TPCx-AI under the Microscope: A Benchmarking Debt Analysis

Ilin Tolovski
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
ilin.tolovski@hpi.de

Philipp Hildebrandt
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
philipp.hildebrandt@hpi.de

Khuzaima Daudjee
Cheriton School of
Computer Science
University of Waterloo
khuzaima.daudjee@uwaterloo.ca

Tilmann Rabl
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
tilmann.rabl@hpi.de

ABSTRACT

TPCx-AI is an industry standard benchmark for evaluating the end-to-end performance of machine learning systems and the underlying hardware configurations. In the database community, individual parts of the dataset and the workloads are used to evaluate preprocessing methods and systems for fast inference. In both of these cases, the datasets and workloads are used based on the characteristics defined in the specification. Upon analysis of TPCx-AI's dataset and use cases, we observe that the official implementation of TPCx-AI's kit diverges from the specification, does not evaluate the capabilities of the system under test, and impacts the overall performance in a benchmark run.

In this paper, we investigate the benchmarking debt accumulated in the TPCx-AI dataset and the workloads. We identify properties that impact the benchmark's performance, including runtime and quality of use cases, the defined metrics and their thresholds, workload discrepancies, and data errors. Our analysis shows that all use cases and datasets contain benchmarking debts, impacting the training and serving runtimes by up to 350× and 800×, respectively. By addressing these debts, we observe an end-to-end throughput increase of up to 3.8× over the default TPCx-AI implementation.

PVLDB Reference Format:
Ilin Tolovski, Philipp Hildebrandt, Khuzaima Daudjee, and Tilmann Rabl. TPCx-AI under the Microscope: A Benchmarking Debt Analysis. PVLDB, 19(6): 1305 - 1318, 2026.
doi:10.14778/3797919.3797936

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hpides/tpcx-ai-analysis.git.

1 INTRODUCTION

There have been significant advances in research and development of machine learning (ML) systems over the past decade. ML systems are developed and optimized for heterogeneous hardware at various scales, ranging from resource-constrained edge devices to powerful single-node servers and multi-node clusters. ML system performance is evaluated by benchmarks measuring fine-grained model performance with multiple metrics on a curated set of datasets and metrics (e.g., MLPerf [29]), or measuring end-to-end system performance by an aggregated performance metric on specification-based datasets and workloads (e.g., TPCx-AI [11]).

TPCx-AI evaluates the end-to-end performance of *ten* diverse machine learning workloads, such as clustering, time-series analysis, image recognition, and speech-to-text transcription. It measures the throughput performance in four stages: *training, sequential serving, parallel serving, and scoring*. For each stage, TPCx-AI generates a comprehensive data schema of structured, numerical datasets, as well as text, audio, and image datasets. Data processing occurs in the loading and preprocessing stages, which vary across use cases and affect their runtime breakdown. The TPCx-AI specification provides only a coarse overview of the benchmarking stages, making it difficult to estimate their runtime [12].

TPCx-AI is used to evaluate systems specialized in individual stages of the ML lifecycle, such as data loading, preprocessing, and storage [16, 20, 22, 26, 27, 33, 41], or to compare the end-to-end performance between different systems and hardware configurations [3, 8, 22, 32]. The former uses parts of the TPCx-AI artifacts and tables to evaluate the impact of optimizations. The latter utilize the complete set of workloads to evaluate the end-to-end performance. In both scenarios, the workloads are used to measure improvements of the runtime on a System under Test (SuT).

The end-to-end workload performance depends on the quality of the generated dataset and its integration with the workload. Using individual datasets or workloads to evaluate novel algorithms and systems carries over their intrinsic properties, which, to the best of our knowledge, have not been identified or analyzed. Related analyses on TPC-H workloads show that such workload characteristics impact the runtime up to an order of magnitude [9, 14].

TPCx-AI evaluates the performance of ML system deployments, which requires realistic workload runtimes and execution profiles, but not necessarily realistic datasets. Other uses of the benchmark can lead to unexpected behavior, when the workload or data does not stress or even contain the evaluated properties. We define such behavior as *benchmarking debt*, which can originate from the dataset or workload.

Dataset-related debt has been investigated in the context of benchmark datasets. Preprocessed collections of real data, such as COCO, ImageNet, and Wikipedia Data, are used to evaluate ML benchmarks [4, 21, 37]. Related work has shown that such data are often biased, containing nested and recurring errors [38]. Inconsistencies when sampling a subset of these datasets often provide inconclusive and non-reproducible results [10, 25].

To address some of these concerns, TPCx-AI generates a synthetic dataset based on a predefined specification. The characteristics of the generated dataset and their impact on the workload performance have not been investigated beyond their scalability. A

Proceedings of the VLDB Endowment, Vol. 19, No. 6 ISSN 2150-8097.
doi:10.14778/3797919.3797936

preliminary scaling analysis of TPCx-AI by Patel et al. [32] shows that the execution time of the workloads does not scale proportionally to the scaling factor due to the properties of the datasets and the learning algorithms in the use cases. This work does not analyze the data properties that are causing the results or the broader impact on the workload. To the best of our knowledge, there is no further analysis of the TPCx-AI dataset and workloads.

Benchmarks such as MLPerf use fixed real-world datasets to evaluate deep learning tasks and implement the training, inference, and storage stages of the ML lifecycle into separate benchmarks [1, 2, 36]. TPCx-AI integrates the end-to-end lifecycle in a single workload and evaluates a variety of generic ML and deep learning workloads. It uses a single, scalable dataset, allowing it to run on edge devices as well as high-end servers [3, 8]. Unlike MLPerf, the TPCx-AI benchmark workload incorporates all stages of an AI workload, making it suitable to evaluate systems comprising both data manipulation and ML operations using scalable datasets and workloads. An initial comparison between MLPerf and TPCx-AI summarizes these differences between the two benchmarks and their usage scenarios [24].

In this paper, we focus on the quality of TPCx-AI's dataset and workloads. We analyze their impact on the runtime and the end-to-end throughput. Due to the scalable and synthetic nature of the artifacts in TPCx-AI, we observe benchmarking debts in all use cases and discuss optimizations specific to the TPCx-AI datasets and workloads. Overall, we make the following contributions:

- We analyze the TPCx-AI dataset in detail, focusing on the validity and performance of the workloads;
- We define data- and workload-related benchmark debts and present possible solutions that enable a more efficient evaluation of the SuT;
- We discuss the impact of the data- and workload-related debt on the ML use cases and the evaluated performance.

The rest of the paper is organized as follows. In Section 2, we define and describe benchmarking debt. In Section 3, we present an overview of the TPCx-AI benchmark and its data model. We present the findings of the data analysis in Section 4 and summarize the main takeaways in Section 5. In Section 6, we present related work on benchmark analyses. In Section 7, we summarize the paper.

2 BENCHMARKING DEBT

TPCx-AI use cases are end-to-end ML workloads consisting of data loading, preprocessing, training, and inference of an ML algorithm, evaluated by a performance metric. Each of these stages, as well as their combination, can introduce unforeseen challenges, i.e., benchmarking debt that prevents the SuT from exhibiting realistic performance. In this paper, we recognize two types of debt based on their source: data- and workload-related debt.

2.1 Data-related Debt

The TPCx-AI dataset is generated according to a specification that defines a set of realistic properties for each use case. These properties change when multiple data tables are joined and grouped. Such changes are not trivial to detect due to differences in the distributions and sample sizes across the tables. We analyze discrepancies between individual and joined tables across use cases to identify common data-related debts in the TPCx-AI dataset. We define *data-related debts (DRDs)* as properties or intrinsic data characteristics not outlined in the specification that impact the use case runtime or model performance. We recognize several data-related debts:

DRD 1. *Data properties impacting the workload runtime.* The properties of the raw datasets, such as the sample size, their cardinality, and label distributions, are defined by the specification. Preprocessing operations and sampling techniques significantly change the data size and label distribution, affecting the runtime of all workload stages. We observe discrepancies in the data size between the generated and loaded data, as well as significant changes in the data size following the `preprocessing` substage. The changes are reflected in the number of samples, feature space, and label distribution, impacting the duration of the preprocessing or training.

DRD 2. *Discrepancies between the specification and the generated data.* The specification of the data generator defines the dataset properties. We observe that discrepancies in the properties between the specification and the generated data can lead to performance degradation and impact the benchmark quality.

DRD 3. *Data properties impacting the model target metric.* The model quality thresholds of the use cases are defined based on the label distribution, task complexity, and expected performance. The label distribution in the training set impacts the setup of the training task; the distribution of the scoring data needs to resemble that in the training data to achieve the defined threshold. We observe discrepancies in the label distributions between the training and scoring datasets, which impact the model performance.

DRD 4. *Data errors.* TPCx-AI's dataset is generated from a predefined specification. We observe formatting errors and semantic errors. Formatting errors occur during the loading stage when the dataset shape changes. We observe tables with undefined columns, leading to incorrectly imputed values, and tables with corrupted rows, leading to incorrect samples. The semantic errors include samples with the same identifier but different labels.

2.2 Workload-related Debt

While the use cases are designed to evaluate the capabilities of the SuT, there are discrepancies in the specification and implementation of the workloads. Clause 5.2.2. in the TPCx-AI specification states: *"If there is a conflict between the TPCx-AI Specification and the TPCx-AI kit, the TPCx-AI kit implementation prevails [12]."* Despite this, such occurrences introduce ambiguity to the workload and impact the performance. We define these deficiencies as *workload-related debts (WRDs)* and group them into the following categories:

WRD 1. *Inconsistent data inputs.* The data tables associated with each use case are defined in the specification. We observe discrepancies between the specified tables and those read in the implementation, affecting the loaded data size and the end-to-end runtime.

WRD 2. *Preprocessing operations not following the specification.* TPCx-AI's specification provides an overview of the data preprocessing steps for each use case. We observe that the specification lacks some of the implemented operators. Discrepancies in the preprocessing operations alter the workload complexity, affecting the runtime and the overall model performance during training and serving. WRD 2 is related to DRD 2, in which discrepancies in the data properties also affect the overall workload quality.

WRD 3. *Cross-stage operators.* Preprocessing operators are implemented in the training and inference stages of each use case. We observe scenarios in which preprocessing steps are distributed across multiple stages. Joining data during the loading stage alters the characteristics of the input tables. Moving oversampling, scaling, and vectorization operators within a training pipeline removes details about the feature space and the label distribution of the final training data. While invoking operators across methods does not impact the benchmark quality, it reduces its interpretability.

WRD 4. *Metric discrepancies in the driver and specification.* Each use case defines a quality metric matching the ML task. A discrepancy in the specified metric and the implementation impacts the convergence rate of the model, resulting in different training times.

WRD 5. *Low quality thresholds.* The quality metrics have defined thresholds that determine the success of the training stage. A low quality threshold influences the training runtime and can be reached by simplistic solutions that do not reflect the task complexity.

WRD 6. *Algorithms not suitable for the workload.* The choice of learning algorithms in the use cases is based on data properties, such as number of samples and features, label distribution and type, and sparsity. Utilizing an algorithm that does not match these properties can lead to long runtimes and poor model performance.

2.3 Performance Implications

The benchmarking debts impact performance, such that each debt implicitly affects multiple benchmarking aspects. The workload runtime is affected by debts stemming from changing data sizes (DRDs 1 and 4, WRDs 1), debts related to hardware utilization (WRDs 2-3), as well as the choice of learning algorithm (WRD 6).

The workload runtime is also impacted by debts stemming from specification discrepancies (DRD 2, WRDs 1-4) that introduce ambiguities and require consolidations of the implementation and specification. Adjusting the workloads according to the specification changes the input and execution artifacts, thus affecting the overall runtime and the achieved model performance.

Model performance is affected by data and algorithm properties that impact the workload scaling properties and the runtime required to achieve the threshold performance (DRD 3, WRDs 5, 6). Utilizing data properties such as the label distribution or the scaling properties of the learning algorithm to optimize hardware throughput will impact the runtime; however, it might jeopardize the validity of the underlying machine learning task. We quantify the impact of the benchmarking debts on the end-to-end throughput, AIUCpm, in Section 4.4.

3 TPCX-AI OVERVIEW

While the initial TPCx-AI paper [11] and the benchmarking specification [12] provide a conceptual overview of the benchmarking run, we focus on data and workload properties that impact its effectiveness and performance. In this section, we present the data model and benchmarking stages, and outline the data and workload vulnerabilities that lead to benchmarking debt in each use case.

3.1 Data Model

The data model of the TPCx-AI dataset is specified in the schema configuration file. It consists of 14 relations that include structured, numerical tables, as well as text, audio, and image data (Figure 1). The numerical and text data are generated as CSV files, audio files are stored in WAV format, and images are generated as JPG files.

The schema consists of independent tables used exclusively in one or more use cases, as well as related tables that, when joined, generate a comprehensive dataset to address the machine learning task. Individual tables, such as *Product_Reviews*, *Marketplace*, and *Failures* are used in Use Cases 4, 5, and 6, respectively. Use Cases 1, 3, 7, 8, and 10 join subsets of the remaining tables to generate the training, serving, and scoring datasets. The tables store the numerical and text data in the schema format in persistent storage. The images and audio files are stored as raw files in persistent storage. The unstructured data are accompanied by metadata tables specifying the file identifiers, filesystem paths, and labels. Use Cases 2 and 9 join the audio files and images with their metadata tables.

The properties of the generated tables, such as the feature ranges, their datatypes, and distributions, are predefined in an immutable configuration file based on the TPCx-AI specification. Fixing the data properties and the random seeds enables reproducible and scalable data generation. However, by joining several tables, the resulting dataset might lose the desired initial properties, resulting in unforeseen data-related debt. To define the data properties that accumulate the data-related debt, we analyze whether the generated tables comply with two aspects: the benchmarking specification and the workload task. Non-compliance with the specification might be a result of data errors (DRD 4) or incomplete data definitions.

To comply with the workload task definition, the tables need to have a consistent feature space and label distribution across the benchmarking stages. Changes throughout the benchmarking run due to joining multiple tables and preprocessing operators result in workload-related debt (WRDs 1, 2, and 6).

3.2 Benchmark Stages

A TPCx-AI run consists of six stages: data generation, data loading, power training, serving, throughput, and scoring tests (see Figure 2). We describe each stage and divide it into logical substages[1].

Data Generation. TPCx-AI uses a parallel data generator for generating its data schema at the beginning of the benchmark run [35]. The schema and the table characteristics are defined in XML specification files defining the data fields, their ranges, distributions, and table dependencies. The data generation is focused on parallelism and efficiency. Discrepancies between the generated data and the provided specification result in data errors, specifically DRD 4. The errors in the generated data are among the main factors that contributing to WRDs 2, 4, and 5. The runtime of the data generation is not included in the benchmarking metric.

Data Loading. During the LOADING STAGE, the data is transferred to the file system of the node running the use case. In the single-node setup, this includes a file copy operation to the defined input folder, whereas in the multi-node setup, this stage copies the data to a distributed file system. The stage consists of parallel copy operations in the local or distributed filesystems, with each copy operation representing a logical substage. The runtime of the loading stage is included in the benchmark metric.

[1]We write substages with `true type` and benchmarking stages with SMALL CAPITAL.

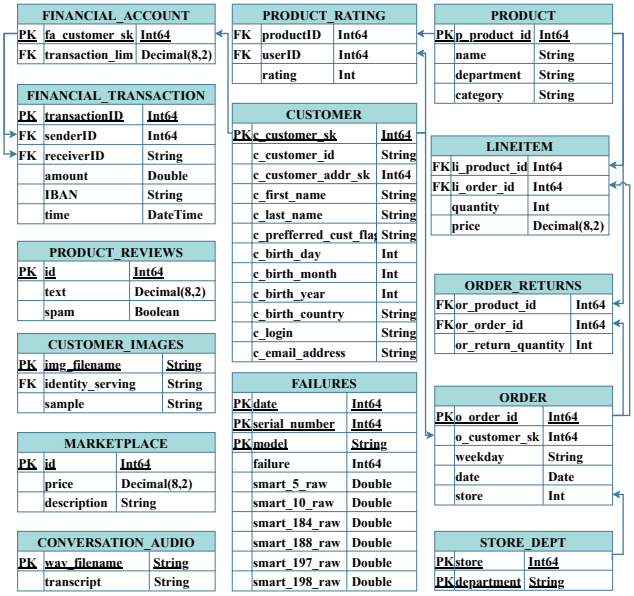


Figure 1: TPCx-AI data model.

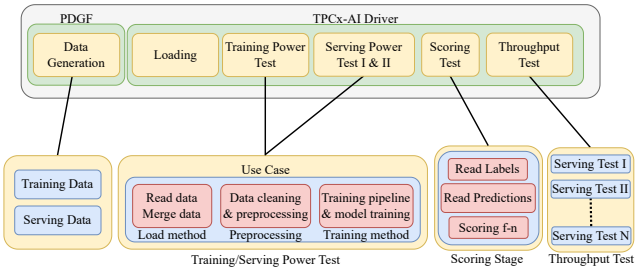


Figure 2: Benchmark stages & their substages in TPCx-AI.

Power Training Test. The POWER TRAINING TEST executes all use cases in a sequence. The outputs are persisted ML models, which are used in the POWER SERVING and THROUGHPUT tests. It is the longest stage of the benchmark, and its runtime significantly impacts the benchmarking quality metric (see Figure 3a). In the use cases, the training stage also takes the longest (see Figure 3b).

In the POWER TRAINING TEST, the generated dataset is loaded, preprocessed, and used to train ML models. These steps are implemented as separate methods in the workload, while the loading and preprocessing operators are shared across methods. To provide a clear distinction between the substages, we distinguish between: `data loading`, `data preprocessing`, and `model training`.

During the `loading` substage, the data is read into individual dataframes and merged into one. The data merging includes join sequences and column filtering, which, for some use cases (*UCs 1, 3*), result in significantly reduced data size compared to the individual table size (up to two orders of magnitude). Such data reductions are not reported in the benchmark specification, resulting in DRD 1.

The `preprocessing substage` consists of data cleaning, feature engineering, and sampling operators. While distributing the preprocessing operators across methods does not affect the runtime of the use cases or their model performance, it results in findings that are misleading when analyzing the workload runtime. When a subset of TPCx-AI's artifacts, such as the preprocessed data or entire workloads, is used to evaluate novel systems, it can lead to wrong evaluation assumptions and inconclusive results (WRD 3).

By dividing the TRAINING stage into logical substages, we isolate the model training in its own `substage`, so we can analyze its impact on the use case runtime and performance. The model parameters are predefined for each use case, and in most use cases, the framework's default values are used. This is in line with TPCx-AI's goal to provide a generic benchmarking framework.

The `training` substage reflects the debts accumulated from the `loading` and `preprocessing` substages. Specifically, DRDs 1, 2, and 4 influence the workload runtime, performance, and quality metric (WRDs 1, 4, and 5). The choice of the learning algorithm generates WRD with respect to the workload and hardware scalability. We observe that half of the use cases are bottlenecked by the training method (*UCs 2, 4-7*), while the other half are bottlenecked by the loading (*UCs 1, 3*) or the preprocessing stage (*UCs 8-10*) (see Figure 3c). DRDs 1,3, and 4 lead to such runtime distribution.

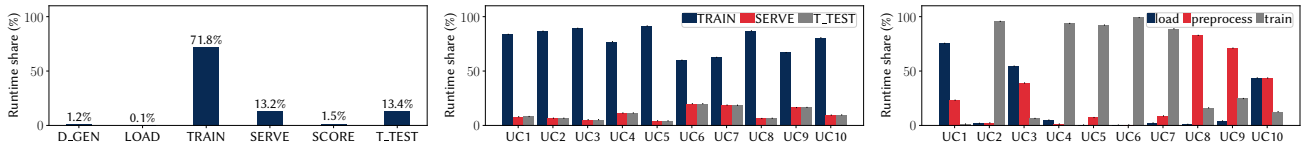
Table 1: Quality metrics per use case.

UC	Quality Metric	Threshold
1	K-Means Clusters	NA
2	Word Error Rate	0.50
3	Mean Squared Logarithmic Error	5.40
4	F1 Score	0.65
5	Root Mean Squared Logarithmic Error	0.50
6	Matthews Correlation Coefficient	0.19
7	Mean Absolute Error	1.80
8	Accuracy	0.65
9	Accuracy	0.90
10	Accuracy	0.70

Power Serving Test. The serving test measures the sequential throughput performance of the trained models across the 10 use cases. The serving dataset contains 10% of the training samples. The `loading` and `preprocessing` substages are identical to the POWER TRAINING TEST and include debts DRDs 1,3, and 4.

The `serving` substage loads the trained model and evaluates the serving data sequentially. The model is loaded into memory, and two serving tests are run in sequence. The complexity of the algorithm and the model performance impact the serving runtime, potentially incurring WRDs 5 and 6.

Scoring Test. The SCORING stage consists of data generation, serving test, and a scoring function. First, the data generator creates a labeled dataset with a fixed size independent of the scale factor. Next, a SERVING stage starts using the generated dataset. In the SCORING stage, the generated labels are compared with the ground truth, and a quality metric is calculated. Each use case metric is based on the machine learning task (see Table 1). The SCORING stage determines the validity of the use cases in the benchmarking run. There are several debts that impact the success scenario of the use cases: specifically DRDs 1 & 3, and WRDs 4, 5, and & 6.



(a) Benchmark stage breakdown. **(b) Use case stage runtime.** **(c) Training stage breakdown.**

Figure 3: Runtime distribution and breakdown of TPCx-AI.

Table 2: Observed benchmark debt per use case

Debt	Use Cases									
Data	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1. Runtime	✓	✓	✓	–	✓	–	–	✓	✓	✓
2. Model	✓	–	✓	✓	–	–	–	✓	✓	✓
3. Specification	✓	✓	✓	✓	✓	✓	–	–	✓	✓
4. Data Errors	✓	–	–	–	✓	–	–	–	–	✓
Workload	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1. Data Inputs	✓	✓	✓	✓	✓	–	–	–	–	–
2. Specification	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
3. Operators	–	–	✓	✓	–	✓	–	✓	✓	–
4. Model	–	–	–	–	–	✓	✓	–	–	✓
5. Thresholds	✓	–	–	–	–	✓	–	–	–	✓
6. Algorithm	✓	✓	–	–	–	✓	✓	–	–	–
✓ : observed				– : not observed						

Throughput Test. The throughput test simulates a concurrent SERVING scenario, where a set of data streams is processed by the models in parallel. The default number of streams is set to two, and there is no upper limit imposed by the framework. The total time required to complete the throughput test is used to calculate the quality performance metric of the benchmark.

Verification & Benchmark Metric. TPCx-AI measures the AI use-cases-per-minute at a scale factor SF (AIUCpmSF):

$$AIUCpm@SF = \frac{SF * N * 60}{\sqrt[4]{T_{LD} * T_{PTT} * T_{PST} * T_{TT}}} \quad (1)$$

where SF denotes the scale factor, N - the number of use cases, and $T_{LD,\ PTT,\ PST,\ TT}$ the times for LOADING, TRAINING, SERVING, and THROUGHPUT TEST. The success of each use case and the full benchmark run is determined by the model quality metrics and their thresholds (see Table 1). The thresholds are chosen to reflect the complexity and the required performance of the use case. Using an inadequate metric or threshold leads to models that over- or under-perform, or workloads with misleading runtimes, WRDs 4-6.

3.3 Benchmarking Debt in Use Cases

In this section, we outline the ten TPCx-AI use cases and focus on their `substages` that contain benchmarking debt (see Table 2).

***UC1* - Customer Segmentation.** *UC1* aggregates data from the *lineitem*, *order_returns*, *order*, and *customer* tables to create clusters of customer profiles. The joins and aggregations in the `loading` and `preprocessing` substages significantly reduce the data size, affecting the benchmarking metric through the runtime (DRDs 1 and 3). The feature engineering operations are not listed in the specification (WRD 2). The customer table is not included in the workload, despite being listed in the specification (DRD 2 and WRD 1). The `training` substage invokes *k-means* with a predefined number of clusters, which can result in unreliable data separation (WRD 6). *UC1* does not specify a quality threshold (WRD 5).

***UC2* - Customer Conversation Transcription.** The use case emulates a speech-to-text transcription service. The raw data is generated with a single voice, resulting in a dataset with a narrow frequency range. RNN models such as DeepSpeech reach the best performance with a diverse set of inputs [17]. The narrow frequency range affects the generalization, resulting in DRDs 1 and 3. In a realistic ML scenario, the dataset needs to contain a wider set of frequencies for better transcriptions. While TPCx-AI does not prioritize model performance, the simplified workflow affects the end-to-end performance. Having a greater number of voices requires more complex data preparation, resulting in higher training and serving runtimes, leaving *UC2* vulnerable to WRDs 1, 2 and 6.

***UC3* - Sales Forecasting.** *UC3* case does sales forecasting consisting of multiple stores and several departments. The merged table is significantly smaller, leading to DRD 1 and affecting the overall training time and masking the model complexity. The operators executed in the `preprocessing substage` are not covered in the specification (DRD 2). The aggregation changes the distribution of the weekly sales numbers, affecting the predictive performance of the model (DRD 3). In the SERVING and SCORING stages, the features of the input data differ from the TRAINING input, which is not described in the specification, leaving *UC3* vulnerable to WRDs 1 and 3. We also observe discrepancies between the specified and implemented preprocessing operators (WRD 2).

***UC4* - Spam Detection.** This use case emulates a spam filter based on the *reviews* table. The `preprocessing` substage includes duplicate removal, and a pipeline consisting of count vectorization and a TF-IDF transformation integrated in the training method. The latter is not listed in the specification, thus obscuring the training data feature space (DRD 3). Storing the product reviews in a single table with the product and customer identifiers relates to WRD 1 and results in DRD 2. The tokenization and the preprocessing are split between the `preprocessing` and `training` substages. Splitting preprocessing steps into separate methods and integrating them with the training pipeline is a common practice. However, it makes the result reporting vulnerable to WRDs 2 and 3.

***UC5* - Price Prediction.** *UC5* emulates a product price estimation scenario based on product characteristics defined in the *marketplace* table containing product descriptions in free text. The `preprocessing` consists of the removal of duplicates and samples without product description. The product descriptions contain repetitive textual labels, adding redundancy and artificially

increasing the token frequency, resulting in DRDs 1, 3 and 4. The `preprocessing` substage tokenizes the textual information without reading the label or the identifier column. Tokenization is implemented in the preprocessing stage, improving workload transparency and simplifying analysis of the training dataset; however, it is not included in the specification (WRDs 1, 2).

***UC6* - Hardware Failure Prediction.** *UC6* predicts hard drive failures from hardware logs in the *failures* table. Failures are rare and oversampled in order to reach a uniform distribution. This is not listed in the specification (DRD 3, WRD 2), and it is integrated in the training pipeline (WRD 3). In the `training` substage, the dataset trains a Support Vector Machine (SVM) with a radial kernel. SVMs with complex kernels scale poorly for large datasets. Related work has confirmed the findings when using TPCx-AI [32]. An alternative algorithm would be more performant for larger scaling factors (WRD 4 and 6). The scoring dataset is highly skewed with sparse failure samples. While Matthews Correlation Coefficient (MCC) [28] is a suitable metric for evaluating imbalanced workloads, the data distribution allows for a low quality threshold (WRD 5).

***UC7* - Product Rating.** The use case emulates an online retail recommender system using the *product_rating* table. This use case has several inconsistencies that lead to WRDs 4 and 6. We observe discrepancies between the quality metrics and learning algorithms in the specification and the implementation. The metric is specified as the mean absolute error, but the median absolute error is used in the workloads. We observe a discrepancy regarding the learning algorithm where the implementation invokes the SVD model, while the specification lists an Alternating Least Squares model (WRD 2).

***UC8* - Trip Classification.** *UC8* classifies shopping trips based on customer habits. It uses the *order, lineitem, and product* tables. The `preprocessing substage` consists of several joins and aggregations. The `preprocessing substage` increases the number of columns by 10x, and is the longest running `substage` of the benchmark. Despite the computational intensity of the joins, the chained aggregations contribute to over 80% of the runtime, which are not listed in the specification (DRDs 1 and 2). We observe a discrepancy in the `substage` execution patterns. The resource-intensive `preprocessing` runs on a single thread, whereas the `training` and `serving` run on all threads by default (WRDs 2 and 3).

***UC9* - Facial Recognition.** This use case emulates a facial matching scenario. The dataset consists of multiple items per class, where the items are images of one person, with one person per class. In the `preprocessing substage`, the images are aligned to the center of the frame and their values are normalized to greyscale. The images used in facial recognition and classification workloads are augmented and modified to improve the robustness of the model in real-world scenarios. In *UC9*, we observe that the aligned images in the training data are not shifted, scaled, or augmented. This leaves the `preprocessing` substage incomplete and the model a subject of overfitting (DRDs 1, 2 and 3, WRDs 2, 5, and 6).

***UC10* - Fraud Detection.** *UC10* classifies fraudulent transactions based on transaction metadata. The transaction data contains ambiguous data samples. Transactions with the same identifier are labeled as fraud and non-fraud. We observe duplicates which are not filtered in the `preprocessing` substage (DRD 4). This information is lost in the `preprocessing` substage since the duplicate columns are dropped, leading to significant differences in the training data size, a mismatch with the specification, and impacting the model target metric (DRD 1, 2, and 3). The transactions dataset contains more than 90% of non-fraudulent samples. The quality threshold is marked at 70% classification accuracy, which can be surpassed by majority-class assignment, leading to WRDs 4 and 5.

4 EVALUATION

We evaluate the data and workload characteristics of the use cases that lead to the benchmarking debt discussed in Sections 2 and 3.

4.1 Experimental Setup

We use the benchmark to detect, evaluate, and discuss the properties incurring data and workload-related debt in each use case. We evaluate the single-node implementation of TPCx-AI on two representative hardware architectures: a CPU node with 2x AMD EPYC 7742 CPUs with 128 cores and 512 GB of RAM, and a GPU node with 2x Intel 8570C CPU with 112 cores, NVIDIA B200 GPU, and 512 GB of RAM. We run the use cases for the official TPCx-AI scale factors 1, 3, 10, and 30 and report the numbers for SF10. We observe low runtime variance across all experiments (less than 10%), which is negligible for the measured runtimes.

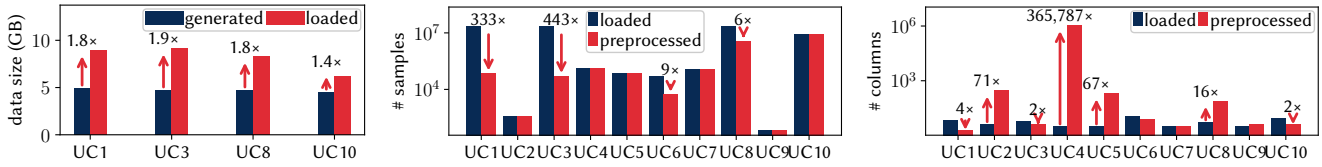
4.2 Data-related Technical Debt

In this section, we investigate the data properties that generate benchmark debt. We discuss the effects of characteristics on the use case quality metric and the benchmarking metric AIUCpm@SF.

Data Size Discrepancies - Generated and Loaded Data. During the loading and preprocessing stages, the data is merged and processed, changing its initial properties, such as the number of samples, features, and label distributions. For most of the use cases, the TPCx-AI specification lists the operations performed in each of these stages; however, the immediate implications on the resulting data are neither discussed nor can they be implicitly derived.

The data is transformed during the `loading` and `preprocessing` substages. In the first, the tables are read and merged into a single dataframe. During this phase, the size of the data is significantly changed. In Figure 4a, we present the differences between the sizes of the individual tables read in the workload and the size of the resulting table. We present *UCs 1, 3, 8, and 10*, which contain a join sequence during the `loading` substage. We omit the use cases that read a single table and do not alter the data at this stage. The presented use cases join four (*UCs 1 & 3*), three (*UC8*) and two (*UC10*) tables in the `loading` substage. We observe that for each of these use cases, the size of the loaded tables increases by up to 2× following the join sequence. The data changes the distribution of the features and labels, which impacts the predictive performance of the trained model and determines the success of the workload.

Specification Discrepancies - Data Preprocessing. Once the data is joined in a single dataframe, it is preprocessed to prepare it for the data training. The TPCx-AI specification provides an overview of the operators executed during the `preprocessing` substage. However, we observe discrepancies in the specification and the workload implementations regarding the listed preprocessing operators. We present use case examples that skew the runtime distribution of the `training` and `serving` stages, as well as omitting information regarding the dataset distribution (DRDs 1 and 2).



(a) Data size increase changes. **(b) Sample ratio in each UC.** **(c) Column ratio in each UC.**

Figure 4: Changes in the data shape and size following the loading stage.

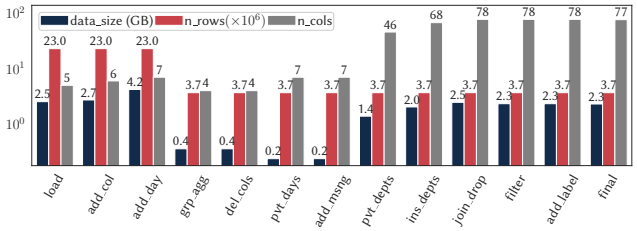


Figure 5: Changes in the *UC8* dataset - preprocessing stage.

UC1 specifies the removal of duplicates and nulls as the main operations of the `preprocessing` substage. However, it omits the operations that aggregate the features of the loaded dataset and are used as final features of the training dataset. These operations reduce the sample size of the preprocessed data by up to 333× (Figure 4b) and the feature space by 4× (Figure 4c). The feature engineering time accounts for up to 90% of the `pre-processing` substage and 18% of the overall TRAINING STAGE (Figure 3c).

We also observe transformations executed in the `training` substage in a training pipeline. *UC4* vectorizes the text and computes the TF-IDF feature vector as a part of the pipeline. These computations change the feature vector dimensions from a 3-feature dense representation to a sparse matrix with over 1 million columns (Figure 4c). Both operators are defined as training operators. However, both are also invoked during the SERVING and SCORING stages to transform the data before loading it into the model.

Transforming the feature space to a sparse representation and changing the column definition are data preparation steps and should be defined as such. It significantly changes the runtime distribution of the use case. The vectorization and computation of the TF-IDF feature transformation take up to 98% of the `training` substage in its current state and 57% of the overall training stage.

In the specification of *UC5*, the `preprocessing` substage consists of duplicate and null-value removal. The implementation of *UC5* includes a tokenization step that transforms the product description to individual features, changing the format and the shape of the training set. It results in a 67× increase in the number of features (Figure 4c). The product description tokens are the training features provided in the `training` substage, therefore, the tokenization step should be defined in the specification (WRD 2).

UC6 loads a hardware failure dataset with an imbalanced distribution of 97% of the samples (see Figure 6). The specification describes `preprocessing` operations, such as duplicate removal and column filtering, which do not alter the label distribution. However, as a part of the `training` substage, an ADASYN oversampling creates new samples from the imbalanced class until a uniform label distribution is reached (Figure 6 - *UC6*) and doubles the sample size of the training set. Oversampling the underrepresented labels is a common technique in imbalanced learning. It increases the runtime and impacts the model's performance. Similar to *UC4*, the transformation of the training samples and their label distribution should be defined as a data preparation step. In the specification, the oversampling is not listed (DRDs 1-3, and WRD 2).

The `preprocessing` substage in *UC8* is the longest stage in the benchmark run. The specification lists several data transformations, such as joining the input tables (in the `loading` substage), encoding categorical values, and aggregating purchased items. However, these operators do not reflect the changes in the data size and the feature space that occur during the `preprocessing` substage. In Figures 4b and 4c, we observe the transformation of the number of samples and columns, which resemble the operations listed in the specification. In Figure 5, we show that the operators not listed in the specification, such as the pivot of the departments and the addition of missing columns, change the properties of the training table (DRDs 1 and 2, WRDs 2).

Active Data Columns. As a result of the join sequences in the `loading` substage, the data loaded in memory has a significantly larger footprint resulting from joining full tables that add unused and redundant (duplicate) columns. After the `loading` stage, only a subset of columns is returned from the joined dataframe. This occurs in use cases that join three or more tables, such as *UCs 1, 3, and 8*, where only 58%, 46%, and 38% of the columns are kept.

In Figure 4c, we show the differences in the number of columns as a result of the data preprocessing and feature engineering steps. We observe that despite filtering a significant part of the columns in memory, there are occurrences of obsolete columns throughout the `preprocessing` substage as well. Specifically, *UC10* utilizes only half of the data columns during the `preprocessing` substage, and trains the model with only two features.

In this regard, we argue that the benchmark can benefit from more selective data loading and utilizing join operators that do not duplicate the join keys. This can significantly improve the runtime of *UCs 1, 3, 8, and 10*, which are bottlenecked by the `loading` and `preprocessing` substages, mitigating DRD 1 (see Figure 3c).

Label Distributions. The label distributions impact the convergence of the trained model, but also its predictive performance in the SERVING and SCORING stages (DRD 3). We analyze the label distributions across the loading, training, and scoring stages. We present the label distributions of *UCs 3-10* in Figure 6. We omit *UC1* since it is a non-supervised use case without a training label, and *UC2* where the labels are non-repeating sequences of words.

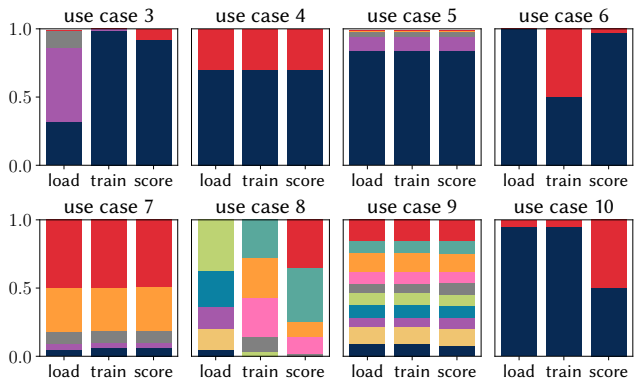


Figure 6: Label distributions of Use Cases 3-10.

For the regression use cases with a continuous label variable, we aggregate the values in five equidistant bins. This can be observed in *UCs 3, 5, and 7*. An overview of the bin distribution in these use cases shows a significant imbalance and change in the bin distributions between the three stages, as shown for *UC3*. The changes in the data distribution in the labels come from the grouping and aggregation of the samples to create a dataset that models the weekly sales for each store department over the past year. In *UCs 5 and 7*, distributions across the three stages do not change.

In the classification use cases (*UCs 4, 6, and 8-10*), we observe mixed label cardinality with use cases addressing binary classification tasks (*UCs 4, 6, and 10*), as well as multiclass classification tasks (*UCs 8 and 9*). In *UC6*, we observe a significant change in the label distribution between the loaded data and the training data, due to oversampling the underrepresented label to enhance the predictive performance of the model. We observe that the scoring data resembles the distribution of the loaded data, where more than 97% of the labels belong to one class. In *UC10*, we observe the opposite case. The label distribution of the loaded and training datasets is heavily imbalanced, with more than 95% of samples belonging to one class. The scoring dataset has a uniform label distribution, which results in lower accuracy when predicting the underrepresented class.

Since the `preprocessing` substage of the review data in *UC4* is targeted towards transforming the free text reviews into a vectorized feature, the label distribution remains consistent across the loading, training, and scoring stages. In the multi-class classification scenarios, we observe a consistent distribution across the three benchmarking stages (*UC9*) and a varying distribution in *UC8*. In the latter, due to the high number of distinct labels (38), we combine sets of neighboring labels in six label bins.

We observe that the distribution of the sets of labels in *UC8* differs significantly between the benchmarking stages. The same finding holds for the individual labels as well, the most-frequent class in the loading stage (label 8 with 13%) is represented with only 3% after the `preprocessing` substage and again with 13% in the scoring dataset. The `preprocessing` substage in *UC8* changes the sample and feature space significantly, which is also represented in the label distribution. We present an overview of the data changes per preprocessing operator in Figure 5. While the data size remains almost the same at approximately 2.5 GB for SF1, the shape of the data changes significantly, with a reduced number of samples by a factor of 6× and increased feature space by 15×. These transformations during the `preprocessing` substage are reflected in different label distributions between the loaded and preprocessed data.

Table 3: An example column shift in the *customer* table.

birth_country	login	cluster_id	new_col
Korea	Republic Of	6MRI7fvUtD	1
French Guiana	hyGbpY	1	unset_val

Feature Ranges. The feature distribution of the training data is the basis for tuning and learning the model during the `training` substage. The training and scoring data need to have similar feature distributions to successfully predict the labels. In TPCx-AI, the model quality is evaluated in three stages: the serving, scoring, and throughput tests. The serving and throughput test data come from a single unlabeled dataset that is utilized to measure the single and batch inference latency of the use case. The data is generated as a fraction of the training data, following the same feature distribution. The scoring data evaluates the predictive quality of the model. Based on a predefined quality metric and a threshold, it is used to determine the success of the workload. The feature distribution should resemble that of the training data.

In *UCs 6 and 10*, we observe significant discrepancies in the feature ranges between training and scoring datasets (Figures 7b and 7c), impacting the model performance (DRD 3). The oversampled failure data in *UC6* has a significantly different distribution compared to the scoring failure data. Their extreme ranges are the same, however, there is no overlap in the middle of the range, impacting the model quality, resulting in WRD 4 (see *Model Quality*).

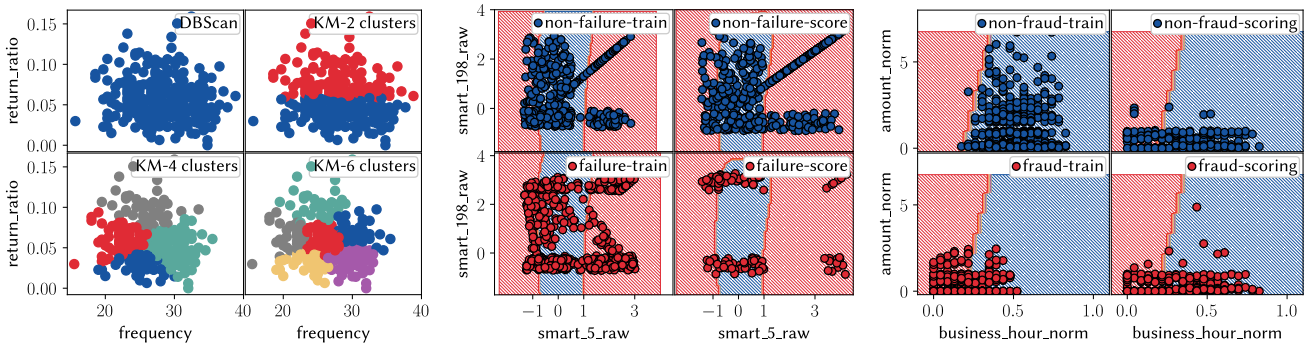
We observe a different discrepancy in *UC10*. The training non-fraud subset has 2.5× wider domain of the *amount_norm*, while the fraud subset has approximately 2× narrower range of *business_hour_norm* and is left-skewed. Both scoring subsets have very similar feature distributions, affecting DRD 2, and WRDs 4 and 5.

Errors in the TPCx-AI Dataset. TPCx-AI's data model provides a multi-modal data schema. Several tables contain a combination of structured numerical data, formatted strings, audio transcripts, images, and free text in the form of product reviews.

Errors in the data content result in shifted row and column values, as well as changed table dimensions (DRD 4). The frameworks used in TPCx-AI, such as pandas and numpy, adapt column shifts and table rows by adding new rows or columns and imputing them with null-values. Without an inspection of the loaded data in the workload, such discrepancies from the specification are difficult to detect since the workload will process the adjusted data.

In Table 3, we present an example of such a column shift found in the *customer* table used in *UC1*. This results from country names consisting of multiple names separated by a comma. The *cluster_id* field is filled with values from the adjacent *email_address* column, while the labels are shifted to a placeholder column, imputed with null-values. In the *marketplace* table, used in *UC5*, we observe a similar discrepancy between the sequential identifier and the table row count, a result of line endings not detected by the file reader.

Improvements of the TPCx-AI Dataset. The synthetic data used in TPCx-AI allows for a scalable end-to-end evaluation while



(a) *UC1.* (b) *UC6.* (c) *UC10.*

Figure 7: Feature ranges and decision boundaries in *UCs 1, 6, & 10.*

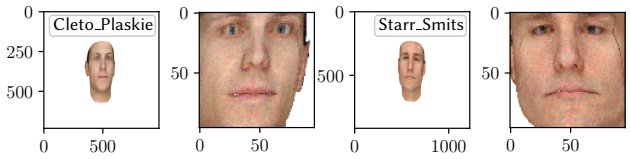


Figure 8: Image alignment in *UC9.*

preserving the specified data properties. One challenge when using a synthetic dataset is to create a realistic representation of the data. This is shown in the audio and image files used in *UCs 2 and 9*.

The conversation audio dataset used in *UC2* is generated with a single voice signature. While the use case manages to transcribe the text from the audio files, we argue that the use case can increase its realism by generating audio files with different voice signatures. The audio files also do not contain any noise or disturbances. An end-to-end evaluation of a speech-to-text transcription would benefit from a noise filtering step in preprocessing.

The `preprocessing` substage in *UC9* formats and aligns the images before loading them in the logistic regression model. Image alignment and other spatial transformations, such as image cropping, augmentation, and normalization, are some of the most common operations used to enhance the sample size and the model robustness. In *UC9*, we observe a center-alignment of the images, where a 96×96 pixel square is cropped from the image and provided as a training sample (Figure 8). A training set consisting only of such examples, without additional noise, limits the generalization of the model and increases the probability of overfitting.

4.3 Workload-related Technical Debt

In this section, we investigate the workload-related debt in the TPCx-AI use cases stemming from the workload implementation, the learning algorithms, and the specified model quality thresholds.

Inconsistent Data Inputs. In the `loading substage`, the individual tables are loaded in memory and merged into a single table. While most use cases process the same set of tables during the TRAINING, SERVING, and SCORING stages, we observe a mismatch between the tables listed in the benchmark specification and the tables read by the use cases, contributing to WRD 1. We observe such discrepancies in *UCs 1 and 3*, with different implications.

UC1 omits the *customer* table, resulting in a smaller dataset in the TRAINING and SERVING stages. When the customer table is included in the workload, the `loading` runtime is increased by up to 3.05 × (Figure 9b). Since *UC1* is bound by the data loading (Figure 3c), the data loading increases the TRAINING runtime by up to 2.2 × (Figure 9a). Systems evaluated with the default implementation of *UC1* will exhibit similar performance decrease [19, 20].

In *UC3*, we observe a mismatch in the tables used between the TRAINING stage and the SCORING and SERVING stages. Due to the model being trained on aggregated store information, the prediction requests in the serving and scoring stages are performed solely on the *store_dept* table, not executing the loading and preprocessing steps performed on the *order, lineitem,* and *product* tables. Since *UC3* is bottlenecked by the `loading` and `preprocessing` substages, omitting these steps impacts the overall runtime of the workload. While these steps are not essential for the correctness of the workload, they result in misleading short serving and scoring runtimes.

Model Quality. In Figure 7, we present the decision boundaries of the models in *UCs 1, 6,* and *10* and their sample distribution. Every use case in TPCx-AI reaches its defined performance threshold. However, our analysis shows that for some workloads, the thresholds, metrics, or algorithms are not suitable for the task.

In *UC1*, we observe two factors that affect the workload quality, the feature space of the training data and the choice of learning algorithm. The workload performs customer segmentation based on observed buying habits. The data is clustered with k-Means, and the workload defines four clusters. The provided dataset does not have any natural clusters. Since k-Means reads the number of centroids as an input parameter, it always returns the defined set of clusters. We expanded the use case to include two and six clusters, and we observe a clear separation of clusters despite having no separation between the individual data points (see Figure 7a, 2, 4, and 6 clusters). When executing the task with DBScan [15], a clustering algorithm that determines the number of clusters dynamically, we observe one cluster, which corresponds to the lack of spatial separation between the clusters (see Figure 7a, DBScan).

We inspect the decision boundary and the model performance for the two imbalanced use cases, *UC6* and *UC10*. In *UC6*, the distribution of the training dataset is changed by generating new samples of the underrepresented class (failing hardware components). The model in the `training` substage is trained with a uniform dataset

with a label distribution shown in Figure 7b. We show the label distribution of the training samples labeled as non-failure and failure on the top and bottom subplots, respectively. In the background, we show the decision boundary of the model against the two most significant smart features (*smart_5_raw* and *smart_198_raw*).

From the training sample distribution, we observe that the decision boundary of the trained SVM model shows a significant overlap between both class labels throughout their range. The model's radial kernel cannot generate a boundary that separates the two classes well. The failure samples in the scoring data have a different feature range from the training set. The overlap with the non-failure sample is significantly smaller; however, the precision is limited by the distribution of the training data.

In order to achieve higher model quality, the training and scoring data should share a similar distribution and feature range. We present the decision boundary based on the two most important features, however, we observe similar overlap when utilizing dimensionality reduction approaches such as Linear Discriminant Analysis (LDA) [40], Principal Component Analysis (PCA) [18], and Uniform Manifold Approximation and Projection (UMAP) [31].

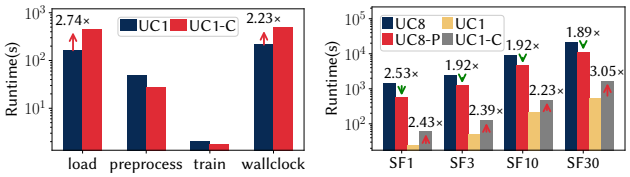
In *UC10*, we observe a discrepancy in the feature ranges between the underrepresented class labels in the training and scoring data (see Figure 7c). The decision boundary follows the distribution of the non-fraudulent samples in the training data. Given the skewed distribution, the model generates a curve that correctly predicts approximately 50% of the fraudulent samples. When applying the same model to the scoring data, the predictive performance drops for both classes. This is due to the significantly wider and narrower feature range for the fraudulent and non-fraudulent samples.

As a result, both classes share the same range distribution, posing a significant challenge for the linear model to make correct predictions. Despite differences in predictive performance between the training and scoring datasets, *UC10* meets the defined quality threshold. We discuss the choice of quality metrics and their thresholds under *Use Case Quality Metric* later in this section.

Workload Runtime - Data Preprocessing. The duration of the TRAINING and SERVING stages contributes significantly to the benchmark metric. The workload should enable an efficient execution of the individual stages to provide a fair comparison between the different SuTs and hardware configurations. The implementation of the use cases combines frameworks that utilize the hardware resources differently. This allows for TPCx-AI to evaluate the single- and multi-threaded performance, but makes full utilization of modern hardware challenging. We observe this behavior for use cases that are bottlenecked by the `loading` and `preprocessing` substages (*UCs 1, 3, 8, 10*). The performance gap is due to the single-threaded execution of the `loading` and `preprocessing` substages.

UC8 has the overall longest `preprocessing` substage. Bayer et al. observe the same bottleneck and propose a simple parallelization of the long-running aggregations [8]. While their evaluation is focused on edge-devices, we evaluate *UC8* on data-center servers for SFs 1-30 (Figure 9b). We observe up to 2.5× lower runtime of the TRAINING stage. The adjusted workload does not change the data layout or the execution order of the operators, but rather starts three parallel processes for sequential aggregation functions.

Workload Runtime - Learning Algorithm. The selection of learning algorithms impacts the runtime of the TRAINING and SERVING stages, as well as the THROUGHPUT TEST. The performance of the algorithm depends on several factors, such as data shape and size, their feature space, and the properties of the underlying hardware. We analyze the learning algorithms with regard to the data and workload properties. In this section, we present the impact of the algorithm selection on the use case runtime.



(a) *UC1* substage runtime. **(b) Runtime change in *UCs 8 & 1.***

Figure 9: Runtime differences upon modifying *UC8 & UC1.*

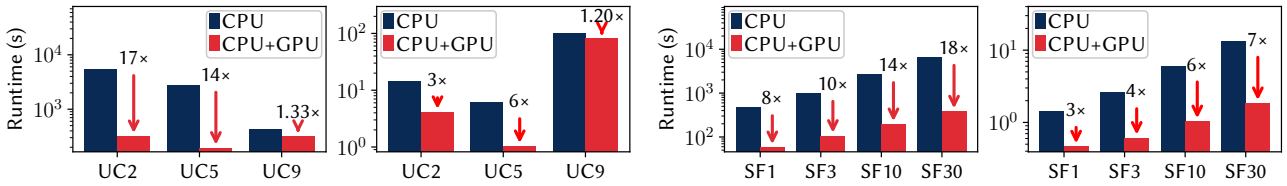
UCs 2, 5, and 9 train neural network models and embeddings, which benefit from using a GPU during the training and serving stages. *UC2* trains a convolutional neural network for 25 epochs to generate text transcripts. *UC5* trains a recurrent neural network for 15 epochs on the tensor vectors generated from the product descriptions. *UC9* updates an existing image recognition neural network with pre-trained weights to generate image embeddings and train a neural network with two dense layers for 15 epochs.

In Figure 10a, we present the runtime performance difference for the three use cases during the TRAINING and SERVING stages. Depending on the model complexity and the number of epochs, we observe that utilizing a GPU reduces the runtime of the training stage from 1.3× for *UC9* up to 17× for *UC2*. Given the similar complexity of the model and the feature space of the datasets between *UCs 2 and 5*, the differences in the speed-ups of 17× and 14× are due to the different number of epochs, whereas the average epoch runtime between the two architectures differs by less than 10%.

In the SERVING stage, we observe lower speed-ups, from 1.2× up to 6× for *UCs 9 and 5*, respectively (see Figure 10a). The runtime improvements are increasing with the scaling factor. In Figure 10b, we present the speedups for *UC5*, while we also observe similar trends for *UCs 2 and 9*. The hardware architecture significantly impacts the duration of the TRAINING and SERVING stages for the deep learning use cases, making TPCx-AI suitable to evaluate the benefits of running the benchmark on heterogeneous hardware.

Utilizing a learning algorithm that scales linearly with the data allows for higher throughput over increasing scaling factors, as well as more predictable performance trends. In *UC6*, we replace a radial kernel Support Vector Machine (SVM) with a random forest algorithm with 100 decision trees. The SVM training runtime increases by 35× from SF3 to SF10 and 38× from SF10 to SF30. The SVM serving runtimes increase by 25× and 29×, respectively, for the same scale factors. The runtime increase when using the random forest algorithm is 2.2× and 4.5× for training, while we observe no increase from SF3 to SF10 and a 2× increase from SF10 to SF30.

For reference, the training and serving data sizes change by 6× in both scaling scenarios, which is significantly closer to the increases shown by the RF implementation. While the training and serving times for SF1 for both algorithms are almost identical, the scaling characteristics result in differences of up to 346× for training and up to 796× for serving for SF30 (Figure 11). Patel et al. have not



(a) GPU's impact on the training (left) and serving (right) runtimes. (b) Scalability of training (left) and serving (right) runtimes in *UC5*.

Figure 10: Performance comparison of CPU and GPU execution in *UCs 2,5,* and *9*.

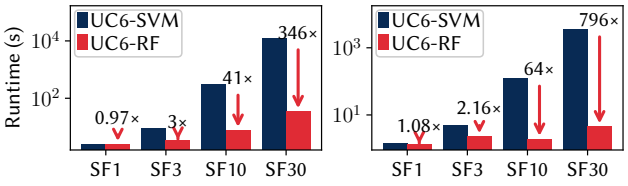


Figure 11: Algorithm impact on *UC6* training (left) and serving (right) runtime.

been successful in running *UC6* for scaling factors larger than SF10, while adjusting the algorithm would enable their analysis [32].

Use Case Quality Metric. The quality of each use case is evaluated by a different quality metric and a threshold. An adequate quality metric and its threshold are essential for determining the model quality. We observe discrepancies between the metrics listed in the specification and the ones in the workload implementation.

UC7 lists the median absolute error in the specification, whereas the implementation uses the mean absolute error. *UC6* measures the Matthew Correlation Coefficient; however, the specification lists the F1 score. In addition to the metric discrepancy, we observe a very low MCC threshold for the scoring data. While the MCC takes all members of the confusion matrix into consideration, we observe that the trained model achieves the required MCC score by correctly predicting only 40% of the failure samples.

UC10 measures the classification accuracy. The training data has highly skewed data, where 97% of the samples belong to one class. The threshold of 70% accuracy can be reached via a majority assignment given the label distribution. The scoring data for *UC10* has a uniform label distribution, requiring only 40% of correct predictions on one class to reach the threshold. Using metrics such as F1 Score or MCC would improve the model quality assessment.

4.4 Performance Implications on AIUCpm

In this section, we analyze the impact of the individual debts on the overall quality of the benchmark. We adjust and fix the use cases where applicable and measure the differences in the AIUCpm.

Data Inputs. We measure the impact of having invalid data inputs on the AIUCpm. We observe these debts across *UCs 1-4*, where in *UC1*, it is manifested by omitting a full table. We adjust the implementation to ingest the tables listed in the specification. We observe that due to the 2× longer runtime of the loading and preprocessing substage when ingesting all specified datasets (see Figure 9a), the total AIUCpm drops by up to 28% (Figure 12a). We observe a higher impact of WRD 1 when running scale factors 3 and 10 due to the larger share of *UC1* in the total runtime. On the other hand, SF1 and SF30 are bottlenecked by the `preprocessing` and `training` substages of *UC8* and *UC6*, respectively.

Hardware Impact. A successful benchmarking run of TPCx-AI assumes efficient utilization of the underlying hardware. With WRDs 2-4, we observe that the information on hardware utilization and its impact is not provided, even though some use cases are implicitly optimized for using a GPU (*UCs 2, 5, 9*), while others can benefit significantly from multi-threading (*UC8*). The GPU utilization is not defined in the specification, whereas the parallelization of *UC8* is neither specified nor implemented. Both impact the overall benchmarking performance significantly. We measure the impact on the AIUCpm by utilizing an accelerator in *UCs 2, 5,* and *9*, as well as parallelizing the `preprocessing substage` in *UC8*. We observe that by addressing WRDs 2-4, we can increase the AIUCpm by 1.7×–1.9× across all scaling factors (Figure 12b).

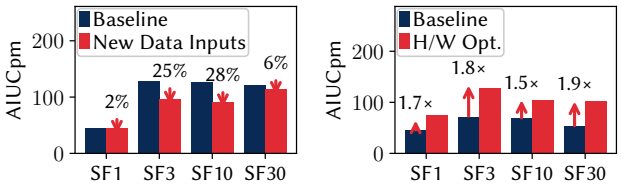
Learning Algorithms. The scaling properties of the learning algorithm increasingly impact the AIUCpm with larger scaling factors. We measure the runtime impact by utilizing a random forest instead of an SVM (Figure 11) on the overall AIUCpm. In Figure 13a, we observe that by adjusting the learning algorithm in *UC6*, we achieve an end-to-end AIUCpm increase of up to 87%.

End-to-end improvement. We measure the overall impact of the defined workload debts on AIUCpm. By addressing WRDs 1-6 in *UCs 1, 2, 5, 6, 8,* and *9*, their aggregated impact results in an overall increase of up to 3.8× in the AIUCpm (Figure 13b). We observe that the impact of the individual adjustments discussed compounds and contributes independently to the overall increase in AIUCpm. Even though addressing WRD 1 marginally lowers the AIUCpm, fixing WRDs 2-6 increases the overall performance.

5 KEY TAKEAWAYS

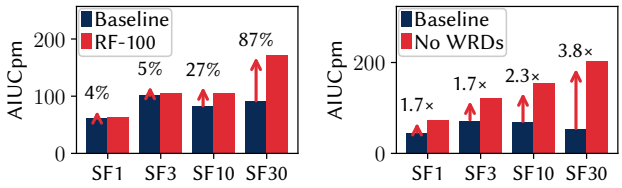
TPCx-AI's implementation is highly scalable and suitable for evaluating the system throughput of single- and multi-node deployments. However, the correctness and efficiency of the benchmark kit can be improved. Next, we present a set of improvements for the future development of TPCx-AI.

Specification Alignment. Even though the TPCx-AI implementation has precedence over the specification [12], we point out several instances where both should be aligned. The changes in the dataset size are a significant factor in the workload runtime and need to be explained in the specification. They contribute to DRDs 1, 3, and WRDs 1, 2. We observe discrepancies of up to 440× in the sample size and up to 365k× in the feature space (*UCs 1, 2, 3, 4, 5, 8*). We also observe differences between the specification and the



(a) New data lowers the AIUCpm. (b) Hardware impact on AIUCpm.

Figure 12: Impact of WRD 1 (a) and WRDs 2-4 (b) on AIUCpm.



(a) Using RFs increases AIUCpm. (b) Total debt impact on AIUCpm.

Figure 13: Impact of WRD 6 (a) and WRDs 1-6 (b) on AIUCpm.

workload implementation. The specification needs to be aligned with the data definitions (*UCs 1 and 3*), preprocessing operations (*UCs 1, 5, 6, 7, and 8*), and quality metrics definitions (*UCs 6 and 7*).

Efficient Hardware Utilization. The hardware utilization of the `preprocessing` and `training` substages is dependent on the execution models of the libraries used in the use cases. We observe that the `preprocessing` substage of *UC8* can be up to 4× slower due to the single-threaded execution of chained aggregations. On the other hand, TPCx-AI utilizes available accelerators such as GPUs. In *UCs 2, 5, and 9*, the utilization of the GPUs in the TRAINING and SERVING stages reduces the runtime from 1.2× up to 18× and results in up to 2× (WRD 1) higher AIUCpm. Parallelizing single-threaded stages and disclosing the optimization opportunities provided in the implementation can improve the overall benchmarking performance as well as the interpretability of the results.

Data Schema Improvements. We observe errors in the data loaded in *UCs 1, 3* (DRD 4) that change the column and row dimensions. These can be fixed by adjusting the data generator schema.

Data Variety and Preprocessing. The `preprocessing` substage for *UCs 2 and 9* omits commonly used preprocessing techniques. The audio files generated for *UC2* are transcribed with a single voice signature, even though the data generator can generate data with several voices. The audio files do not contain any noise. As a result, the workload omits noise filtering, a common feature in real-world speech-to-text transcription workloads. The `preprocessing` substage in *UC9* does not include image augmentation due to their synthetic nature. Despite the lack of image augmentation, *UC9* reaches 100% accuracy for SFs 1-30 (DRD 3, WRD 5). *UCs 2 and 9* require a more diverse dataset and a more elaborate preprocessing stage to accurately represent the task complexity.

6 RELATED WORK

We present related work on the adoption TPCx-AI for systems evaluation, end-to-end ML benchmarks, and TPC benchmark analyses.

Adoption of TPCx-AI. TPCx-AI has been used to evaluate a spectrum of machine learning system configurations based on the general characteristics of the workloads. Subset of use cases have been used to evaluate the performance of embedded GPU devices [8], analyze the performance of an ML system where the data is hosted in PostgreSQL [22], as well as optimize the performance of ML pipelines executed in feature stores [23]. TPCx-AI use cases are also used to evaluate data processing tools and code analysis libraries in data science [20, 26, 41]. Our analysis gives insight into the suitability of the data and the use cases for the scenarios of interest and system configurations. Future users can benefit from the insights to select suitable use cases, benchmarking stages, and scaling factors based on the properties and size of the datasets.

Machine Learning Benchmarks. MLPerf consolidates a set of benchmarks for several machine learning scenarios: data center training and inference [30, 36], data and model storage [1], edge device training and inference [5], as well as power consumption [39]. The MLPerf suite evaluates the time required to achieve a threshold performance on a predefined set of deep learning workloads. AISBench [13] is a benchmark that evaluates AI workloads on a wide set of metrics, including efficiency measures, fine-grained execution time analysis, training performance, resource utilization, and throughput. AISBench provides insights necessary to spot execution bottlenecks. Bayer et al. [7] propose an end-to-end performance benchmark focusing on stage reuse and modularity. TPCx-AI evaluates the end-to-end hardware throughput of systems running generic ML and deep learning workloads on synthetic datasets.

Analyses of TPC Benchmarks. Several benchmarking efforts from the TPC have addressed big data workloads in database systems [6, 34]. The frameworks are distributed either as a specification or an implementation-based express benchmarks. Related work analyzes and quantifies the chokepoints in known benchmarks [9, 14]. Characterization work of TPCx-AI has been so far in the context of scalability [32] and comparisons to previous benchmarks [24]. In this paper, we define the data characteristics in TPCx-AI that lead to general performance hurdles for individual use cases and the benchmarking run in general. We quantify their impact on the use cases, as well as the benchmark quality metric.

7 CONCLUSION

We conduct a detailed analysis of the TPCx-AI benchmark, focusing on data and workload-related sources of technical debt embedded in the benchmark. On studying the generated datasets across all benchmarking stages and evaluating the workload implementations, we identify several opportunities for improvement. These include improving the data quality, runtime performance and predictive accuracy of the workloads, as well as addressing discrepancies in the benchmark specification. Based on our findings, we identify opportunities for improving the benchmark quality, as well as highlight the debt implications on related work.

ACKNOWLEDGMENTS

This work was partially funded by the German Research Foundation (ref. 414984028 and ref. 556566056) and supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant RGPIN-2024-04657.

REFERENCES

[1] [n.d.]. *Benchmark MLPerf Storage | MLCommons V1.1 Results.* https://mlcommons.org/benchmarks/storage/

[2] [n.d.]. *Benchmark MLPerf Training | MLCommons Version 2.0 Results.* https://mlcommons.org/benchmarks/training/

[3] [n.d.]. *TPCx-AI Results.* https://www.tpc.org/tpcx-ai/results/tpcxai_results5.asp?version=1

[4] [n.d.]. *Wikipedia Data 01.01.2020.* https://github.com/mlcommons/training/tree/master/language_model/tensorflow/bert

[5] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. 2021. MLPerf Tiny Benchmark. https://doi.org/10.48550/arXiv.2106.07597 arXiv:2106.07597 [cs]

[6] Chaitanya Baru, Milind Bhandarkar, Carlo Curino, Manuel Danisch, Michael Frank, Bhaskar Gowda, Hans-Arno Jacobsen, Huang Jie, Dileep Kumar, Raghunath Nambiar, Meikel Poess, Francois Raab, Tilmann Rabl, Nishkam Ravi, Kai Sachs, Saptak Sen, Lan Yi, and Choonhan Youn. 2015. Discussion of BigBench: A Proposed Industry Standard Performance Benchmark for Big Data. In *Performance Characterization and Benchmarking. Traditional to Big Data*, Raghunath Nambiar and Meikel Poess (Eds.). Vol. 8904. Springer International Publishing, Cham, 44–63. https://doi.org/10.1007/978-3-319-15350-6_4

[7] Robert Bayer, Ties Robroek, and Pinar Tözün. 2025. Towards A Modular End-To-End Machine Learning Benchmarking Framework. In *Proceedings of the 3rd International Workshop on Testing Distributed Internet of Things Systems* (Rotterdam, Netherlands) *(TDIS '25)*. Association for Computing Machinery, New York, NY, USA, 23–26. https://doi.org/10.1145/3719159.3721223

[8] Robert Bayer, Jon Voigt Tøttrup, and Pınar Tözün. 2023. TPCx-AI on NVIDIA Jetsons. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Vol. 13860. Springer Nature Switzerland, Cham, 49–66. https://doi.org/10.1007/978-3-031-29576-8_4

[9] Peter Boncz, Thomas Neumann, and Orri Erling. 2014. TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In *Performance Characterization and Benchmarking*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Raghunath Nambiar, and Meikel Poess (Eds.). Vol. 8391. Springer International Publishing, Cham, 61–76. https://doi.org/10.1007/978-3-319-04936-6_5

[10] Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Nazanin Mohammadi Sepahvand, Edward Raff, Kanika Madan, Vikram Voleti, et al. 2021. Accounting for variance in machine learning benchmarks. *Proceedings of Machine Learning and Systems* 3 (2021), 747–769.

[11] Christoph Brücke, Philipp Härtling, Rodrigo D Escobar Palacios, Hamesh Patel, and Tilmann Rabl. 2023. TPCx-AI - An Industry Standard Benchmark for Artificial Intelligence and Machine Learning Systems. *Proceedings of the VLDB Endowment* 16, 12 (Aug. 2023), 3649–3661. https://doi.org/10.14778/3611540.3611554

[12] Transaction Processing Performance Council. 2022. *TPCx-AI.* https://tpc.org/tpcx-ai/default5.asp

[13] Jian Dong, Wei Bao, Xiaoqi Cao, Yang Xu, Yuze Yang, Binbin Li, Qi Zhang, and Heng Ye. 2025. AISBench: An Performance Benchmark for AI Server Systems. *The Journal of Supercomputing* 81, 2 (Jan. 2025), 441. https://doi.org/10.1007/s11227-024-06778-3

[14] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. 2020. Quantifying TPC-H Choke Points and Their Optimizations. *Proceedings of the VLDB Endowment* 13, 8 (April 2020), 1206–1220. https://doi.org/10.14778/3389133.3389138

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon) *(KDD'96)*. AAAI Press, 226–231.

[16] Hong Guan, Saif Masood, Mahidhar Dwarampudi, Venkatesh Gunda, Hong Min, Lei Yu, Soham Nag, and Jia Zou. 2023. A Comparison of End-to-End Decision Forest Inference Pipelines. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 200–215. https://doi.org/10.1145/3620678.3624656

[17] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition. arXiv:1412.5567 [cs.CL] https://arxiv.org/abs/1412.5567

[18] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. *An Introduction to Statistical Learning: with Applications in R.* Springer Publishing Company, Incorporated.

[19] Michael Jungmair, Alexis Engelke, and Jana Giceva. 2024. *Artifact for "HiPy: Extracting High-Level Semantics From Python Code For Data Processing".* https://doi.org/10.5281/zenodo.13323059

[20] Michael Jungmair, Alexis Engelke, and Jana Giceva. 2024. HiPy: Extracting High-Level Semantics from Python Code for Data Processing. *Proceedings of the ACM on Programming Languages* 8, OOPSLA2 (Oct. 2024), 736–762. https://doi.org/10.1145/3689737

[21] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*. Springer, 740–755.

[22] Leonhard Liu and Patrick K. Erdelt. 2025. Benchmarking Machine Learning Pipelines in PostgreSQL with TPCx-AI. In *Performance Evaluation and Benchmarking: 16th TPC Technology Conference, TPCTC 2024*. 118–133. https://doi.org/10.1007/978-3-031-93858-0_8

[23] Rui Liu, Kwanghyun Park, Fotis Psallidas, Xiaoyong Zhu, Jinghui Mo, Rathijit Sen, Matteo Interlandi, Konstantinos Karanasos, Yuanyuan Tian, and Jesús Camacho-Rodríguez. 2023. Optimizing Data Pipelines for Machine Learning in Feature Stores. *Proceedings of the VLDB Endowment* 16, 13 (Sept. 2023), 4230–4239. https://doi.org/10.14778/3625054.3625060

[24] Yingrui Liu Olesiuk, Miro Hodak, David Ellison, and Ajay Dholakia. 2023. More the Merrier: Comparative Evaluation of TPCx-AI and MLPerf Benchmarks for AI. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Nature Switzerland, Cham, 67–77. https://doi.org/10.1007/978-3-031-29576-8_5

[25] Rachel Longjohn, Markelle Kelly, Sameer Singh, and Padhraic Smyth. 2024. Benchmark Data Repositories for Better Benchmarking. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Article 2744, 23 pages.

[26] Weizheng Lu, Kaisheng He, Xuye Qin, Chengjie Li, Zhong Wang, Tao Yuan, Xia Liao, Feng Zhang, Yueguo Chen, and Xiaoyong Du. 2024. Xorbits: Automating Operator Tiling for Distributed Data Science. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 5211–5223. https://doi.org/10.1109/ICDE60146.2024.00392

[27] Weizheng Lu, Chao Hui, Yunhai Wang, Feng Zhang, Yueguo Chen, Bao Liu, Chengjie Li, Zhaoxin Wu, and Xuye Qin. 2025. Decentralized Actor Scheduling and Reference-Based Storage in Xorbits: A Native Scalable Data Science Engine. *Proc. VLDB Endow.* 18, 9 (May 2025), 2955–2963. https://doi.org/10.14778/3746405.3746420

[28] B.W. Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405, 2 (1975), 442–451. https://doi.org/10.1016/0005-2795(75)90109-9

[29] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Atsushi Ike, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Tsuguchika Tabaru, Carole-Jean Wu, Lingjie Xu, Masafumi Yamazaki, Cliff Young, and Matei Zaharia. 2020. MLPerf Training Benchmark. arXiv:1910.01500 [cs.LG] https://arxiv.org/abs/1910.01500

[30] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. 2020. MLPerf Training Benchmark. *Proceedings of Machine Learning and Systems* 2 (March 2020), 336–349.

[31] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. 2018. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software* 3, 29 (2018), 861. https://doi.org/10.21105/joss.00861

[32] Hamesh Patel, Kacper Ufa, Sammy Nah, Amandeep Raina, and Rodrigo Escobar. 2023. Preliminary Scaling Characterization of TPCx-AI. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Nature Switzerland, Cham, 78–93. https://doi.org/10.1007/978-3-031-29576-8_6

[33] Yuchen Peng, Zhongle Xie, Ke Chen, Gang Chen, and Lidan Shou. 2025. Towards Automatic and Efficient Prediction Query Processing in Analytical Database. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. 2253–2266. https://doi.org/10.1109/ICDE65448.2025.00171

[34] Meikel Poess, Tilmann Rabl, and Hans-Arno Jacobsen. 2017. Analysis of TPC-DS: The First Standard Benchmark for SQL-based Big Data Systems. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, Santa Clara California, 573–585. https://doi.org/10.1145/3127479.3128603

[35] Tilmann Rabl, Michael Frank, Hatem Mousselly Sergieh, and Harald Kosch. 2011. A Data Generator for Cloud-Scale Benchmarking. In *Performance Evaluation, Measurement and Characterization of Complex Systems*, Raghunath Nambiar and Meikel Poess (Eds.). Vol. 6417. Springer Berlin Heidelberg, Berlin, Heidelberg, 41–56. https://doi.org/10.1007/978-3-642-18206-8_4

[36] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark

Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 446–459. https://doi.org/10.1109/ISCA45697.2020.00045

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115 (2015), 211–252.

[38] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "Everyone Wants to Do the Model Work, Not the Data Work": Data Cascades in High-Stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–15. https://doi.org/10.1145/3411764.3445518

[39] Arya Tschand, Arun Tejusve Raghunath Rajan, Sachin Idgunji, Anirban Ghosh, Jeremy Holleman, Csaba Kiraly, Pawan Ambalkar, Ritika Borkar, Ramesh Chukka, Trevor Cockrell, Oliver Curtis, Grigori Fursin, Miro Hodak, Hiwot Kassa, Anton Lokhmotov, Dejan Miskovic, Yuechao Pan, Manu Prasad Manmathan, Liz Raymond, Tom St John, Arjun Suresh, Rowan Taubitz, Sean Zhan, Scott Wasson, David Kanter, and Vijay Janapa Reddi. 2025. MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from Microwatts to Megawatts for Sustainable AI. https://doi.org/10.48550/arXiv.2410.12032 arXiv:2410.12032 [cs]

[40] Petros Xanthopoulos, Panos M. Pardalos, and Theodore B. Trafalis. 2013. *Linear Discriminant Analysis*. Springer New York, New York, NY, 27–33. https://doi.org/10.1007/978-1-4419-9878-1_4

[41] Chenyang Zhang, Junxiong Peng, Chen Xu, Quanqing Xu, and Chuanhui Yang. 2024. IMBridge: Impedance Mismatch Mitigation between Database Engine and Prediction Query Execution. In *Companion of the 2024 International Conference on Management of Data*. ACM, Santiago AA Chile, 456–459. https://doi.org/10.1145/3626246.3654754