_E(x) = r] \propto \exp(\epsilon \cdot q(D, r)), \tag{5}$$

where $\mathcal{R}$ is the set of all possible candidates.

The key idea is that, instead of adding noise to a numerical value (Laplace) or flipping categorical values with a certain probability (k-ary RR), the Exponential Mechanism scores every possible candidate r using a quality function $q(D, r)$. This function assigns higher scores to better candidates. The mechanism then samples a candidate from the resulting distribution so that the probability of selecting a given r grows exponentially with its quality.

This naturally balances utility and privacy: high-quality candidates are exponentially more likely to be chosen, while the inherent stochasticity ensures that no single individual's presence can overly influence the final selection.

In this way, the Exponential Mechanism provides a mathematically rigorous way to perform discrete choices, such as deciding which attribute pairs to correlate, while preserving differential privacy guarantees.

Furthermore, and crucial to enable the real-world implementation, differential privacy provides an essential and powerful property: the Post-Processing Theorem [10].

Theorem 1 (*Post-Processing*)**.** *Let $\mathcal{K}$ be a randomized function that is ϵ-differentially private. Let g be any arbitrary randomized or deterministic function mapping from the range of $\mathcal{K}$ to an alternative arbitrary output space $\mathcal{Z}$. Then, the composite mechanism $g \circ \mathcal{K}$ is also ϵ-differentially private.*

This theorem states that differential privacy, once applied, can not be reversed by manipulating the output of a given DP mechanism.

Since the post-processing function g operates only on the private output $\mathcal{K}(x)$ it is mathematically impossible for it to uncover or leak any extra individual information as long as it has no access to the original sensitive data. As a result, an analyst can apply any subsequent data transformation and perform any visualization without any risk of breaching differential privacy.

This property is key for this work's implementation and more broadly for any real-life application of Differential Privacy, enabling down-stream analysis with the assurance that the mathematical privacy guarantees won't be compromised.

3.2.1 Local vs Global DP

There is an important nuance to consider. As defined in the paper by Solórzano et al. [47], Local and Global DP are two different paradigms, two different approaches, to the implementation of differential privacy.

Global DP is the more traditional approach and is the one originally introduced by Dwork [8]. In Global DP, we assume that there is a centralized, trusted curator who holds all of the raw, sensitive data. This curator aggregates the statistics and calculates exactly the amount of noise required based on the global sensitivity, injecting it into said aggregates. Afterwards, the privatized results are released to the public.

In the Local DP approach, however, this changes completely. There is no centralized trusted curator to aggregate the raw data. Instead, individuals inject noise into their data before it ever leaves their possession. An example of this is how mobile operating systems collect usage statistics, as demonstrated by Min et al. [30]. In this framework, before the data is sent to a central server, it is locally perturbed. As a result, the server never sees the true, raw data, only the noisy version, to which it now computes the aggregate statistics.

That said, there is a critical issue with the Local approach: it implies the injection of massive amounts of noise. Since there is no central coordinator to calculate the precise amount of noise given the global sensitivity, to add to the

aggregated statistics, every single record must carry enough noise by itself to protect its own individual privacy. When an aggregator tries to collect, sum up, or analyze these records, all of this individual noise adds up together.

Statistically, this means the variance of any estimators grows drastically with the number of records, potentially destroying the utility the data. The paper by Solórzano et al. [47] has a great visual diagram showing exactly how these two data flows diverge (Figure 1).

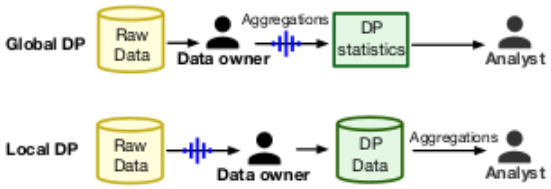


Figure 1: Local DP and Global DP diagrams

In this project both approaches were tested and Global DP was found to be the more adequate way forward after the Local DP approach was experimentally proven to destroy utility. This induced us to pivot to a different approach: Differentially Private Data Generation (DPDG), which operates safely under the Global DP framework.

This approach works by observing all the data to learn a differentially private generative model instead of perturbing every trace line locally. Examples of such frameworks are DataSynthesizer [39] or Maximum Spanning Trees [28], both of which leverage Global DP by adding a strictly gauged amount of noise during the model-learning phase, preserving statistics and correlations far more proficiently than with a Local DP approach.

3.3 Differentially Private Data Generation (DPDG)

These are a family of methods whose goal is not to simply provide a private statistic such as a mean or deviation, but rather an entire private dataset to be used. An example of an application of such approach for publishing a dataset is the United Nations High Commissioner for Refugees (UNHCR) synthetic registration data [46].

DPDG encompasses a wide array of algorithms. Some, such as differentially private generative adversarial networks (DP-GANs), are built on differentially private stochastic gradient descent (DP-SGD) [3]. Others avoid neural networks altogether and rely on more classical statistical methods, an example being DataSynthesizer [39]. Yet another category combines statistics with optimization, as exemplified by the Maximum Spanning Tree (MST) approach [28].

Nonetheless, all of the approaches mentioned work similarly: given the original sensitive dataset they generate a synthetic, differentially private version to

be used and published.

In addition, we would like to point out that synthetic data generation by itself (without Differential Privacy) is not private. There exists a misconception that simply because a dataset is "fake" (synthesized through Deep Learning or statistics in general), it automatically protects individual privacy, since no real individual is in it. This is absolutely not so.

This misconception is quickly dispelled by thinking about how generative models actually work, be them Machine Learning (ML)/Deep Learning (DL) based or otherwise. These models learn patterns and characteristics about the original data in order to generate data of similar characteristics, meaning that if left unchecked, they can also end up memorizing, copying, real individuals of the sensitive dataset or exposing the original data's distributions.

Without a strict mathematical constraint, such as Differential Privacy, actively limiting what the model can remember, the generator might simply overfit and memorize the underlying training data, especially the rare outliers. This exact vulnerability is stated in benchmarking frameworks like CLOVER [40] and generative architectures like PATE-GAN [48], which demonstrated that standard generative models don't provide rigorous privacy guarantees.

This has implications for HPC workload traces which as mentioned feature skewed, long-tailed distributions where some jobs can be rare but hold much influence in the overall statistics, as it can be seen in Figure 2.

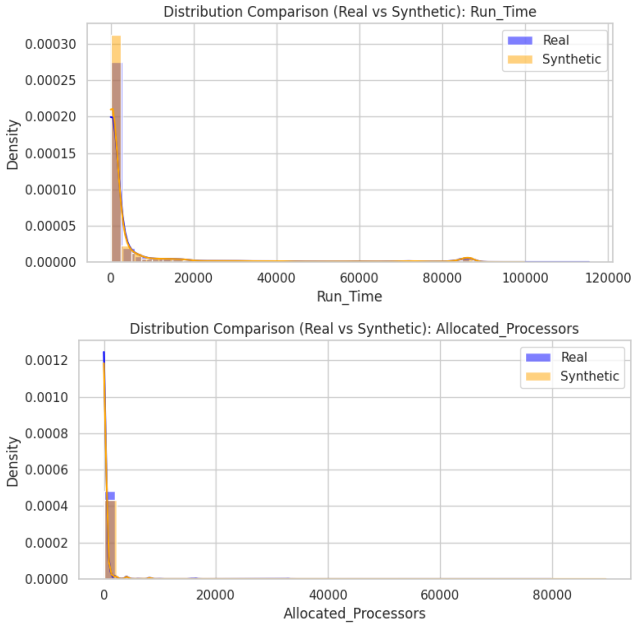


Figure 2: Run_Time (up) and Allocated_Processors (down) distributions from the Curie dataset showing their skewness and long-tailed nature.

This is exactly why Solórzano et al. [47] argues that bringing actual Differential Privacy guarantees to HPC traces is non-negotiable. Synthetic data generation by itself does not do so, thus, without the mathematically gauged noise of DP to actively blind the model from focusing on individual records, a synthetic dataset remains open to attacks, rendering void any possible privacy guarantees.

3.3.1 Maximum Spanning Trees and Theoretical Preliminaries

Currently, given their success at several competitions such as NIST [28], Maximum Spanning Trees (MST) is one of the methods standing at the forefront of differentially private synthetic data generation.

Peer research and comparative benchmarks [45, 40] have indicated that this algorithm is well suited for generating high-dimensional datasets and preserving the relations of their data. For this reasons, this work's proposed pipeline employed this algorithm as its synthesizer, as proposed by McKenna et al. [28].

Prior to explaining the mechanics of MST, however, we need reevaluate classical, pure ϵ-Differential Privacy (ϵ-DP), which has been proven to be too strict for some real-world applications, such as is the case of MST, as shown by McKenna et al. [28] and Mironov [31].

Pure ϵ-DP provides a strict, worst-case bound on privacy loss. In consequence, this means that when an algorithm requires multiple sequential steps of multiple Differential Privacy mechanisms, as is the case for MST, we must rely on the property of composition, a property of Differential Privacy. Under pure DP, privacy loss accumulates linearly. Yet, if we attempt to measure the hundreds of attribute combinations necessary to capture a complex workload linearly, the total privacy budget would explode. To prevent this under a pure DP framework, one would be forced to inject destructive amounts of noise into each individual measurement, rendering the final synthetic data utility-less.

To solve this issue, the strict privacy definition of ϵ-DP must be relaxed. Consequently, before detailing the MST pipeline, two advanced formulations of Differential privacy are introduced, the second improving on the first: Approximate Differential Privacy and Rényi Differential Privacy.

Approximate Differential Privacy

To overcome the strict constraints of pure DP, researchers introduced a minor margin of error, a relaxation, formulating (ϵ, δ)-Differential Privacy, widely known as Approximate DP, first introduced by Dwork et al. in "Our Data, Ourselves: Privacy via Distributed Noise Generation" [11].

This framework recognizes that guaranteeing absolute privacy in every theoretical worst-case scenario damages data utility. Therefore, Approximate DP introduces a secondary parameter, δ, which represents the probability that the strict ϵ-DP guarantee fails.

Mathematically, it modifies the fundamental definition of differential privacy as shown in equation 6.

$$\Pr[\mathcal{K}(D_1) \in \mathcal{S}] \leq e^{\epsilon} \times \Pr[\mathcal{K}(D_2) \in \mathcal{S}] + \delta \tag{6}$$

To maintain rigorous privacy guarantees, δ must be kept cryptographically small (typically $\delta < 1/N$, where N denotes the size of the database) so that the probability of a privacy leak remains statistically unlikely.

However, Approximate-DP still has one critical issue: as stated, it allows for an absolute breach of privacy with probability δ.

Given this shortcoming, the more advanced definition known as Rényi Differential Privacy [31], which never allows such total breach of differential privacy and provides with better composition, is introduced.

Rényi Differential Privacy (RDP)

While Approximate DP accommodates the Gaussian mechanism (required for MST to function), Approximate DP has several, critical, issues. On one hand, it has a probability for an absolute breach of privacy and, furthermore, composition of ϵ and δ under this approach is, as stated by Mironov [31], computationally expensive (#P-hard in some cases) and difficult.

To address these shortcomings, Mironov [31] proposed Rényi Differential Privacy (RDP). RDP is a generalization of ϵ-DP based on the concept of the Rényi divergence, a measure that quantifies the difference between two probability distributions. In this work, for simplicity, we restrict our attention to discrete distributions, since MST operates only on discrete categorical data.

Definition 5 (Rényi divergence [31])**.** *For two discrete probability distributions P and Q defined over a finite support $\mathcal{X} = \{x_1, \ldots, x_n\}$, the Rényi divergence of order $\alpha > 1$ is*

$$D_\alpha(P \parallel Q) = \frac{1}{\alpha - 1} \log \sum_{i=1}^{n} \frac{P(x_i)^\alpha}{Q(x_i)^{\alpha-1}},$$

where all logarithms are natural and $P(x)$ denotes the probability mass function of P at x. For the endpoints of the interval $(1, \infty)$, the Rényi divergence is defined by continuity:

$$D_1(P \parallel Q) = \lim_{\alpha \to 1} D_\alpha(P \parallel Q) = \sum_{i=1}^{n} P(x_i) \log \frac{P(x_i)}{Q(x_i)},$$

which is the Kullback–Leibler divergence, and

$$D_\infty(P \parallel Q) = \lim_{\alpha \to \infty} D_\alpha(P \parallel Q) = \log \sup_i \frac{P(x_i)}{Q(x_i)}.$$

Where α is a term introduced in the definition of the Rényi Divergence to generalize the Kullback-Leibler divergence, and in RDP works as a knob controlling the strictness of our privacy guarantees. If infinite it is equal to the strict definition of ϵ-DP and if lower it behaves similar to Approximate

DP. Nonetheless, whatever finite value the α might be given, RDP preserves plausible deniability for all outcomes as proven by Mironov [31].

As to how this related with Differential Privacy, when looking at the case with $\alpha = \infty$ a random mechanism f is DP if and only its distribution over any two neighboring datasets D_1 and D_2 satisfies: $D_\infty(f(D_1) \parallel f(D_2)) <= \epsilon$, which brings us to the definition of RDP:

Definition 6 (Rényi Differential Privacy (RDP) [31])**.** *A randomized mechanism $f : \mathcal{D} \mapsto \mathcal{R}$ is said to have ϵ-Rényi differential privacy of order α, or (α, ϵ)-RDP for short, if for any neighboring datasets $D_1, D_2 \in \mathcal{D}$ it holds that*

$$D_\alpha(f(D_1) \parallel f(D_2)) \leq \epsilon.$$

This complex definition of DP has many advantages. For instance, it inherits many of the properties of the standard differential privacy: any change in the outcome is bound by ϵ, it is robust against auxiliary information and the post-processing theorem holds.

Furthermore, RDP advanced composition theorem allows for a tighter accounting of the privacy loss and for a definition of the Gaussian mechanism under RDP:

Let $f : \mathcal{D} \mapsto \mathcal{R}$ be a randomized mechanism. The Gaussian mechanism for approximating f is defined as

$$\mathbf{G}_\sigma f(D) = f(D) + \mathcal{N}(0, \sigma^2),$$

where $\mathcal{N}(0, \sigma^2)$ denotes the normal distribution with mean 0 and variance σ^2.

Proposition 1 (Rényi divergence of shifted Gaussians)**.** *For any $\alpha \geq 1$, $\sigma > 0$, and $\mu \in \mathbb{R}$,*

$$D_\alpha\big(\mathcal{N}(0, \sigma^2) \parallel \mathcal{N}(\mu, \sigma^2)\big) = \frac{\alpha\mu^2}{2\sigma^2}.$$

Corollary 1 (Gaussian mechanism under RDP)**.** *If f has sensitivity 1, then the Gaussian mechanism $\mathbf{G}_\sigma f$ satisfies $(\alpha, \alpha/(2\sigma^2))$-RDP*

This is all we need, since MST's sensitivity is, precisely, 1. In any case, this corollary is easily generalized given the ℓ_2 sensitivity, used in MST, which states:

Definition 7 (ℓ_2*-Sensitivity*)**.** *For a random function $f : D \to \mathbb{R}^d$, the ℓ_2-sensitivity of f, denoted as $\Delta_2 f$, is defined as:*

$$\Delta_2 f = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|_2 \tag{7}$$

for all neighboring datasets D_1, D_2.

Given this, the corollary can be generalized to:

Corollary 2 (Gaussian mechanism under RDP)**.** *For f, given ℓ_2 sensitivity, then the Gaussian mechanism $\mathbf{G}_\sigma f$ satisfies $(\alpha, \alpha\frac{(\Delta_2 f)^2}{2\sigma^2})$-RDP.*

This complex definition of privacy now allows us to do our Adaptive Composition of Mechanisms in an straightforward manner:

Proposition 2 (Adaptive Composition of RDP Mechanisms [28])**.** *Let* $\mathcal{M}_1 : \mathcal{D} \to \mathcal{R}_1$ *be* (α, ϵ_1)*-RDP and* $\mathcal{M}_2 : \mathcal{D} \times \mathcal{R}_1 \to \mathcal{R}_2$ *be* (α, ϵ_2)*-RDP. Then the mechanism* $\mathcal{M}$ *defined by* $\mathcal{M}(D) = \mathcal{M}_2\big(D, \mathcal{M}_1(D)\big)$ *is* $(\alpha, \epsilon_1 + \epsilon_2)$*-RDP.*

Which a is simpler, tighter and more straightforward composition rule than what would have to be done in Approximate Differential Privacy [31], which as mentioned its composition can be #P-hard.

The Maximum Spanning Tree (MST) Algorithm

Having established better differential privacy definitions, we now detail the inner mechanics of the Maximum Spanning Tree (MST) algorithm [28]. We begin by clarifying how MST internally uses both privacy definitions introduced before. MST works by initially "masquerading" as an algorithm designed under Approximate-DP, requiring the parameters ϵ and δ, however, internally it works by RDP, "translating" the Approximate-DP parameters into RDP's to work under the better privacy-loss compositions of said definition. At its paper, they state that this design choice is due to ϵ and δ being already of widespread usage and of easier comprehension than RDP's α and ϵ. This translation is, as stated by McKenna [28], the following:

Proposition 3 (RDP to DP)**.** *If a mechanism* $\mathcal{M}$ *satisfies* (α, ϵ)*-Rényi differential privacy, it also satisfies* $\left(\epsilon + \frac{\log(1/\delta)}{\alpha - 1}, \delta\right)$*-differential privacy for all* $\delta \in (0, 1]$.

As for the algorithm, conceptually, MST follows a paradigm: it measures key statistical properties of the sensitive data, adds noise to them and then generates synthetic records that preserve those measurements. The key statistical properties that MST measures, around which the algorithm's framework is designed, are marginals.

Formally, a marginal is a summary statistic that captures the low-dimensional structure of a high-dimensional distribution. For a specific subset of attributes (1-way, 2-way, 3-way marginals involve a subset of 1, 2 or 3 attributes), a marginal is a table or matrix counting the occurrences of each possible combination of values within those chosen attributes. This means that marginals at their core are counts, and counts have a global sensitivity of 1.

To protect utility while preventing budget explosion, MST privately selects a highly correlated subset of these marginals via a three-stage pipeline:

1. **Query Selection:** The algorithm models dataset attributes as nodes in a graph and determines a subset of low-dimensional marginals (specifically 1-way and 2-way attribute combinations) that capture the critical correlation structure of the dataset [28].

2. **Query Measurement:** The selected low-dimensional marginals are measured in the sensitive dataset and perturbed using the Gaussian mechanism to satisfy formal privacy bounds [28].

3. **Data Generation:** The resulting collection of noisy marginals is processed via Private-PGM [29], a post-processing tool that estimates a high-dimensional joint data distribution represented as a graphical model, from which synthetic records are directly sampled.

While certain variants (such as NIST-MST [28]) rely on an available public provisional dataset to perform non-private query selection, the general version of MST used in this work handles settings where no public reference data exists [28]. This version spends a fraction of the total privacy budget to execute a differentially private version of Kruskal's algorithm directly on the sensitive data to perform said selection. The complete systematic execution pipeline is composed of four distinct operational phases.

1. Initial 1-Way Marginal Measurement The algorithm first computes and measures all 1-way marginals for every attribute $\mathcal{A} = \{A_1, \ldots, A_d\}$ using the Gaussian mechanism [28]. This consumes a fixed allocation of the privacy budget and establishes the baseline individual distributions of the attributes.

2. Differentially Private Measurement Selection To select the most informative 2-way marginals without exposing individual records, MST evaluates the correlations between feature pairs in a private manner. It invokes Private-PGM (an optimization algorithm internally used in MST as described in McKenna et al. [28]) to construct an estimate of all 2-way marginals, denoted as $\bar{M}_{ij}$, derived from the previously measured noisy 1-way marginals under an assumption of mutual independence.

The quality score $q_{ij}(D)$ for selecting a given pair (i, j) is defined as the ℓ_1 distance between the true 2-way marginal $M_{ij}(D)$ and the estimated ones $\bar{M}_{ij}$ estimated with Private-PGM. This is because, since Private-PGM's estimation considers the attributes to be independent, the difference between the real measurements and the estimations under an assumption of independence results in the correlations:

$$q_{ij}(D) = \|M_{ij}(D) - \bar{M}_{ij}\|_1 \tag{8}$$

Since adding or removing a single individual's record can alter the count of exactly one cell in a marginal table by at most 1, the maximum l_2 sensitivity ($\Delta_2 f$) of any marginal query is strictly bounded at 1 [28] (remember, MST uses the l_2 sensitivity as shown in definition 7). Thus, $q_{ij}(D)$ constitutes a low-sensitivity metric ideal for the Exponential Mechanism, which for it to comply with RDP, as stated by McKenna [28]:

Proposition 4 (Rényi-DP of the Exponential Mechanism)**.** *The Exponential Mechanism applied to the quality score function $q : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$ satisfies $(2\epsilon\Delta, 0)$-DP and $\left(\alpha, \alpha\frac{(2\epsilon\Delta_2)^2}{8}\right)$-RDP for all $\alpha \geq 1$, where $\Delta_2 = \max_{r \in \mathcal{R}} \Delta_2 q(\cdot, r)$ is the maximum sensitivity of q.*

3. Target 2-Way Marginal Measurement. Once we know what marginals are to be selected, the true 2-way marginal tables for all selected pairs $(i, j) \in C$ are extracted from the sensitive data D [28]. These target tables are perturbed by adding independent Gaussian noise scaled according to the remaining RDP privacy budget allocation [28].

4. Distribution estimation and Generation. MST passes the collection of noisy 1-way and 2-way marginals to the algorithm Private-PGM as stated by McKenna [28]. This algorithm now finds a probability distribution that matches the noisy marginals measured as closely as possible and generates a synthetic dataset using an alternative approach, rather than sampling directly from the distribution since this would introduce a sampling error [28].

3.3.2 Empirical Validation and Privacy Vulnerabilities

The design choices of MST, such as using the Rényi DP definition, translate directly into a strong performance, though they present some trade-offs given a high privacy budget.

- **Utility and Structural Fidelity:** Benchmarks conducted by Tao et al. [45] confirm that MST is a robust synthesizer. It achieves positive results across multiple evaluation metrics. Specifically, it excels at preserving low-dimensional marginals, thus preserving complex feature correlations, and maintaining downstream machine learning classification accuracy.
- **Privacy Calibration and Vulnerabilities:** Despite its rigorous formal bounds, there still persists a utility-privacy trade-off. As demonstrated by recent research into the behavior of these algorithms under high-epsilon budgets by Golob et al.'s [14], if MST are configured with an excessively large ϵ budget, the algorithm fits the training marginals too accurately. This overfitting then introduces a subtle avenue for Membership Inference Attacks (MIA) [17], allowing an adversary to determine with confidence whether a specific individual's record was part of the original sensitive data.

The membership inference analysis by Golob et al., however, indicates that these vulnerabilities emerge only in highly experimental settings where the attacker possesses precise, extensive auxiliary population data. In realistic settings, most adversaries lack access to such comprehensive background information making these MIA attacks almost impossible.

Nonetheless, this behavior demonstrates the need for a well-chosen privacy budget, confirming that the algorithm performs safely and reliably as long as restricted to moderate, privacy-preserving ϵ allocations.

3.4 The Impact of Discretization in MST

During this research and experimentation the discretization has been found to have a great impact on the results for MST. This is because this algorithm only

work with discrete, categorical data. Consequently, any continuous numerical attributes need be discretized, and it was found that different discretization techniques heavily impact the results ultimately obtained by MST. This realization made discretization not merely a preprocessing requirement but a key step, especially because applying it in a differentially private manner that allows the discretization to be unmade (after the synthetic data has been generated) and not breaching the privacy guarantees proved to be non-trivial.

In the literature, it was seen that the main solution used was equal binning, however, we decided to do an exploration of better methods, since equal-binning tends to destroy most correlations within the data and would have difficulties when applied to the skewed, long-tailed data of HPC systems. Peer research by Ganev et al. [13] further demonstrated that the choice of discretization strategies and bin configurations impact the overall privacy-utility tradeoffs in end-to-end differentially private synthetic data pipelines. Ganev et al. showed that utility in DP generative models tend to follow an inverted U-shaped trend depending on the number of bins. If too fine a granularity is chosen, in other words too many bins are chosen, then data tables become incredibly sparse. When the Global DP mechanism tries to measure marginal distributions over these sparse tables, the noise overwhelms the actual data, turning any distribution into pure noise.

On the other hand, if too few bins are selected (too coarse a granularity), completely different individuals are aggregated into the same "bucket", losing all nuance long before the generative model ever analyzes the data. Furthermore, Ganev et al. showed that simply using default strategies (such as using 20 uniform bins) achieve worse utility with the same epsilon values, whereas optimizing the discretizer can improve utility by up to 43%.

Moreover, during this project's experimentation, an issue was identified when applying standard discretization approaches on HPC logs. Due to the fact that our data distributions are skewed and feature long tails, a traditional equi-width histogram approach performs poorly. With HPC data, most data points simply fall into the very first couple of bins, while the long tail is divided in sparse or empty bins.

Consequently, alternative discretization strategies were researched. One such alternative is quantile binning, of widespread use in Data Science. Quantile binning partitions continuous data ranges based on the data distribution, however, its integration with differential privacy presents significant theoretical and computational challenges. As demonstrated by Kaplan et al. [21], computing precise quantiles under differential privacy constraints is highly resource-intensive.

This is because a naive application of a differentially private mechanism to individual bin edges results in sequential privacy composition penalties that rapidly deplete the total privacy budget (ϵ) before downstream data generation can occur. To mitigate this issue, Kaplan et al. [21] established that robust boundary estimation requires optimized recursive tree-splitting frameworks, such as their proposed approximate quantiles algorithm, which bound privacy loss logarithmically and preserved the allocation of the privacy budget.

Following an extensive survey of possible discretization approaches, the PrivTree

algorithm by Zhang et al. [49] was selected due to its density-adaptive partitioning mechanism and its strong performance in the benchmark evaluation by Ganev et al. [13].

3.4.1 PrivTree-Based Discretization

PrivTree, originally formulated by Zhang et al. [49] for multi-dimensional hierarchical spatial decomposition, provides an adaptive algorithm for density-based space partitioning. In this work, the algorithm was adapted and implemented with some modifications tailored specifically for single-variable numeric discretization.

The algorithm operates by recursively splitting a continuous data domain into increasingly finer-grained sub-domains (or bins), provided the local data density satisfies a specified threshold check. The principal advantage of PrivTree over alternative hierarchical decomposition techniques lies in its complete removal of a pre-defined maximum tree height (h). In traditional differentially private trees, fixing a strict maximum depth introduces a utility dilemma: a conservative, shallow depth results in coarse and uninformative bins, whereas an aggressive deep tree forces the sequential composition privacy budget to spread thin. This requires injecting massive amounts of Laplace noise that scale proportionally with h, ultimately destroying data utility.

PrivTree successfully solves this limitation by dynamically, and privately, evaluating the termination criteria based on the local data distribution. It permits deep, fine-grained splits within dense regions while terminating early in sparser regions, all without incurring a privacy penalty that scales with the depth of the tree. This adaptive nature guarantees robust performance across highly heterogeneous and skewed datasets, such as the HPC logs. This theoretical robustness is empirically supported by Ganev et al. [13], who demonstrated that PrivTree's proficiency, achieving a robust average performance and minimal variability among state-of-the-art differentially private discretizers.

The Two-Phase Execution Framework
To preserve differential privacy while generating high-utility bins, a strict two-phase execution algorithm is enforced:

1. **Phase 1: Structural Discovery.** The algorithm allocates a portion of the total privacy budget to exclusively to establish the optimal bin boundaries. This phase explores the tree interactively and returns only the finalized, private geometric interval structures, completely omitting any true or intermediate noisy data counts.

2. **Phase 2: Density Estimation.** Once the bin intervals are fixed, they become a data-independent structure. The remaining privacy budget is then used to compute the final noisy counts for each bin (each leaf of the tree). Because the leaf bins partition the domain into entirely disjoint intervals, this step relies on parallel composition where the injected noise

scale is strictly bounded (1 divided this section's privacy budget) and remains completely independent of the tree's height.

Mathematical Formulation and Parameterization
The structural exploration phase satisfies pure ϵ_{tree}-differential privacy. PrivTree injects zero-mean Laplace noise scaled by a localized parameter λ, equivalent to the aforementioned b in 3:

$$\text{Lap}(x\,,\,\lambda) = \frac{1}{2\lambda}\exp\left(-\frac{|x|}{\lambda}\right). \tag{9}$$

By establishing a depth attenuation parameter δ (which isn't related in any way to Approximate DP's) that accumulates linearly down the tree paths, PrivTree penalizes deeper exploration of sparse regions. Given a branching factor $\beta \geq 2$, the scale parameter λ and the attenuation factor δ are parameterized as:

$$\lambda = \left(\frac{2\beta - 1}{\beta - 1}\right)\frac{1}{\epsilon_{\text{tree}}}, \tag{10}$$

$$\delta = \lambda \ln(\beta). \tag{11}$$

Let c_v denote the true number of data points falling within the boundaries of a given node v at depth d. For a target configuration threshold θ, set to 0.0 to facilitate autonomous structural discovery and eliminate hyperparameter tuning (this being the standard configuration as stated by Zhang et al. [49]) the adjusted count b_v is defined as:

$$b_v = \max(\theta - \delta,\ c_v - d \cdot \delta). \tag{12}$$

A node v is successfully split into β children if and only if its noisy adjusted score satisfies the threshold condition:

$$b_v + \eta > \theta, \quad \text{where } \eta \sim \text{Lap}(\lambda). \tag{13}$$

Subcritical Branching Intuition
The theoretical advantages of the truncation mechanism $\max(\theta - \delta, \cdot)$ becomes apparent when examining an empty sub-domain ($c_v = 0$) at $\theta = 0.0$. For any depth $d \geq 1$, the formulation guarantees that the adjusted count is bounded exactly at $b_v = -\delta$. The probability of this empty node erroneously triggering a split simplifies via the cumulative distribution function of the Laplace noise:

$$\Pr[\eta > \theta - b_v] = \Pr[\eta > \delta] = \frac{1}{2}\exp\left(-\frac{\delta}{\lambda}\right) = \frac{1}{2}\exp\left(-\ln\beta\right) = \frac{1}{2\beta} \tag{14}$$

Because a split generates exactly β children, the expected number of empty child nodes that will continue to split is $\beta \times \frac{1}{2\beta} = 0.5$. This guarantees that the random expansion of empty space finishes, bounding the privacy leakage cleanly without a pre-defined tree height.

One-Dimensional Binary Specialization
Our implementation instantiates this multi-dimensional framework into a one-dimensional continuous discretizer by setting $\beta = 2$ (while in its paper β was set to 4). Under this binary configuration, each node split represents a midpoint bisection of a continuous interval $[x_{\min}, x_{\max}]$ into two symmetric sub-intervals. Owing to this fact, the noise scale collapses to a constant $\lambda = \frac{3}{\epsilon_{\text{tree}}}$ and the step attenuation penalty simplifies to $\delta = \lambda \ln(2)$.

This formulation guarantees that the cumulative privacy budget consumed during the splitting remains invariant to the ultimate depth and resolution of the generated bins.

Domain Boundaries and Post-Processing Inversion
In addition to the already stated, as emphasized by Ganev et al. [13], extracting the domain limits themselves (the minimum and maximum bounds of the underlying data) must be executed via Differential Privacy or derived from public domain knowledge. Otherwise, the pipeline remains vulnerable to membership inference attacks.

Furthermore, once the marginal-based model completes the generation of synthetic data, the resulting output is discrete, whereas the original sensitive data may have had attributes that were continuous. Because of this, an inversion process must be implemented to reverse the discretization. This reconstruction process is straightforward enough (provided that differentially private or public domain knowledge bounds are available) as calculating exact bounds directly from the sensitive data would violate the post-processing theorem and breach DP. Empirically, Ganev et al. [13] demonstrated that performing simple uniform sampling within the bin boundaries for data reconstruction, given a sufficiently large number of bins, outperforms more complex geometric or statistical sampling techniques. Thus, we simply get the public domain bounds and do a uniform sampling within them in order to reconstruct the continuous numeric attributes.

3.5 SLURM

SLURM is a priority based batch scheduler for HPC clusters. Its purpose is to manage the execution and tracking of the cluster's jobs, also taking into account the handling of the queue of jobs when resources are scarce [25]. An scheduler is an algorithm whose job is to decide which job to execute at any point in time in order to optimize the utilization of the machine and minimize queue time. Furthermore, in SLURM's case, it is priority based. A priority based scheduler orders jobs decreasingly by an integer, the priority. This priority is calculated as a weighted sum of the following different characteristics of the job request:

- Size: The proportion of CPUs used by the job with respect to the available total. Administrators can choose if they want to prioritize small jobs or bigger jobs. SLURM by default prioritizes bigger jobs.

- Quality of Service (QoS): this factor benefits smaller jobs. This means higher priority upon submission, meaning normally a smaller wait time. Typically there are several QoSs according to job length and size and each is configured with an associated priority (an integer) and the total QoS is calculated as the proportion with respect to the largest priority of all QoSs.
- Age: the time the job's been in the queue. It grows linearly until reaching a max time (and then it is flagged as max time). This is calculated as the time passed since submission.
- Fair share: this factor favors users who haven't yet utilized their quota of resources and penalizes those who overuse the machine. Moreover, it normally has a bigger weight in the weighted sum.

In addition to this, SLURM has a second algorithm called backfill. Backfill is an optimization technique which schedules jobs of lower priority in between the "holes" of the jobs (with higher priority) being executed as long as they don't interfere with them. As stated, one of the main objectives of this project is to replicate the findings of the Autosubmit study [25] utilizing synthetic data rather than the original dataset. This replication aims to evaluate the utility of differentially private data and determine its viability for enabling external researchers to analyze safely the supercomputer's traces. Consequently, utilizing this specific scheduler is critical, as it is the exact same SLURM scheduler employed in the study to replicate, and which was provided by this work's supervisor.

4 Experimental Development

In this section we describe the iterative, empirical development process that lead to our final pipeline. This consisted on a empirical approach where possible solutions were tested, evaluated and either discarded or used and errors were corrected as they were localized. Said solutions naturally began more simple and grew in complexity as empirically we added methods and techniques to improve the results, leading to a pipeline that balances rigorous privacy guarantees with high utility.

In order to apply said experimental approach, a series of Datasets were employed.

4.1 Datasets

Three public HPC workload logs were used for this work, each offering different trade-offs. The first, derived from the Fugaku supercomputer [41], is rich in variables and free of missing values; the second, from the older Curie system [12], is sparser and has missing values but was the basis of my supervisor's earlier experiments; the third was added to ensure the final scripts were general enough

and came from the ANL Intrepid Supercomputer, a system of similar age as Curie (both of them have long been shut-down).

It must be stated that the practical needs of the project eventually led us to primarily utilize Curie's dataset. Autosubmit's analysis, which we aim to replicate, used Curie data, and the associated SLURM simulator scripts were tailored to that dataset's specific structure and idiosyncrasies, making it the natural choice for reproducible validation.

4.1.1 Fugaku Workload Dataset

This dataset was initially selected for experimentation due to its number of variables, its complete absence of missing values and errors, and because it comes from a more modern HPC system. Created from the operations of Supercomputer Fugaku [41], one of the world's most powerful supercomputers, located in Kobe, Japan, it features up to 45 variables. The complete original dataset is massive, containing 24 million jobs executed at Fugaku over 3 years and is divided in files. Of these files, only the first one, 21_03.parquet, was used (it is transformed into a .csv within the script). The variables deemed most important are found in Table 1:

Attribute	Description	Type
jid	The id assigned to the job when it is submitted.	string
usr	The username of the user submitting the job.	string
cnumr	Number of cores requested for the job.	int
nnumr	Number of nodes requested for the job.	int
adt	Arrival date time (submission time).	date time
qdt	Time of insertion in the job queue.	date time
ec	Job exit code.	int
elpl	Elapsed time limit (the time limit set to the job duration).	float
idle_time_ave	Average idle time of the job.	float
msza	Memory size allocated.	float
mmszu	Memory size used.	float
uctmut	User CPU time total use.	float
jobenv_req	Job environment requested.	string
flops	Number of floating point operations per second.	float
pclass	Performance class (either compute-bound or memory-bound).	string
exit state	Exit state of the job execution (either completed or failed).	string
duration	Duration of the job execution in seconds.	float

Table 1: Table of the variables retained for the fugaku dataset and their description

Only these attributes were used since what we care most about is about execution times and time in queue, besides the percentage of use of the system of the users, rather than details about the machine itself, since those are the attributes that slurm utilizes [20].

4.1.2 CEA-Curie Dataset

This is another dataset of a different, older, supercomputer: Curie. This supercomputer was a cluster operated by CEA (French Atomic Energy Comission) from 2011 to 2012[12]. This dataset is, in truth, a log containing 20 months of job accounting records from one partition of the supercomputer and is in a .swf [4] format that is later transformed into a .csv via a script developed for that task I.2. Said .swf stands for "Standardized Workload Format" and is a standardized data format used by researchers and system administrators to record, share, and model how jobs are executed on supercomputers and large-scale parallel clusters (and is the only format accepted by the slurm simulator [20]).

This log was generously provided by Joseph Emeras and is distributed through the Parallel Workloads Archive [12].

Compared to Fugaku's dataset, Curie is much simpler (and yet more complex to process). It has fewer variables, it has frequent missing values or errors (encoded as -1) and reflects an older HPC system.

Nevertheless, this is the dataset used in Autosubmit. Thus, to validate whether differentially private synthetic data is able to reproduce those findings, this dataset must be used. Owing to the fact that many of the available fields are completely absent (all -1) for a large fraction of the jobs, we only synthesize a subset of variables that are actually populated and meaningful for scheduling studies (the only ones SLURM uses [20]). The 11 variables retained are found in Table 2:

Attribute	**Description**	**Type**
Job_ID	Unique identifier assigned when the job is submitted.	string
User_ID	The username of the user submitting the job.	string
Group_ID	ID of the group the user belongs to (important because BSC uses a similar accounting structure).	string
Requested_Processors	Number of processors requested for the job.	int
Allocated_Processors	Number of processors allocated for the job.	int
Submit_Time	Submission time, expressed in seconds since the start of the log (from year one not 0).	int
Wait_Time	Time spent in the queue (seconds).	int
Run_Time	Time the job spent executing (seconds).	int
Requested_Time	Wall-time limit requested by the user (seconds).	int
Status	Numerical identifier for the job final status (categorical variable).	string
Partition_Number	Numerical identifier of the job's Partition(categorical variable).	string

Table 2: Table of the variables retained for the CEA-Curie dataset and their description

4.1.3 ANL-Intrepid Dataset

This is another dataset of a different supercomputer: Intrepid. This supercomputer was a Blue Gene/P system operated by the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory from 2009 onward [5].

This dataset is alike to Curie's, a log containing the first 8 months of job accounting records from the 40-rack production partition of the supercomputer

and is in a .swf format that is later transformed into a .csv via my own script. This log was generously provided by Susan Coghlan and Narayan Desai and is distributed through the Parallel Workloads Archive [5].

Compared to Fugaku's dataset, ANL-Intrepid has structural characteristics unique to its architecture and is more similar to Curie's. This dataset was used to corroborate the generalized qualities of the developed scripts and ensure that they where able to ingest and properly manipulate different datasets. Additionally, the Utility and Privacy was evaluated and a trace was also simulated using SLURM scheduler as a simulator [25]. Because many of the available fields are completely absent or uninformative for scheduling modeling, only a subset of variables deemed populated and meaningful (the ones that slurm uses [20]) were synthesized. The 11 variables retained are found in Table 3:

Attribute	**Description**	**Type**
Job_Number	Unique identifier assigned when the job is submitted.	string
User_ID	The username of the user submitting the job.	string
Requested_Processors	Number of processors requested for the job.	int
Allocated_Processors	Number of processors allocated for the job (often rounded to match the hardware partition size).	int
Submit_Time	Submission time, expressed in seconds since the start of the log.	int
Wait_Time	Time spent in the queue (seconds)	int
Run_Time	Time the job spent executing (seconds).	int
Requested running Time	Wall-time limit requested by the user (seconds).	int
Queue_Number	Numerical identifier of the job's queue (categorical variable).	string

Table 3: Table of the variables retained for the ANL-Intrepid dataset and their description

4.2 Validation Metrics Employed

For the validation of the Differentially Private (DP) synthetic data generators, there are two main dimensions we must evaluate: Utility and Privacy.

Utility measures the analytical value of the synthetic data and is generally straightforward to evaluate by comparing statistical properties. Privacy, on the other hand, is formally guaranteed by the mathematical properties of Differential Privacy (DP) [8, 10]. Therefore, when evaluating privacy empirically, we are essentially verifying that these theoretical guarantees hold in practice, therefore measuring the difficulty for an adversary to successfully execute data

reconstruction and re-identification attacks, primarily Linkage and Membership Inference Attacks.

Regarding utility, we mainly evaluated it by measuring the synthetic data's resemblance to the real data and with the accuracy of Machine Learning models trained on the synthetic data.
Specifically, we use the following:

- **Bound difference and bound error**: These metrics measure how well the synthetic data preserves the absolute domain boundaries (minimum and maximum values) of the numerical attributes compared to the real dataset. We evaluate this by calculating the difference between the bounds of the original and synthetic data, and calculate the mean of the difference.

- **Total Variation Distance (TVD)**: TVD is a standard statistical distance measure used to quantify the maximum difference between the probability distributions of categorical attributes in the real and synthetic datasets. A lower TVD indicates that the synthetic distributions closely mimic the real ones. This measure was used by Tao et al. in his DP generation benchmarking paper [45].

- **Wasserstein Distance**: Also known as the Earth Mover's Distance, this metric evaluates the cost of transforming the synthetic probability distribution of numerical variables into the real probability distribution . This measure was used by Tao et al. in his DP generation benchmarking paper [45]. It is highly effective for capturing the shape of continuous distributions.

- **Plots of Real vs. Synthetic distributions**: A visual inspection of the overlaid distributions of the synthetic and sensitive datasets. It provides intuitive confirmation of the generative model's performance. They serve as a visual confirmation rather than a robust evaluation method.

- **Machine Learning (ML) Utility**: This is a metric geared towards evaluating the predictive power of the synthetic data using the Train-on-Synthetic, Test-on-Real (TSTR) paradigm. In order to do this we train ML models on the synthetic data and test their predictive accuracy on the real data [40]. High performance indicates that the synthetic data preserves relationships and is similar to the real data, or in other words, useful. With this objective in mind, we train a logistic regression (a linear classifier), which shows us if linear relations are preserved and a Random Forest (a tree-based, inherently non-linear classifier) to show us if non-linear relations are preserved. This is shown in the form of a LR loss (the difference between the accuracy of a real-on-real logistical regression model and a synthetic-on-real) and a RF loss (the same but with Random Forest instead).

- **Spearman Correlation Error**: This computes the difference between the Spearman rank-order correlations of the real and synthetic data. It

evaluates how well the non-linear relationships between variables are preserved.

- **Pearson Correlation Error**: Similar to the Spearman error, this computes the difference between the Pearson correlation matrices, explicitly evaluating the preservation of linear relationships between variables.
- **3-way Marginals**: Evaluates the preservation of relationships across three variables simultaneously. By comparing the higher-dimensional marginal distributions between real and synthetic data (when both have been discretized), we observe how effectively these relations were preserved [28]. The closest it is to 0, the better.
- **Higher-Order Conjunctions (HOC)**: Used in the challenges by the National Institute of Standards and Technology (NIST) from the US [28], HOC measures the ability to answer complex, multi-attribute queries safely, verifying that high-dimensional combinations of attributes are well preserved. The closer it is to 0, the better. In addition, as with the three way marginal metric, both datasets involved must have all of its attributes discrete.

Finally, for utility, we use the synthetic HPC workload logs to simulate a SLURM trace, which we subsequently evaluate using standard trace metrics [25]. In order to accomplish this, we first format our synthetic results into the Standard Workload Format (`.swf`) ingested by SLURM using a conversion script (presented in Appendix I.5), which builds upon the `pyswf` script [27]. Following this preparation step, we now execute the SLURM simulator using the following Docker image [26]. Next, we simply use the `launcher.py` script, found in the same repository as the SLURM Docker environment, to launch the simulator. In order to do this, we must first download the corresponding git repository:

```
$ git clone https://gitlab.earth.bsc.es/mgimenez/docker-ubuntu-ce⌋
↪  s-slurm-sim.git
```

Figure 3: Downloading the SLURM repository.

And then go to the folder and build the corresponding docker container:

```
$ cd docker-ubuntu-ces-slurm-sim
$ docker build . --tag
↪  mmarciani/docker-ubuntu-ces-slurm-sim:latest
```

Figure 4: Bulding the docker container.

After this we now create an empty file where slurm will write its simulated logs and we can launch the job with its launch.py file:

```
$ python3 launcher.py
↪  /your/path/to/job/trace/your_synth0roriginal.swf \
  your/path/to/the/results/file/your_results_file.swf \
  put_in_simulation_time --image
  ↪  marciani/docker-ubuntu-ces-slurm-sim \
  --tag latest
```

Figure 5: Executing SLURM.

The simulator returns .swf job logs at the file you created.

To later obtain the statistics and plots of the results for its evaluation, a script provided by the project's supervisor Manuel G. Marciani I.9 was used. As for the privacy, we mainly evaluate its resistance to Linkage and MIA attacks as well as its possible memorization and overfitting with the following metrics:

- **Distance to Closest Record (DCR) Metrics**: DCR measures, in our case, the Euclidean distance between a synthetic record and its nearest neighbor in the real dataset [16]. If the synthetic data overly resembles the original data, the distance will be near zero. We evaluate multiple facets of this:
  - Mean 1 DCR: The average distance to the single closest real record.
  - Mean 5 DCR: The mean distance from each synthetic record to its 5th closest real record. This metric captures the broader local density around synthetic points. A larger value implies greater privacy, as synthetic records do not cluster closely around real records (more dispersed), making them harder to link to real individuals.
  - Mean DCR: The average distance across all nearest-neighbor pairs.
  - Min DCR: The absolute minimum distance found between any synthetic record and any real record, representing the worst-case scenario for privacy. However, this value being very close to 0 (or 0) does not mean that privacy is breached since this can naturally happen given a dataset with many repetitions or closely clustered similar individuals.
- **Nearest Neighbor Distance Ratio (NNDR)**: Instead of measuring only the closes neighbor, it measures the ratio between the Euclidean distance for the closest and second closest real neighbor to any corresponding synthetic record as stated by Zhao et al. [51]. This ratio is between zero and one, and higher values indicate a better privacy while lower values mean that sensitive information may be revealed.
- **Exact Clones**: We explicitly count the number of synthetic records that are exact duplicates of real records. It is important to note that the presence of clones does not necessarily indicate a privacy breach: natural repetitions in the real data or clusters of similar individuals can naturally

produce identical records. However, when clones are present, the Nearest Neighbor Distance Ratio (NNDR) grows in importance. If the NNDR is close to 1, the clones are unlikely to be problematic, as they do not indicate that synthetic records are abnormally close to their real counterparts. This is also the case for the DCR min when it is zero.

- **Maximum Mean Discrepancy (MMD)**: MMD measures the distance between the distributions of synthetic and real data, where values closer to zero suggest the two are indistinguishable [16].

- **Discriminator Test**: A binary classification model is trained to distinguish between real and synthetic records [40] which effectively is a type of linkage attack. If the classifier's accuracy is near 50%, the synthetic data is indistinguishable from the real data. High classification accuracy implies the generator produced easily identifiable synthetic data.

The script implementing the evaluation functions and evaluation execution can be found in Appendix I.7 and I.8 respectively.

Additionally, the CLOVER library's evaluation framework is used [40] for more complex validation primarily focused on privacy, specifically targeting Membership Inference Attacks (MIA) and Linkage Attacks. As for what metrics, the CLOVER framework extends the standard distance metrics used for the evaluation by actively simulating adversarial attacks. For MIA, CLOVER trains detector networks or utilizes black-box statistical tests to compute the Precision, Recall, and Accuracy of an attacker attempting to flag training records. For Linkage attacks, CLOVER assesses the probability that a synthetic record can uniquely map back to a single sensitive real record given varying amounts of background knowledge. By computing the success rates of these simulated attacks, CLOVER provides a metric, a risk score, that complements the mathematical bounds provided by the DP ϵ parameter.

4.3 Initial experiments

After reading the corresponding literature first, we began testing differential privacy with simple practical applications. We began with two of the main DP Python libraries: OpenDP [36] and PyDP [37]. Through applied implementations of both libraries, it was soon discovered that they approach global privacy from fundamentally different perspectives. PyDP, a Python wrapper for Google's Differential Privacy project, operates as a collection of "black boxes", it is highly user-friendly, allowing a user to simply specify a target statistic (sum, mean, variance, percentile), choose a Laplace or Gaussian noise mechanism, set the privacy budget ϵ, and let the library handle sensitivity bounds and missing categories internally. Opposite to this we have OpenDP, developed by Harvard's Privacy Tools Project, and which functions at a much lower level, providing fundamental "bricks" rather than pre-assembled functions. Natively

written in Rust for memory safety and performance, OpenDP requires the researcher to explicitly construct processing graphs, define domain spaces, input and output distance metrics, and the precise privacy unit, while manually deriving query sensitivity to calibrate the noise scale. Every operation is explicitly separated into dataset transformations (clamping, resizing) and mathematical measurements that inject noise.

Through the interactive tutorials and documentation hosted on OpenDP's website, we implemented successfully privacy-preserving algorithms to calculate DP means and counts for the first time, and later on further expanded this by independently implementing more complex calculations such as differentially private standard deviations with both libraries.

These implementations, allowed us to conclude, after the use of both these libraries, that OpenDP is more difficult but ultimately more powerful and of broader potential use.

4.4 Experiments on HPC logs: choosing a synthesizer

After experimenting with simple testing datasets and simple applications, we decided to began experimentation to solve the thesis main objectives. For that, we began with implementations of DataSynthesizer and a Local DP approach.

4.4.1 DataSynthesizer

As stated in Haoyue Ping et al.'s paper [39], DataSynthesizer is a Python library that uses a series of techniques to receive a sensitive dataset as input and output a statistically similar synthetic version with strong privacy guarantees. Since this aligned with this work's objectives, and given its simplicity and ease of implementation, we explored it experimentally. As for the method, its main idea is to create a synthetic dataset whose summary statistics are similar to its private counterparts. In order to do this, the method infers the domain of the attributes, derives a description of their specific distributions (approximates the distribution), and injects noise to them in a differentially private way, then using this description to sample the new data generating thus synthetic datasets of any size.

The method offers mainly two ways of approximating said distributions, a simpler mode which fits said distribution with an equal-bin frequency histogram or bar chart depending on the attribute's type, injecting noise through a Laplacian mechanism into the counts of each bin.

The other mode is more complex and approximates said distributions using a differentially private Bayesian Network, a method called PrivBayes explained in Zhang et al.'s study [50], in order to better preserve the data's correlations, then sampling from said network.

To begin with, our experimentation started with DataSynthesizer's GitHub repository examples and the well-known California Housing dataset as a testing ground. After successfully replicating the examples we decided to begin experimentation with the Fugaku supercomputer dataset [41]. This dataset was

selected because it possessed many variables of interest, such as User ID, allocated and requested CPU resources and more, and critically, contained no missing values (unlike Curie's Dataset) eliminating the need for preprocessing at this early stage.

However, experimentation showed that this method's usage of equi-width histograms and/or Bayesian networks, and uniform sampling were not the best way to properly synthesize HPC log data. Given HPC's skewed, long-tailed distributions (as seen in Figure 2), most data points simply fell into a couple of bins, losing all nuance and information to which then noise is added. By the time the uniform sampling is performed, utility has been destroyed.

4.4.2 Local DP Approach

The other of our initial implementations was a Local DP approach implemented with OpenDP. A custom script was developed to implement a local DP approach, job by job and variable by variable. For numerical variables, both the Laplacian and Gaussian mechanisms were tested, ultimately determining that the Laplacian mechanism was better suited for this case. As for categorical variables, a k-ary randomized response mechanism was employed. In addition, logarithmic transformations were also employed followed by standard normalization to reduce to better represent the data and allow the mechanism to better introduced noise. Both of these transformations were experimentally tested and seen to improve results.

Regarding date time variables, these attributes were converted from timestamps to UNIX seconds (seconds from January first 1970) and a modified Laplacian mechanism with bounded noise [47] was applied. This bounded Laplace mechanism consists in having the Laplacian noise truncated to a finite interval and re-normalized, keeping timestamps within realistic bounds while preserving ϵ-differential privacy. Concerning string identifiers (user and job names), a deterministic hashing was combined with k-ary randomized response.

However, the evaluation of this approach indicated this method's shortcomings.

4.4.3 Results of the experimentation with DataSynthesizer and Local DP approach

After implementing these approaches and performing rudimentary evaluations, at this stage still primarily visual comparisons of original versus synthetic distribution plots, it became apparent that neither approach was sufficient for producing high-utility HPC synthetic data (thus with the visualization we had sufficient insight). The local DP approach suffered from severe utility degradation due to the noise accumulation across high-dimensional records, while DataSynthesizer struggled with the unique characteristics of HPC workloads: date time handling was problematic (converting date-times to seconds and back introduced errors), string identifiers were replaced with random strings losing all frequency structure, and the long-tailed numerical distributions were poorly

captured by equal-width histogram binning. This is shown in Figure 6, where we can see the aforementioned inability by DataSynthesizer (first image) and the local DP based implementation (second image) to properly synthesize our distributions.

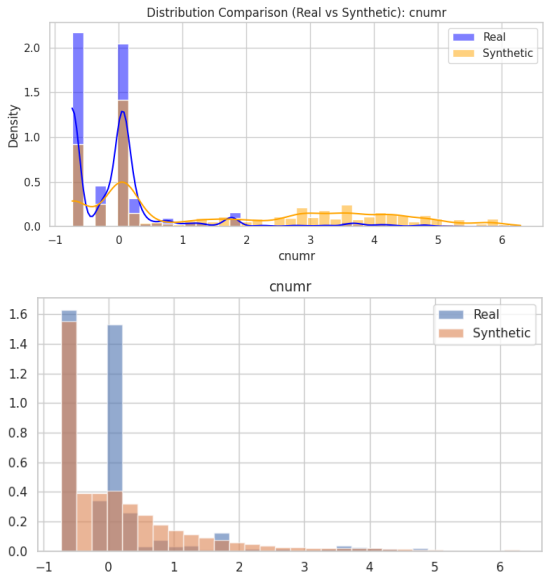


Figure 6: Comparison of original Fugaku data distributions against early synthetic versions produced by local DP and DataSynthesizer, highlighting the challenges of long-tailed distributions.

This realization motivated the pivot to more advanced methods. We began with a rudimentary experimentation on DP-GANs (following Goodfellow's foundational work [15] and the PATE-GAN approach [48]).

4.4.4 GAN based methods

These methods are based off Generative Adversarial Networks (GAN) [15] that learn from the real data to then generate synthetic data that is similar. GANs by themselves, however, don't guarantee any privacy. There have been different approaches to make GANs differentially private, in this case the example of PATE-GANs [48] by Jordon et al was explored. Jordon et al proposed a method that modified the GAN's inner workings so as to guarantee differential privacy. Jordon et al does this by modifying the training procedure to be differentially private, specifically by using a modified version of the Private Aggregation of Teacher Ensembles (PATE) framework. Only the discriminator is explicitly modified to be differentially private, then the generator trained from it is also proven to be differentially private.

However, it is necessary to clarify that this family of generative DP methods

was scarcely tested due to the Nature of GANs themselves and their high computational cost. GANs are Neural Networks notoriously hard to train (training convergence is hard to ensure) and are very sensitive to hyperparameter choices. Furthermore, their performance typically does not generalize proficiently to different datasets, meaning that even if such a Neural Network successfully generated a synthetic, differentially private, version of the Curie dataset, the main dataset used in this work, their dependence in hyperparameter tuning means that most likely it would require substantial re-tuning, and possibly re-training, in order to work for any other dataset. Paired with this, multiple Competitions and benchmarking studies[40, 45] have suggested that these methods offer neither better utility nor privacy, performing similarly to other alternatives. For these reasons reasons this approach was discarded. Nevertheless, these methods remain an avenue of further exploration since they do posses the theoretical advantage of being very capable at finding latent relationships in data.

This disadvantages drew us to quickly discard them instead for MST [28], found after researching for alternatives, and which after prototypes were implemented and initial testing and evaluation done, was quickly found to be the most promising direction.

With a DP Synthetic Data Generation method now chosen, we now moved from using the Fugaku dataset to using CEA-Curie for the remaining experiments.

4.4.5 Evaluation of the pipeline in a new dataset

This pivot presented new challenges. Unlike Fugaku, Curie contained missing values across several variables. This required the implementation of preprocessing steps before further experimentation could be performed.

We began with a KNN imputer to handle missing values. This method imputes a missing value by finding its k closest neighbors with a distance metric to then aggregate their observed values or by the mode (in the case of categorical attributes). We reasoned that this was a promising imputing option since by looking at neighboring records we would naturally produce values that exist in the dataset, avoiding the introduction of impossible values such as zeros or negatives. However, empirically, KNN was proved to be too slow, not properly scaling with the dataset's size. After investigating alternatives, we adopted an iterative imputation strategy by using scikit-learn's `IterativeImputer` with `ExtraTreesRegressor` as the estimator [43]. This approach models each attribute with missing values as a function of the other attributes iteratively refining estimates until convergence, and in this case said estimations are done via a ExtraTreesRegressor which is a randomized tree ensemble. Experimental testing confirmed that this imputer is faster, scales properly, and avoids estimating values outside the data's domain.

That said, this preprocessing introduced an issue: that any data-dependent preprocessing, including imputation, can compromise the end-to-end Differential Privacy guarantee if not done properly. In simpler words, this imputation was breaching privacy, since it was learning complex, exact relationships between

variables in order to impute the missing values, and this overfitted information was passed to the MST synthesizer before any DP noise was applied.

In addition to this, as the pipeline was systematically evaluated for improvement, several other errors that compromised the privacy guarantees surfaced. The most significant of these were related to the post-processing phase. In the initial implementation, after the MST synthesizer produced binned categorical data, the exact bin boundaries calculated from the sensitive data were being used to "un-bin" the synthetic records, transforming them back to continuous values. This was a direct, and severe, privacy leak. This is because, even if the binned data itself is differentially private (having passed through the MST mechanism) and protected by the post-processing theorem 1, the bin edges were derived from the raw, sensitive, data without any privacy protections, which breaches said theorem. Now, an attacker who observed the synthetic data could reverse-engineer the exact quantiles and distributional properties of the original dataset.

Issues with the normalization pipeline were also uncovered. A MinMax scaling was being applied, which as it name indicates, uses the min and max value of the original sensitive dataset, an also severe privacy leak, which only grew in severity when said normalization was undone in the post-processing stage since, to undo this normalization, it once again used the exact min and max values of the original sensitive data.

With these issues recognized, corrections had to be made. The issue with the normalization was easily solved: we ceased to use it. The imputation leaks where initially solved in a similar manner: we simply dropped the missing values. This may seem to be a straightforward, private, correction of the imputation strategy, but it is not the case. Dropping missing values is done by looking at the sensitive data, it is akin to a function of the raw counts of missing values, function which is not being perturbed with any noise, allowing an adversary to have the possibility to infer the existence of specific records from the "filtered" dataset. As a consequence, our solution was to impute numerical and categorical attributes with 0.0 or a "missing" category respectively. Although this is a crude imputation, it is data-independent and therefore does not breach DP. Furthermore, it has several advantages over alternative DP imputation strategies such as imputing with a DP mean: it consumes no privacy budget, and it avoids introducing artificial spikes in sections of the distributions that formerly were the tail (a common problem when imputing heavily skewed, long-tailed data with a fixed number).

In addition, the pipeline includes an attribute-dropping step where, in a DP fashion, those attributes with more than 50% missings are removed, with said missing count being perturbed with a Laplacian Mechanism.

The discretization, however, proved a more challenging issue to solve.

4.5 Discretization selection

After the experimentation to set the synthesizer and fixing the privacy leaks, we began the process of tackling the bigger challenge: Discretization. Specifically,

the core challenge was finding a way to safely discretize continuous variables that (1) respected the end-to-end DP guarantee, (2) preserved the complex distributional structure of HPC data (unlike equal-width binning), and (3) maintained multivariate relationships. Extensive research was conducted in this regard.

In the literature we found Ganev et al.'s paper "The Importance of Being Discrete" [13], a study that evaluated the impact of discretization on end-to-end DP synthetic data generation, just as we had realized ourselves. The paper compared DP versions of uniform binning, quantile binning, k-means clustering, and PrivTree [49]. Its key findings were, to summarize, that PrivTree consistently either outperformed or performed equally to other discretizers across various metrics, privacy budgets, datasets, and downstream models, demonstrating a robust privacy-utility tradeoff with minimal variability. Furthermore, this method was uniquely suited for HPC log data, since it would naturally bin the long tails of the distribution together into a not-so-sparse bin while further partitioning the denser regions of the distribution.

Given these reasons, this method was selected for the discretization. However, in addition to this, we had an issue that had been until now overshadowed: all discretization methods researched required bounds to be provided, bounds which needed to either be estimated in a differentially private manner or obtained with publicly available domain knowledge. At this stage our aim was to introduce as little parameters as possible, thus, a differentially private bound estimation function was implemented based on the Desfontaines algorithm [7] for privately estimating the domain bounds of each numerical variable given coded-in broad bounds. This method used a logarithmic histogram with a Laplace mechanism to inject DP noise into the counts and a coded-in threshold to find upper and lower bounds that satisfied differential privacy.

4.6 Experimental testing of two possible MST implementations

Reviewing the literature Qi et al.'s CLOVER framework [40] was found. This is a framework which provides a comprehensive benchmarking library for synthetic data generation methods, with built-in implementations of MST and robust evaluation metrics for both utility and privacy. A test was done where CLOVER was integrated into the developed system as an alternative preprocessing and synthesizing pipeline, since CLOVER is a library that does its own preprocessing and discretization, and began a systematic comparison against our custom implementation.

The evaluation revealed a classic privacy-utility tradeoff. Our pipeline (around this time called PT-MST, meaning PrivTree-Maximum Spanning Trees) preserved correlations far better, achieving a mean absolute correlation error of 0.0552 compared to CLOVER's 0.12, due to PrivTree's adaptive binning protecting the rank-order structure that Spearman correlation depends on more proficiently. However, CLOVER demonstrated superior privacy metrics: its Distance to Closest Record (DCR) values were significantly higher, with no exact clones produced, while PT-MST occasionally generated a small number of

exact matches for isolated outliers at this point in time (which was due to some privacy leaks still present at this stage).

The root cause was CLOVER's use of equal-width binning with a fixed number of bins (100 by default). Even though this destroyed most rank-order information, it spread synthetic records more uniformly across the domain, naturally pushing DCR values higher. Our PrivTree-MST pipeline, conversely, concentrated bins where real data was dense, producing more realistic distributions but placing synthetic records closer to real ones.

It was also discovered that CLOVER internally applied both standard normalization and MinMax scaling, the latter overwriting the former, and used KBinsDiscretizer with uniform strategy. For HPC data with massive outliers, this meant most data points could be crushed into a couple of bins, erasing their internal rank-order, with this error further compounding when the discretization was undone with a uniform sampling mechanism.

After careful consideration, It was decided to keep both options in the overall pipeline, but continued developing the PrivTree-MST approach as the primary method, given its superior correlation preservation and more realistic distributional fidelity, as seen in Table 4.

Metric	**PrivTree-MST**	**CLOVER-MST**
RF TSTR	0.9025	0.8939
Mean DCR	208,010	10,373,466
Mean Abs Corr Error	0.0552	0.1200

Table 4: Comparison of custom PrivTree-MST versus CLOVER-MST on the CEA-Curie dataset ($\epsilon = 1.0$).

4.7 Logarithmic Transformation Experimentation

So far in our research we had applied few, if any, transformations in our data, however, given its skewness, we explored what effects a logarithmic transformation had on the pipeline's performance. Three configurations were tested: no log transformation, full log transformation (all numerical variables), and selective log transformation (only heavily skewed variables: allocated processors, requested processors, and runtime).

The logarithmic transformation, by compressing large values, expanding resolution among small values, and converting multiplicative relationships into additive ones, improved the preservation of non-linear (Spearman) correlations. In log space, relationships that appeared non-linear in raw space became closer to linear, making them easier for the MST to capture accurately under DP noise constraints.

The selective log transformation emerged as the apparent best trade-off: it achieved the best Wasserstein distances across all continuous variables, strong ML utility, solid privacy, and excellent non-linear correlation preservation. Full log transformation, while offering better DCR privacy values, caused bound

inflation on some variables making it second in performance improvement to the selective log transformation. A comparison of the tested configurations is presented in Table 5.

However, later reasoning made us realize that even if selective log transformation was the best approach for our particular dataset (Curie), it was not generalizable. Thus, it was decided to use the full log transformation instead. This evaluations were already being done with the last iteration of the validation script, and all of its metrics, shown in the validation section 4.2.

Method	**RF Loss**	**Mean DCR**	**NNDR**
PrivTree Selective Log	0.0704	208,010	0.8436
PrivTree Full Log	0.0698	1,748,583,459	0.8751
PrivTree No Log	0.0757	215,688	0.8458
CLOVER	0.1044	6,780,844	0.7518

Table 5: Comprehensive comparison of all tested configurations on the CEA-Curie dataset ($\epsilon = 1.0$).

4.8 Generalization to multiple datasets and SLURM simulation experimentation

With the core of the PrivTree-MST pipeline established, the script was generalized in order for it to be able to accept multiple different datasets. This generality was tested on two additional HPC workload logs: the RIKEN Fugaku dataset [41] and the ANL Intrepid dataset [5]. The pipeline successfully synthesized private counterparts of both datasets, with a metadata file of each dataset as the only requirement, demonstrating its generality.

The synthetic data generated from ANL Intrepid was also used to conduct SLURM simulation experiments, although only with the original, non synthesized, trace since our objective here was demonstrating the pipeline to be able to ingest diverse datasets, following the approach of Marciani [25]. Using the BSC SLURM Simulator Docker environment [26] and the PySWF library [27], the workload traces were converted to SLURM's SWF format and an evaluation of how proficiently the synthetic data, generated with $\epsilon = 1$ preserved the statistical properties was done. This end-to-end validation confirmed that the privacy-preserving pipeline was capable of being used against diverse datasets other than CEA-Curie. The results of the statistical evaluation can be found in Appendix H.6, while the slurm trace simulation can be found in Appendix H.7.

4.8.1 Fixing of Errors

During the previous experimentation, several privacy leaks were found.

This time the issue was located in the categorical variable handling. Our initial approach used deterministic pseudonymization (replacing User_ID and Group_ID with random integers), which leaked the exact categorical domain,

from which an attacker could infer the presence of rare users from the set of pseudonyms. Following the approach described by Korolova et al. [23], this was instead replaced with a DP categorical processing algorithm: under approximate differential privacy we count occurrences of each category, add Laplace noise, and only retain categories whose noisy count exceeds a threshold, a dynamic threshold properly set up according to approximate differential privacy as shown in Korolova's work. Any dropped categories are merged into an "OTHER" label. This step consumes a tiny fraction of the privacy budget (1%) but ensures the categorical domain itself is protected.

To establish this dynamic threshold rigorously, the algorithm relies on specific parameters that define the boundary between safe data and high-risk "tail" categories:

- **Threshold (K):** The minimum baseline value that a category's noisy count must reach to be safely released to the public dataset.
- **User Contribution Limit (d):** The maximum number of distinct categories a single individual is permitted to contribute to. To prevent highly active users from having an unbounded influence on the noise scale, each user's history is clipped to their first d entries.
- **Privacy Parameters (ϵ, δ):** Given that this algorithm satisfies Approximate DP, its standard parameters of (ϵ, δ) are used.
- **Laplace Noise Scale (b):** The magnitude of the random noise added to the true counts, drawn from a Laplace mechanism. To successfully mask a user's potential impact across d categories, the scale must be set to:

$$b = \frac{d}{\epsilon} \tag{15}$$

By analytically solving for the exact boundary where the probability of a privacy leak is bounded by δ, Korolova et al. derived the optimal mathematical formula for the dynamic threshold K:

$$K = d\left(1 - \frac{\ln\left(\frac{2\delta}{d}\right)}{\epsilon}\right) \tag{16}$$

This formula demonstrates a direct trade-off between privacy and utility. As the privacy requirements become stricter (smaller values for ϵ or δ), the second term inside the parentheses grows, causing the overall threshold K to rise. A higher threshold forces the algorithm to filter out more sparse categories, safely aggregating them into the "OTHER" label to preserve user anonymity.

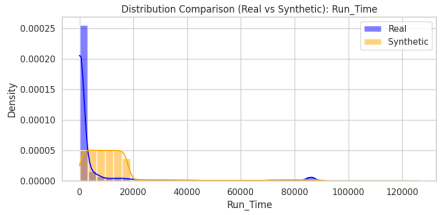
Moreover, the private bound finder based in the Desfontaines algorithm (mentioned in section 4.5) was removed. We realized, after reading Korolova's study, that it also possessed a privacy leak: its threshold was coded in, instead of properly established as shown in Korolova's study with equation 16. However, after correcting its privacy leaks with the proper thresholding, it was

found impossible to guarantee that it would not cut off distribution tails. Since in our work we preferred overestimating bounds (thereby keeping the distributions intact) rather than underestimating them and cutting of their tails, we pivoted to purely using broad domain-knowledge-based bounds combined with PrivTree's native adaptive binning. Testing confirmed that PrivTree performed proficiently with these wider, data-independent bounds. Now bounds may be overestimated, but no tails would be cut.

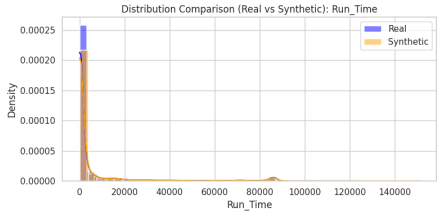
However, it must be noted that this also made the pipeline dependent on proper domain-knowledge bounds.

4.9 PrivTree Additions Experimentation

Testing was conducted to prove PrivTree to be better than an equal-binning counterpart implemented with the number of bins controlled dynamically using the Friedman-Diaconis rule [42]. This evaluation was done because this equal-binning discretization required no parameters and was simpler. Nevertheless, as with CLOVER, results showed PrivTree to be preferable. The difference in results is exemplified by the following plots in Figures 7a and 7b.



(a) Pipeline results with equal-binning as discretizer and epsilon 10.



(b) Pipeline results with PrivTree as discretizer and epsilon 10.

Figure 7: Comparison of runtime performance across different discretization methods.

In Figures 7a and 7b, we clearly observe the effect equal binning mixed with uniform sampling to rebuild the continuous numerical attributes has: most attributes fall in a few broad bins and all information and nuance was lost.

Finally, the "safety measure" for PrivTree's maximum depth was addressed. Until now a fixed parameter had been used as a safety measure which stop the algorithm once the bin's "width", their density, dropped below a threshold. A more principled approach was, however, devised. One that did not require any tuning: using the square root rule [42] as an upper bound to dynamically calculate the maximum tree depth from the dataset's size. In addition to this, the splitting threshold of PrivTree, θ, was coded-in to zero, the standard PrivTree configuration, as Zhang et al. [49] demonstrates, $\theta = 0$ ensures that nodes with sufficient point counts are properly handled.

4.10 Choosing an Epsilon

This is an inevitable step in differential privacy that is harder to perform than what it would seem at first glance.

As stated, ϵ represent the tradeoff between utility and privacy, the lower the ϵ the greater your privacy (and worse your utility) and vice versa. This means that the selection of ϵ depends on your work's specific objectives and in how critical the guarantee of privacy is. As Dwork et al discovered in her study, "Differential Privacy in practice: expose your epsilons!" [9], many times people choose epsilon arbitrarily. For this work, where privacy is paramount (thus the selection of the privacy budget is of critical importance), research was done about informed ways to make this decision.

First we have a simple but efficient approach widely used in statistics and Machine Learning for parameter selection: the elbow method. This basically consists, in the setting of differential privacy, in calculating, for a given set of epsilon values to test, a measure of the utility of the results achieved with said budget to then plot said utility measure against the epsilon. As ϵ increases, the measurement should fall, then you choose the epsilon at the "elbow" which provides the better tradeoff.

Other options explored were much more complex and proved not applicable to this work, working instead for simpler differentially private applications that do not require multiple compositions of differential privacy mechanisms and were based in the stricter definition of differential privacy.

One such method was proposed by Justin Hsu et al in "An economic method for choosing epsilon" [18]. In his work he exposes how the question of how to choose the key parameter, ϵ, has no rigorous answer and proposes a model where the aim is to balance two parties with conflicting goals, an analyst who wants to learn from the data and a prospective participant who needs to decide wether to include their data or not in the analysis.

His aim with this model is to take into account the impact of ϵ (and δ) in a differentially private study with the two "characters" mentioned. This model assumes rationality by both characters considering that the participant will choose to contribute their data to the study if their expected (monetary) benefits outweigh the possible risks (bad events that should befall them if their privacy is exposed). This approach is based in a interpretation of differential privacy in terms of bad events such that for a participant fearing a set of real word events, his participation doesn't increase or decrease the probabilities of this events happening by much (their participation increases this risk by a probability bounded by epsilon). On the other hand it also takes into account cost, because no events are equally harmful, thus, a cost is assigned to each of those "events" and using DP this cost can be bounded (demonstrated mathematically in the paper [18]).

However, we still have the same issue, ϵ is a knob that if high, costs increase (meaning the participant will ask for more money) whereas if low utility drops. One reason as to why the choosing of this parameter is so difficult is due to it packing a complex scenario of two parties with opposing interests.

That is why they propose a model to do this selection, a model involving the two parties. The analyst's concern is the accuracy of his study and, since the greater the population the less noise is needed to be added and the more accuracy we have, we need a constraint (or the analyst would use infinite populations). This constraint is the study's budget and the fact that each individual need be compensated for their participation from that budget. The participant meanwhile will contribute his data if the monetary compensation overturns the increase in risk taken.

This is an interesting proposal, but It was deemed unfit for this work. Mainly, because it would involve devising a budget and compensations in a domain of HPC job traces, which would in its core be a shrouded arbitrary decision.

Another, more promising, explored proposal was by Jaewoo Lee in his paper "How much is enough: Choosing ϵ for Differential Privacy" [24]. This proposal was considered for this pipeline and an attempt to adapt it to this work's pipeline was made, albeit unsuccessfully.

Jaewoo Lee proposes a formula, derived from the differential privacy definition, that allows the calculation of a privacy budget, ϵ, that bounds the probability of an adversary learning from the data. Essentially, this approach works as an Upper Bound Method. It requires you to define two main variables: the attacker's prior belief and the maximum allowed posterior belief (the maximum certainty we are willing to let an adversary have after observing the synthetic data about whether or not an individual participated). For instance, if a security team states, "We cannot let attackers be more than X% sure that a specific individual is in this database," you can use that strict mathematical constraint as input for Lee's formula and in this way extract the exact ϵ needed to guarantee it [24].

This approach seemed to be a perfect method for making an informed selection of ϵ, however, it was ultimately found to not be applicable to this work's pipeline due to a combination of practical data limitations and mismatches.

Practically, to make this formula work, you need to accurately quantify that initial prior knowledge. In the context of HPC workload traces, defining a realistic prior probability for an attacker, for instance, trying to pinpoint a specific rare backfilled job is practically impossible. An estimation of worst-case baseline probabilities was also attempted, but quickly became apparent that guessing the prior incorrectly meant that the formula would propose an ϵ that either completely destroyed utility or offered zero real protection by being far too high. It relies on a theoretical assumption of attacker knowledge that we simply cannot confidently measure in our scenario.

Furthermore, even if a perfect prior estimation could be made, Lee's method hits a fundamental mathematical constraint: it is explicitly designed for single simple query functions with no composition working under the stricter ϵ-DP definition. Because of this, Lee's method proved to be incompatible with this project's proposed pipeline, involving both multiple complex compositions and different definitions of differential privacy.

Given that both Hsu's and Lee's frameworks either required devising non-existent monetary budgets or were incompatible, the data-driven elbow method

approach was ultimately the one used for the informed epsilon selection of this work.

This was done by plotting the trade-off between Differential Privacy's two competing goals: privacy (the ϵ budget) versus utility. For the utility, two different measures measures were used for better robustness: the average distances (between the original and synthetic distributions) and the correlation error (of the original and synthetic dataset). By plotting these metrics against ϵ, we now look for the "elbow" of the curve, the exact point where the trade-off shifts, and decreasing privacy (increasing ϵ) no longer brings any significant or meaningful improvements to the utility. The epsilons plotted in this way were 0.1, 1, 3, 5, 10 and 100 whereas an additional evaluation was done with epsilon 10000. This evaluation can be seen in the validation section 4.2.

5 Final Pipeline: PT-MST

The following diagram is an overview of the architecture of the system. Subsequent sections do a deeper analysis of the implementation of the main modules.

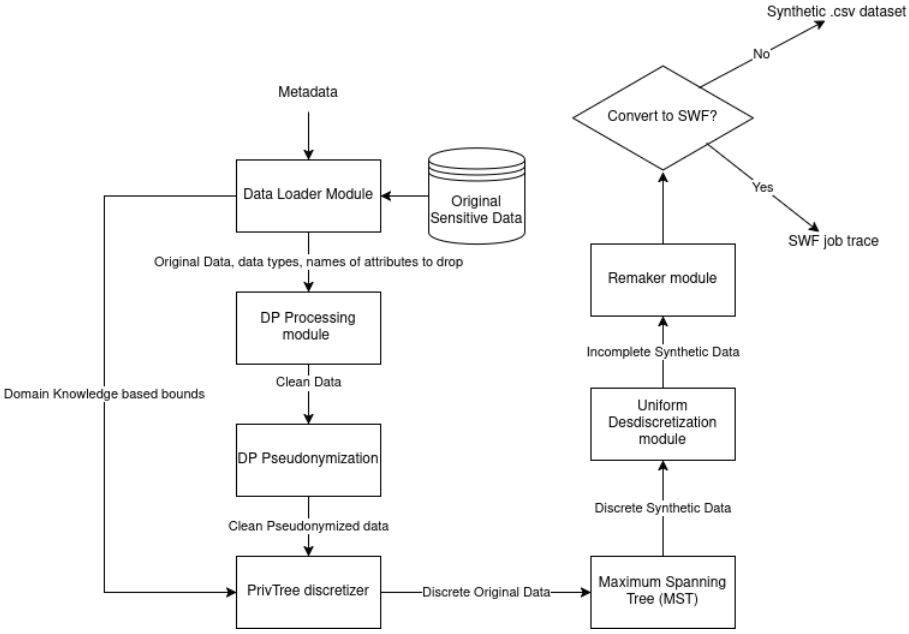


Figure 8: PT-MST Diagram

Figure 8 illustrates the architecture of the final synthesis pipeline, named as PT-MST. The complete system is composed of seven main modules that handle the cycle of data loading, preprocessing, private discretization, synthesis, and reformatting:

1. **Data Loader Module:** This module ingests structural metadata specified in JSON format. Using this metadata, it locates the target dataset, loads it into memory supporting multiple file encodings (`.swf`, `.parquet`, or `.csv`), and parses schema parameters. It isolates the variables targeted for synthesis, identifies columns designated to be dropped, validates native data types, and incorporates domain knowledge domain boundaries. An example of the metadata that this pipeline ingests is provided in Appendix A, specifically Figure 33.

2. **Preprocessing Module:** This module enforces data type consistency across all attributes, standardizes date-time text fields into numerical Unix timestamps (seconds since January 1970), and cleans structural anomalies. Rows with extreme missing rates are securely discarded using a differentially private schema pruning algorithm, while missing values are handled via safe default mapping. Additionally, highly sensitive identifiers (e.g., `User_ID` and `Group_ID`) are pseudonymized using a differentially private domain discovery mechanism based on Korolova thresholding [23], which retains only high-frequency categories to prevent privacy leakage from rare identifiers.

3. **PrivTree Discretizer Module:** This module receives the sanitized data along with domain bounds. It executes a tree-based hierarchical space decomposition algorithm modified for single-variable discretization.

4. **MST Module:** This module initializes the generative model using the discretized, pseudonymized data. The underlying Maximum Spanning Tree synthesizer models strong attribute correlations and outputs a fully synthetic, discrete dataset matching the safe, privately estimated size of the original data (if no length was provided in the metadata).

5. **Uniform Dediscretization Module:** This module reverts the discrete (binned) synthetic values back into continuous numerical distributions. It samples real-valued values uniformly within the lower and upper boundaries discovered by the discretization tree for each specific bin.

6. **Re-maker Module:** Standard Workload Format (SWF) job logs structurally utilize a value of -1 to represent missing, unrecorded, or invalid attributes. This module restores columns that were omitted during preprocessing due to sparsity or explicit user directives. It populates these reattached attributes entirely with the structural indicator -1, ensuring backward compatibility with standardized scheduling formats.

7. **CSV to SWF Converter Module:** This final component transforms the continuous synthetic dataset from tabular comma-separated values into a standardized `.swf` workload trace. This allows the synthesized output to be deployed directly into infrastructure simulation engines, such as the SLURM scheduler.

5.1 Epsilon and Delta allocation

In this pipeline multiple Differential Privacy techniques are being used. In order to guarantee a proper composition of the privacy budget, the pipeline divides the global ϵ_{total} across the five distinct operations that use it, ensuring that the cumulative sum of the individual privacy components strictly satisfies sequential composition:

- **Global Noisy Count (ϵ_{count}):** Allocated 2% of the total privacy budget ϵ to compute the safe dataset size.
- **Schema Pruning ($\epsilon_{\text{schema\_prune}}$):** Allocated 3% of the total privacy budget ϵ, uniformly divided across all columns such that: $\epsilon_{\text{per\_prune}} = \epsilon_{\text{schema\_prune}}/N_{\text{total\_initial\_cols}}$.
- **Pseudonymization ($\epsilon_{\text{pseudonymize}}$):** Allocated 5% of the total privacy budget ϵ, split evenly among categorical identifiers as $\epsilon_{\text{per\_pseudo}} = \epsilon_{\text{pseudonymize}}/N_{\text{initial\_identicat\_cols}}$.
- **PrivTree Discretization ($\epsilon_{\text{discretization}}$):** Allocated 20% of the total privacy budget ϵ, distributed across numeric features where: $\epsilon_{\text{per\_col\_tree}} = \epsilon_{\text{discretization}}/N_{\text{initial\_num\_cols}}$.
- **MST Synthesis ($\epsilon_{\text{generation}}$):** MST is allocated the remaining 70% of the global privacy budget ϵ.

Besides the ϵ, the probability of failure, necessary for Approximate DP, and used in the pipeline by the DP Pseudonymization and MST, must also be allocated since both MST and the DP pseudonymization used said variable. This is done by splitting it equally, 0.5% of it dedicated for the preprocessing stage (the pseudonymization) and 0.5% for MST.

$$\delta_{\text{preprop}} = 0.5 \cdot \delta_{\text{total}}, \quad \delta_{\text{mst}} = 0.5 \cdot \delta_{\text{total}} \tag{17}$$

5.2 Preprocessing Module and Pseudonymization

Before executing the discretizing and synthesizing algorithms, the raw dataset must be formatted, its data types properly casted, and sanitized to ensure the proper down-steam generation of synthetic data. The used CEA-Curie logs, as well as ANL-Intrepid's, missing values, sparse features, and identifiers. The preprocessing module addresses these issues in a general manner, that is, preprocessing is not tailored to one specific dataset.

The pipeline first computes the dataset size in a private manner. This is done because releasing the exact row count of a database can expose the presence or absence of an individual, thus, a differentially private count is implemented by injecting noise with the Laplace mechanism:

```
        n_raw = len(df)
        scale_count = 1.0 / eps_count
        noisy_count = max(1.0, n_raw + np.random.laplace(0,
        ↪  scale_count))
```

Figure 9: Implemented calculation of the Noisy Count with differential privacy using the Laplace mechanism.

Secondly, categorical attributes are parsed as strings to avoid any possible data type misclassifications incurred when reading categorical attributes that use numbers as categories. After this, numerical columns are strictly cast to floats, since noise injection causes many numeric values to be decimal, and date-time configurations are transformed into standard UNIX style integer representations.

Missing data values are handled through a data-independent mapping strategy rather than standard imputation. Traditional approaches, such as mean, median, or K-Nearest Neighbors imputation or the dropping of missing values, are incompatible with differential privacy since they rely on making decisions based on the sensitive data. As a result, in order to preserve DP, we instead do a fixed mapping: categorical missing values are assigned a "Missing" label, while numerical missing values are mapped to 0.0. This ensures that no privacy is leaked during the missing value handling.

Following this, and in order to prevent overly sparse or empty attributes from needlessly consuming the privacy budget, a differentially private schema pruning algorithm is implemented. For each attribute we calculate the proportion of missing values, which is then perturbed with a Laplacian Mechanism in order to guarantee DP. Those attribute whose missing value proportion exceed a coded-in threshold (set at 50%), are removed from the dataset:

```
def dp_should_drop(series, eps_prune, noisy_n, fraction=0.5):
    missing = series.isna().sum() + (series == 'Missing').sum() +
    ↪  (series == 0.0).sum()
    noisy_missing = missing + np.random.laplace(0, 1.0 /
    ↪  eps_prune)
    return noisy_missing > (noisy_n * fraction)

for c in numerical:
    if dp_should_drop(df[c], eps_per_prune, noisy_count,
    ↪  fraction=0.5):
        to_remove.append(c)
    else:
        new_numerical.append(c)
```

Figure 10: Implemented algorithm to drop empty or sparse attributes in a differentially private way.

Following this initial missing value and data type processing, the data undergoes a private pseudonymization to replace unique user-identifying attributes with random integer numbers.

Rare, low-frequency categorical variables pose significant re-identification risks. The pipeline mitigates this by applying a differentially private thresholding mechanism based on the framework by Korolova et al. [23]. This approach calculates a frequency threshold to ensure that rare outliers are suppressed:

```
def calculate_korolova_threshold(eps_col, delta):
    """
    Calculates the exact theoretical threshold K from Korolova et
    ↪  al.
    for a user contributing at most 1 record (d=1).
    """
    b = 1.0 / eps_col
    K = 1.0 - (b * math.log(2.0 * delta))
    return K
```

Figure 11: Calculation of the Korolova based DP threshold.

Given that each job entry represents an independent entity contributing at most $d = 1$ record to any given categorical distribution, the operational threshold K reduces to:

$$K = 1 - \frac{1}{\epsilon} \ln(2\delta). \tag{18}$$

By utilizing 18, the pseudonymization pipeline computes the noisy frequencies of all distinct values within the targeted attribute. Categories whose noisy counts fall below K are stripped and remapped to a generic baseline of 0 (representing an aggregate "OTHER" class). High-frequency keys that exceed K are securely mapped to randomized IDs, suppressing unique, easily-identifiable outliers:

```
def dp_pseudonymize(df, col, eps_col, delta_col):
    val_counts = df[col].value_counts()
    vals = val_counts.index.values
    counts = val_counts.values

    noisy_counts = counts + np.random.laplace(0, 1.0 / eps_col,
    ↪  size=len(counts))

    threshold = calculate_korolova_threshold(eps_col, delta_col)
    valid_mask = noisy_counts >= threshold
    valid_vals = vals[valid_mask]

    num_unique = len(valid_vals)
    max_range = max(1000, num_unique * 2)
    id_pool = np.arange(1, max_range)
```

```
    np.random.shuffle(id_pool)

    mapping = dict(zip(valid_vals, id_pool[:num_unique]))
    df[col] = df[col].map(mapping).fillna(0).astype(int)
    return df

for var in identicat:
    if var in df.columns:
        df = dp_pseudonymize(df, var, eps_per_pseudo,
        ↪  delta_per_pseudo)
```

Figure 12: Implementation of the DP pseudonymization.

5.3 PrivTree Discretization Nuances

This section details specific modifications of the PrivTree algorithm implemented in the pipeline. The theoretical explanation of the algorithm can be found in section 3.4.1.

> **Note on Notation Alignment:** It is critical to distinguish between the distinct privacy frameworks executing within this preprocessing pipeline. While the pseudonymization module utilizes approximate (ϵ, δ)-DP founded on Korolova thresholding, the PrivTree structural exploration operates strictly under pure ϵ-DP. The parameter `delta` (δ) instantiated within our PrivTree source code does not denote Approximate DP's breach of privacy probability; rather, it represents a data-independent, deterministic depth attenuation step-penalty used to scale down threshold expectations as the tree deepens.

Algorithmic Additions and Modifications

While the theoretical formulation of PrivTree permits an infinite tree depth, large privacy budgets ϵ compress the noise scale λ and rapidly diminish the depth attenuation penalty δ. Under these conditions, the splitting condition 13 is easily satisfied even within highly sparse or uniform intervals. This can cause an exponential expansion of micro bins, a phenomenon we deemed as bin explosion. This severe fragmentation can overfit outliers, inflate cross-bin variance, and critically hamper downstream modules such as the Maximum Spanning Trees DP Data Generator. Specifically, the explosion in the number of categories increases the total size of each measured marginal (which since marginals are akin to tables it can be thought of as increasing its amount of cells) forcing the fixed privacy budget to be spread across a larger parameter space (over more cells) causing the data utility to degrade.

To mitigate this behavior this implementation introduces four architectural and numerical modifications to the base algorithm:

1. **Binary Tree Discretization (1D Specialization):** Unlike the original multi-dimensional spatial formulation designed for coordinate indexing, each numerical feature in our pipeline is isolated and discretized independently. This maps the problem to a one-dimensional space where continuous intervals are recursively binary split at their midpoints:

```
mid = (node.low + node.high) / 2.0
node.left = PrivTreeNode(node.low, mid, node.depth + 1)
node.right = PrivTreeNode(mid, node.high, node.depth + 1)
```

Figure 13: Implementation of the binary tree splitting.

2. **Dynamic Max Depth Ceiling via the Square Root Rule:** To avoid the "bin explosion" without introducing manual, data-dependent hyperparameters, we implement a dynamic ceiling based on Pearson's Square Root Rule [42]. The bin resolution for a dataset of size n is bounded by $\sqrt{n}$. To preserve differential privacy, this ceiling is calculated using the globally computed noisy row count $\tilde{n}$, which is perturbed via a Laplace mechanism using a percentage of the privacy budget, ϵ_{count}:

$$\text{max\_bins} = \sqrt{\tilde{n}} \tag{19}$$

Since the underlying layout is a strict binary tree, the maximum depth $d_{\max}$ required to accommodate such a bin resolution is formalized as:

$$d_{\max} = \lceil \log_2(\text{max\_bins}) \rceil = \left\lceil \log_2 \left(\sqrt{\tilde{n}} \right) \right\rceil \tag{20}$$

By the post-processing theorem of Differential Privacy 1, since $\tilde{n}$ is a valid differentially private primitive, any deterministic transformation applied to it, such as the computation of $d_{\max}$, doesn't incur any privacy leakage. This dynamic restriction is enforced within the core queue loop:

```
max_bins = np.sqrt(noisy_n)
max_depth = math.ceil(np.ceil(np.log2(max_bins)))

# Inside the queue loop
if noisy_b > theta and node.depth <= max_depth:
    # ... Split Node ...
else:
    leaves.append(node)
```

Figure 14: Dynamic Maximum Tree Depth Implementation.

This constraint allows node splitting up to and including depth $d_{\max}$ (providing an absolute leaf depth boundary of $d_{\max}+1$), successfully restricting excessive bin generation while allowing PrivTree full autonomy to determine optimal cell layouts within those safety boundaries.

3. **Logarithmic-Space Scaling:** Workload logs frequently exhibit skewed, long-tailed numeric distributions. Uniform binary splitting applied directly in linear space often fails to capture granular variations across lower boundaries, clustering them into single macro-bins. To solve this, our implementation encloses the tree execution in a logarithmic transformation. To elaborate, we apply a log transform to the numerical values, with a shift computed from the domain knowledge lower bound to safely handle possible negative domains or 0 values.

$$x_{\log} = \log_{10}(x + \text{shift}). \tag{21}$$

After this PrivTree processes the log-scaled intervals to discover cut-points that are more spaced across orders of magnitude.

4. **Numerical Stability and Epsilon-Separation:** Once the tree's construction is finalized, the discovered log-space edges and midpoints are mapped back to linear space via exponential inversion:

$$x_{\text{linear}} = 10^{x_{\log}} - \text{shift}. \tag{22}$$

However, this can introduce floating-point precision errors or duplicate edges at dense domain boundaries. To ensure numerical stability and guarantee valid non-zero widths for subsequent classification via `pd.cut`, the implementation runs a monotonic accumulation check and injects deterministic noise to separate overlapping boundaries:

```
edges = np.maximum.accumulate(edges)
for i in range(1, len(edges)):
    if edges[i] <= edges[i - 1]:
        edges[i] = edges[i - 1] + 1e-9

# Lock outer edges to original explicit bounds
edges[0] = low
edges[-1] = high
```

Figure 15: Implementation of the edges and its small noise injection.

Finally, the outer boundary bounds are forcefully locked back to the explicit wide bounds to eliminate precision rounding errors.

PrivTree's internal budget allocation

From an architectural perspective, a privacy budget ϵ_{tree} is allocated to each attribute to be discretized. Our preprocessing pipeline allocates a 20% fraction of ϵ_{total} to be used by PrivTree for the discretization of all attributes. This privacy budget is subsequently equally divided among all validated numerical attributes (m), establishing an independent, uniform per-column budget of $\epsilon_{\text{tree}} = \frac{0.20 \cdot \epsilon_{\text{total}}}{m}$.

5.4 Maximum Spanning Tree (MST) Synthesis

This section details specific structural and operational constraints of our MST's implementation. Because MST constructs discrete marginal contingency Table, all continuous training features must be previously discretized by the PrivTree module, and unique identifiers must be removed before its execution, otherwise, since identifiers have a massive amount of unique categories, which causes an exponential expansion of the marginal tables, consuming most of MST's privacy budget.

Additionally, as a safety measure, all features are cast to string representations (since in this point in the pipeline are all discrete). This sanitized data is now provided to `MSTSynthesizer` from OpenDP's SmartNoise library [35].

The synthesizer is thus initialized and fitted purely on the categorical representations. Its implementation is simple thanks to the aforementioned libraries:

```
synth = MSTSynthesizer(epsilon=user_eps, delta=delta)

synth.fit(
    df_train,
    categorical_columns=cat_cols,
    continuous_columns=num_cols,
)
```

Figure 16: Implementation of MST.

Generating the synthetic dataset is a straightforward process thanks to SmartNoise's implementation. The number of output records is determined by a simple conditional: if the metadata explicitly specifies a target sample size `length_t_gen`, that value is used. Otherwise, the pipeline uses the previously computed noisy count `noisy_counts`. The corresponding logic is shown in Code Snippet 17.

```
if length_t_gen:
    synthetic_df =
    ↪  generate_synthetic_data_with_mst(epsilon_generation,
    ↪  df_binned, length_t_gen, delta_f_mst, identifiers)
else:
    synthetic_df =
    ↪  generate_synthetic_data_with_mst(epsilon_generation,
    ↪  df_binned, noisy_counts, delta_f_mst, identifiers)
```

Figure 17: Logic for selecting the number of synthetic records to generate.

Internally, the generation function relies on Private-PGM's sampling method, which directly invokes `synth.sample()` to draw the designated number of records as illustrated in Code Snippet 18.

```
synthetic_df = synth.sample(int(noisy_counts))
```

Figure 18: Internal sampling routine used by SmartNoise/Private-PGM.

The result at this stage is a discrete synthetic dataset. We now need to undo the discretization in a DP manner.

5.5 Uniform De-discretization module

This function, located in script `mst_synth_func` I.4, iterates through the continuous attributes, maps the synthesized bin indices back to their linear space boundaries, and draws a uniform random distribution between the lower and upper bounds of each respective bin interval:

$$x_{\text{continuous}} \sim \mathcal{U}(\text{bin}_{\text{low}},\ \text{bin}_{\text{high}}) \tag{23}$$

Because this process depends entirely on the privatized interval boundaries generated by PrivTree, it acts as a post-processing step that adds zero additional privacy leakage as by the post-processing theorem.

5.6 Remaker module (The Inversion Mechanism)

The pipeline concludes with a structural reconstruction function named "remake", which can be found in the script `generalized_main` I.1.

This module is designed to prepare the synthesized data frame for downstream High-Performance Computing (HPC) validation and Standard Workload Format (SWF) conversion. Columns classified as structurally unsafe or designated to be dropped (to_remove) are scrubbed and backfilled with a safe default sentinel value (-1).

Simultaneously, the unique identifier categories dropped prior to training are synthetic-reconstructed using a pseudo-random permutation of a sequential array matching the size of the generated dataframe:

```
def remake(df, to_remove, identifiers):
    for col in to_remove:
        df[col] = pd.Series(-1, index=df.index, dtype='Int64')
    for col in identifiers:
        x = np.arange(1, len(df) + 1)
        df[col] = np.random.permutation(x)
    return df
```

Figure 19: Remaker module.

This permutation step restores necessary structural identifier properties, thus in this way ensuring every synthesized job maintains a unique, non-repeating

identifier sequence, without exposing any historical indexing identities or compromising differential privacy guarantees.

6 Results Evaluation

In this section, we present the empirical results of our final synthetic data generation pipeline as well as our epsilon selection.

> **Data Preprocessing Note:** Crucially, before conducting the distances related tests as well as the ML utility tests a Min-Max normalization was applied to both the original and synthetic datasets. This ensures that variables with vastly different natural scales (such as `Submit_Time` versus `Allocated_Processors`) contribute proportionally to our utility measurements and distance metrics, preventing variables with massive absolute bounds from disproportionately warping the evaluation.

6.1 Empirical Epsilon Selection via the Elbow Method

Choosing a fitting privacy budget ϵ is rarely straightforward and can often be quite arbitrary [9, 24]. To ground our selection in empirical evidence, we utilized the elbow method (as stated in section 4.10), plotting a range of ϵ values against corresponding utility measurements. Specifically, we analyzed two separate axes of error metrics:

1. Average Wasserstein and Total Variation Distance (TVD) deviations.
2. Average Spearman (non-linear) and Pearson (linear) correlation errors.

These two metrics were chosen because they are methods that directly quantify the closeness of the synthesized and real data. Distances and correlations are preferred over the bound difference since it only captures numerical variables, while ML Utility (as seen in Table 6) remains largely stable with negligible changes. As for the 3-way marginals and HOC, they are applied on the undiscretized synthetic data rather than the the final synthetic dataset, making them less trustworthy.

To clarify, we also included two exceptionally high privacy budgets ($\epsilon = 100$ and $\epsilon = 10000$) during the evaluation phase. These were at no point considered viable candidates since too big a privacy budget can be risky, specially for MST [14]. Rather, they served as an experimental baseline check to verify that the Differential Privacy (DP) framework behaved as theoretically expected, gradually reducing noise injection as the budget increases.

As for the elbow plot, we plotted the aforementioned utility metrics for the $\epsilon = 0.1, 1, 3, 5, 10$ and 100. These values were selected to span an spectrum groing from strict privacy ($\epsilon = 0.1$) to a loose privacy ($\epsilon = 100$), with intermediate values covering the typically used ϵ as observed in the literature for DP synthetic data generation [28].

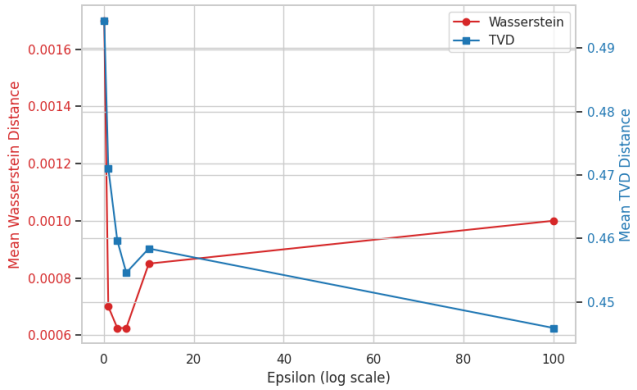


Figure 20: Two-axis elbow plot showcasing Normalized Wasserstein and TVD distances across evaluated ϵ values.

As observed in Figure 20, the inflection point (or "elbow") of the curve is between $\epsilon = 5$ and $\epsilon = 10$. While the TVD distances strictly diminish with larger epsilon values, the Wasserstein distance appears to stagnate or slightly increase. This behavior, however, is a known issue driven by the number of discretization bins increasing alongside higher ϵ due to our implementation of PrivTree and the pipeline's natural stochasticity.

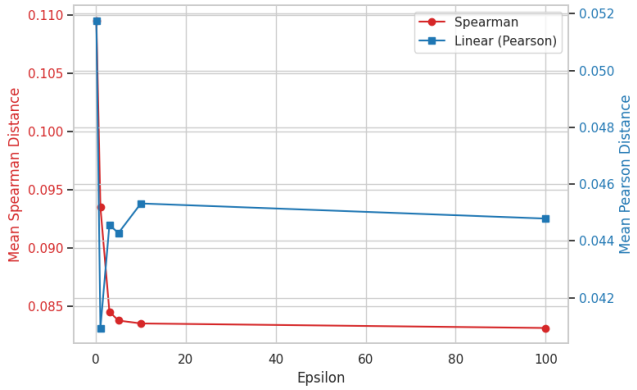


Figure 21: Spearman and Pearson Correlation Error vs. Epsilon.

In Figure 21, a similar elbow can be found between $\epsilon = 5$ and $\epsilon = 10$. Non-linear correlations (Spearman) visibly improve as more privacy budget is allocated, whereas linear correlations (Pearson) stagnates after initially demonstrating better preservation under tighter privacy restrictions (since smaller epsilons imply, in this pipeline, less bins which helps maintain linear relations at the expense of the non-linear ones).

As a result of these elbow plots, it can be concluded that the most promising

trade-offs offering strong data utility alongside robust privacy guarantees are $\epsilon = 5$ and $\epsilon = 10$.

Additional evaluation results with the main, important, metrics of alternative epsilon configurations are provided for in Appendix H were we can find the results for epsilon 0.1 H.1, for epsilon 1 H.2, for epsilon 3 H.3, for epsilon 100 H.4 and for epsilon 10000 H.5.

In addition, the elbow_method script can also be found in Appendix I.6.

6.2 Results analysis

The main evaluation results are summarized in Table 6.

Metric (Lower is Better, unless specified)	**Epsilon = 5.0**	**Epsilon = 10.0**
Avg. Bound Difference	346867.582455	237516.743931
Mean Wasserstein Distance	0.000600	0.000875
Mean Total Variation Distance (TVD)	0.006100	0.005475
Random Forest (RF) Loss	0.025509	0.025861
Linear Regression (LR) Loss	-0.000016	0.000128
Discriminator Test (Closer to 0.5 is better)	0.646116	0.642423
Mean Synth to Train DCR	0.004600	0.004299
Min Synth to Train DCR	1.6459e-07	1.1921e-07
Mean 1-DCR	6847.872143	6799.597659
Mean 5-DCR	10126.858115	10049.513175
NNDR	0.787758	0.787114
Exact Clones (Count)	0	0
Spearman Correlation Error	0.084411	0.082880
Linear Correlation Error	0.044578	0.045031
Maximum Mean Discrepancy (MMD)	0.020814	0.020764
3-Way Marginal Score	0.671411	0.690834
Higher-Order Correlation (HOC) Score	0.000715	0.000866

Table 6: Main evaluation Metrics for $\epsilon = 5, 10$.

From the results in Table 6 we can observe a great decrease in the average boundary deviations, showing how PrivTree improves the discretization as less noise is injected with higher ϵ values.

However, the mean Wasserstein distance for $\epsilon = 10$ increases, which would indicate a worse synthesization of the numerical variables. This can be explained by the interaction between PrivTree and MST. Larger ϵ values allow PrivTree to produce more bins, which in turn increases the size of the categorical space MST must model. This forces MST to spread its privacy budget across more cells, injecting additional noise which degrades the utility for certain variables. In practice, this means that increasing ϵ results in smaller improvements in utility while privacy losses remain relatively low.

Nevertheless, the increase in the Wasserstein distance is balanced by the TVD distance and MMD decreasing, which both indicate an improvement in the quality of the synthetic data. In addition, neither RF loss nor LR loss seem to change much, only slightly worsening, which can be due to a wide array of

different things including, for instance, that the more noisy values of a lower ϵ keep the models from overfitting, which means that the model memorization of the training data.

In addition, we can observe that the Discriminator Test's accuracy is slightly lower for epsilon 10 indicating that the synthetic data generated from that pipeline is less distinguishable from the real data which indicates both that a linkage attack would be harder to perform and that said data is more similar to the real data.

As for the privacy, we can see a clear trend perfectly corresponding with what we expect from an increase in epsilon: they worsen. The mean DCR and Min DCR decreased, as did the mean 1 and 5 DCRs and neither have clones. The NNDR grew smaller as well. Nevertheless, this worsening of the privacy metrics is very small, showing that both ϵ have similar privacy.

Finally we can observe that linear correlations had a negligible worsening while spearman correlations improved, indicating that the relationships of our data were better preserved. As for the 3-way marginal score and HOC scores, those metrics are not very fit to be used because of a simple matter: they measure the discretized dataset's preservation of marginals and conjunctions, and the bigger the epsilon the more discrete categories (the more marginals and conjunctions) making their values higher, even if the synthesization was better.

So, in conclusion, from this Table 6 we would conclude that epsilon 10 slightly improves in the overall synthesization quality as shown by TVD and MMD distances and the Spearman correlation with a slight worsening of the numerical attributes and linear correlations, while privacy worsens slightly but remains roughly equal.

As a consequence of these mixed results, we now look closer a to the specific attributes metric themselves. Table 7 highlights boundary deviations across core continuous features.

Attribute Group / Column	**Original Bounds**	$\epsilon = 5.0$ **Synthesized**	$\epsilon = 10.0$ **Synthesized**
`Submit_Time` (Min)	31,656,837.00	31,547,877.17	31,547,899.46
`Submit_Time` (Max)	53,580,159.00	53,862,077.24	53,862,066.23
`Wait_Time` (Max)	7,610,035.00	9,956,690.18	9,070,994.00
`Run_Time` (Max)	124,615.00	161,878.17	172,763.26
`Allocated_Processors`(Max)	79,808.00	79,951.24	79,988.92
Average Bound Difference	–	**346,867.58**	**237,516.74**

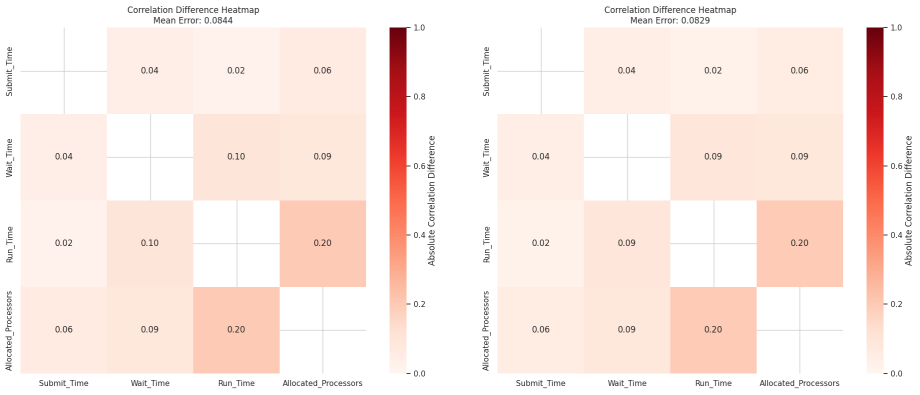
Table 7: Boundary error breakdown for $\epsilon = 5.0$ and $\epsilon = 10.0$.

These boundaries indicate clearly that $\epsilon = 10$ achieves tighter bounds, which means a better synthesization, reducing extreme, false, tails created due to wide domain knowledge bounds (specifically for `Wait_Time` and `Run_Time`). Table 8 showcases the Wasserstein and TVD distances values obtained by each attribute at the different ϵ values.

Attribute Name	Metric Type	$\epsilon = 5.0$ Distance	$\epsilon = 10.0$ Distance
`Submit_Time`	Wasserstein	0.0020	0.0020
`Wait_Time`	Wasserstein	0.0001	0.0001
`Run_Time`	Wasserstein	0.0002	0.0012
`Allocated_Processors`	Wasserstein	0.0001	0.0002
`Status`	TVD	0.0000	0.0004
`User_ID`	TVD	0.0137	0.0127
`Group_ID`	TVD	0.0096	0.0082
`Partition_Number`	TVD	0.0011	0.0006
Mean Wasserstein	Universal Avg.	**0.0006**	**0.0009**
Mean TVD	Universal Avg.	**0.0061**	**0.0055**

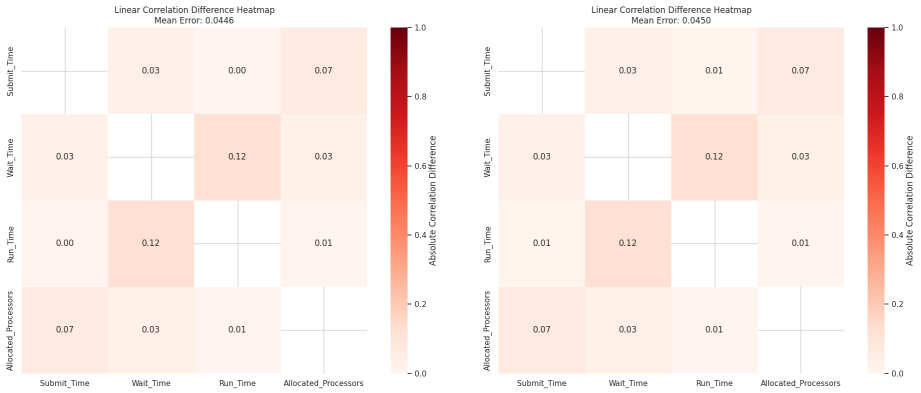
Table 8: Features distance comparisons.

We can observe in Table 8 that both synthesizations achieve very low distances indicating good individual variable synthesization quality. Nevertheless, it can also be seen that, among the numerical variables, the Wasserstein distances increase for Run_Time indicating a worse synthesization of said variable given this epsilon increase. However, as we'll see in Figure 24, that is barely noticeable. Furthermore, all TVD distance decrease, or stay the same, in other words, only one attribute (two counting the slight increase of Status) worsen indicating an overall improvement in the individual variables synthesization.



(a) $\epsilon = 5$ *Spearman error heatmap.* *(b)* $\epsilon = 10$ *Spearman error heatmap.*

Figure 22: Spearman correlation error heatmaps.

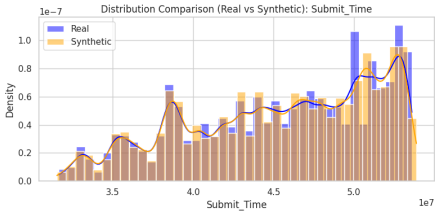


(a) $\epsilon = 5$ *Pearson Error Heatmap.*

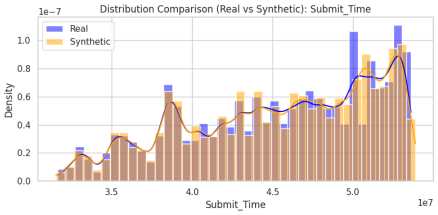
(b) $\epsilon = 10$ *Pearson Error Heatmap.*

Figure 23: Pearson correlation error heatmaps.

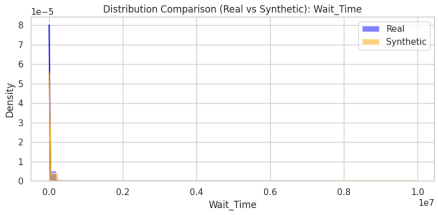
Observing Figures 22 and 23 we can observe that seemingly the Spearman and Pearson errors were the same. Looking more closely, however, we can see that the only difference in both pairs of heatmaps comes from Runtime. In the Spearman heatmaps its correlation error decreases by 0.1 and in the Pearson heatmaps its correlation error with submit time increases by 0.1. Both the increase and decrease can be deemed negligible. What is more interesting to analyze however is were the correlation error is bigger, and it is in the Spearman case in the correlations between Run_Time and the Allocated_Processors and in Pearson's case the main error comes from the correlations between Run_Time and Wait_Time. Clearly, Run_Time has an important role in the CEA-Curie dataset. In addition, we can also visually observe the similarity of the real and synthesized attributes through the plots in Figure 24.



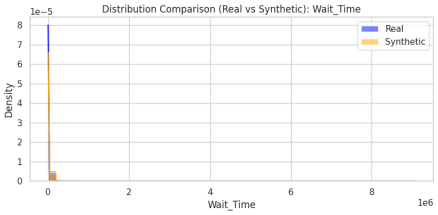
(a) $\epsilon = 5$ ***Submit_Time***



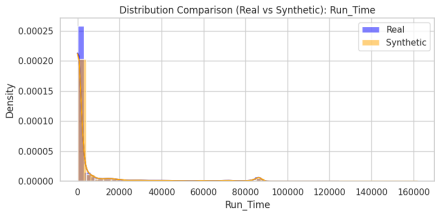
(b) $\epsilon = 10$ ***Submit_Time***



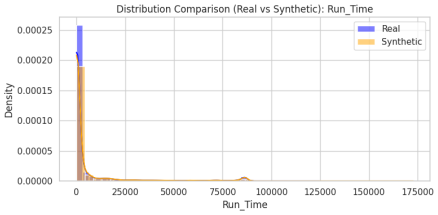
(c) $\epsilon = 5$ ***Wait_Time***



(d) $\epsilon = 10$ ***Wait_Time***



(e) $\epsilon = 5$ ***Run_Time***



(f) $\epsilon = 10$ ***Run_Time***

Figure 24: Visual univariate distribution comparisons for continuous variables across $\epsilon = 5$ *and* $\epsilon = 10$.

There were more variables but they were perfectly synthesized in both cases and remained the same, they can be found in the Appendix H.8 and H.9 for $\epsilon = 5$ and $\epsilon = 10$ respectively. As it can be observed in Figure 24, both synthetic results match closely the real original distributions with the more pronounced frequency spikes being, however, shorter and with a certain smoothing effect due to the introduced noise. In addition, we can observe what we saw in the distances table: Submit_Time is just as well synthesized in both epsilon instances, wait_time slightly improves and Run_Time slightly worsens.

After analyzing these results only, we would select $\epsilon = 5$ over $\epsilon = 10$ since the utility scores achieved were almost identical and epsilon 5 has strictly better privacy guarantees. However, there are deeper discrepancies which emerge once these traces are used to execute the SLURM scheduling simulations, thus showing the need of measuring utility via a real-world application instead of merely statistics.

6.3 SLURM Simulation Results

To evaluate the capacity of this synthetic data of recreating realistic HPC traces, we executed a series of simulations of how a super-computing cluster would behave given these synthesized jobs by using the BSC Slurm simulator [20]. Additionally, this evaluation allows us to go beyond purely statistical, mainly distance-based metrics to measure how well the synthetic data preserves the correlations needed to properly recreate a supercomputer. With this in mind, in this section we present a direct three-way comparison between the simulation driven by the original data and the simulations driven by our synthetic datasets provided by our pipeline under $\epsilon = 5$ and $\epsilon = 10$.

6.3.1 Comparative Analysis of Scheduling Metrics

The scheduling behavior of each dataset within the SLURM environment is summarized in Table 9. In this Table we observe job counts, allocated CPUS, Wait Time, and priority for both Backfilled and Non-Backfilled jobs.

Simulation Metric	**Original Baseline**	**Final Pipeline** ($\epsilon = 5$)	**Final Pipeline** ($\epsilon = 10$)
Total Simulated Jobs	9,115	8,915	8,877
Backfilled (BF) Job Count	880	4,966	3,215
Mean Allocated CPUs (BF)	7,296.53	1,313.78	1,528.49
Mean Priority (BF)	10,853.23	16,768.77	9,978.61
Mean Priority (Non-BF)	12,760.64	28,166.81	25,567.06
Mean Wait Time (Seconds)	213.64	108.54	177.63

Table 9: SLURM Simulation Performance Profiles: Baseline vs. Synthetic Workloads.

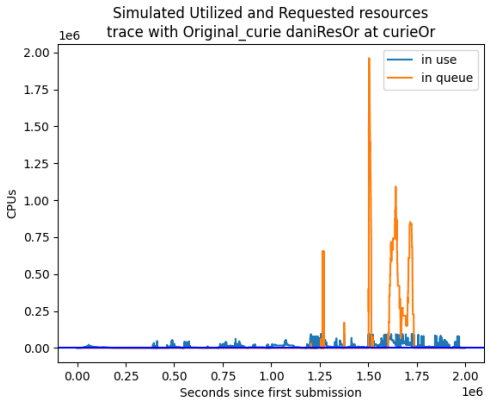
An analysis of Table 9 immediately showcases the better synthesis of the $\epsilon = 10$ privacy budget over $\epsilon = 5$.

Under $\epsilon = 5$, the stronger privacy guarantees results in more noise that increase distortions in the scheduling trace. It artificially halves the global mean wait time to 108.54 seconds and greatly increases the priority assigned to both backfilled and non-backfilled jobs.

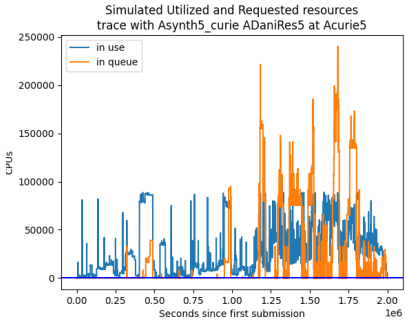
On the other hand, $\epsilon = 10$ demonstrates more structural fidelity, though it still is far from the original baseline. It establishes a global mean wait time of 177.63 seconds, closer to the 213.64-second ground truth compared with $\epsilon = 5$. Furthermore, its mean backfilled priority (9,978.61) is also closer to the baseline threshold of 10,853.23. However, both of the synthetic data examples struggle to correctly mimic the average core requests of the baseline's backfilled jobs (7,296 CPUs) as well as the actual number of backfilled jobs. This is because DP removes the one-off large requests that are easily identifiable. Nonetheless, it can be concluded that $\epsilon = 10$ achieves results closer to the original baseline than $\epsilon = 5$.

6.3.2 Visual Trace Analysis and Queue Smoothing Effects

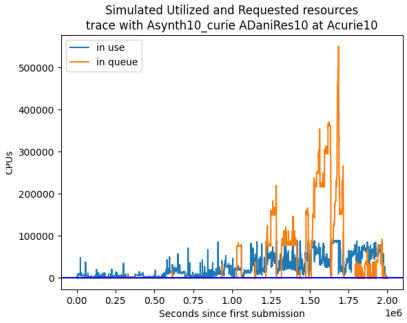
The nuances of these workloads are better shaped via a visual comparison of the simulated resource queue traces and the pie-plot of the backfilled proportion of jobs, shown in Figures 25 and 26 respectively.



(a) Simulated utilized and requested resources obtained with the real data.

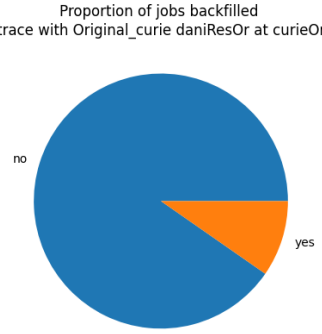


(b) Simulated utilized and requested resources obtained by the synthetic trace (using $\epsilon = 5.0$).

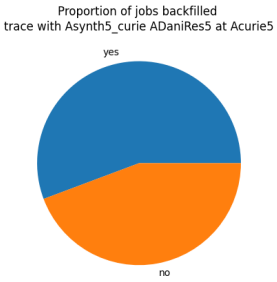


(c) Simulated utilized and requested resources obtained by the synthetic trace (using $\epsilon = 10.0$).

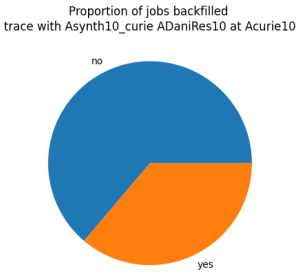
Figure 25: SLURM simulated utilization and trace request tracking of the Real and synthetic data (three way comparison between the original trace and the traces obtained using $\epsilon = 5, 10$).



(a) Proportion of backfilled jobs obtained by the Real trace.



(b) Proportion of backfilled jobs obtained by the synthetic trace (using $\epsilon = 5.0$).



(c) Proportion of backfilled jobs obtained by the synthetic trace (using $\epsilon = 10.0$).

Figure 26: SLURM simulated backfill proportions across real and synthetic data (three way comparison between the original trace and the traces obtained using $\epsilon = 5, 10$).

In Figure 25 we can observe that both synthetic traces remain, to a degree, reminiscent of the original scheduling timeline. However, the $\epsilon = 10$ visualization stands as noticeably more reminiscent of the original results.

In addition, there is a crucial behavior observable across both synthetic traces: the peaks representing "CPUs in queue" are shorter and wider than those of the original data, although they appear around the same submission times as in the original trace. This phenomenon can be linked as a direct visual observation of Differential Privacy's nature. The core goal of differential privacy is the elimination of highly distinct outliers, since it would be easy to detect their presence in the dataset, to protect those individuals. By truncating extreme spikes and distributing that mass into the neighboring distributions (with the noise), the synthetic data shows a "smoothing effect" where the spikes of the original dataset are now wider and shorter.

Moreover, we can also clearly observe in Figure 26 the issue with the pro-

portion of jobs being backfilled, with both synthetic traces having a far bigger backfilled job count.

From these results we can conclude that $\epsilon = 10$ preserves better the structure of the original data than $\epsilon = 5$, since it brings the backfilled job count closer to the real data (3,215 vs. 4,966). Nonetheless, it is still far from the real count (880).

More results of different epsilon values can be found at the Appendix H.

The Backfilling Dilemma

Another intriguing characteristic introduced by the synthetic traces that can be clearly observed in these simulation results is the massive increase in backfilled jobs. While the original workload triggered 880 backfilled executions, $\epsilon = 5$ generated 4,966 and $\epsilon = 10$ generated 3,215.

To ground this phenomenon, other experimental tests conducted at ultra-low and ultra-high privacy budgets of $\epsilon = 0.1$ and $\epsilon = 10000$ (which can be found in the Appendix H.1 and H.5 respectively). These traces also showed this massive increase in the backfill count, revealing a persistent issue in how differential privacy warps the traces.

Our hypothesis is that, despite the specific ϵ value, the implicit smoothing effect of differential privacy reduces how large outliers are when compared to the original distribution, while at the same such individuals are made more frequent. As a result, although large, resource-heavy jobs are still successfully generated, the workload ends up with "spikes" that are wider and shorter (we have more big jobs, but they aren't as big) and the workload also contains more "gaps", more "empty spaces" in the system (more resources that can be allocated via the Backfill algorithm). These gaps are readily filled by the scheduler, leading to a higher overall backfill count. This abnormality is due to differential privacy: it shrinks the influence of any single original job record, and because the dataset is large, that effect is amplified, making it extremely difficult to trace the generated output back to any specific individual job, and reducing any single original job record's influence results in the, formerly big and rare jobs, being "smaller" and more frequent (thus harder to identify).

Having both the insights from our statistical validation and the results of the scheduling simulations, we can conclude that for our final pipeline, an epsilon budget of 10 is preferable than an epsilon budget of 5 given similar privacy guarantees, similar statistical utility and better simulation results.

While isolated statistical distance metrics initially favored $\epsilon = 5$ due to its superior theoretical privacy and seemingly good synthesis quality and utility, the SLURM simulations revealed that it injected excessive noise causing the simulated wait times to be roughly half of the Real data's. Epsilon 10 mitigates this, capturing more authentic wait times and priority values, besides a better Utilized and Requested resources trace and less backfilling.

6.4 CLOVER Privacy Evaluation

As observed, $\epsilon = 10$ yields our best utility, thus, we performed this evaluation only for this value. To rigorously confirm that this setting also upholds privacy, we employ the comprehensive evaluation of the Clover library [40], covering both re-identification distance metrics and several membership inference attacks. The results are detailed below.

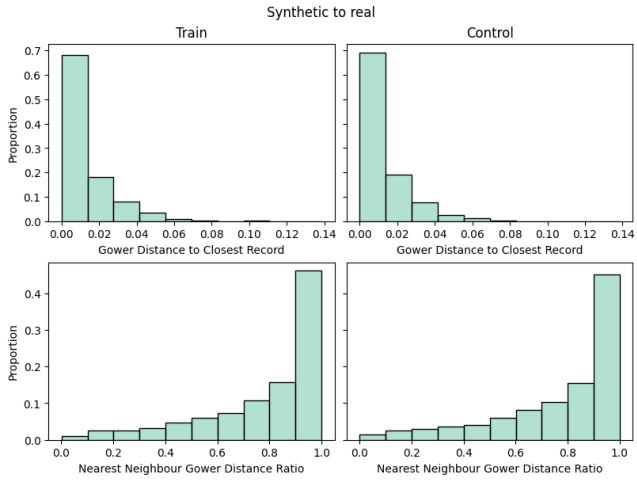


Figure 27: Distance to closest record (DCR) and nearest neighbor distance ratio (NNDR) of synthetic records relative to training and control real data.

Figure 27 evaluates the proximity of generated synthetic records to real-world data using the Gower Distance to Closest Record (DCR, top row) and the Nearest Neighbor Gower Distance Ratio (NNDR, bottom row) (while we used the euclidean distance of numerical values, Clover utilizes the Gower distance which also takes into account categorical values), compared against both the training dataset (Train) and an unseen holdout dataset (Control). Both metrics show a sharp spike near 0.0, indicating that approximately 70% of the synthetic records are highly similar to real records. However, the distributions between the Train and Control columns are virtually identical, demonstrating that the synthetic data captures the general structure of the real data without replicating individual records; the observed proximity reflects instead the high density and frequent categorical repetitions inherent to HPC traces and doesn't mean low privacy (it would be low privacy if the sensitive dataset didn't have many similar individuals).

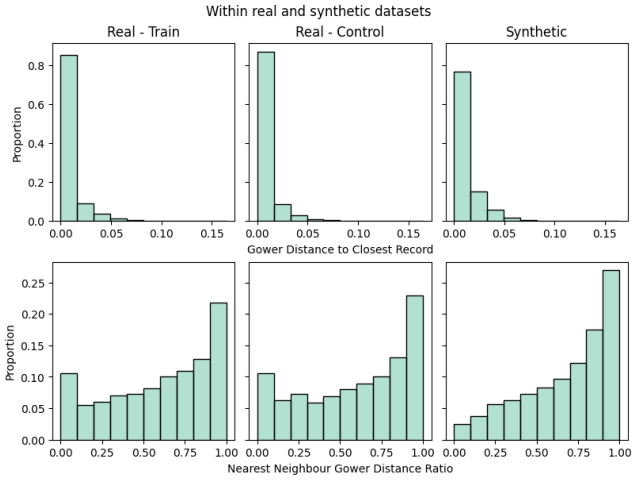


Figure 28: Internal dataset distance evaluation: DCR and NNDR computed within the real training, real control, and synthetic datasets individually.

Figure 28 provides a baseline of internal structural diversity by independently evaluating the training data (Real – Train), holdout data (Real – Control), and the generated synthetic data. The top row displays the internal Gower Distance to Closest Record (DCR), where all three datasets exhibit an overwhelming spike ($> 75\%$) at 0.0. This validates that the real data population is dense, containing a high volume of naturally occurring identical or near-duplicate records (Characteristic of HPC data as we've seen).
The bottom row shows the internal Nearest Neighbour Gower Distance Ratio (NNDR). The synthetic data faithfully replicates the upward distribution profile of the real data towards 1.0, although with a slightly lower proportion of near-zero ratios ($\approx 2.5\%$ vs. $\approx 10\%$ in the real data). This demonstrates that our pipeline captures the dense structure of the original data while introducing a slight smoothing effect that reduces exact internal pairwise duplication (since as we've seen the data has no exact clones).

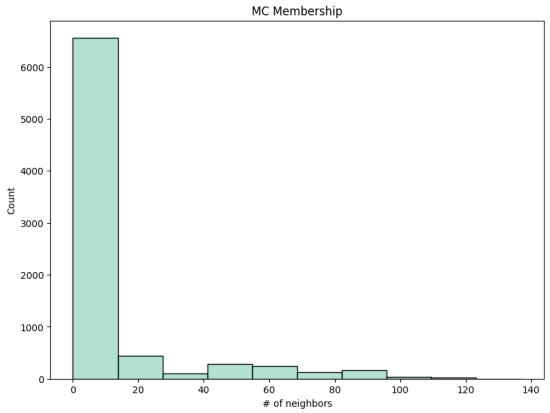


Figure 29: Monte Carlo membership evaluation: distribution of synthetic neighbour counts for real training records.

Figure 29 shows the Monte Carlo (MC) membership inference metric [40], which counts the number of synthetic neighbours within a predetermined radius for each real record in the training set. The distribution is heavily right-skewed, with an overwhelming majority of records (over 6,500) falling into the lowest bin, indicating fewer than 15 synthetic neighbours. A long, sparse tail extends past 120 neighbours. Because the Monte Carlo attack relies on elevated neighbour counts to flag candidate members, this low neighbour density means that the adversary gains no reliable signal: the vast majority of training records are not tightly surrounded by synthetic copies, so the attack cannot separate members from non-members. Thus, our pipeline effectively resists the Monte Carlo membership inference attack.

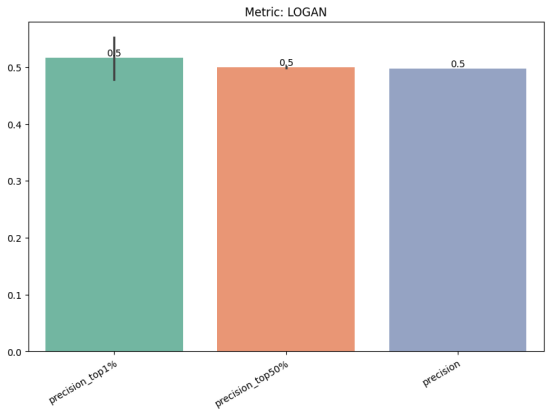


Figure 30: LOGAN membership inference attack evaluation.

Figure 30 presents the results of the LOGAN membership inference attack,

which trains a classifier to distinguish first-generation synthetic samples from second-generation synthetic samples (which were obtained by doing a secong synthetic data generation with our final pipeline). The evaluation reports precision at the attacker's top 1% most confident predictions (precision_top1%), top 50% most confident predictions (precision_top50%), and overall baseline precision. All three metrics reside exactly at or near 0.5 (random chance), with confidence intervals crossing the 0.5 threshold. This demonstrates that the pipeline provides strong resistance against membership inference, as an adversary has no statistical advantage over a random guess when attempting to identify training-set participants from the generated synthetic data.

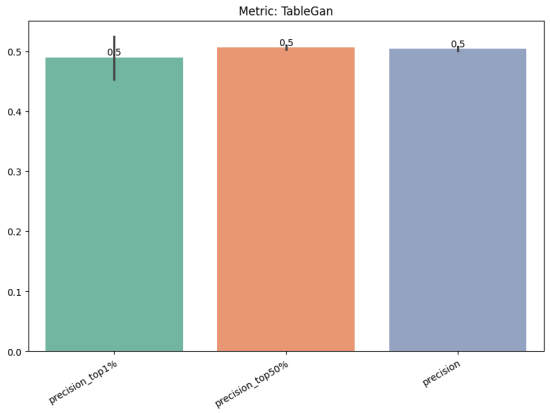


Figure 31: Privacy evaluation under the TableGAN membership inference attack.

Figure 31 shows the outcome of the TableGAN membership inference attack [40], where an adversarial model composed of a discriminator and a classifier is trained to distinguish between a generative model's training data (our Data synthesizing pipeline in this case) and unseen holdout data. Precision is measured at the top 1% most confident predictions, the top 50%, and overall. Across all scopes, precision remains at approximately 0.5. This shows us that the discriminator-classifier attack model performs no better than random guessing, confirming that our pipeline successfully protects against this specific membership inference attack.

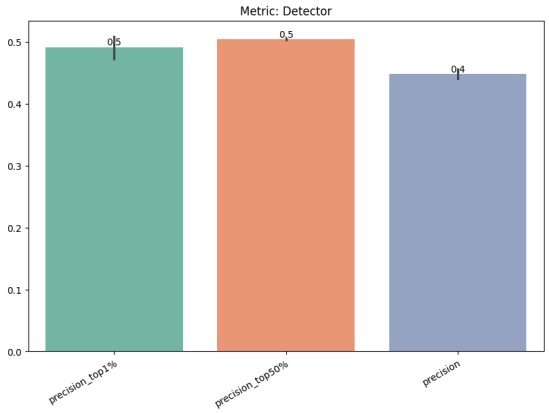


Figure 32: Membership inference attack evaluation using the Detector method from the Clover *library.*

Figure 32 simulates a membership inference attack by training an adversarial Detector model to classify our synthetic data against real holdout data (records never used to train the generator). The chart displays attack precision at different confidence thresholds once again: top 1%, top 50%, and overall. Precision remains firmly at 0.5 in all cases, meaning the Detector model does no better than random guessing. This demonstrates that the generated synthetic data does not leak identifiable traces of its original training records.

Across the full spectrum of re-identification and membership inference metrics, the model trained with $\epsilon = 10$ consistently exhibits privacy behavior indistinguishable from the ideal random baseline, validating its suitability for safe data sharing.

7 Conclusions

This thesis aimed to explore the gap between data utility and privacy in High Performance Computing (HPC) systems. Motivated by the need to safely share MareNostrum's workload traces to validate the Autosubmit scheduling mechanism, we explored, implemented, and evaluated a Differential Privacy (DP) framework tailored for complex HPC traces data.

Our initial investigations showed that traditional anonymization techniques and Local Differential Privacy approaches are insufficient for HPC environments, since local, record-level noise addition greatly decreased the statistical utility of data, which encouraged the adoption of a Global DP Data Synthesis approach. By leveraging privacy-preserving generative models (MST in particular) we aimed to bound the synthetic data generation with strict mathematical guarantees, preventing the model from memorizing specific jobs.

7.1 Validation of the Differentially Private Pipeline

While our theoretical privacy objectives were met, the validation phase, particularly the SLURM simulations, revealed significant limitations. By feeding our DP-synthesized traces into a SLURM simulator and comparing the scheduling outcomes against the real logs, we found severe differences. The validation showcased several key insights:

- **Privacy:** The synthesized datasets successfully masked highly unique, recognizable job submissions, with DCR and NNDR distances clearly showing that an adversary utilizing auxiliary information cannot confidently link specific synthetic jobs back to the real data, fully satisfying our privacy goal as shown in the results Table 6, with values for Mean DCR of 0.004299 and an NNDR of 0.787114.

- **Distortion of Scheduling Dynamics:** Although statistical metrics demonstrate utility for some applications such as ML or DL, the real-world scheduler simulations exposed critical anomalies. The synthetic traces suffered from a DP-induced smoothing by truncating extreme outlier jobs. The pipeline altered queue behavior, resulting in less pronounced but wider resource peaks and smaller wait times that deviated from the real data. This can be seen in Figure 25 and by the results of our best synthetic trace, which obtained a mean wait time of 177.63 seconds while the real data obtained a mean wite time of 213.64 seconds instead.

- **The Backfilling Dilema:** The most glaring anomaly that surfaced in the Slurm simulations executed from synthetic data was related to the proportion of backfilled jobs. The DP noise artificially inflated the number of backfilled jobs, jumping from 880 in the original baseline to over 3,200 even at our best-performing $\epsilon = 10$ threshold. This indicates that the synthesized traces deviate from real data in key aspects, making them non-applicable for this specific use-case.

- **The Privacy-Utility Trade-off Reality:** Through iterative testing, we identified $\epsilon = 10$ as the most functional threshold, offering a better reflection of the real data than tighter privacy bounds ($\epsilon = 5$), as can be seen in Table 6 and more importantly the slurm simulations Figures 25 and 26.

Ultimately, this work demonstrates that Global DP synthesis provides robust privacy guarantees, and generates useful data, capturing many of the underlying statistics and multivariate relationships of the original workload, that is applicable for some use in multiple instances such as for training Deep Learning models and Machine Learning models to, for instance, predict job run times or classify job completion statuses, as shown by our positive ML utility metrics (a RF loss of ≈ 0.026 and a LR loss of ≈ 0.0001 as seen in 6).

Beyond supervised learning, results indicate that the synthetic traces are also suitable for studying workload distributions or performing exploratory data analysis as shown by its low correlation errors (as shown by Figures 23 and 22) and outstanding synthesization of single attributes (as shown in Figure 24). However, results showed that DP's nature, smoothing out outliers, made Slurm traces simulated from synthetic data deviate and behave abnormally, ultimately implying that the results from the Autosubmit study would be considerably different with the synthetic data and indicating that there is still avenue for further research.

In short, while the synthetic data may not yet be a private replacement for simulation input, it offers a shareable, and privacy-compliant alternative for other data-driven research on HPC workloads.

As for the main goals of this thesis, all were achieved, even if in a deviated fashion. Various differential privacy techniques and synthetic data generation methods were tested, we identified and utilized tools to efficiently implement the privacy methods and the final pipeline was evaluated practically through replication of Autosubmit slurm results. However, it was demonstrated that the traces simulated from SLURM to heavily deviated from reality.

In addition, this work also showcased the great impact that discretization can have when generating DP synthetic data, especially when using Maximum Spanning Trees.

8 Future Work

As it has been proved in this study, while our pipeline offer positive results, it ultimately struggles to properly simulate with slurm synthetic traces better approximating reality. To overcome this limitation several future avenues of work can be explored.

First, the integration of deep generative models, such as Private Aggregation of Teacher Ensembles for GANs (PATE-GAN) [48], can be promising to evaluate the capacity of Deep Neural Networks at learning latent workload patterns that may allow for synthetic data that results in better simulations 4.4.4.

Second, the application of Reinforcement Learning (RL) to establish highly adaptive local Differential Privacy (LDP) mechanisms capable of dynamically adjusting noise levels based on real-time workload or systemic variations [30] may also overcome this limitation.

Lastly, incorporating a robust domain-specific post-processing could rectify structural anomalies or physically impossible values within the synthetic outputs allowing for better simulations. In this work, such post-processing was not performed since designing such a tailored postprocessing would require direct access to the proprietary sensitive dataset (MareNostrum's), access which in BSC's case is restricted, thus the reason of being of this whole project.

References

[1] Michael Barbaro and Tom Zeller. "A Face is exposed for AOL searcher no. 4417749". In: *New York Times* (Jan. 2006). URL: https://www.researchgate.net/publication/265660320_A_Face_is_exposed_for_AOL_searcher_no_4417749.

[2] Barcelona Supercomputing Center. *Centro Nacional de Supercomputación*. https://www.bsc.es. Accessed: May 2026. 2026.

[3] Alex Bie, Gautam Kamath, and Guojun Zhang. *Private GANs, Revisited*. 2023. arXiv: 2302.02936 [cs.LG]. URL: https://arxiv.org/abs/2302.02936.

[4] Steve J. Chapin et al. "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror G. Feitelson and Larry Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 67–90. ISBN: 978-3-540-47954-3. DOI: https://doi.org/10.1007/3-540-47954-6_4.

[5] Susan Coghlan and Narayan Desai. *Argonne National Laboratory (ANL) Intrepid Workload Log*. https://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/index.html. Distributed via the Parallel Workloads Archive. 2009.

[6] Chris Culnane, Benjamin I. P. Rubinstein, and Vanessa Teague. *Health Data in an Open World: A Re-identification Attack on the Australian MBS/PBS Data*. Technical report, University of Melbourne. Available at https://doi.org/10.48550/arXiv.1712.05627 (pre-print). 2017.

[7] Damien Desfontaines. "Lowering the cost of anonymization". PhD thesis. ETH Zurich, 2020. DOI: 10.3929/ETHZ-B-000508570. URL: https://hdl.handle.net/20.500.11850/508570.

[8] Cynthia Dwork. "Differential Privacy". In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12. ISBN: 978-3-540-35908-1. DOI: https://doi.org/10.1007/11787006_1.

[9] Cynthia Dwork, Nitin Kohli, and Deirdre Mulligan. "Differential Privacy in Practice: Expose your Epsilons!" In: *Journal of Privacy and Confidentiality* 9.2 (Oct. 2019). DOI: 10.29012/jpc.689. URL: https://journalprivacyconfidentiality.org/index.php/jpc/article/view/689.

[10] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Found. Trends Theor. Comput. Sci.* 9.3–4 (Aug. 2014), pp. 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: https://doi.org/10.1561/0400000042.

[11] Cynthia Dwork et al. “Our Data, Ourselves: Privacy Via Distributed Noise Generation”. In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by Serge Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 486–503. ISBN: 978-3-540-34547-3. DOI: https://doi.org/10.1007/11761679_29.

[12] Joseph Emeras. *CEA Curie Workload Log*. Parallel Workloads Archive, Hebrew University of Jerusalem. Provided by Joseph Emeras. Accessed: 2026-05-06. 2012. URL: https://www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie/index.html.

[13] Georgi Ganev et al. *The Importance of Being Discrete: Measuring the Impact of Discretization in End-to-End Differentially Private Synthetic Data*. 2025. arXiv: 2504.06923 [cs.CR]. URL: https://arxiv.org/abs/2504.06923.

[14] Steven Golob et al. *High Epsilon Synthetic Data Vulnerabilities in MST and PrivBayes*. 2024. arXiv: 2402.06699 [cs.CR]. URL: https://arxiv.org/abs/2402.06699.

[15] Ian J. Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

[16] Arthur Gretton et al. “A kernel two-sample test”. In: *J. Mach. Learn. Res.* 13.1 (Mar. 2012), pp. 723–773. ISSN: 1532-4435. URL: https://dl.acm.org/doi/10.5555/2503308.2188410#bibliography.

[17] Jamie Hayes et al. “LOGAN: Membership Inference Attacks Against Generative Models”. In: *Proceedings on Privacy Enhancing Technologies* 2019.1 (2019), pp. 133–152. DOI: 10.2478/popets-2019-0008.

[18] Justin Hsu et al. “Differential Privacy: An Economic Method for Choosing Epsilon”. In: *2014 IEEE 27th Computer Security Foundations Symposium*. 2014, pp. 398–410. DOI: 10.1109/CSF.2014.35.

[19] Intersoft Consulting. *GDPR-Info.eu*. URL: https://gdpr-info.eu/ (visited on 06/20/2026).

[20] Ana Jokanovic, Marco D’Amico, and Julita Corbalan. “Evaluating SLURM Simulator with Real-Machine SLURM and Vice Versa”. In: *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, Nov. 2018, 12, p. 12. DOI: 10.1109/PMBS.2018.8641556.

[21] Haim Kaplan, Shachar Schnapp, and Uri Stemmer. “Differentially Private Approximate Quantiles”. In: *CoRR* abs/2110.05429 (2021). arXiv: 2110.05429. URL: https://arxiv.org/abs/2110.05429.

[22] Hillol Kargupta et al. “On the Privacy Preserving Properties of Random Data Perturbation Techniques”. In: *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*. 2003, pp. 99–106. DOI: 10.1109/ICDM.2003.1250908.

[23] Aleksandra Korolova et al. “Releasing search queries and clicks privately”. In: *Proceedings of the 18th International Conference on World Wide Web*. WWW ’09. Madrid, Spain: Association for Computing Machinery, 2009, pp. 171–180. ISBN: 9781605584874. DOI: 10.1145/1526709.1526733. URL: https://doi.org/10.1145/1526709.1526733.

[24] Jaewoo Lee and Chris Clifton. “How Much Is Enough? Choosing ϵ for Differential Privacy”. In: *Information Security*. Ed. by Xuejia Lai, Jianying Zhou, and Hui Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 325–340. ISBN: 978-3-642-24861-0. DOI: https://doi.org/10.1007/978-3-642-24861-0_22.

[25] M. G. Marciani et al. “Evaluating the impact of task aggregation in workflows with shared resource environments: use case for the MONARCH application”. In: *Geoscientific Model Development* 18.23 (2025), pp. 9709–9721. DOI: 10.5194/gmd-18-9709-2025. URL: https://gmd.copernicus.org/articles/18/9709/2025/.

[26] M. G. Marciani. *docker-ubuntu-ces-slurm-sim: SLURM Simulator Docker Environment*. https://gitlab.earth.bsc.es/mgimenez/docker-ubuntu-ces-slurm-sim. Accessed: June 2026. 2024.

[27] M. G. Marciani. *pyswf: Python Standard Workload Format Library*. https://gitlab.earth.bsc.es/mgimenez/pyswf. Accessed: June 2026. 2024.

[28] Ryan McKenna, Gerome Miklau, and Daniel Sheldon. “Winning the NIST Contest: A scalable and general approach to differentially private synthetic data”. In: *CoRR* abs/2108.04978 (2021). arXiv: 2108.04978. URL: https://arxiv.org/abs/2108.04978.

[29] Ryan McKenna, Daniel Sheldon, and Gerome Miklau. “Graphical-model based estimation and inference for differential privacy”. In: *CoRR* abs/1901.09136 (2019). arXiv: 1901.09136. URL: http://arxiv.org/abs/1901.09136.

[30] Minghui Min et al. “Task Offloading With Differential Privacy in Multi-Access Edge Computing: An A3C-Based Approach”. In: *IEEE Transactions on Cognitive Communications and Networking* 12 (2026), pp. 5676–5689. DOI: 10.1109/TCCN.2026.3657108.

[31] Ilya Mironov. “Renyi Differential Privacy”. In: *CoRR* abs/1702.07476 (2017). arXiv: 1702.07476. URL: http://arxiv.org/abs/1702.07476.

[32] Yves-Alexandre de Montjoye et al. “Unique in the Crowd: The privacy bounds of human mobility”. In: *Scientific Reports* 3 (2013), p. 1376. DOI: 10.1038/srep01376.

[33] Arvind Narayanan and Vitaly Shmatikov. "Robust De-anonymization of Large Sparse Datasets". In: *Proceedings of the 2008 IEEE Symposium on Security and Privacy (S&P)*. 2008, pp. 111–125. DOI: 10.1109/SP.2008.33.

[34] Muhammad Naveed, Seny Kamara, and Charles V. Wright. "Inference Attacks on Property-Preserving Encrypted Databases". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2015, pp. 644–655. DOI: 10.1145/2810103.2813651.

[35] OpenDP. *SmartNoise Synthesizers Documentation*. https://docs.smartnoise.org/synth/index.html. Accessed: 2026-05-31. 2023.

[36] OpenDP Team. *OpenDP: A community-building effort for differential privacy*. https://opendp.org. Accessed: June 2026. 2020.

[37] OpenMined. *PyDP: A Python wrapper for Google's Differential Privacy project*. https://github.com/OpenMined/PyDP. Accessed: June 2026. 2020.

[38] Vijay Pandurangan. *On Taxis and Rainbows: Lessons from NYC's badly anonymized taxi logs*. Medium post. https://medium.com/@vijayp/of-taxis-and-rainbows-f6bc289679a1. 2014.

[39] Haoyue Ping, Julia Stoyanovich, and Bill Howe. "DataSynthesizer: Privacy-Preserving Synthetic Datasets". In: *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. SSDBM '17. Chicago, IL, USA: Association for Computing Machinery, 2017. ISBN: 9781450352826. DOI: 10.1145/3085504.3091117. URL: https://doi.org/10.1145/3085504.3091117.

[40] Yue Qi et al. *CLOVER: A framework for benchmarking synthetic data generation methods balancing utility and privacy in healthcare*. 2026. DOI: https://doi.org/10.1016/j.ailsci.2026.100155. URL: https://www.sciencedirect.com/science/article/pii/S2667318526000036.

[41] RIKEN Center for Computational Science. *F-DATA: A comprehensive workload dataset from Supercomputer Fugaku*. Zenodo, 2024. DOI: 10.5281/zenodo.11467483. URL: https://doi.org/10.5281/zenodo.11467483.

[42] José Moral de la Rubia. "Rice University Rule to Determine the Number of Bins". In: *Open Journal of Statistics* 14 (2024), pp. 119–149. DOI: 10.4236/ojs.2024.141006.

[43] scikit-learn developers. *IterativeImputer*. scikit-learn documentation. Accessed: 2026-06-21. URL: https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html.

[44] Latanya Sweeney. "Simple Demographics Often Identify People Uniquely". eng. 2000. URL: http://dataprivacylab.org/projects/identifiability/.

[45] Yuchao Tao et al. *Benchmarking Differentially Private Synthetic Data Generation Algorithms*. 2022. arXiv: 2112.09238 [cs.CR]. URL: https://arxiv.org/abs/2112.09238.

[46] UNHCR. *Differentially Private Synthetic Microdata: A practical guide on how to create synthetic datasets using differential privacy*. https://microdata.unhcr.org/index.php/synthetic-data. UNHCR Microdata Library, accessed: YYYY-MM-DD. 2025.

[47] Ana Luisa Veroneze Solórzano et al. "Bringing Differential Privacy to HPC: Privacy-Preserving Transformations of HPC Traces". In: *Proceedings of the 34th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '25. University of Notre Dame Conference Facilities, Notre Dame, IN, USA: Association for Computing Machinery, 2025. ISBN: 9798400718694. DOI: 10.1145/3731545.3731573. URL: https://doi.org/10.1145/3731545.3731573.

[48] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. "PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=S1zk9iRqF7.

[49] Jun Zhang, Xiaokui Xiao, and Xing Xie. "PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions". In: *CoRR* abs/1601.03229 (2016). arXiv: 1601.03229. URL: http://arxiv.org/abs/1601.03229.

[50] Jun Zhang et al. "PrivBayes: Private Data Release via Bayesian Networks". In: *ACM Trans. Database Syst.* 42.4 (Oct. 2017). ISSN: 0362-5915. DOI: 10.1145/3134428. URL: https://doi.org/10.1145/3134428.

[51] Zilong Zhao et al. *CTAB-GAN: Effective Table Data Synthesizing*. 2021. arXiv: 2102.08369 [cs.LG]. URL: https://arxiv.org/abs/2102.08369.

A How to Use

The pipeline is designed with a focus on generality and ease of use, allowing it to handle arbitrary datasets (even including non-HPC data). This flexibility is achieved via a user-defined metadata configuration file in JSON format. This file provides the pipeline with essential execution parameters: the input dataset path, the target path for the generated synthetic data, the privacy budget parameters (ϵ and δ), the length of the synthetic dataset to generate and a structural definition of the dataset's variables (including their data types, domain bounds, and whether they should be synthesized or dropped).

> **Domain-Knowledge Bounds:** Before defining the available data types, it is critical to clarify the role of domain-knowledge bounds. These bounds are used to discretize continuous data during the privacy-preserving transformation. Crucially, to prevent violating differential privacy guarantees, these bounds must not be derived from inspecting the sensitive dataset itself. Instead, they must rely strictly on external, public, domain knowledge or broad, conservative estimates (e.g., knowing a simulation runtime will absolutely not exceed two years). They can be very broad numbers, that won't affect the results much.

A.1 Supported Data Types

The pipeline processes columns based on four distinct data types defined in the metadata:

- `numerical`: Quantitative variables, whether continuous or discrete (e.g., number of CPU cores or execution runtime).
- `categorical`: Variables containing a fixed, discrete set of classes, whether encoded as strings or integers.
- `identifier`: Unique string tokens assigned to individual data points. To ensure privacy, identifiers are stripped at the execution's start and reappended post-synthesis as a sequential index starting from 1.
- `identicat`: A specialized hybrid type for non-unique identifiers (e.g., `User_ID` or `Group_ID`) whose frequencies and distributions must be preserved for analytical utility. These variables are pseudonymized in a differentially private way before being processed as a standard categorical variable. As seeb ub section 5.2

A.2 Execution Workflow

To use the pipeline, you must first construct a valid JSON metadata file that matches your dataset's schema. Below is an example configuration file:

```
{
    "data_path": "your/path/CEA-Curie-2011-2.1-cln.swf",
    "synthetic_data_path": "your/path/curie_synthetic_df.csv",
    "epsilon": 10.0,
    "delta": 0.001,
    "length_synth_data": null,
    "variables": [
        {"name": "Job_Number", "type": "identifier", "bounds":
        ↪  null, "wanted": "synthesize"},
        {"name": "Submit_Time", "type": "numerical", "bounds":
        ↪  {"min": 0, "max": 63072000}, "wanted": "synthesize"},
        {"name": "Wait_Time", "type": "numerical", "bounds":
        ↪  {"min": 0, "max": 10000000}, "wanted": "synthesize"},
        {"name": "Run_Time", "type": "numerical", "bounds":
        ↪  {"min": 0, "max": 10000000}, "wanted": "synthesize"},
        {"name": "Allocated_Processors", "type": "numerical",
        ↪  "bounds": {"min": 1, "max": 80000}, "wanted":
        ↪  "synthesize"},
        {"name": "Average_CPU_Time", "type": "numerical",
        ↪  "bounds": null, "wanted": "synthesize"},
        {"name": "Used_Memory", "type": "numerical", "bounds":
        ↪  null, "wanted": "drop"},
        {"name": "Requested_Processors", "type": "numerical",
        ↪  "bounds": null, "wanted": "drop"},
        {"name": "Status", "type": "categorical", "bounds": null,
        ↪  "wanted": "synthesize"},
        {"name": "User_ID", "type": "identicat", "bounds": null,
        ↪  "wanted": "synthesize"},
        {"name": "Group_ID", "type": "identicat", "bounds": null,
        ↪  "wanted": "synthesize"}
    ]
}
```

Figure 33: CEA-Curie Metadata example

Note: Setting ***`length_synth_data`*** *to* ***`null`*** *defaults the output length to the noisy, differentially private row count computed from the original dataset.*

Once the metadata file is ready, execute the generalized main script from your terminal:

```
$ python3 /your/path/generalized_main.py
```

Figure 34: Command for executing the pipeline

The main script will now initialize and prompt you for the configuration path:

```
Enter the metadata file's path: /your/path/to/metadata.json
```

Figure 35: System prompt.

After this, the execution will proceed automatically, exporting the generated differentially private synthetic dataset to your designated `synthetic_data_path`. At the end it will also prompt you asking if you wish to transform these .csv synthetic results to .swf.

B Work Plan

The overall project was planned according to the Gantt chart shown in Figure 36. The chart covers the period from February to June 2026 and is divided into eight main activity blocks.

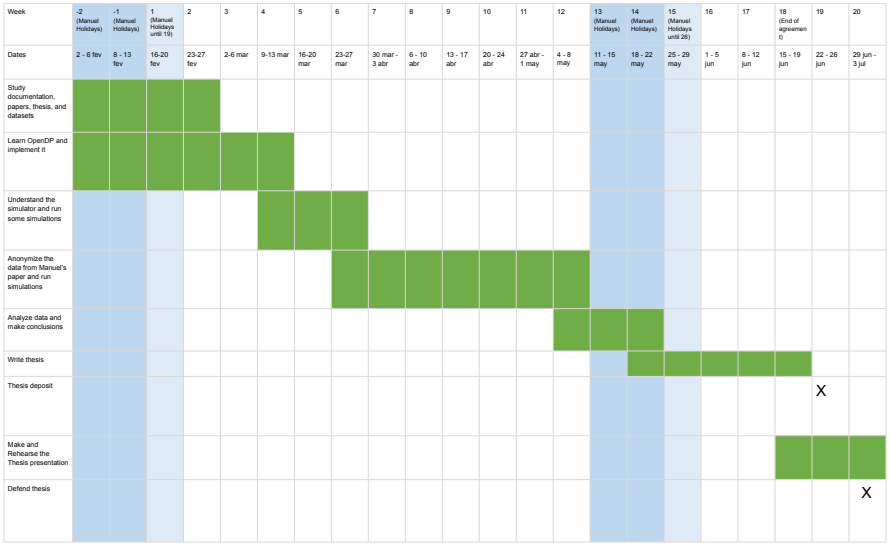


Figure 36: Gantt chart of the research internship and thesis development.

The chart visually summarizes the weekly distribution of the following high-level tasks:

- Study of documentation, papers, and datasets.
- Learning and implementing OpenDP.

- Understanding the SLURM simulator and running initial simulations.
- Anonymizing HPC data (iterative development of the DP pipeline).
- Analyzing data and drawing conclusions.
- Writing the thesis.
- Depositing the thesis.
- Preparing and rehearsing the defense.
- Final defense.

Detailed task-oriented work plan

The following Table provides a more detailed breakdown of the thesis, aligned with the Gantt chart and the actual research process described in Chapters 5 and 6.

Period	Activities
2–6 Feb	Study of introductory literature (Dwork 2006, Solórzano et al. 2025); read HPC scheduling papers (Marciani et al. 2025, Cirne 2001).
8–13 Feb	Install OpenDP and PyDP; implement basic DP primitives (means, counts).
16–20 Feb	Understand SLURM simulator; run example simulations with public traces.
23–27 Feb	First anonymization attempts on HPC logs: local DP with Laplace and randomized response.
2–6 Mar	Experiment with DataSynthesizer on Fugaku dataset; compare independent vs correlated modes.
9–13 Mar	Switch to Curie dataset; handle missing values (initial imputation).
16–20 Mar	Develop complete local DP pipeline (job-level noise); evaluate utility.
23–27 Mar	Discover privacy leaks in post-processing (bin edges, MinMax scaling, imputation).
30 Mar–3 Apr	Read Ganev et al. (2025); implement PrivTree discretization (binary splitting).
6–10 Apr	Integrate CLOVER framework; compare PrivTree-MST vs CLOVER-MST.
13–17 Apr	Test logarithmic transformation; adopt selective log for skewed variables.
20–24 Apr	Generalize pipeline to multiple datasets (ANL Intrepid).
27 Apr–1 May	Fix categorical pseudonymization using Korolova thresholding (DP domain discovery).
4–8 May	Automate parameters (dynamic max depth, square root rule); remove private bound finder.
11–18 May	Research about informed ways to choose ϵ; Final validation runs for multiple ϵ values; apply elbow method to select $\epsilon = 10$.
18–26 May	Run SLURM simulations for $\epsilon = 5$ and $\epsilon = 10$; compare scheduling metrics.
26 May - 20 Jun	Write thesis chapters (Introduction, Background, Process, Final Pipeline, Validation, Ethics, Environment, Conclusions, Future Work).
20 – 22 Jun	Complete appendices (code listings, extra results); deposit thesis.
22 – 25 Jun	Prepare defense slides.
25 – 29 Jun	Rehearse presentation.
30 Jun	Thesis defense.

Table 10: Weekly breakdown of main tasks (February–July 2026).

The work plan was overall followed with only minor adjustments made. All objectives and milestones were achieved within the scheduled timeline.

C Cost Analysis

This section provides an economic evaluation of the resources used during the development of the thesis. Costs are broken down into personnel (student and supervisors), hardware, software, and energy consumption. All Figures are estimates.

C.1 Personnel costs

- **Student (research intern)**: 6 months (February – July 2026) and full-time 37.5 hours/week. Total hours: 652 hours. Monthly gross salary: 1416.67 €. **Subtotal: 17000 €**.
- **Supervisor (Manuel Marciani)**: Estimated 2 hours/week for meetings and feedback. Total: $5 \times 4 \times 2 = 40$ hours. Estimated Hourly cost (PhD researcher): 24 €/h [1]. **Subtotal: 960 €**.
- **Tutor (Gladys Utrera)**: Estimated 1 hour/week. Total 20 hours, 31 €/h[2] → **620 €**.

Total personnel cost: $17,000 + 960 + 620 = 18,580$ €.

C.2 Hardware and infrastructure

- **Laptop (Dell Latitude 7450, Intel Core Ultra 5/i5, 16 GB RAM)**: Already owned by BSC; amortized over 3 years (Specifications verified via Dell Latitude 7450 Owner's Manual[3]).
  Purchase price $\approx 1,000$ €. Usage fraction for this project: 6 months = 1/6 of useful life → **200 €**.
- **No dedicated HPC nodes were used**: all experiments ran locally.
- **External storage and backups**: negligible (cloud storage free tier).

C.3 Software and libraries

All software used is open source and freely available:

- Python 3.10, pandas, numpy, scikit-learn, matplotlib, seaborn.
- OpenDP (SmartNoise), PyDP, CLOVER framework.

[1] See Glassdoor: `https://www.glassdoor.es/Sueldo/Barcelona-Supercomputing-Center-Sueldos-E382342.htm`

[2] See UPC salary tables: `https://portal.personal.upc.edu/ca/pdi/fitxers/sistema-retributiu/taules-retributives`

[3] See Dell Latitude 7450 Owner's Manual `https://www.dell.com/support/manuals/es-es/latitude-14-7450-2-in-1-laptop/latitude_7450_om/introduction?guid=guid-5ad2b015-4ee3-4b1c-9828-2b02fa65099d&lang=en-us`

- SLURM simulator Docker image (provided by BSC).
- LaTeX (Overleaf free plan).

Software cost: 0 €.

C.4 Energy consumption

The laptop consumes approximately 30 W under typical load according to its specifications (Dell Latitude 7450 Owner's Manual [3]).

Total approximate compute hours: ≈ 652 hours.
Energy: $652 \times 0.03 = 14.4$ kWh.
Electricity price (Spain, 2026): ≈ 0.15 €/kWh.
Energy cost: $14.4 \times 0.15 = 2.16$ €.

C.5 Total cost

This cost is entirely covered by the BSC internship program and the UPC academic budget, thus, no external funding was required.

$$\text{Personnel} + \text{Hardware} + \text{Energy} = 18,580 + 200 + 2.16 \approx \mathbf{18,782.16}\ €.$$

C.6 Economic viability

It must be stated that this project does not aim for commercialization, but the developed PT-MST pipeline can be reused by BSC and other research institutions at zero marginal cost (open source). Its main economic benefit is the avoidance of privacy breaches related costs (legal fines, reputational damage, security risks) that could arise from releasing raw HPC logs. Additionally, from a research perspective, the pipeline would allow sharing of synthetic traces, thus accelerating collaborative projects (such as the Autosubmit study).

D Other internship related activities

So far only the research itself has been described, which occupied most of the time during the stay as a research intern at BSC. Beyond this core work, however, other complementary activities were also engaged with.

On the one hand, almost every week, various meetings of different nature were attended. These meetings allowed the observation of how the entity (BSC) operates. Most of those meetings were to share the progress updates of different people and groups of the department (of Earth Sciences).

In other meetings, individual researchers working in their master's thesis or PhD presented their work, mainly focused in research around weather, fire and other related to Earth sciences topics. More interestingly, most works involved practical applications of AI, mainly Machine Learning, and most importantly, Deep learning, applied in those fields, many times state-of-the-art. Last but not least, I also participated in compulsory trainings sessions:

- BSC introduction: A first training where they explained to the newcomers what apps BSC uses, how to set our BSC accounts, use its online tools and what first compulsory steps to take.
- Equity, Diversity & Inclusion training where they expose situations and provide tips to avoid possible discrimination.
- Health and Safety risk training: Covered the protocols for possible health hazards at BSC as well as the fire safety procedures at the installation (including how to use,when to use and where to find fire extinguishers and water hoses) and ergonomic advice for maintaining good postures at the desks to avoid back and other body aches. These activities, though limited in size and scope in comparison to the main work, enriched my experience at BSC and has enabled me to see the inner workings of an institution such as this.

E Gender and Bias Analysis

This section analyses potential gender biases and other possible fairness issues that may arise in the data, the model, and the overall pipeline.

E.1 Bias in the Input Data

HPC workload logs (Curie, Fugaku, ANL Intrepid) contain no demographic attributes (such as age, gender or ethnicity). The only user-related fields are `User_ID` and `Group_ID`, which are pseudonymized using differential privacy (Section 5.2). No gender information is stored nor used, thus the pipeline cannot introduce or propagate gender biases. Nevertheless, historical HPC user bases may be gender-imbalanced since the DP synthesizer preserves the original distribution of user IDs, thus any existing imbalances are maintained, although since said User IDs are gender neutral this is not reflected.

E.2 Bias in the Model and Synthesis

The MST synthesizer constructs marginals based on correlations. It does not use any protected attribute, therefore, it cannot produce outputs that discriminate on gender, race, or other sensitive categories since no attributes of this kind were used. The exponential mechanism used for edge selection is data-dependent but does not amplify existing societal biases.

E.3 Bias Mitigation Strategies

- Differential privacy itself can act as a bias mitigation mechanism: by limiting the influence of any single record, DP prevents the model from over-learning biases of rare subpopulations (for instance, a single female user with a unique job pattern).

- Korolova thresholding (Section 4.8.1) removes rare categories that could be used for re-identification, which also reduces the risk of targeting small, possibly vulnerable groups.

E.4 Gender Equality in the Research Team

The work was conducted by a male student under the supervision of a male researcher and a female tutor (Gladys Utrera). Nonetheless, the Earth Sciences Department at BSC (and more widely BSC) promotes gender diversity, shown by the strong female presence of the department and at BSC in general. Furthermore, during the internship mandatory equity, diversity, and inclusion trainings organized by BSC were attended.

E.5 Summary

The PT-MST pipeline is gender-neutral by design. No step introduces or exacerbates gender bias and the project itself is almost entirely unrelated from any population, focused instead on genderless job logs. Furthermore, the use of differential privacy and DP pseudonymization inherently protects against disparate impact that could arise from overfitting to sparse groups.

F Ethical Considerations and Privacy

This work has a straightforward and deeply important ethical side: Privacy.

For a long time Artificial Intelligence, more specifically, generative models, have been creating concerns for the impacts that these techniques, when applied without proper moral and ethical guidelines, have on society. More specifically, about individual's privacy among many other concerns. It is only fitting, then, that this same techniques provide a solution such as the one exposed in this work. Privacy is not a mere regulatory imposition or a superficial preference. It is a fundamental human right, essential for individual autonomy and institutional trust.

This work proposes a solution to a privacy concern, it enforces the right that people, companies, organizations etc have for keeping their data private while at the same time it doesn't curtail research, allowing for the created data to still be useful. This research is thus not only innocuous from an ethical standpoint, but beneficial, allowing research to be done without causing harm, without breaching the human right that is privacy, ensuring that helping science to move forward by participating in a medical study, or by allowing your data to be gathered, is not punished instead of rewarded.

F.1 Risks of Re-identification in HPC Logs

In the domain of High-Performance Computing (HPC), data sharing is frequently hampered by the inherent security risks of exposure.

As an example, through linkage attacks, seemingly benign data can be traced back to specific researchers, projects, or companies. This not only compromises individual user privacy but also threatens to leak industrial secrets or the strategic direction of scientific research.

F.2 Responsible Data Sharing and GDPR Compliance

To foster open science and collaborative innovation without compromising security, developing mechanisms for responsible data sharing is of key importance. Traditional anonymization techniques, such as deterministic Pseudonymization or data obfuscation, have proven insufficient against modern, data-driven deanonymization attacks 3.1. This failure creates a tension with modern legal frameworks, most notably the European Union's General Data Protection Regulation (GDPR)[19], which mandates strict adherence to the principles of "privacy by design" and data minimization.

Through the use of Differential Privacy (DP), this work directly resolves this issue. Differential Privacy provides a mathematically rigorous guarantee that the presence or absence of any single individual's log entry does not significantly affect the output distribution of the generative model. Consequently, it allows organizations to share highly realistic synthetic logs that comply with the requirements of the GDPR, and their own desires of keeping specific details of their system secret, ensuring that data sharing is both legally compliant, ethically sound and, moreover, beneficial for the organization and research overall.

However, just this year, a significant legal hurdle has surfaced. On June 4, 2026 the United State's Department of Commerce issued the Disclosure Avoidance for Statistical Products[4] order. This order explicitly bans the usage of "noise infusion" (a category including differential privacy) for statistical products disseminated by the Census Bureau and the Bureau of Economic Analysis.

F.3 Contribution to Ethical AI/ML

Ultimately, this research contributes to a vital paradigm shift within the broader landscape of ethical AI and Machine Learning. Instead of viewing technological advancement and privacy protection as a choice where one must be sacrificed for the other, this project demonstrates that utility and privacy can coexist and be treated rather like a dial, where you choose the sacrifices you want to make.

By demonstrating that synthetic data generation models can also be used to safeguard the environments they document, this work actively enforces the right that people, companies, and organizations have for keeping their data private. At the same time, it ensures we do not curtail this expanding field, turning generative AI from a potential threat into one of privacy's most resilient shields, proving that no technology or technique is inherently harmful, but the uses people give it are, be it purposely or not.

[4] `https://www.commerce.gov/opog/disclosure-avoidance-statistical-products`

G Environmental and Sustainability Analysis

This section evaluates the environmental, social, and economic sustainability of the project and maps it to the United Nations Sustainable Development Goals (ODS)[5].

G.1 Environmental sustainability

The main environmental concern of this work is the energy consumption associated with generating large, differentially private datasets. This was a factor considered when making architectural and algorithmic choices: a marginal-based data generation technique (MST) was selected over highly hyperparameter dependent and resource-intensive alternatives such as Generative Adversarial Networks (GANs). The marginal-based approach significantly reduces the parameter space, implying less experimentation, lower computational cost, and a smaller carbon footprint.

In addition, for this research the concrete energy usage was very low. All experiments ran locally on a single laptop (Dell Latitude 7450, Intel Core i5, 16 GB RAM) provided by BSC. No HPC nodes were used. As calculated in Appendix C, the total energy consumed was approximately 14.4 kWh, resulting in roughly 3 kg of CO_2 emissions (Spanish grid average: ≈ 0.2 kg CO_2/kWh). This is negligible compared to typical HPC workloads.

Moreover, the laptop was not purchased exclusively for this research: it will be reused by other researchers at BSC, amortizing its production footprint. Overall, the project has a minimal carbon footprint.

G.2 Social sustainability

The project directly addresses a societal challenge: protecting individual privacy while enabling data-driven research. By providing a practical differential privacy (DP) pipeline for HPC logs, it fosters responsible data sharing, which is essential for open science. The work aligns with the GDPR's "privacy by design" principle, reducing legal risks for institutions like BSC. It also helps build trust between data subjects (users of supercomputers) and data holders (HPC centers), encouraging participation in scientific studies.

G.3 Economic sustainability

The pipeline is open source and requires no proprietary software. Its use can prevent costly data breaches (fines, reputation loss) and reduces the need for manual anonymization, which is time-consuming and error-prone. The cost analysis (Appendix C) shows a total investment of 18,782.16 € that almost in its entirety come from salaries, making the solution economically sustainable.

[5] `https://www.un.org/sustainabledevelopment/sustainable-development-goals/`

G.4 Contribution to the Sustainable Development Goals (ODS)

The following Table maps the thesis outcomes to specific ODS targets:

ODS	Contribution
ODS 9 (Industry, Innovation and Infrastructure)	Enables secure sharing of supercomputer workload traces, fostering innovation in HPC scheduling and resource management without compromising user privacy.
ODS 16 (Peace, Justice and Strong Institutions)	Protects the right to privacy (Article 12 of the Universal Declaration of Human Rights). The DP pipeline provides strong mathematical guarantees, helping institutions comply with data protection laws (GDPR).
ODS 11 (Sustainable Cities and Communities)	HPC centers are critical infrastructure. By reducing the risk of data leaks, the work contributes to the resilience of these systems.
ODS 4 (Quality Education)	The thesis and the open-source code serve as educational material for students learning about differential privacy, synthetic data, and ethical AI.
ODS 13 (Climate Action)	The low energy footprint of the proposed pipeline supports climate-conscious computing.

Table 11: Alignment with UN Sustainable Development Goals

G.5 Summary

The project has a minimal environmental footprint, strong social benefits (privacy protection), and is economically viable. It directly contributes to ODS 9, 16, and 11, and indirectly supports ODS 4 and 13. Even though the research does not actively reduce carbon emissions, it addresses a crucial ethical concern (privacy) and brings us closer to a time when researchers can safely benefit from BSC's data without harming individuals or the environment.

H Other experimental results

In this section of the appendices we present the results obtained by other ϵ values along a brief analysis.

H.1 Final Pipeline of the results of epsilon 0.1

As it can be seen in Table 12, at $\epsilon = 0.1$, the extreme noise injection significantly increases both Wasserstein and TVD distances compared to higher ϵ values. In addition, the average bound difference also increases when compared with all epsilons (excluding epsilon 1). Remarkably, despite the noise, machine learning (ML) utility is preserved. Furthermore, privacy guarantees are very strong, as

evidenced by the all DCR metrics being very high, fact further compounded by the NNDR being outstandingly high as well. However, as expected under tight privacy constraints, MMD distances and correlation errors see a noticeable increase. The improved 3-way marginal score is most likely due to PrivTree, under such tight constraints, having generated far lower bins.

Metric	**Value**
Avg Bound Diff	353579.185452
Mean Wasserstein	0.001875
Mean TVD	0.003375
RF Loss	0.025797
LR Loss	-0.000032
Discriminator AUC	0.620802
Mean Synth to Train DCR	0.006983
Min Synth to Train DCR	0.000001
Mean 1-DCR	9770.474923
Mean 5-DCR	14072.927499
NNDR	0.793794
Exact clones	0.000000
Spearman Corr Error	0.108936
Linear Corr Error	0.049351
MMD	0.039510
3-Way Marginal Score	0.520732
HOC Score	0.001179

Table 12: Summary of evaluation metrics of $\epsilon = 0.1$.

As shown in Figure 37, the severe noise injection results in a chaotic, noisy trace of simulated resources that bears no visual resemblance to the original data.

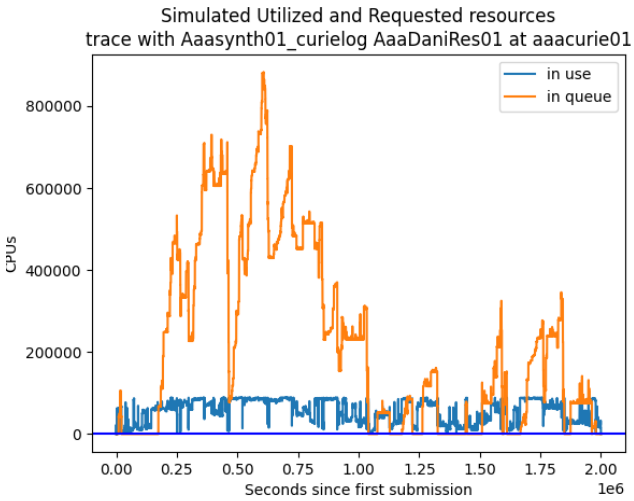


Figure 37: Simulated utilized and requested resources obtained by synthetic trace generated using epsilon 0.1

Figure 38 demonstrates that the proportion of backfilled jobs is artificially inflated compared to the original dataset. The accompanying summary tables provide further breakdown: Backfilled jobs average 1,682 allocated CPUs with a mean priority of 7,784. In contrast, Non-Backfilled jobs exhibit a much higher mean priority of 25,183. Finally, the Wait Time Table reveals massive variance due to noise; while the median wait time is brief (13), the mean is pulled to 1,341 by extreme outliers reaching over 533,000.

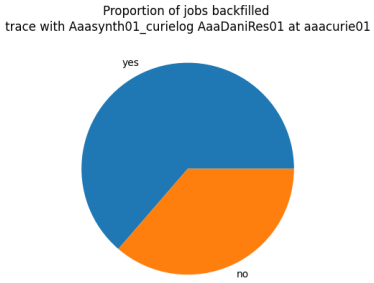


Figure 38: Proportion of backfilled jobs obtained by the synthetic trace generated using epsilon 0.1

Statistic	alloc_cpus	priority
Count	5036	5036
Mean	1682.913423	7784.904289
Std	7506.040746	13459.994359
Min	16	3335
25%	16	3339
50%	64	3349
75%	512	3537.25
Max	79648	103923

Table 13: Summary statistics for Backfilled (BF) jobs.

Statistic	priority
Count	2874
Mean	25183.645442
Std	28751.446174
Min	3335
25%	3336
50%	3448
75%	53340
Max	107045

Table 14: Summary statistics for Non-Backfilled (Non-BF) jobs: Priority.

Statistic	wait_time
Count	7910
Mean	1341.161820
Std	13757.320769
Min	0
25%	0
50%	13
75%	39
Max	533418

Table 15: Summary statistics for Wait Time.

H.2 Final Pipeline averaged results of epsilon 1.0

In Table 16 we can observe a noticeable increase in Avg Bound difference, however, this does not indicate that it achieved worse synthesization results than lower epsilon values, as evidence by the improvement of Wasserstein and TVD distances. Meanwhile, RF and LR losses remain roughly the same while DCR

metrics and the NNDR have decreased compared with epsilon 0.1, as expected, but are still greater than those for bigger epsilon values. In addition, the Spearman and Linear Correlation errors have both improved while HOC has decreased, despite having more bins, and 3-way marginal score have increased because of having more bins.

Metric	**Value**
Avg Bound Diff	386284.477228
Mean Wasserstein	0.000675
Mean TVD	0.002275
RF Loss	0.026052
LR Loss	-0.000016
Discriminator AUC	0.629771
Mean Synth to Train DCR	0.004614
Min Synth to Train DCR	1.8672e-07
Mean 1-DCR	7499.858006
Mean 5-DCR	10894.619109
NNDR	0.788066
Exact clones	0.000000
Spearman Corr Error	0.087451
Linear Corr Error	0.044833
MMD	0.022196
3-Way Marginal Score	0.604374
HOC Score	0.000574

Table 16: Summary of evaluation metrics for $\epsilon = 1$.

Figure 39 shows that the simulated resource usage begins to visually resemble the original data, though the queue spikes are wider and shorter due to DP noise.

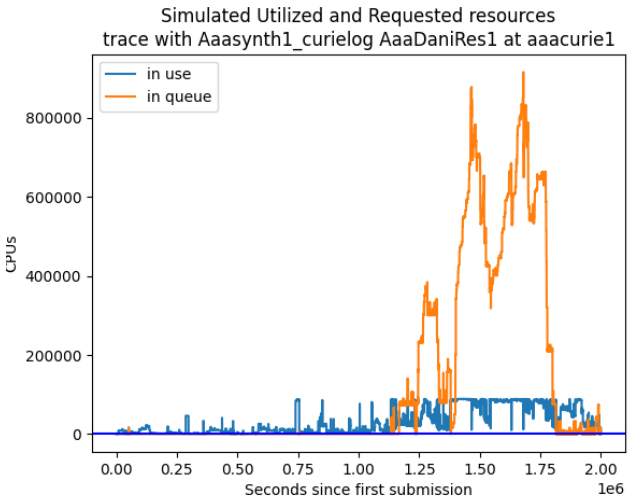


Figure 39: Simulated utilized and requested resources obtained by synthetic trace generated using epsilon 1

In Figure 40 the over-representation of backfilled jobs persists at this ϵ level (and as we will see, any level). The summary tables indicate that Backfilled jobs consume fewer CPUs on average (1,259) and hold lower average priorities (10,573) compared to Non-Backfilled jobs, which average a priority of 22,048. The Wait Time Table shows a mean of 1,042, but a median of just 9, demonstrating that the vast majority of synthetic jobs process quickly despite a heavily skewed maximum.

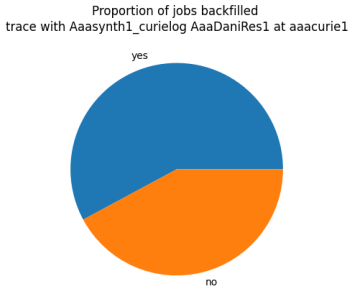


Figure 40: Proportion of backfilled jobs obtained by synthetic trace generated using epsilon 1

Statistic	alloc_cpus	priority
Count	5139	5139
Mean	1259.080755	10573.694688
Std	5588.095343	13237.950785
Min	16	593
25%	16	2962
50%	64	7694
75%	320	9479
Max	78112	100943

Table 17: Summary statistics for Backfilled (BF) jobs.

Statistic	priority
Count	3743
Mean	22048.674058
Std	29103.227236
Min	593
25%	1185
50%	5974
75%	35517
Max	103739

Table 18: Summary statistics for Non-Backfilled (Non-BF) jobs: Priority.

Statistic	wait_time
Count	8882
Mean	1042.792839
Std	10039.285617
Min	0
25%	0
50%	9
75%	37
Max	337805

Table 19: Summary statistics for Wait Time.

H.3 Final Pipeline averaged results of epsilon 3.0

As it can be observed in Table 20, with $\epsilon = 3.0$, the average bounds error and mean Wasserstein distance decrease as the privacy budget relaxes, being however worse than most of the bigger epsilons. Conversely, the TVD distance error increases, as the higher number of bins forces more noise into the categorical

synthesis. This is because the more categories we have, the more noise MST is forced to inject to maintain privacy. Nevertheless, the Wasserstein distances decrease as also does MMD indicating better synthesization. In addition, RF and LR losses hold steady and correlations both decrease further compounding that there was a better synthesization. As expected, privacy guarantees (DCR, NNDR) decrease as the privacy guarantees are relaxed.

Metric	**Value**
Avg Bound Diff	327995.347792
Mean Wasserstein	0.000650
Mean TVD	0.004500
RF Loss	0.025621
LR Loss	-0.000016
Discriminator AUC	0.657816
Mean Synth to Train DCR	0.004630
Min Synth to Train DCR	1.2288e-07
Mean 1-DCR	6921.756654
Mean 5-DCR	10190.334252
NNDR	0.788134
Exact clones	0.000000
Spearman Corr Error	0.084916
Linear Corr Error	0.043427
MMD	0.020160
3-Way Marginal Score	0.678886
HOC Score	0.000377

Table 20: Summary of evaluation metrics of $\epsilon = 3$.

Figure 41 reveals that the resource utilization trace at $\epsilon = 3.0$ mirrors the results of $\epsilon = 5.0$. The queue spikes are narrower than those observed at $\epsilon = 1.0$ and less pronounced than the original data.

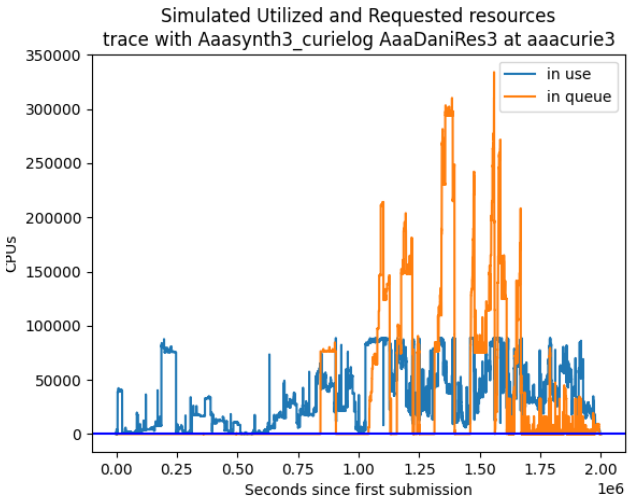


Figure 41: Simulated utilized and requested resources obtained by synthetic trace generated using epsilon 3

Figure 42 highlights that the backfill inflation issue remains highly pronounced. Backfilled jobs average 1,301 allocated CPUs with a mean priority of 13,203. Non-Backfilled jobs maintain predictably higher priorities, averaging 31,196. The Wait Time distribution drops significantly at this epsilon, showing a mean of 252 and a median of 0, indicating minimal wait times for the synthetic majority.

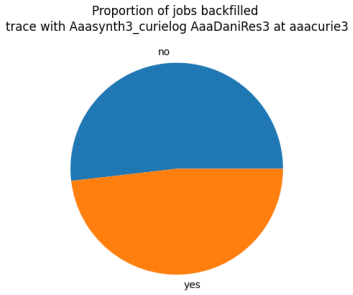


Figure 42: Proportion of backfilled jobs obtained by synthetic trace generated using epsilon 3

Statistic	alloc_cpus	priority
Count	4324	4324
Mean	1301.406105	13203.224561
Std	5595.881775	15390.415448
Min	16	326
25%	16	2715.75
50%	32	6175
75%	400	19876.5
Max	78848	107652

Table 21: Summary statistics for Backfilled (BF) jobs.

Statistic	priority
Count	4654
Mean	31196.599055
Std	30123.857152
Min	326
25%	4222
50%	20726
75%	54222
Max	104392

Table 22: Summary statistics for Non-Backfilled (Non-BF) jobs: Priority.

Statistic	wait_time
Count	8978
Mean	252.845511
Std	3093.357085
Min	0
25%	0
50%	0
75%	31
Max	100951
e	

Table 23: Summary statistics for Wait Time.

H.4 Final Pipeline averaged results of epsilon 100.0

As seen in Table 24, with a massive privacy budget of $\epsilon = 100.0$, the average bound difference decreases when compared with most of the previous epsilon values. However, the Wasserstein distance increases while the TVD distances

remain roughly the same, demonstrating that extremely large ϵ values do not guarantee linear improvements in synthesization. ML utility, DCR metrics, and correlation errors also stagnate, indicating that, interestingly, increasing the budget has not translated in noticeable improvements in Utility, nor in Privacy loss as indicated by the DCR and NNDR metrics also remaining the same, nevertheless, even if the DCR and NNDR metrics seem to indicate privacy, with an ϵ of 100 our privacy guarantees are flimsy. This lackluster results are further compounded by this trace's performance in the SLURM simulation as seen in Figure 43.

Metric	**Value**
Avg Bound Diff	246781.649040
Mean Wasserstein	0.001075
Mean TVD	0.006175
RF Loss	0.025669
LR Loss	0.000112
Discriminator AUC	0.670061
Mean Synth to Train DCR	0.004968
Min Synth to Train DCR	1.3160e-07
Mean 1-DCR	7116.159478
Mean 5-DCR	10460.997253
NNDR	0.787580
Exact clones	0.000000
Spearman Corr Error	0.083097
Linear Corr Error	0.045401
MMD	0.019846
3-Way Marginal Score	0.672200
HOC Score	0.000681

Table 24: Summary of evaluation metrics of $\epsilon = 100$.

Figure 43 shows fewer but significantly wider and taller resource utilization spikes, moving closer to the amplitude of the original data as a result of the relaxed privacy constraint.

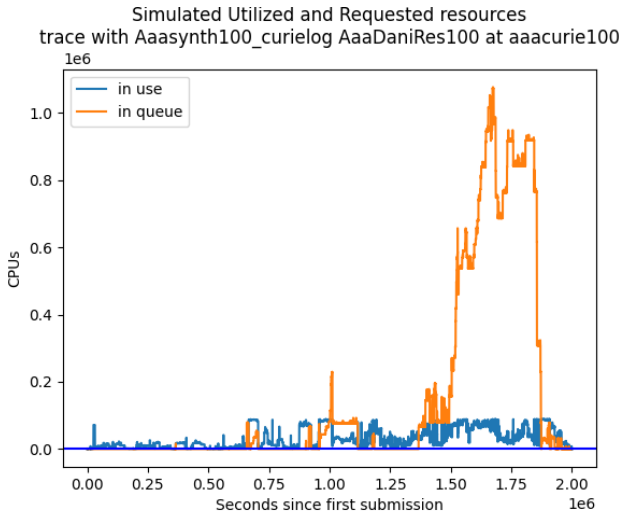


Figure 43: Simulated utilized and requested resources obtained by synthetic trace generated using epsilon 100

As it can be seen in Figure 44, despite the large ϵ value, the backfilling proportion remains much higher than the original trace, pointing to an underlying pipeline artifact rather than a strict DP limitation. Backfilled jobs utilize an average of 1,502 CPUs with a mean priority of 15,405, whereas Non-Backfilled jobs average a priority of 28,477. Wait times remain highly skewed, with a median of 0 but a mean pulled to 552 by extreme upper-bound outliers.

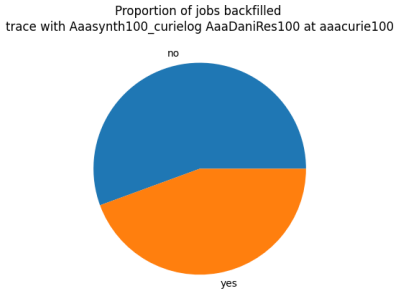


Figure 44: Proportion of backfilled jobs obtained by synthetic trace generated using epsilon 100

Statistic	alloc_cpus	priority
Count	3986	3986
Mean	1502.783743	15405.350728
Std	6847.468997	15074.194528
Min	16	187
25%	16	3161
50%	32	12067
75%	256	18908
Max	77104	100910

Table 25: Summary statistics for Backfilled (BF) jobs.

Statistic	priority
Count	4992
Mean	28477.611979
Std	29045.769082
Min	187
25%	4268
50%	17256
75%	43732
Max	101500

Table 26: Summary statistics for Non-Backfilled (Non-BF) jobs: Priority.

Statistic	wait_time
Count	8978
Mean	552.061595
Std	10958.990817
Min	0
25%	0
50%	0
75%	27
Max	474904

Table 27: Summary statistics for Wait Time.

H.5 Final Pipeline averaged results of epsilon 10000.0

In Table 28 we very clearly observe that the massive increase in ϵ has translated in a very noticeable improvement in the Avg Bound difference. However, both Wasserstein and TVD distances worsened while MMD remains the same as seen far lower epsilon. Furthermore, the correlations errors don't show much

improvements. As for the privacy, as was he case with $\epsilon = 100$, it has mostly remained the same, nevertheless, this privacy guarantees are very flimsy and as seen in Golob et al. [14] MST can be vulnerable against MIA attacks given to big an epsilon value.

What it is interesting to observe, however, is how in our pipeline results do not improve linearly with ϵ but seem to stagnate, to plateau. One of the biggest reasons behind this is that the amount of bins increases with epsilon (at an epsilon of 100 the increase stops) and this pushes MST to inject more noise reducing the improvements of utility given by higher epsilon values.

Metric	**Value**
Avg Bound Diff	53879.020292
Mean Wasserstein	0.001225
Mean TVD	0.006800
RF Loss	0.025189
LR Loss	0.000112
Discriminator AUC	0.643329
Mean Synth to Train DCR	0.005318
Min Synth to Train DCR	9.3058e-08
Mean 1-DCR	7984.223750
Mean 5-DCR	11616.066441
NNDR	0.787831
Exact clones	0.000000
Spearman Corr Error	0.083230
Linear Corr Error	0.046542
MMD	0.020984
3-Way Marginal Score	0.681530
HOC Score	0.000423

Table 28: Summary of evaluation metrics of $\epsilon = 10000.0$.

The resource trace in Figure 45 behaves similarly to the $\epsilon = 100.0$ trace, exhibiting fewer, wider spikes that more closely approximate the amplitude of the original dataset.

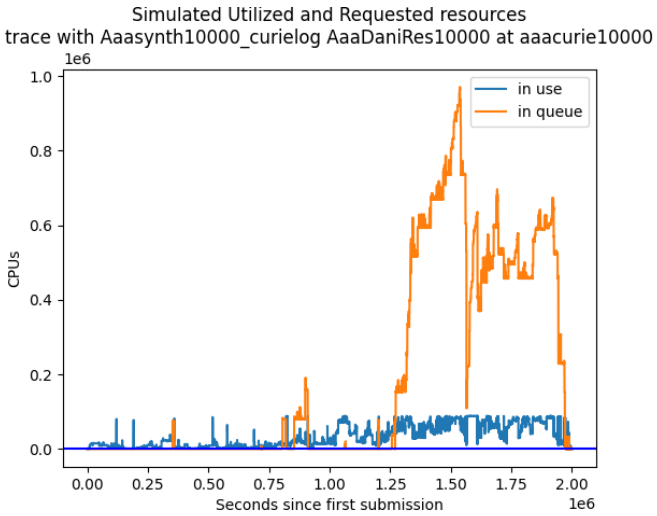


Figure 45: Simulated utilized and requested resources obtained by synthetic trace generated using epsilon 10000

In Figure 46, the backfilling dilemma remains unresolved even at this extreme epsilon. Backfilled jobs show a mean CPU allocation of 1,411 and an average priority of 15,864. Non-Backfilled jobs continue to reflect higher average priorities (30,050). The Wait Time Table displays a mean of 802 and a median of 5, reiterating that certain distributional skews in the synthetic trace are independent of the privacy budget.

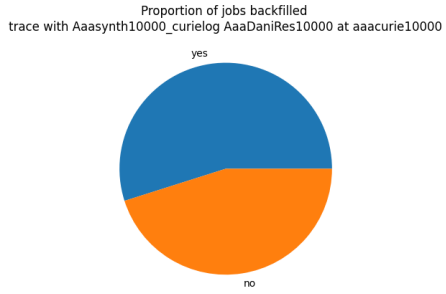


Figure 46: Proportion of backfilled jobs obtained by synthetic trace generated using epsilon 10000

Statistic	alloc_cpus	priority
Count	4939	4939
Mean	1411.333468	15864.338530
Std	6357.080244	15445.002188
Min	16	178
25%	16	2851.5
50%	32	14330
75%	256	20710
Max	79600	115562

Table 29: Summary statistics for Backfilled (BF) jobs.

Statistic	priority
Count	4057
Mean	30050.160217
Std	30609.167759
Min	178
25%	7108
50%	15967
75%	51506
Max	100925

Table 30: Summary statistics for Non-Backfilled (Non-BF) jobs: Priority.

Statistic	wait_time
Count	8996
Mean	802.298133
Std	12803.192334
Min	0
25%	0
50%	5
75%	36
Max	526255

Table 31: Summary statistics for Wait Time.

H.6 ANL-Intrepid statistical averaged results with epsilon 1.0

These results represent the statistical evaluation of the generation pipeline configured with $\epsilon = 1.0$ for the ANL-Intrepid dataset. However, rather than strictly

serving as an evaluation of differential privacy guarantees, this execution functions primarily as a validation of the pipeline's generality and adaptability to external workloads outside of the primary Curie dataset. Notably, feature normalization was omitted during this baseline run to test the pipeline's resilience under raw data conditions.

As seen in Table 32, the metrics indicate highly promising generalizability. While the lack of feature normalization naturally compounds the unscaled feature spaces, resulting in an elevated Mean Wasserstein distance and a high Discriminator AUC (0.990), the pipeline preserves structural distributions remarkably well, as evidenced by a low Total Variation Distance (TVD) of 0.027 and a low Maximum Mean Discrepancy (MMD) of 0.014. Downstream machine learning utility is sustained with low random forest (RF) and linear regression (LR) loss deltas. Furthermore, privacy preservation remains robust: Distance to Closest Record (DCR) values are high, the Nearest Neighbor Distance Ratio (NNDR) sits securely at 0.877, and zero exact clones are generated. Correlation errors are slightly higher than normalized variants but remain well within manageable thresholds.

Metric	**Value**
Avg Bound Diff	180981.053958
Mean Wasserstein	7054.778833
Mean TVD	0.027450
RF Loss	0.067813
LR Loss	-0.063969
Discriminator AUC	0.990270
Mean Synth to Train DCR	13899.694014
Min Synth to Train DCR	96.506378
Mean 1-DCR	14147.215803
Mean 5-DCR	18784.174481
NNDR	0.877573
Exact clones	0.000000
Spearman Corr Error	0.149657
Linear Corr Error	0.088066
MMD	0.014142

Table 32: Evaluation metrics for $\epsilon = 1.0$ on the ANL-Intrepid dataset.

H.7 ANL-Intrepid original dataset slurm results

These scheduling results were obtained by passing a .csv version of the ANL original dataset through the generalized `csvtswf` transformation script to later use for simulation with SLURM. Due to time constraints, this assessment does not aim to provide an evaluation of the performance of synthetic traces generated from ANL. Instead, it serves as a structural proof-of-concept demonstrating that the underlying execution scripts are thoroughly decoupled from specific log

formats and can successfully ingest diverse high-performance computing (HPC) traces.

As illustrated in Figure 47, the resource allocation patterns depart significantly from those observed in the Curie dataset logs. The ANL trace demonstrates a more complex queuing behavior characterized by a higher frequency of queue spikes found in the initial job submission intervals.

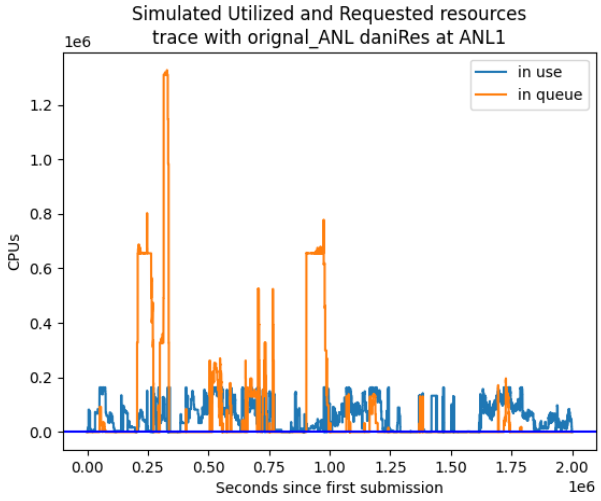


Figure 47: Simulated utilized and requested resources obtained with the ANL-Intrepid dataset.

Additionally, as seen in Figure 48 the underlying scheduler logic under this workload shows an increased propensity for backfilling operations, which is further detailed by the partitioned summary statistics below.

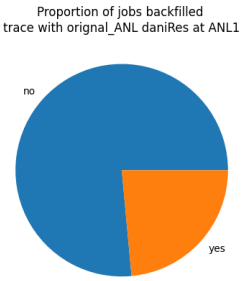


Figure 48: Proportion of backfilled jobs obtained with the ANL-Intrepid dataset.

Statistic	alloc_cpus	priority
Count	878	878
Mean	11033.949886	7381.259681
Std	27763.654858	13129.039885
Min	256	437
25%	2048	2770
50%	2048	4231
75%	8192	7573
Max	163840	116280

Table 33: Summary statistics for Backfilled (BF) jobs (ANL-Intrepid).

Statistic	priority
Count	2853
Mean	8709.296530
Std	12909.182102
Min	437
25%	3453
50%	6344
75%	10564
Max	107578

Table 34: Summary statistics for Non-Backfilled (Non-BF) jobs: Priority (ANL-Intrepid).

Statistic	wait_time
Count	3731
Mean	909.415974
Std	4594.255391
Min	0
25%	0
50%	0
75%	0
Max	82246

Table 35: Summary statistics for Non-Backfilled (Non-BF) jobs: Wait Time (ANL-Intrepid).

H.8 Further variables of the $\epsilon = 5$ evaluation

These are the synthesized vs Real results of the variables not shown in the validation section. First we have the allocated processors variable in Figure 49:

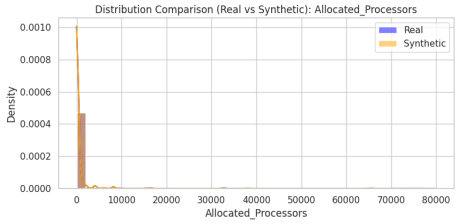


Figure 49: Distribution Comparison of the Real vs Synthetic distribution of the Allocated Processors attribute for the Final Pipeline with $\epsilon = 5$

As we can observe in Figure 49, we have a seemingly perfect synthesis. Now, we have the Status bar chart in Figure 50:

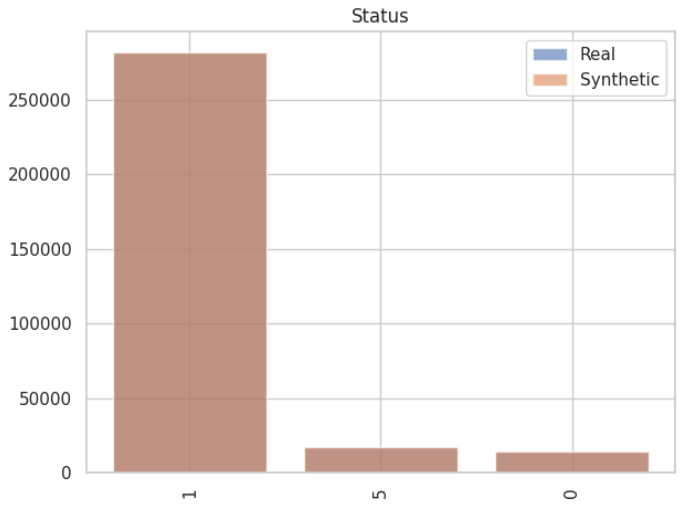


Figure 50: Distribution Comparison of the Real vs Synthetic distribution of the Status attribute for the Final Pipeline with $\epsilon = 5$

As we can observe in Figure 50, we have another seemingly perfect synthesis. Now the last not-shown-in-validation attribute, shown in Figure 51:

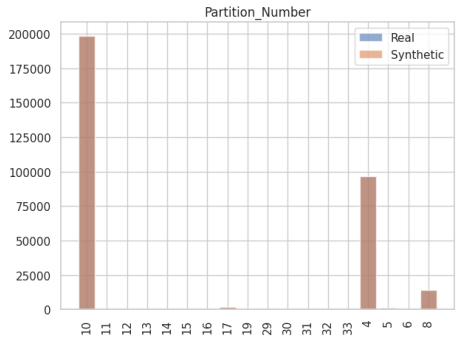


Figure 51: Distribution Comparison of the Real vs Synthetic distribution of the Partition Number attribute for the Final Pipeline with $\epsilon = 5$

As we can observe in Figure 51, we have another seemingly perfect synthesis.

H.9 Further variables of the $\epsilon = 10$ evaluation

These are the synthesized vs Real results of the variables not shown in the validation section. First we have the allocated processors variable in Figure 52:

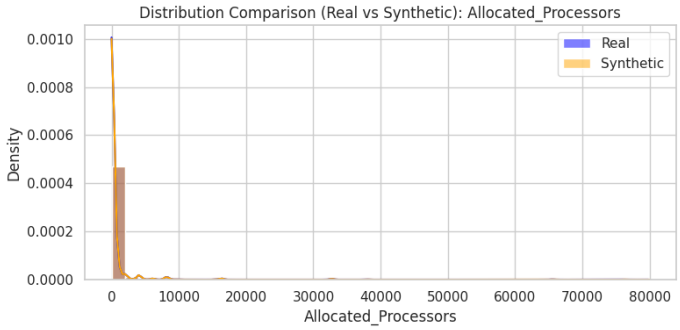


Figure 52: Distribution Comparison of the Real vs Synthetic distribution of the Allocated Processors attribute for the Final Pipeline with $\epsilon = 10$

As we can observe in Figure 52, we have a seemingly perfect synthesis. Now, we have the Status bar chart, shown in Figure 53:

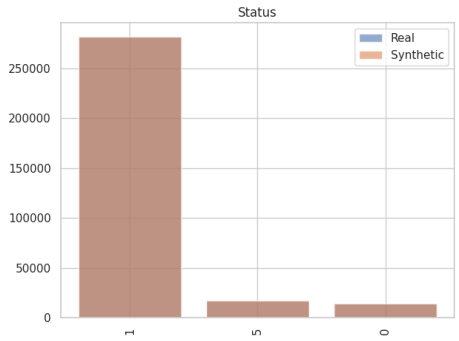


Figure 53: Distribution Comparison of the Real vs Synthetic distribution of the Status attribute for the Final Pipeline with $\epsilon = 10$

As we can observe in Figure 53, we have another seemingly perfect synthesis. Now the last not-shown-in-validation attribute, shown in Figure 54:

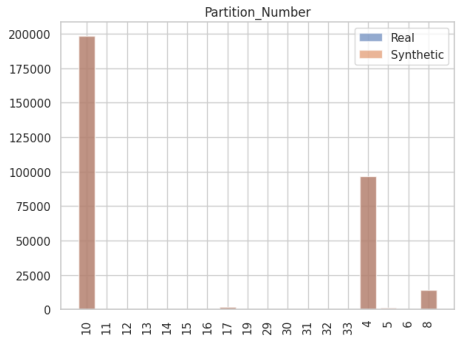


Figure 54: Distribution Comparison of the Real vs Synthetic distribution of the Partition Number attribute for the Final Pipeline with $\epsilon = 10$

As we can observe in Figure 54, we have another seemingly perfect synthesis.

I Code repository

In this section we provide the scripts of the implemented code of the final pipeline and its evaluation scripts.

I.1 generalized_main.py

This is the script that executes step by step the different parts of the pipeline.

```
    #!/usr/bin/env python3
"""
Main pipeline script for HPC synthetic data generation.
↪  Orchestrates metadata
loading, PrivTree discretization, MST synthesis, and SWF format
↪  conversion.

Copyright (C) 2026 Daniel Lopez Garcia

This program is free software: you can redistribute it and/or
↪  modify
it under the terms of the GNU General Public License as published
↪  by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

REFERENCES

[1] Cynthia Dwork. \Differential Privacy". In: Proceedings of the
↪  33rd International Colloquium on Automata, Languages and
↪  Programming (ICALP). Lecture Notes in Computer Science 4052.
↪  Springer, 2006, pp. 1{12. doi: 10.1007/11787006_1. url:
↪  https://doi.org/10.1007/11787006_1.

[2] joseph Emeras. CEA Curie Workload Log. Parallel Workloads
↪  Archive, Hebrew University of Jerusalem. Provided by Joseph
↪  Emeras. Accessed: 2026-05-06. 2012. url: https://www.cs.huji.
↪  ac.il/labs/parallel/workload/l_cea_curie/index.html

[3] Parallel Workloads Archive Workload Logs repository can be
↪  found at
↪  https://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[4] Aleksandra Korolova et al. \Releasing search queries and
↪  clicks privately". In: Proceedings of the 18th International
↪  Conference on World Wide Web (WWW). ACM, 2009, pp. 171{180.
↪  doi: 10.1145/1526709.1526733.

[5] Manuel G. Marciani et al. \Evaluating the impact of task
↪  aggregation in workflows with shared resource environments:
↪  use case for the MONARCH application". In: Geoscientific Model
↪  Development 18 (2025), pp. 9709{9721. doi: 10.5194/gmd- 18-
↪  9709- 2025. url: https://doi.org/10.5194/gmd-18-9709-2025
```

```
[6] Ryan McKenna, Gerome Miklau, and Daniel Sheldon. \Winning the
↪ NIST Contest: A Scalable and General Approach to
↪ Differentially Private Synthetic Data". In: arXiv preprint
↪ arXiv:2108.04978 (2021). url:
↪ https://arxiv.org/abs/2108.04978.

[7] Jun Zhang, Xiaokui Xiao, and Xing Xie. \PrivTree: A
↪ Differentially Private Algorithm for Hierarchical
↪ Decompositions". In: Proceedings of the 2016 International
↪ Conference on Management of Data (SIGMOD). 2016, pp. 155{170.
↪ doi: 10.1145/2882903.2882946. url:
↪ https://doi.org/10.1145/2882903.2882946.

[8] Manuel Giménez de Castro Marciani. pyswf. GitLab Repository.
↪ url: https://gitlab.earth.bsc.es/mgimenez/pyswf.
BSC-CNS - Earth Sciences
Daniel López García
2026
"""
from generalized_preprop import *
from mst_synth_funcs import *
from generalized_csvTswf import *
from metadata_loading import *
# USER INPUT
metadata_path = input("Enter the metadata file's path: ")
df_raw,numerical_cols,categorical_cols,identicat_cols,identifier_
↪ cols,datetimes_cols,dom_know_bounds,_,_,drop_cols,
↪ synthetic_data_path, o_data_path,epsilon,delta,length_t_gen=
↪ load_dataset_from_metadata(metadata_path=metadata_path)
# MAIN LOOP
print(f"\n--- Running pipeline for epsilon={epsilon} ---")
# Returns dictionary with all necessary components
delta_f_preprop=delta*0.5
delta_f_mst=delta*0.5
result, to_remove, identifiers,noisy_counts =
↪ preprocessing_curie(df_raw,numerical_cols,categorical_cols,id
↪ entifier_cols,identicat_cols,datetimes_cols,epsilon,dom_know_
↪ bounds,delta=delta_f_preprop)
# Extract components from dictionary
df_binned = result['binned_data']           # For MST (binned
↪ only)
edges_dict = result['edges_dict']
midpoints_dict = result['midpoints_dict']
num_cols = result['num_cols']
epsilon_generation = result['eps_generation']
bounds_dict = result['bounds_dict']
# SYNTHESIS
```

```
print(f"Using epsilon={epsilon_generation} for MST generation (90%
↪  of total budget)")

# Pass epsilon_generation (not full epsilon) to MST
if length_t_gen:
    synthetic_df =
    ↪  generate_synthetic_data_with_mst(epsilon_generation,
    ↪  df_binned, length_t_gen,delta_f_mst, identifiers)
else:
    synthetic_df =
    ↪  generate_synthetic_data_with_mst(epsilon_generation,
    ↪  df_binned, noisy_counts,delta_f_mst, identifiers)
print('------------------------------------------------------------')
print("Synthetic dataset generated with MST!")
# UNBINNING
print("Unbinning synthetic data using uniform sampling...")

_synthetic_df = uniform_debinning(
    synthetic_binned_df=synthetic_df,
    edges_dict=edges_dict,
    num_cols=num_cols
)
final_synthetic_df = _synthetic_df.copy()
print("Unbinning complete!")
def remake(df,to_remove,identifiers):
    for col in to_remove:
        df[col] = pd.Series(-1, index=df.index, dtype='Int64')
    for col in identifiers:
        x=np.arange(1,len(df)+1)
        df[col]=np.random.permutation(x)
    return df
to_remove=to_remove+drop_cols
df = remake(final_synthetic_df, to_remove, identifiers)
first_part_synthetic_data_path = synthetic_data_path.rsplit('.',
↪  1)[0]
second_part_synthetic_data_path = synthetic_data_path.rsplit('.',
↪  1)[-1]
synthetic_data_path =
↪  f"{first_part_synthetic_data_path}_{epsilon}.csv"
df.to_csv(synthetic_data_path, index=False)
# CONVERSION TO SWF
print(f"Converting to swf...")
convert_to_swf=input('Is conversion to swf desired?(yes/no)')
if convert_to_swf=='yes':
    print(df.columns)
    convert_dataset_to_swf(
        input_path=synthetic_data_path,
        output_path=f"{first_part_synthetic_data_path}_{epsilon}.
        ↪  swf",
```

```
        original_reference_path=o_data_path
    )

    print("Conversion complete!")
    print(f"Synthetic dataset saved as '{synthetic_data_path}'")
    print(f"SWF dataset saved as
    ↪  '{first_part_synthetic_data_path}_{epsilon}.swf'")
else:
    print('Everything is finisehd, synthetic dataset is ',
    ↪  synthetic_data_path)
```

Figure 55: Script of the orchestrator of the pipeline

I.2 metadata_loading.py

This is the script where the metadata is read the dataset loaded and all the metadata's information gathered.

```
#!/usr/bin/env python3
"""
Loads and processes configuration schemas and JSON metadata
↪  definitions
for the target dataset, establishing domain bounds, types, and
↪  constraints.

Copyright (C) 2026 Daniel López García

This program is free software: you can redistribute it and/or
↪  modify
it under the terms of the GNU General Public License as published
↪  by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see
↪  <http://www.gnu.org/licenses/>.

REFERENCES
[1] Parallel Workloads Archive Workload Logs repository can be
↪  found at
↪  https://www.cs.huji.ac.il/labs/parallel/workload/logs.html.
```

```
[2] joseph Emeras. CEA Curie Workload Log. Parallel Workloads
↪  Archive, Hebrew University of Jerusalem. Provided by Joseph
↪  Emeras. Accessed: 2026-05-06. 2012. url: https://www.cs.huji.
↪  ac.il/labs/parallel/workload/l_cea_curie/index.html

BSC-CNS - Earth Sciences
Daniel López García
2026
"""
import json
import pandas as pd
import csv

def load_metadata(metadata_path: str) -> tuple:
    """
    Loads a JSON metadata file with variable definitions.
        Returns: (data_path, variables_list)
        Each variable dict contains: name, type, bounds
        ↪  (optional), wanted (optional), granularity (optional)
    Valid types: numerical, categorical, identicat, identifier,
    ↪  string, datetime
    """
    with open(metadata_path, 'r') as f:
        metadata = json.load(f)
    data_path = metadata['data_path']
    variables = metadata['variables']
    synthetic_data_path = metadata['synthetic_data_path']
    eps=metadata['epsilon']
    delta=metadata['delta']
    length_t_gen=metadata['length_synth_data']
    return data_path, variables, synthetic_data_path, eps,
    ↪  delta,length_t_gen


def load_dataset_from_metadata(metadata_path: str) -> tuple:
    """
    Loads any dataset (CSV, Parquet, SWF) and returns:
      - df                  : DataFrame with dropped columns
      ↪  removed
      - numerical_cols      : numerical columns to synthesize
      - categorical_cols    : categorical columns to synthesize
      - identicat_cols      : identicat columns to synthesize
      - identifier_cols     : identifier columns not dropped
      - datetime_cols       : all datetime columns (any wanted
      ↪  status) present in df
      - bounds_specs        : dict col -> {"min": ..., "max": ...}
      ↪  or None
      - preserve_cols       : all columns marked "preserve"
```

```
  - wanted_map          : dict col -> wanted status
  ↪  ("synthesize","preserve","drop")
  - drop_cols           : columns marked "drop"
"""
data_path, variables, synthetic_data_path, epsilon,
↪  delta,length_t_gen = load_metadata(metadata_path)

col_names = [v['name'] for v in variables]
col_type_map = {}
bounds_specs = {}
wanted_map = {}
allowed_types = {'numerical', 'categorical', 'identicat',
↪  'identifier', 'string', 'datetime'}

for v in variables:
    col = v['name']
    t = v['type']
    if t not in allowed_types:
        print(f"Warning: unknown type '{t}' for column '{col}'
        ↪  { treating as 'string'")
        t = 'string'
    col_type_map[col] = t
    wanted_map[col] = v.get('wanted', 'synthesize')

    b = v.get('bounds')
    if b and isinstance(b, dict) and 'min' in b and 'max' in
    ↪  b:
        bounds_specs[col] = {'min': b['min'], 'max': b['max']}
    else:
        bounds_specs[col] = None

ext = data_path.rsplit('.', 1)[-1].lower() if '.' in data_path
↪  else ''

if ext == 'swf':
    rows = []
    num_expected = len(col_names)
    with open(data_path, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith(';'):
                continue
            fields = line.split()
            if len(fields) < num_expected:
                fields += [''] * (num_expected - len(fields))
            elif len(fields) > num_expected:
                fields = fields[:num_expected]
            row = dict(zip(col_names, fields))
            rows.append(row)
```

```
    df = pd.DataFrame(rows, columns=col_names)
    for col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='ignore')
elif ext == 'csv':
    df = pd.read_csv(data_path)
elif ext == 'parquet':
    df = pd.read_parquet(data_path)
else:
    df = pd.read_csv(data_path)

drop_cols = [c for c in col_names if wanted_map.get(c) ==
↪  'drop' and c in df.columns]
df.drop(columns=drop_cols, inplace=True, errors='ignore')

preserve_cols = [c for c in col_names if wanted_map.get(c) ==
↪  'preserve' and c in df.columns]
synthesize_cols = [c for c in col_names if wanted_map.get(c)
↪  == 'synthesize' and c in df.columns]

final_cols = [c for c in col_names if c in df.columns]
df = df[final_cols]

numerical_cols   = [c for c in synthesize_cols if
↪  col_type_map.get(c) == 'numerical']
categorical_cols = [c for c in synthesize_cols if
↪  col_type_map.get(c) == 'categorical']
identicat_cols   = [c for c in synthesize_cols if
↪  col_type_map.get(c) == 'identicat']

identifier_cols  = [c for c in df.columns if
↪  col_type_map.get(c) in ('identifier', 'string')
                    and wanted_map.get(c) != 'drop']

datetime_cols    = [c for c in df.columns if
↪  col_type_map.get(c) == 'datetime']

print(f"Loaded data from {data_path}")
print(f"  Columns to SYNTHESIZE : {synthesize_cols}")
print(f"    Numerical   : {numerical_cols}")
if numerical_cols:
    print(f"      Bounds:")
    for c in numerical_cols:
        b = bounds_specs.get(c)
        if b:
            print(f"        {c}: {b}")
        else:
            print(f"        {c}: no bounds specified")
print(f"    Categorical : {categorical_cols}")
print(f"    Identicat   : {identicat_cols}   ← hybrid")
```

```
    print(f"    Datetime    : {datetime_cols}")
    print(f"  Columns to PRESERVE   : {preserve_cols}")
    print(f"  Columns DROPPED       : {drop_cols}")
    print(f"  Identifier columns    : {identifier_cols}")

    return df, numerical_cols, categorical_cols,
    ↪  identicat_cols,identifier_cols, datetime_cols,
    ↪  bounds_specs, preserve_cols,wanted_map, drop_cols,
    ↪  synthetic_data_path,data_path, epsilon, delta,length_t_gen
```

Figure 56: Script of the algorithm that loads the data from the given metadata path

I.3 generalized_preprop

This is the script centralizing all the preprocessing steps.

```
#!/usr/bin/env python3
"""
Handles data preprocessing for synthetic workload generation,
↪  specifically
implementing PrivTree binning for data discretization and
↪  calculating
theoretical privacy thresholds.

Copyright (C) 2026 Daniel López García

This program is free software: you can redistribute it and/or
↪  modify
it under the terms of the GNU General Public License as published
↪  by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see
↪  <http://www.gnu.org/licenses/>.

REFERENCES
```

```
[1] Cynthia Dwork. \Differential Privacy". In: Proceedings of the
↪  33rd International Colloquium on Automata, Languages and
↪  Programming (ICALP). Lecture Notes in Computer Science 4052.
↪  Springer, 2006, pp. 1{12. doi: 10.1007/11787006_1. url:
↪  https://doi.org/10.1007/11787006_1.

[2] Aleksandra Korolova et al. \Releasing search queries and
↪  clicks privately". In: Proceedings of the 18th International
↪  Conference on World Wide Web (WWW). ACM, 2009, pp. 171{180.
↪  doi: 10.1145/1526709.1526733.

[3] Jun Zhang, Xiaokui Xiao, and Xing Xie. \PrivTree: A
↪  Differentially Private Algorithm for Hierarchical
↪  Decompositions". In: Proceedings of the 2016 International
↪  Conference on Management of Data (SIGMOD). 2016, pp. 155{170.
↪  doi: 10.1145/2882903.2882946. url:
↪  https://doi.org/10.1145/2882903.2882946.

BSC-CNS - Earth Sciences
Daniel López García
2026
"""
import pandas as pd
import numpy as np
import random
import math

def calculate_korolova_threshold(eps_col, delta):
    """
    Calculates the exact theoretical threshold K from Korolova et
    ↪  al.
    for a user contributing at most 1 record (d=1).
    """
    b = 1.0 / eps_col
    K = 1.0 - (b * math.log(2.0 * delta))
    return K

class PrivTreeNode:
    def __init__(self, low, high, depth=0):
        self.low = low
        self.high = high
        self.depth = depth
        self.left = None
        self.right = None
        self.is_leaf = True

def privtree_binning(values, low_bound, high_bound, epsilon,
↪  noisy_n, theta=0.0):
    clean_values = np.asarray(values)
```

```
    eps_tree = epsilon

    # Enforce Square Root Bound Ceiling
    max_bins = np.sqrt(noisy_n)
    max_depth = math.ceil(np.ceil(np.log2(max_bins)))
    print(f"[PrivTree-DP] Noisy Row Count: {noisy_n:.1f} | Square
    ↪  Root Bin Ceiling: {max_bins:.1f} -> Max Tree Depth:
    ↪  {max_depth}")

    beta = 2
    lam = ((2 * beta - 1) / (beta - 1)) * 1.0 / eps_tree
    delta = lam * np.log(beta)

    root = PrivTreeNode(low_bound, high_bound, depth=0)
    queue = [root]
    leaves = []

    while queue:
        node = queue.pop(0)
        mask = (clean_values >= node.low) & (clean_values <=
        ↪  node.high)
        c_v = np.sum(mask)

        b_v = max(theta - delta, c_v - node.depth * delta)
        noisy_b = b_v + np.random.laplace(0, lam)

        if noisy_b > theta and node.depth <= max_depth:
            mid = (node.low + node.high) / 2.0
            node.is_leaf = False
            node.left = PrivTreeNode(node.low, mid, node.depth +
            ↪  1)
            node.right = PrivTreeNode(mid, node.high, node.depth +
            ↪  1)
            queue.append(node.left)
            queue.append(node.right)
        else:
            leaves.append(node)

    leaves = sorted(leaves, key=lambda x: x.low)
    edges = [leaves[0].low]
    for leaf in leaves:
        edges.append(leaf.high)

    edges = np.array(edges)
    edges = np.maximum.accumulate(edges)

    for i in range(1, len(edges)):
        if edges[i] <= edges[i - 1]:
            edges[i] = edges[i - 1] + 1e-9
```

```
    midpoints = (edges[:-1] + edges[1:]) / 2.0
    print(f"[PrivTree-DP] Generated {len(leaves)} bins")
    return edges, midpoints

def preprocessing_curie(df, numerical, categorical, identifiers,
↪  identicat, datetimes, user_eps, bounds, delta=10e-5,
↪  use_log_transform=True):
    df = df.copy()
    print(f"use_log_transform: {use_log_transform}")
    df.replace([-1, '-1'], np.nan, inplace=True)
    for c in categorical + identifiers + identicat:
        if c in df.columns:
            df[c] = df[c].fillna('Missing').astype(str)
    for c in numerical:
        if c in df.columns:
            df[c] = df[c].fillna(0.0)

    initial_num_cols = max(1, len(numerical))
    initial_identicat_cols = max(1, len(identicat))
    total_initial_cols = max(1, len(numerical) + len(categorical)
    ↪  + len(identifiers) + len(identicat) + len(datetimes))

    # 1. SECURE GLOBAL BUDGET ALLOCATION
    eps_count          = user_eps * 0.02
    eps_schema_prune   = user_eps * 0.03
    eps_pseudonymize   = user_eps * 0.05
    eps_discretization = user_eps * 0.20
    eps_generation     = user_eps * 0.70

    eps_per_prune      = eps_schema_prune / total_initial_cols
    eps_per_pseudo     = eps_pseudonymize / initial_identicat_cols
    eps_per_col_tree   = (eps_discretization) / initial_num_cols
    delta_pseudo       = delta
    delta_per_pseudo   = delta_pseudo / initial_identicat_cols

    print(f"Total epsilon: {user_eps}")
    print(f"  - Count / Pruning / Pseudo eps: {eps_count +
    ↪  eps_schema_prune + eps_pseudonymize:.4f}")
    print(f"  - Discretization epsilon: {eps_discretization:.4f}")
    print(f"  - MST Generation epsilon: {eps_generation:.4f}")

    # 2. GLOBAL NOISY COUNT
    n_raw = len(df)
    scale_count = 1.0 / eps_count
    noisy_count = max(1.0, n_raw + np.random.laplace(0,
    ↪  scale_count))

    # 3. DP SCHEMA PRUNING
```

```
to_remove, new_numerical, new_categorical, new_identifier,
↪  new_new_identicat, new_datetimes = [], [], [], [], [], []

def dp_should_drop(series, eps_prune, noisy_n, fraction=0.5):
    missing = series.isna().sum() + (series ==
    ↪  'Missing').sum() + (series == 0.0).sum()
    noisy_missing = missing + np.random.laplace(0, 1.0 /
    ↪  eps_prune)
    return noisy_missing > (noisy_n * fraction)

for c in numerical:
    if dp_should_drop(df[c], eps_per_prune, noisy_count,
    ↪  fraction=0.5):
        to_remove.append(c)
    else:
        new_numerical.append(c)

for c in categorical:
    if dp_should_drop(df[c], eps_per_prune, noisy_count,
    ↪  fraction=0.5):
        to_remove.append(c)
    else:
        new_categorical.append(c)

for c in identifiers:
    if dp_should_drop(df[c], eps_per_prune, noisy_count,
    ↪  fraction=0.5):
        to_remove.append(c)
    else:
        new_identifier.append(c)

for c in identicat:
    if dp_should_drop(df[c], eps_per_prune, noisy_count,
    ↪  fraction=0.5):
        to_remove.append(c)
    else:
        new_new_identicat.append(c)

for c in datetimes:
    if dp_should_drop(df[c], eps_per_prune, noisy_count,
    ↪  fraction=0.5):
        to_remove.append(c)
    else:
        new_datetimes.append(c)

df = df.drop(columns=to_remove)

numerical = new_numerical.copy()
categorical = new_categorical.copy()
```

```
identifiers = new_identifier.copy()
identicat = new_new_identicat.copy()
datetimes = new_datetimes.copy()

df = df.drop(columns=[c for c in identifiers if c in
↪  df.columns], errors='ignore')
num_cols = numerical

for col in datetimes:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col],
        ↪  errors='coerce').astype('int64') / 1e9

# 4. DP PSEUDONYMIZATION
def dp_pseudonymize(df, col, eps_col, delta_col):
    val_counts = df[col].value_counts()
    vals = val_counts.index.values
    counts = val_counts.values

    noisy_counts = counts + np.random.laplace(0, 1.0 /
    ↪  eps_col, size=len(counts))

    threshold = calculate_korolova_threshold(eps_col,
    ↪  delta_col)
    valid_mask = noisy_counts >= threshold
    valid_vals = vals[valid_mask]

    num_unique = len(valid_vals)
    max_range = max(1000, num_unique * 2)
    id_pool = np.arange(1, max_range)
    np.random.shuffle(id_pool)

    mapping = dict(zip(valid_vals, id_pool[:num_unique]))
    df[col] = df[col].map(mapping).fillna(0).astype(int)
    return df

for var in identicat:
    if var in df.columns:
        df = dp_pseudonymize(df, var, eps_per_pseudo,
        ↪  delta_per_pseudo)

original_data = df.copy()

# 5. PRIVTREE DISCRETIZATION (WITH LOG-TRANSFORM)
bounds_dict = {}
print(f"\n--- Binning (PrivTree | Log-Space:
↪  {use_log_transform}) ---")
for col in num_cols:
    col_bounds = bounds.get(col)
```

```
        low = col_bounds['min']
        high = col_bounds['max']
        bounds_dict[col] = (low, high)

    df_binned = df.copy()
    edges_dict = {}
    midpoints_dict = {}

    for col in num_cols:
        low, high = bounds_dict[col]

        df[col] = np.clip(df[col].values, low, high)

        if use_log_transform:
            if low >= 0:
                to_log = lambda x: np.log10(x + 1.0)
                to_lin = lambda x: 10.0**x - 1.0
            else:
                shift = abs(low) + 1.0
                to_log = lambda x: np.log10(x + shift)
                to_lin = lambda x: 10.0**x - shift
        else:
            to_log = lambda x: x
            to_lin = lambda x: x

        eval_values = to_log(df[col].values)
        eval_low = to_log(low)
        eval_high = to_log(high)

        eval_edges, eval_midpoints = privtree_binning(
            eval_values,
            low_bound=eval_low,
            high_bound=eval_high,
            epsilon=eps_per_col_tree,
            noisy_n=noisy_count,
        )

        edges = to_lin(eval_edges)
        midpoints = to_lin(eval_midpoints)

        edges[0] = low
        edges[-1] = high

        edges_dict[col] = edges
        midpoints_dict[col] = midpoints

        binned_series = pd.cut(
            df[col],
            bins=edges,
```

```
            labels=list(range(len(edges) - 1)),
            include_lowest=True
        )

        nan_count = binned_series.isna().sum()
        if nan_count > 0:
            print(f"WARNING: [{col}] {nan_count} values outside
            ↪  bins. Assigning to nearest bin.")

        df_binned[f"{col}_binned"] =
        ↪  binned_series.fillna(0).astype(int)

    df_binned = df_binned.drop(columns=num_cols)

    return {
        'binned_data': df_binned,
        'edges_dict': edges_dict,
        'midpoints_dict': midpoints_dict,
        'num_cols': num_cols,
        'bounds_dict': bounds_dict,
        'eps_generation': eps_generation
    }, to_remove, identifiers, noisy_count
```

Figure 57: Script of the preprocessing step

I.4 mst_synth_funcs.py

This is the script centralizing some functions, including the one for the MST synthesizer.

```
#!/usr/bin/env python3
"""
Applies the Maximum Spanning Tree (MST) synthesizer to binned
↪  datasets
for differentially private synthetic data generation and debinning
↪  mapping.

Copyright (C) 2026 Daniel López García

This program is free software: you can redistribute it and/or
↪  modify
it under the terms of the GNU General Public License as published
↪  by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
```

```
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see
↪  <http://www.gnu.org/licenses/>.

REFERENCES

[1] Cynthia Dwork. \Differential Privacy". In: Proceedings of the
↪  33rd International Colloquium on Automata, Languages and
↪  Programming (ICALP). Lecture Notes in Computer Science 4052.
↪  Springer, 2006, pp. 1{12. doi: 10.1007/11787006_1. url:
↪  https://doi.org/10.1007/11787006_1.

[2] Ryan McKenna, Gerome Miklau, and Daniel Sheldon. \Winning the
↪  NIST Contest: A Scalable and General Approach to
↪  Differentially Private Synthetic Data". In: arXiv preprint
↪  arXiv:2108.04978 (2021). url:
↪  https://arxiv.org/abs/2108.04978.

BSC-CNS - Earth Sciences
Daniel López García
2026
"""
import pandas as pd
import numpy as np
from snsynth.mst import MSTSynthesizer
# Standard library
import sys
sys.path.append("..")
import random
def generate_synthetic_data_with_mst(user_eps, df_binned,
↪  noisy_counts,delta, identifiers=['Job_Number']):
    """
    This function takes the fully preprocessed and binned dataset,
    ↪  applies the MST synthesizer
    with the specified epsilon for differential privacy, and
    ↪  generates a synthetic dataset.
    It then un-bins the synthetic data with DCR protection and
    ↪  returns the final synthetic dataframe.
    """
    df = df_binned.copy()

    cols_to_drop = identifiers
    df_train = df.drop(columns=[c for c in cols_to_drop if c in
    ↪  df.columns])
    cat_cols = df_train.columns.tolist()
```

```
    df_train[cat_cols] = df_train[cat_cols].astype(str)
    num_cols = []  # No continuous columns for MST, as we are
    ↪  feeding it binned data

    synth = MSTSynthesizer(epsilon=user_eps,delta=delta)

    print("Fitting the MST model (This may take a moment to
    ↪  calculate marginals)...")


    synth.fit(
        df_train,
        categorical_columns=cat_cols,
        continuous_columns=num_cols,
    )

    print("Generating synthetic data...")

    synthetic_df = synth.sample(int(noisy_counts))
    print("Synthesis Complete!")
    return synthetic_df

def uniform_debinning(synthetic_binned_df, edges_dict, num_cols):
    df_continuous = synthetic_binned_df.copy()

    for col in num_cols:
        binned_col = f"{col}_binned"
        if binned_col not in df_continuous.columns:
            continue

        edges = edges_dict[col]
        idx = df_continuous[binned_col].astype(int).values

        # Clamp indices to valid range
        idx = np.clip(idx, 0, len(edges) - 2)

        lows = edges[idx]
        highs = edges[idx + 1]

        sampled_values = np.random.uniform(lows, highs)

        df_continuous[col] = sampled_values
        df_continuous.drop(columns=[binned_col], inplace=True)

    return df_continuous
def remake_data(df,to_remove):
    for col in to_remove:
        df[col] = pd.Series(-1, index=df.index, dtype='Int64')
```

```
    df['job_id']=np.random.randint(100000, 999999, size=len(df))
    return df
```

Figure 58: Script implementing varius functions, among them MST

I.5 generalized_csvTswf.py

This script transforms the .CSV results of the MST synthesizer into the format required by slurm.

```
#!/usr/bin/env python3
"""
Handles mapping, parsing, and data format standardization to
↪  transform between
CSV/Parquet payloads and standard HPC Standard Workload Formats
↪  (SWF).

Copyright (C) 2026 Daniel López García

This program is free software: you can redistribute it and/or
↪  modify
it under the terms of the GNU General Public License as published
↪  by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see
↪  <http://www.gnu.org/licenses/>.

REFERENCES

[1] Parallel Workloads Archive Workload Logs repository can be
↪  found at
↪  https://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[2] joseph Emeras. CEA Curie Workload Log. Parallel Workloads
↪  Archive, Hebrew University of Jerusalem. Provided by Joseph
↪  Emeras. Accessed: 2026-05-06. 2012. url: https://www.cs.huji.
↪  ac.il/labs/parallel/workload/l_cea_curie/index.html
```

```
[4] Manuel G. Marciani et al. \Evaluating the impact of task
↪  aggregation in workflows with shared resource environments:
↪  use case for the MONARCH application". In: Geoscientific Model
↪  Development 18 (2025), pp. 9709{9721. doi: 10.5194/gmd- 18-
↪  9709- 2025. url: https://doi.org/10.5194/gmd-18-9709-2025

[5] Manuel Giménez de Castro Marciani. pyswf. GitLab Repository.
↪  url: https://gitlab.earth.bsc.es/mgimenez/pyswf.

BSC-CNS - Earth Sciences
Daniel López García
2026
"""
import os
import pandas as pd
from pyswf import SWFtrace

REQUIRED_TARGETS = [
    'status', 'uid', 'gid', 'partition_id', 'submit_time',
    ↪  'wait_time',
    'runtime', 'alloc_cpus', 'requested_cpus', 'requested_time',
    'avg_cpu_used', 'used_time', 'requested_memory', 'qos_id',
    'preceding', 'thinktime', 'job_id'
]

def load_dataframe(file_path: str) -> pd.DataFrame:
    """
    Safely loads a dataframe supporting both CSV and Parquet
    ↪  extensions.
    """
    ext = file_path.rsplit('.', 1)[-1].lower() if '.' in file_path
    ↪  else ''
    if ext == 'parquet':
        return pd.read_parquet(file_path)
    else:
        return pd.read_csv(file_path)

def normalize_and_rename_columns(df: pd.DataFrame) ->
↪  pd.DataFrame:
    """
    Normalizes column variations, runs the strict mapping rules,
    and forces any completely missing target variable to be
    ↪  created with -1.
    """
    df.columns = df.columns.str.lower().str.replace(' ',
    ↪  '_').str.replace('-', '_')

    # 2. Map known naming formats to your strict targets
    strict_rename_map = {
```

```
        'status': 'status',
        'user_id': 'uid', 'uid': 'uid',
        'group_id': 'gid', 'gid': 'gid',
        'partition_number': 'partition_id', 'partition_id':
        ↪  'partition_id',
        'submit_time': 'submit_time',
        'wait_time': 'wait_time',
        'run_time': 'runtime', 'runtime': 'runtime',
        'allocated_processors': 'alloc_cpus', 'allocated_cpus':
        ↪  'alloc_cpus', 'alloc_cpus': 'alloc_cpus',
        'requested_processors': 'requested_cpus',
        ↪  'requested_cpus': 'requested_cpus',
        'requested_time': 'requested_time',
        'average_cpu_time': 'avg_cpu_used', 'avg_cpu_time':
        ↪  'avg_cpu_used', 'avg_cpu_used': 'avg_cpu_used',
        'used_memory': 'used_time', 'used_time': 'used_time',
        'requested_memory': 'requested_memory',
        'queue_number': 'qos_id', 'queue_id': 'qos_id', 'qos_id':
        ↪  'qos_id',
        'preceding_job_number': 'preceding', 'preceding':
        ↪  'preceding',
        'think_time': 'thinktime', 'thinktime': 'thinktime',
        'job_number': 'job_id', 'job_id': 'job_id'
    }

    rename_payload = {col: strict_rename_map[col] for col in
    ↪  df.columns if col in strict_rename_map}
    df = df.rename(columns=rename_payload)

    # 3. Check for missing columns and fill them with -1
    for target in REQUIRED_TARGETS:
        if target not in df.columns:
            df[target] = -1

    return df

def extract_system_metadata(reference_path: str) -> dict:
    """
    Extracts structural configuration metadata from an original
    ↪  SWF file header
    or derives cluster limits automatically from an original
    ↪  CSV/Parquet trace.
    """
    meta = {}
    if not reference_path or not os.path.isfile(reference_path):
        return meta

    ext = reference_path.rsplit('.', 1)[-1].lower() if '.' in
    ↪  reference_path else ''
```

```
    if ext == 'swf':
        with open(reference_path, 'r', errors='ignore') as f:
            for line in f:
                if not line.startswith(';'):
                    break
                line = line[1:].strip()
                if ':' in line:
                    key, val = line.split(':', 1)
                    meta[key.strip().lower()] = val.strip()

    elif ext in ['csv', 'parquet']:
        try:
            df_orig = load_dataframe(reference_path)
            df_orig = normalize_and_rename_columns(df_orig)

            meta['maxjobs'] = str(len(df_orig))
            meta['maxrecords'] = str(len(df_orig))

            max_procs = 1
            if 'alloc_cpus' in df_orig.columns:
                val = pd.to_numeric(df_orig['alloc_cpus'],
                ↪  errors='coerce').max()
                if not pd.isna(val) and val > max_procs:
                    max_procs = val
            if 'requested_cpus' in df_orig.columns:
                val = pd.to_numeric(df_orig['requested_cpus'],
                ↪  errors='coerce').max()
                if not pd.isna(val) and val > max_procs:
                    max_procs = val

            meta['maxprocs'] = str(int(max_procs))
            meta['maxnodes'] = '1'
            meta['unixstarttime'] = '0'
        except Exception as e:
            print(f"Warning: Failed to auto-extract limits from
            ↪  reference data {reference_path}: {e}")

    return meta

def convert_dataset_to_swf(input_path: str, output_path: str,
↪  original_reference_path: str = None):
    """
    Converts a single target dataset (CSV or Parquet) back to an
    ↪  SWF trace file.
    """
    if not os.path.isfile(input_path):
        print(f"Error: The input file '{input_path}' was not
        ↪  found. Skipping.")
```

```
        return

    df = load_dataframe(input_path)

    df = normalize_and_rename_columns(df)

    cat_vars = ['status', 'uid', 'gid', 'partition_id', 'job_id']
    num_vars = [c for c in REQUIRED_TARGETS if c not in cat_vars]

    for col in cat_vars:
        df[col] = df[col].astype(str).str.split('.').str[0].astyp
        ↪  e('string')

    for col in num_vars:
        df[col] = pd.to_numeric(df[col],
        ↪  errors='coerce').fillna(-1).astype(int)

    df['submit_time'] = df['submit_time'] -
    ↪  df['submit_time'].min()
    df = df.sort_values(by='submit_time').reset_index(drop=True)

    df['job_id'] = (df.index + 1).astype('string')

    df = df[REQUIRED_TARGETS]

    system_meta = extract_system_metadata(original_reference_path)

    max_jobs = int(system_meta.get('maxjobs', len(df)))
    max_records = int(system_meta.get('maxrecords', len(df)))
    max_nodes = int(system_meta.get('maxnodes', 1))
    max_procs = int(system_meta.get('maxprocs', 1))
    unix_start = int(system_meta.get('unixstarttime', 0))

    swf_conversor = SWFtrace(
        swf_version=2.2,
        preemption=system_meta.get('preemption', 'No'),
        max_runtime='suda',
        max_memory='suda',
        queue=1,
        queues=[1],
        allow_overuse='suda',
        max_jobs=max_jobs,
        max_records=max_records,
        max_nodes=max_nodes,
        max_procs=max_procs,
        job_list=df,
        computer_name=system_meta.get('computer',
        ↪  'unknown_cluster'),
        location=system_meta.get('location', 'Unknown'),
```

```
        acknowledge=system_meta.get('acknowledge', 'Anonymous'),
        conversion='pd-2-swf generalized script',
        unix_start_time=unix_start,
        time_zone_string=system_meta.get('timezonestring', 'UTC'),
        start_time=system_meta.get('starttime', ''),
        end_time=system_meta.get('endtime', ''),
        max_queues=1,
        max_partitions=33,
        partition='suda',
        partitions='suda',
        note=system_meta.get('note', 'Converted workload data
        ↪  package'),
        file_location=output_path
    )

    swf_conversor.to_swf(output_path)
    print(f"Successfully converted and saved to: {output_path}")

def batch_convert_datasets(datasets_config: list):
    """
    Iterates over a batch processing configuration array.
    """
    for index, config in enumerate(datasets_config):
        print(f"\nProcessing trace pipeline [{index +
        ↪  1}/{len(datasets_config)}]...")
        convert_dataset_to_swf(
            input_path=config.get('input_path'),
            output_path=config.get('output_path'),
            original_reference_path=config.get('original_referenc
            ↪  e_path')
        )
```

Figure 59: Script that converts a .csv to a .swf

Additionally, this script requires the use of pyswf.py by Manuel Marciani [27].

I.6 elbow_method.py

This script performs the elbow method given a series of previously generated synthetic datasets.

```
from synth_evaluation_script import *
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

```
epsilon_to_test = [1.0, 3.0, 5.0, 8.0, 10.0, 100.0]  # Add more
↪  epsilon values as needed
eps_results_dict = {}

for eps in epsilon_to_test:
    print(f"Evaluation for epsilon={eps}:")
    real_df = pd.read_csv(f"/home/dlope2/Documents/Work/final_pip
    ↪  eline/original_binned_{eps}.csv")
    synth_df = pd.read_csv(f"/home/dlope2/Documents/Work/final_pi
    ↪  peline/curie_synthetic_df_binned_{eps}.csv")
    evaluator = DPEvaluator(real_data=real_df,
    ↪  synth_data=synth_df[real_df.columns])  # Ensure columns
    ↪  are aligned

    print("Calculating 3-Way Marginals...")
    # Limiting combinations is recommended for datasets with many
    ↪  columns (18 columns = 816 possible 3-way combinations)
    k_marginal_score = evaluator.evaluate_k_marginals(k=3,
    ↪  num_combinations=50)
    print(f"3-Way Marginal Score: {k_marginal_score:.4f}")

    print("Calculating HOC...")
    hoc_score = evaluator.evaluate_hoc(num_queries=100,
    ↪  max_features_per_query=5)
    print(f"HOC Score: {hoc_score:.4f}")

    eps_results_dict[eps] = {'3-way marginal': k_marginal_score,
    ↪  'HOC': hoc_score}

eps_values = sorted(eps_results_dict)
three_way_errors = [eps_results_dict[e]['3-way marginal'] for e in
↪  eps_values]

plt.figure(figsize=(8, 5))
plt.plot(eps_values, three_way_errors, marker='o')
plt.xlabel('Epsilon')
plt.ylabel('3-way marginal error')
plt.title('Elbow Plot: Epsilon vs 3-way marginal Error')
plt.grid(True, which='both', linestyle='--', alpha=0.5)
plt.show()

eps_values = sorted(eps_results_dict)
hoc_errors = [eps_results_dict[e]['HOC'] for e in eps_values]

plt.figure(figsize=(8, 5))
plt.plot(eps_values, hoc_errors, marker='o')
plt.xlabel('Epsilon')
plt.ylabel('HOC error')
plt.title('Elbow Plot: Epsilon vs HOC Error')
```

```
plt.grid(True, which='both', linestyle='--', alpha=0.5)
plt.show()

print("\n--- Running Statistical Tests ---")
results_dict = []

for eps in epsilon_to_test:
    comparison_results = []

    # Read the dataframes
    real_df = pd.read_csv('/home/dlope2/Documents/Work/TFG/CEA-Cu
    ↪  rie-2011-2.1-cln_original.csv')
    synth_df = pd.read_csv(f'/home/dlope2/Documents/Work/TFG/curi
    ↪  e_synthetic_df_{eps}.csv')

    for col in real_df.columns:
        real_series = real_df[col].dropna()
        synth_series = synth_df[col].dropna()

        if col not in ['Submit_Time', 'Wait_Time', 'Run_Time',
        ↪  'Allocated_Processors']:
            stat = calculate_tvd(real_series, synth_series)
            test_name = "TVD"
        else:
            # Min-Max Normalization using Sklearn
            scaler = MinMaxScaler()

            synth_2d = synth_series.values.reshape(-1, 1)
            real_2d = real_series.values.reshape(-1, 1)

            synth_series_norm =
            ↪  scaler.fit_transform(synth_2d).flatten()
            real_series_norm = scaler.transform(real_2d).flatten()

            stat = wasserstein_distance(real_series_norm,
            ↪  synth_series_norm)
            test_name = "Wasserstein"

        comparison_results.append({
            "Attribute": col,
            "Metric": test_name,
            "Distance (Lower is Better)": round(stat, 4)
        })

    stats_df = pd.DataFrame(comparison_results)
    print(f"\nResults for epsilon: {eps}")
    print(stats_df.to_string(index=False))
    metric_means = stats_df.groupby('Metric')['Distance (Lower is
    ↪  Better)'].mean()
```

```
    # 6. Spearman Correlation (non-linear) error
    num_cols = ['Submit_Time', 'Wait_Time', 'Run_Time',
    ↪  'Allocated_Processors']
    real_corr = real_df[num_cols].corr(method='spearman')
    synth_corr = synth_df[num_cols].corr(method='spearman')
    corr_diff = (real_corr - synth_corr).abs()
    np.fill_diagonal(corr_diff.values, np.nan)
    mean_corr_error = corr_diff.mean().mean()
    print(f"Mean Absolute Spearman Correlation Error:
    ↪  {mean_corr_error:.4f}")
    print("(Closer to 0.0 is better. > 0.15 usually indicates
    ↪  significant loss of joint distributions.)")

    # 7. Linear Correlation error
    num_cols_all = ['Submit_Time', 'Wait_Time', 'Run_Time',
    ↪  'Allocated_Processors']
    corr_real_lin = real_df[num_cols_all].corr()
    corr_synth_lin = synth_df[num_cols_all].corr()
    lin_diff = (corr_real_lin - corr_synth_lin).abs()
    np.fill_diagonal(lin_diff.values, np.nan)
    mean_pearson_correrr = lin_diff.mean().mean()
    print(f"Mean Linear Correlation Error:
    ↪  {mean_pearson_correrr:.4f}")

    # Append to final results
    results_dict.append({
        'epsilon': eps,
        'Mean Wasserstein': metric_means.get('Wasserstein',
        ↪  np.nan),
        'Mean TVD': metric_means.get('TVD', np.nan),
        'Mean Spearman Corr Error': mean_corr_error,
        'Mean Linear Corr Error': mean_pearson_correrr
    })

    print("Metric Means:")
    print(metric_means)
    print("-" * 50)

# %%
df_results = pd.DataFrame(results_dict)

# %%
fig, ax1 = plt.subplots(figsize=(8, 5))
ax1.set_xlabel('Epsilon')

# 1. Left Y-axis (Spearman Correlation)
color1 = 'tab:red'
ax1.set_ylabel('Mean Spearman Distance', color=color1)
```

```
line1 = ax1.plot(df_results['epsilon'], df_results['Mean Spearman
↪  Corr Error'],
                 marker='o', color=color1, label='Spearman')
ax1.tick_params(axis='y', labelcolor=color1)

# 2. Right Y-axis (Linear/Pearson Correlation)
ax2 = ax1.twinx()
color2 = 'tab:blue'
ax2.set_ylabel('Mean Pearson Distance', color=color2)
line2 = ax2.plot(df_results['epsilon'], df_results['Mean Linear
↪  Corr Error'], marker='s', color=color2, label='Linear
↪  (Pearson)')
ax2.tick_params(axis='y', labelcolor=color2)

# Combine legends
lines = line1 + line2
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels, loc='best')

fig.tight_layout()
plt.show()

# %%
fig, ax1 = plt.subplots(figsize=(8, 5))
ax1.set_xlabel('Epsilon (log scale)')

# 1. Left Y-axis (Wasserstein)
color1 = 'tab:red'
ax1.set_ylabel('Mean Wasserstein Distance', color=color1)
line1 = ax1.plot(df_results['epsilon'], df_results['Mean
↪  Wasserstein'],
                 marker='o', color=color1, label='Wasserstein')
ax1.tick_params(axis='y', labelcolor=color1)

# 2. Right Y-axis (TVD)
ax2 = ax1.twinx()
color2 = 'tab:blue'
ax2.set_ylabel('Mean TVD Distance', color=color2)
line2 = ax2.plot(df_results['epsilon'], df_results['Mean TVD'],
                 marker='s', color=color2, label='TVD')
ax2.tick_params(axis='y', labelcolor=color2)

# Combine legends
lines = line1 + line2
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels, loc='best')

fig.tight_layout()
plt.show()
```

Figure 60: Script implementing the elbow method

I.7 synth_evaluation_script.py

This script provides evaluation functions used in the evaluation execution script.

```
"""
Synthetic HPC Workload Evaluation Suite (Utility & Privacy)

"""

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import wasserstein_distance
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score,
↪  normalized_mutual_info_score
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics.pairwise import rbf_kernel
import random
from itertools import combinations

sns.set(style="whitegrid")
class DPEvaluator:
    def __init__(self, real_data: pd.DataFrame, synth_data:
    ↪  pd.DataFrame):
        assert set(real_data.columns) == set(synth_data.columns)
        self.real_data = real_data
        self.synth_data = synth_data
        self.n_real = len(real_data)
        self.n_synth = len(synth_data)

    def evaluate_k_marginals(self, k=3, num_combinations=None):
        cols = self.real_data.columns.tolist()
        all_combos = list(combinations(cols, k))
        if num_combinations and num_combinations <
        ↪  len(all_combos):
            test_combos = random.sample(all_combos,
            ↪  num_combinations)
        else:
```

```
            test_combos = all_combos
        total = 0
        for combo in test_combos:
            combo = list(combo)
            real_cnt = self.real_data.groupby(combo).size() /
            ↪  self.n_real
            synth_cnt = self.synth_data.groupby(combo).size() /
            ↪  self.n_synth
            df_cnt = pd.DataFrame({'real': real_cnt, 'synth':
            ↪  synth_cnt}).fillna(0)
            total += (df_cnt['real'] -
            ↪  df_cnt['synth']).abs().sum()
        return total / len(test_combos)

    def evaluate_hoc(self, num_queries=50,
    ↪  max_features_per_query=5):
        cols = self.real_data.columns.tolist()
        total_diff = 0
        for _ in range(num_queries):
            target = self.real_data.sample(1).iloc[0]
            n_feat = random.randint(2, min(max_features_per_query,
            ↪  len(cols)))
            feats = random.sample(cols, n_feat)
            mask_real = pd.Series(True,
            ↪  index=self.real_data.index)
            mask_synth = pd.Series(True,
            ↪  index=self.synth_data.index)
            for f in feats:
                val = target[f]
                mask_real &= self.real_data[f] == val
                mask_synth &= self.synth_data[f] == val
            total_diff += abs(mask_real.sum()/self.n_real -
            ↪  mask_synth.sum()/self.n_synth)
        return total_diff / num_queries
def plot_real_vs_synth_distributions(real_df, synth_df,
↪  columns_to_plot, bins=40):
    """
    Generates overlay plots (Histograms + KDE for numericals, Bar
    ↪  charts for categoricals)
    to visually inspect the 1D marginal distributions.
    """
    sns.set(style="whitegrid")
    for col in columns_to_plot:
        if col not in real_df.columns or col not in
        ↪  synth_df.columns:
            print(f"Skipping '{col}': Not found in both
            ↪  DataFrames.")
            continue
        if pd.api.types.is_numeric_dtype(real_df[col]):
```

```
            plt.figure(figsize=(8, 4))
            real_data = real_df[col]
            synth_data = synth_df[col]
            sns.histplot(real_data, color='blue', label='Real',
            ↪  stat='density',
                        alpha=0.5, kde=True, bins=bins)
            sns.histplot(synth_data, color='orange',
            ↪  label='Synthetic', stat='density',
                        alpha=0.5, kde=True, bins=bins)
            plt.title(f'Distribution Comparison (Real vs
            ↪  Synthetic): {col}')
            plt.xlabel(col)
            plt.ylabel('Density')
            plt.legend()
            plt.tight_layout()
            plt.show()
        else:
            real_counts = real_df[col].value_counts()
            synth_counts = synth_df[col].value_counts()
            all_idx = real_counts.index.union(synth_counts.index)
            plt.bar(all_idx.astype(str),
            ↪  real_counts.reindex(all_idx, fill_value=0),
                    alpha=0.6, label="Real")
            plt.bar(all_idx.astype(str),
            ↪  synth_counts.reindex(all_idx, fill_value=0),
                    alpha=0.6, label="Synthetic")
            plt.xticks(rotation=90)
            plt.title(col)
            plt.legend()
            plt.tight_layout()
            plt.show()
def bounds(original,synth):
    total_min_diff=0
    total_max_diff=0
    num_cols=0
    for i in original.columns:
        if original[i].dtype == 'object' or
        ↪  pd.api.types.is_categorical_dtype(original[i]):
            continue
        else:
            min_diff = abs(original[i].min() - synth[i].min())
            max_diff = abs(original[i].max() - synth[i].max())
            print(f"Column: {i}, Original Min:
            ↪  {original[i].min():.4f}, Synth Min:
            ↪  {synth[i].min():.4f}, Min Diff: {min_diff:.4f}")
            print(f"Column: {i}, Original Max:
            ↪  {original[i].max():.4f}, Synth Max:
            ↪  {synth[i].max():.4f}, Max Diff: {max_diff:.4f}")
            total_min_diff += min_diff
```

```
            total_max_diff += max_diff
            num_cols+=1
    avg_total_bound_diff = ((total_min_diff + total_max_diff) / (2
    ↪  * num_cols)) if num_cols > 0 else 0.0
    return avg_total_bound_diff
def calculate_tvd(real_s, synth_s):
    all_cats = set(real_s.unique()).union(set(synth_s.unique()))
    real_props =
    ↪  real_s.value_counts(normalize=True).reindex(all_cats,
    ↪  fill_value=0)
    synth_props =
    ↪  synth_s.value_counts(normalize=True).reindex(all_cats,
    ↪  fill_value=0)
    return 0.5 * np.sum(np.abs(real_props - synth_props))
def evaluate_hpc_privacy(real_train_df, synthetic_df, k=5):
    num_cols = real_train_df.select_dtypes(include=['int64',
    ↪  'float64']).columns
    real_num = real_train_df[num_cols].dropna()
    synth_num = synthetic_df[num_cols].dropna()
    nn = NearestNeighbors(n_neighbors=k, metric='euclidean',
    ↪  n_jobs=-1)
    nn.fit(real_num)
    distances, _ = nn.kneighbors(synth_num)
    d1 = distances[:, 0]
    d2 = distances[:, 1]
    dk = distances[:, k-1]
    nndr = d1 / (d2 + 1e-10)
    exact_clones = np.sum(d1 < 1e-5)
    return {'mean_1dcr': np.mean(d1), 'mean_5dcr': np.mean(dk),
            'nndr': np.mean(nndr), 'exact_clones': exact_clones,
            'd1': d1, 'dk': dk, 'nndr_array': nndr}
def calculate_mmd(X, Y, gamma=1.0, subsample_size=10000):
    X = np.array(X)
    Y = np.array(Y)
    n_samples = min(len(X), len(Y), subsample_size)
    idxX = np.random.choice(len(X), n_samples, replace=False)
    idxY = np.random.choice(len(Y), n_samples, replace=False)
    X_sub = X[idxX]; Y_sub = Y[idxY]
    XX = rbf_kernel(X_sub, X_sub, gamma=gamma)
    YY = rbf_kernel(Y_sub, Y_sub, gamma=gamma)
    XY = rbf_kernel(X_sub, Y_sub, gamma=gamma)
    mmd2 = XX.mean() + YY.mean() - 2*XY.mean()
    return np.sqrt(max(0, mmd2))
```

Figure 61: Script with evaluation functions

I.8 evaluation_proper.py

This script executes the evaluation given a previously generated synthetic dataset, the original dataset, the synthetic binned dataset and the original binned dataset.

```
from synth_evaluation_script import *
from generalized_preprop import *
from mst_synth_funcs import *
from generalized_csvTswf import *
from metadata_loading import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.neighbors import NearestNeighbors

# %%
eps_to_test = [5.0, 10.0]

# %%
metadata_path = "curie_metadata.json"

from sklearn.preprocessing import MinMaxScaler

def single_evaluation(epsilon=False, synth_path=None,
↪  real_path=None, binreal_path=None, binsynth_path=None):
    real_df = pd.read_csv(real_path)
    synth_df = pd.read_csv(synth_path)
    real_bin_df = pd.read_csv(binreal_path)
    synth_bin_df = pd.read_csv(binsynth_path)

    try:
        real_df = real_df.drop(columns=['Job_ID'])
        real_bin_df = real_bin_df.drop(columns=['Job_ID'])
    except KeyError:
        try:
            real_df = real_df.drop(columns=['job_id'])
            real_bin_df = real_bin_df.drop(columns=['job_id'])
        except KeyError:
            pass

    try:
        synth_df = synth_df.drop(columns=['job_id'])
```

```
    synth_bin_df = synth_bin_df.drop(columns=['job_id'])
except KeyError:
    pass

categorical = ['User_ID', 'Group_ID', 'Status',
↪  'Partition_Number']
for i in categorical:
    real_df[i] = real_df[i].astype('str')
    synth_df[i] = synth_df[i].astype('str')
    real_bin_df[i] = real_bin_df[i].astype('str')
    synth_bin_df[i] = synth_bin_df[i].astype('str')

common_cols = [c for c in real_df.columns if c in
↪  synth_df.columns]
df_real_train, df_real_holdout = train_test_split(real_df,
↪  test_size=0.2, random_state=42)

results = {}

# 1. Univariate Utility (Wasserstein & TVD)
print("\n--- Running Statistical Tests ---")
comparison_results = []

results['Avg Bound Diff'] = bounds(real_df, synth_df)
print(f"Average Bound Difference: {results['Avg Bound
↪  Diff']:.4f}")

for col in common_cols:
    real_series = real_df[col].dropna()
    synth_series = synth_df[col].dropna()

    if real_df[col].dtype == 'object':
        stat = calculate_tvd(real_series, synth_series)
        test_name = "TVD"
    else:
        # Normalizing continuous variables using Synthetic as
        ↪  baseline
        scaler = MinMaxScaler()
        synth_2d = synth_series.values.reshape(-1, 1)
        real_2d = real_series.values.reshape(-1, 1)

        synth_series_norm =
        ↪  scaler.fit_transform(synth_2d).flatten()
        real_series_norm = scaler.transform(real_2d).flatten()

        stat = wasserstein_distance(real_series_norm,
        ↪  synth_series_norm)
        test_name = "Wasserstein"
```

```
        comparison_results.append({
            "Attribute": col,
            "Metric": test_name,
            "Distance (Lower is Better)": round(stat, 4)
        })

    stats_df = pd.DataFrame(comparison_results)
    print(stats_df.to_string(index=False))
    metric_means = stats_df.groupby('Metric')['Distance (Lower is
    ↪  Better)'].mean()
    results['Mean Wasserstein'] = metric_means.get('Wasserstein',
    ↪  np.nan)
    results['Mean TVD'] = metric_means.get('TVD', np.nan)
    print(f"Mean Wasserstein: {results['Mean Wasserstein']:.4f}")
    print(f"Mean TVD: {results['Mean TVD']:.4f}")

    # 2. Multivariate Utility (TSTR)
    print("\n--- Machine Learning Utility ---")
    TARGET = 'Status'
    features = [c for c in common_cols if c != TARGET and c !=
    ↪  'User_ID' and c != 'Group_ID']
    cat_features = [c for c in features if real_df[c].dtype ==
    ↪  'object']
    num_features = [c for c in features if c not in cat_features]

    # Added MinMaxScaler to the numerical features step
    preprocessor = ColumnTransformer([
        ('cat', OneHotEncoder(handle_unknown='ignore',
        ↪  drop='if_binary'), cat_features),
        ('num', MinMaxScaler(), num_features)
    ], remainder='passthrough')

    X_real_train, X_real_test, y_real_train, y_real_test =
    ↪  train_test_split(
        real_df[features], real_df[TARGET], test_size=0.2,
        ↪  random_state=42)
    X_synth_train = synth_df[features]
    y_synth_train = synth_df[TARGET]

    preprocessor.fit(X_synth_train)

    X_real_train_proc = preprocessor.transform(X_real_train)
    X_real_test_proc = preprocessor.transform(X_real_test)
    X_synth_train_proc = preprocessor.transform(X_synth_train)

    models = {
        "RF": RandomForestClassifier(n_estimators=50,
        ↪  random_state=42),
        "LR": LogisticRegression(max_iter=1000, random_state=42)
```

```
    }
    losses = {}
    for name, model in models.items():
        model.fit(X_real_train_proc, y_real_train)
        acc_real = accuracy_score(y_real_test,
        ↪  model.predict(X_real_test_proc))
        model.fit(X_synth_train_proc, y_synth_train)
        acc_synth = accuracy_score(y_real_test,
        ↪  model.predict(X_real_test_proc))
        loss = acc_real - acc_synth
        losses[name] = loss
        print(f"{name}: TRTR={acc_real:.4f}, TSTR={acc_synth:.4f},
        ↪  Loss={loss:.4f}")

    results['RF Loss'] = losses['RF']
    results['LR Loss'] = losses['LR']

    # 3. Discriminator
    print("\n--- Holistic Similarity: Discriminator Test ---")
    df_real_discrim = real_df[features].copy()
    df_real_discrim['is_synthetic'] = 0
    df_synth_discrim = synth_df[features].copy()
    df_synth_discrim['is_synthetic'] = 1
    df_combined = pd.concat([df_real_discrim, df_synth_discrim],
    ↪  axis=0)
    X_discrim = df_combined.drop(columns=['is_synthetic'])
    y_discrim = df_combined['is_synthetic']
    X_disc_train, X_disc_test, y_disc_train, y_disc_test =
    ↪  train_test_split(
        X_discrim, y_discrim, test_size=0.3, random_state=42)

    # Use synthetic-based preprocessor transformations (preserving
    ↪  baseline)
    X_disc_train_proc = preprocessor.transform(X_disc_train)
    X_disc_test_proc = preprocessor.transform(X_disc_test)

    disc_model = RandomForestClassifier(n_estimators=50,
    ↪  max_depth=5, random_state=42)
    disc_model.fit(X_disc_train_proc, y_disc_train)
    disc_preds = disc_model.predict_proba(X_disc_test_proc)[:, 1]
    disc_auc = roc_auc_score(y_disc_test, disc_preds)
    results['Discriminator AUC'] = disc_auc
    print(f"Discriminator AUC: {disc_auc:.4f}")

    # 4. Privacy: DCR (outlier assessment)
    print("\n--- Privacy: Outlier DCR ---")
    df_real_priv = real_df.copy().drop(columns=['User_ID',
    ↪  'Group_ID'])
```

```
df_synth_priv = synth_df.copy().drop(columns=['User_ID',
↪  'Group_ID'])

preproc_priv = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore',
    ↪  drop='if_binary'), cat_features),
    ('num', MinMaxScaler(), num_features)
], remainder='passthrough')

preproc_priv.fit(synth_df[features])

X_real_train_dcr =
↪  preproc_priv.transform(df_real_train[features])
X_real_holdout_dcr =
↪  preproc_priv.transform(df_real_holdout[features])
X_synth_dcr = preproc_priv.transform(synth_df[features].sampl
↪  e(min(len(synth_df), 5000), random_state=42))

nn = NearestNeighbors(n_neighbors=1, metric='euclidean',
↪  n_jobs=-1)
nn.fit(X_real_train_dcr)
holdout_distances, _ = nn.kneighbors(X_real_holdout_dcr)
dcr_holdout = holdout_distances.flatten()
synth_distances, _ = nn.kneighbors(X_synth_dcr)
dcr_synth = synth_distances.flatten()

print(f"Holdout to Train DCR (Baseline) - Mean:
↪  {dcr_holdout.mean()}, Min: {dcr_holdout.min()}")
print(f"Synth to Train DCR (Evaluated)  - Mean:
↪  {dcr_synth.mean()}, Min: {dcr_synth.min()}")
results['Mean Synth to Train DCR'] = dcr_synth.mean()
results['Min Synth to Train DCR'] = dcr_synth.min()
plt.figure(figsize=(10, 6))
sns.kdeplot(dcr_holdout, color='#1f77b4', fill=True,
↪  alpha=0.4,
            label='Baseline (Real unseen data)', clip=(0, 2),
            ↪  linewidth=2)
sns.kdeplot(dcr_synth, color='#ff7f0e', fill=True, alpha=0.4,
            label='Evaluated (Synthetic data)', clip=(0, 2),
            ↪  linewidth=2)
mean_holdout = np.mean(dcr_holdout)
mean_synth = np.mean(dcr_synth)
plt.axvline(mean_holdout, color='#1f77b4', linestyle='--',
↪  linewidth=2,
            label=f'Baseline Mean: {mean_holdout:.4f}')
plt.axvline(mean_synth, color='#ff7f0e', linestyle='--',
↪  linewidth=2,
            label=f'Synth Mean: {mean_synth:.4f}')
```

```
plt.title("Empirical Privacy Check: Distance to Closest Record
↪  (DCR)\n(Further right = Better Privacy Protection)",
↪  fontsize=14, fontweight='bold', pad=15)
plt.xlabel("euclidean Distance\n← Danger (0.0 = Exact Match)
↪  |  Safe (Distant) →", fontsize=12, labelpad=10)
plt.ylabel("Density of Records", fontsize=12)
max_plot_val = np.percentile(dcr_synth, 99) * 1.5
plt.xlim(-0.005, max_plot_val)
plt.legend(loc='upper right', frameon=True, shadow=True)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.grid(axis='x', linestyle=':', alpha=0.5)
plt.tight_layout()
plt.show()
# 5. HPC Cluster Safety (k-DCR, NNDR)
hpc = evaluate_hpc_privacy(df_real_priv, df_synth_priv, k=5)
results['Mean 1-DCR'] = hpc['mean_1dcr']
results['Mean 5-DCR'] = hpc['mean_5dcr']
results['NNDR'] = hpc['nndr']
results['Exact clones'] = hpc['exact_clones']

# 6. Spearman Correlation (non-linear) error
num_cols = [c for c in common_cols if
↪  pd.api.types.is_numeric_dtype(real_df[c])]
real_corr = real_df[num_cols].corr(method='spearman')
synth_corr = synth_df[num_cols].corr(method='spearman')
corr_diff = (real_corr - synth_corr).abs()
np.fill_diagonal(corr_diff.values, np.nan)
results['Spearman Corr Error'] = corr_diff.mean().mean()
mean_corr_error = corr_diff.mean().mean()
print(f"Mean Absolute Correlation Error:
↪  {mean_corr_error:.4f}")
print("(Closer to 0.0 is better. > 0.15 usually indicates
↪  significant loss of joint distributions.)")
plt.figure(figsize=(10, 8))
sns.heatmap(corr_diff, annot=True, fmt=".2f", cmap="Reds",
↪  vmin=0, vmax=1, square=True,
            cbar_kws={'label': 'Absolute Correlation
            ↪  Difference'})
plt.title(f"Correlation Difference Heatmap\nMean Error:
↪  {mean_corr_error:.4f}")
plt.tight_layout()
plt.show()
# 7. Linear Correlation error
num_cols_all = [c for c in common_cols if
↪  pd.api.types.is_numeric_dtype(real_df[c])]
if num_cols_all:
    corr_real_lin = real_df[num_cols_all].corr()
    corr_synth_lin = synth_df[num_cols_all].corr()
    lin_diff = (corr_real_lin - corr_synth_lin).abs()
```

```
    np.fill_diagonal(lin_diff.values, np.nan)
    results['Linear Corr Error'] = lin_diff.mean().mean()
else:
    results['Linear Corr Error'] = np.nan
print('mean linear correlation error:', results['Linear Corr
↪  Error'])
plt.figure(figsize=(10, 8))
sns.heatmap(lin_diff, annot=True, fmt=".2f", cmap="Reds",
↪  vmin=0, vmax=1, square=True,
            cbar_kws={'label': 'Absolute Correlation
            ↪  Difference'})
plt.title(f"Linear Correlation Difference Heatmap\nMean Error:
↪  {results['Linear Corr Error']:.4f}")
plt.tight_layout()
plt.show()
# 8. MMD
mmd_scaler = MinMaxScaler()
bin_num_cols = [c for c in real_bin_df.columns if
↪  pd.api.types.is_numeric_dtype(real_bin_df[c])]

real_bin_df_norm = real_bin_df.copy()
synth_bin_df_norm = synth_bin_df.copy()

if bin_num_cols:
    synth_bin_df_norm[bin_num_cols] =
    ↪  mmd_scaler.fit_transform(synth_bin_df[bin_num_cols])
    real_bin_df_norm[bin_num_cols] =
    ↪  mmd_scaler.transform(real_bin_df[bin_num_cols])

X_mmd = real_bin_df_norm
Y_mmd = synth_bin_df_norm
results['MMD'] = calculate_mmd(X_mmd, Y_mmd)

evaluator = DPEvaluator(real_data=real_bin_df_norm,
↪  synth_data=synth_bin_df_norm[real_bin_df_norm.columns])

print("Calculating 3-Way Marginals...")
k_marginal_score = evaluator.evaluate_k_marginals(k=3,
↪  num_combinations=50)
print(f"3-Way Marginal Score: {k_marginal_score:.4f}")
results['3-Way Marginal Score'] = k_marginal_score
print("Calculating HOC...")
hoc_score = evaluator.evaluate_hoc(num_queries=100,
↪  max_features_per_query=5)
print(f"HOC Score: {hoc_score:.4f}")
results['HOC Score'] = hoc_score
# 9. Visual Comparison of Distributions (for key attributes)
columns_to_plot = [c for c in common_cols if c not in
↪  ['User_ID', 'Group_ID', 'Job_ID']]
```

```
    plot_real_vs_synth_distributions(real_df, synth_df,
    ↪  columns_to_plot)

    # ---- Print final summary and return ----
    print("\n" + "="*60)
    print("SUMMARY OF EVALUATION METRICS")
    print("="*60)
    for key, value in results.items():
        if isinstance(value, float):
            print(f"{key:<30}: {value:.6f}")
        else:
            print(f"{key:<30}: {value}")
    print("="*60)

    return results
dict_results = {eps: {} for eps in eps_to_test}
for eps in eps_to_test:
    df_raw,numerical_cols,categorical_cols,identicat_cols,identif
    ↪  ier_cols,datetimes_cols,dom_know_bounds,_,_,drop_cols,
    ↪  synthetic_data_path, o_data_path,_,delta,length_t_gen=
    ↪  load_dataset_from_metadata(metadata_path=metadata_path)
    delta_f_preprop=delta*0.5
    delta_f_mst=delta*0.5
    result, to_remove, identifiers,noisy_counts =
    ↪  preprocessing_curie(df_raw,numerical_cols,categorical_col
    ↪  s,identifier_cols,identicat_cols,datetimes_cols,eps,dom_k
    ↪  now_bounds,delta=delta_f_preprop)
    df_binned = result['binned_data']           # For MST (binned
    ↪  only)
    edges_dict = result['edges_dict']
    midpoints_dict = result['midpoints_dict']
    num_cols = result['num_cols']
    epsilon_generation = result['eps_generation']
    bounds_dict = result['bounds_dict']
    original_data = result['original_data']
    original_data.to_csv(f"original_supereval_{eps}.csv",index=Fa
    ↪  lse)
    df_binned.to_csv(f"original_supereval_binned_{eps}.csv",index
    ↪  =False)
    synthetic_df =
    ↪  generate_synthetic_data_with_mst(epsilon_generation,
    ↪  df_binned, noisy_counts,delta_f_mst, identifiers)
    first_part_synthetic_data_path =
    ↪  synthetic_data_path.rsplit('.', 1)[0]
    second_part_synthetic_data_path =
    ↪  synthetic_data_path.rsplit('.', 1)[-1]
    bin_synthetic_data_path = f"{first_part_synthetic_data_path}_
    ↪  binned_supereval_{eps}.csv"
    synthetic_df.to_csv(bin_synthetic_data_path, index=False)
```

```
    _synthetic_df = uniform_debinning(
        synthetic_binned_df=synthetic_df,
        edges_dict=edges_dict,
        num_cols=num_cols
    )
    final_synthetic_df = _synthetic_df.copy()
    print("Unbinning complete!")
    def remake(df,to_remove,identifiers):
        for col in to_remove:
            df[col] = pd.Series(-1, index=df.index, dtype='Int64')
        for col in identifiers:
            x=np.arange(1,len(df)+1)
            df[col]=np.random.permutation(x)
        return df
    to_remove=to_remove+drop_cols
    df = remake(final_synthetic_df, to_remove, identifiers)
    first_part_synthetic_data_path =
    ↪  synthetic_data_path.rsplit('.', 1)[0]
    second_part_synthetic_data_path =
    ↪  synthetic_data_path.rsplit('.', 1)[-1]
    synthetic_data_path =
    ↪  f"{first_part_synthetic_data_path}_supereval_{eps}.csv"
    df.to_csv(synthetic_data_path, index=False)
    dict_results[eps] = single_evaluation(epsilon=eps,
    ↪  synth_path=synthetic_data_path, real_path=f"original_supe
    ↪  reval_{eps}.csv",binreal_path=f"original_supereval_binned
    ↪  _{eps}.csv",binsynth_path=bin_synthetic_data_path)
```

Figure 62: Script to perform the statistical evaluation

I.9 manuel_slurm_evaluation.py

This is the script, provided by the supervisor Manuel G. Marciani, that executes the evaluation of the simulated SLURM trace.

```
# coding: utf-8
"""
Parse BSC's SLURM Simulator

 - Plot utilization and required resources per instant
 - Share of backfilled jobs
 - If present, compute the average priority of the non backfilled
 ↪  job
```

```
BSC-CNS - Earth Sciences
Manuel Gimenez de Castro Marciani
2023
"""
import pandas as pd
import argparse
import matplotlib.pyplot as plt
import numpy as np
import pyswf

def workflow_statistics(trace_dataframe: pd.DataFrame,
↪  target_uid:int, machine:str, workflow_name:str, wrapped:str,
                        job_alloc_cpus: int, job_runtime: int,
                        ↪  trace_path:str) -> None:
    """
    Generates run statistics of the target workflow
    :trace_dataframe: dataframe representation of the simulator
    ↪  output trace
    :target_uid: uid that launched the simulation
    :machine: name of the machine that was simulated
    :workflow_name: name of the workflow
    :wrapped: if the workflow was wrapped
    :trace_path: path to save the figure
    """
    # TARGET WORKFLOW
    # manuel: slowfown for a horizontal workflow is not really
    ↪  well defined.
    with open(trace_path+f"{machine}_{job_alloc_cpus}_{job_runtim
    ↪  e}_{workflow_name}_{wrapped}.csv", "w") as wf_outputfile:
        target_workflow = trace_dataframe[trace_dataframe["uid"]
        ↪  == target_uid]
        if "horizontal" in workflow_name:
            wf_runtime = target_workflow["runtime"].iloc[0]  # we
            ↪  assume every job has the same runtime
        else:  # vertical
            wf_runtime = target_workflow["runtime"].sum()
        wf_total_wait_time = target_workflow["wait_time"].sum()
        wf_total_time = target_workflow["end_time"].max() -
        ↪  target_workflow["submit_time"].min()
        wf_slowdown = wf_total_time/wf_runtime
        wf_outputfile.write(f"simulation, total_runtime,
        ↪  total_time, slowdown, total_wait_time,
        ↪  runtime_diff\n")
        wf_outputfile.write(f"{machine}_{job_alloc_cpus}_{job_run
        ↪  time}_{workflow_name}_{wrapped}, {wf_runtime},
        ↪  {wf_total_time}, {wf_slowdown}, {wf_total_wait_time},
        ↪  {wf_total_time - wf_runtime}\n")
```

```
def utilization_plot(trace_dataframe: pd.DataFrame,
↪  target_uid:int, machine:str, workflow_name:str, wrapped:str,
       job_alloc_cpus: int, job_runtime: int, trace_path:str,
       ↪  total_cpus: int = 0, xbot: int = 0, xtop: int = 0,
       ↪  interactive: bool = False) -> None:
    """
    Plots the utilization and requested plot of the data.
    :trace_dataframe: dataframe representation of the simulator
    ↪  output trace
    :target_uid: uid of the user that launched the simulation
    :machine: name of the machine that was simulated
    :workflow_name: name of the workflow
    :wrapped: if the workflow was wrapped
    :total_cpus: total available cpus in the machine. If set to 0,
    ↪  absolute values will be plotted
    :xbot: left limit of the window that will be saved
    :xtop: right limit of the window that will be saved
    :trace_path: path to save the figure
    :interactive: boolean flag indicating if pop up the
    ↪  interactive window for the plot
    """

    # only compute the utilization where there were changes, and
    ↪  the previous instant.
    time_steps = list(trace_dataframe["start_time"]) +
    ↪  list(trace_dataframe["end_time"]) +
    ↪  list(trace_dataframe["submit_time"]) +
    ↪  list(trace_dataframe["start_time"]-1) +
    ↪  list(trace_dataframe["end_time"]-1) +
    ↪  list(trace_dataframe["submit_time"]-1)
    time_steps = list(set(time_steps))  # remove duplicates
    time_steps.sort()

    utilization = []
    requested = []

    for time in time_steps:
        active_jobs =
        ↪  trace_dataframe[(trace_dataframe["start_time"] <=
        ↪  time) & (trace_dataframe["end_time"] > time)]
        utilization.append(active_jobs["alloc_cpus"].sum())
        queueing_jobs =
        ↪  trace_dataframe[(trace_dataframe["submit_time"] <=
        ↪  time) & (trace_dataframe["start_time"] > time)]
        requested.append(queueing_jobs["alloc_cpus"].sum())

    if total_cpus != 0:
```

```
        plt.plot(time_steps, np.array(utilization)/total_cpus,
        ↪  label="util")
        plt.plot(time_steps, np.array(requested)/total_cpus,
        ↪  label="req")
    else:
        plt.plot(time_steps, utilization, label="in use")
        plt.plot(time_steps, requested, label="in queue")

    # draw vertical lines where the workflow was launched
    for i, wf_submit_time in
    ↪  enumerate(trace_dataframe[trace_dataframe["uid"] ==
    ↪  target_uid]["submit_time"]):
        if i == 0:
            plt.axvline(wf_submit_time, color="r", label=f"wf job
            ↪  submission")
        else:
            plt.axvline(wf_submit_time, color="r")

    # draw vertical lines where each job started
    for i, wf_submit_time in
    ↪  enumerate(trace_dataframe[trace_dataframe["uid"] ==
    ↪  target_uid]["start_time"]):
        if i == 0:
            plt.axvline(wf_submit_time, color="g", label=f"wf job
            ↪  start")
        else:
            plt.axvline(wf_submit_time, color="g")

    plt.axhline(1, color="b")

    plt.title(f"Simulated Utilized and Requested resources\ntrace
    ↪  with {workflow_name} {wrapped} at {machine}")
    plt.legend()

    plt.ylabel("CPUs")
    plt.xlabel("Seconds since first submission")

    if xbot != 0 or xtop != 0:
        plt.xlim(xbot, xtop)

    if interactive:
        plt.show()
    else:
        plt.savefig(trace_path+f"{machine}_{job_alloc_cpus}_{job_
        ↪  runtime}_{workflow_name}_{wrapped}_util_req.png")
        plt.clf()  # make sure to clear the canvas always
```

```
def backfill_plot(trace_dataframe: pd.DataFrame, target_uid:int,
↪  machine:str, workflow_name:str, wrapped:str, job_alloc_cpus:
↪  int, job_runtime: int, trace_path:str):
    """
    Plots the utilization and requested plot of the data.
    :trace_dataframe: dataframe representation of the simulator
    ↪  output trace
    :target_uid: uid of the user that launched the simulation
    :machine: name of the machine that was simulated
    :workflow_name: name of the workflow
    :wrapped: if the workflow was wrapped
    :trace_path: path to save the figure
    """

    # Backfilled PLOT
    total = trace_dataframe["backfill"].describe()[0]
    count = trace_dataframe["backfill"].describe()[3]
    common = trace_dataframe["backfill"].describe()[2]
    # super sketchy way of picking the other label

    uncommon_set = set(trace_dataframe["backfill"].unique()) -
    ↪  {common}
    if uncommon_set:
        uncommon = uncommon_set.pop()
    else:
        uncommon = common
    plt.pie([count, total-count], labels=[common, uncommon])
    plt.title(f"Proportion of jobs backfilled \ntrace with
    ↪  {workflow_name} {wrapped} at {machine}")
    plt.savefig(trace_path+f"{machine}_{job_alloc_cpus}_{job_runt
    ↪  ime}_{workflow_name}_{wrapped}_backfill.png")
    plt.clf()


def parse_args():
    # PARSE STUFF
    parser = argparse.ArgumentParser(
        description='Parse slurm simulator trace file (log of
        ↪  executed files), compute statistics and plot usage and
        ↪  '
                    'requested resources')
    parser.add_argument('trace_file', type=str, help='path where
    ↪  the original trace is located')
    parser.add_argument('target_uid', type=int, help='uid of the
    ↪  user that launches the workflow')
    parser.add_argument('machine', type=str, help="name of the
    ↪  machine we are simulating")
    parser.add_argument('workflow_name', type=str, help='name of
    ↪  the workflow')
```

```
    parser.add_argument('type', type=str, help='type of the
    ↪  workflow')
    parser.add_argument('--total-cpus', default=0, type=int,
    ↪  help='total number of cpus in the machine')
    parser.add_argument('--xbot', default=0, type=int, help='left
    ↪  corner of the visualization window')
    parser.add_argument('--xtop', default=0, type=int, help='right
    ↪  corner of the visualization window')
    parser.add_argument('--fairshare-cutout', default=0, type=int,
    ↪  help='indicate until when are the jobs of the fairshare
    ↪  setting running')
    return parser.parse_args()
if __name__ == "__main__":

    args = parse_args()

    trace_file = args.trace_file
    trace_path = args.trace_file.rsplit("/",1)[0]+"/"
    target_uid = args.target_uid
    machine = args.machine
    workflow_name = args.workflow_name
    wf_type = args.type
    total_cpus = args.total_cpus
    xbot = args.xbot
    xtop = args.xtop

    trace_out = pyswf.SWFtrace.from_slurm_trace(trace_file)
    # file_name = trace_file.split("/")[-1]
    # job_alloc_cpus = file_name.split("_")[3]
    # job_runtime = file_name.split("_")[4]
    file_name = "f"
    job_alloc_cpus = "96"
    job_runtime = "1800"

    # filter out those jobs that are to set the fairshare
    trace_workload =
    ↪  trace_out.job_list[trace_out.job_list["submit_time"] >=
    ↪  args.fairshare_cutout]

    # output wait time stats to csv
    trace_workload["wait_time"].describe().to_csv(trace_path+f'wa
    ↪  it_time_{machine}_{job_alloc_cpus}_{job_runtime}_{workflo
    ↪  w_name}_{wf_type}.csv')
    non_backfilled = trace_workload[trace_workload["backfill"] ==
    ↪  "no"]
    backfilled = trace_workload[trace_workload["backfill"] ==
    ↪  "yes"]

    # output the stats of allocated cpus of the backfilled jobs
```

```
backfilled["alloc_cpus"].describe().to_csv(trace_path+f'bf_al
↪ loc_cpus_{machine}_{job_alloc_cpus}_{job_runtime}_{workfl
↪ ow_name}_{wf_type}.csv')
try:
    non_backfilled["priority"].describe().to_csv(trace_path+f
    ↪ 'non_bf_prio_{machine}_{job_alloc_cpus}_{job_runtime}
    ↪ _{workflow_name}_{wf_type}.csv')
    backfilled["priority"].describe().to_csv(trace_path+f'bf_
    ↪ prio_{machine}_{job_alloc_cpus}_{job_runtime}_{workfl
    ↪ ow_name}_{wf_type}.csv')
except KeyError:  # It could be that we don't have the
↪ priority key
    pass

workflow_statistics(trace_workload, target_uid, machine,
↪ workflow_name, wf_type, job_alloc_cpus, job_runtime,
                    trace_path)
utilization_plot(trace_workload, target_uid, machine,
↪ workflow_name, wf_type, job_alloc_cpus, job_runtime,
                 trace_path, total_cpus, xbot if xbot >=
                 ↪ args.fairshare_cutout else
                 ↪ args.fairshare_cutout, xtop, False)
backfill_plot(trace_workload, target_uid, machine,
↪ workflow_name, wf_type, job_alloc_cpus, job_runtime,
↪ trace_path)
```

Figure 63: Manuel G. Marciani's script to perform the evaluation of the SLURM simulation