TPCx-AI - An Industry Standard Benchmark for Artificial Intelligence and Machine Learning Systems

Christoph Brücke
bankmark
Germany
christoph.bruecke@bankmark.de

Philipp Härtling
bankmark
Germany
philipp.haertling@bankmark.de

Rodrigo D Escobar Palacios
Intel
Hillsboro, Oregon
rodrigo.d.escobar.palacios@intel.com

Hamesh Patel
Intel
Hillsboro, Oregon
hamesh.s.patel@intel.com

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

ABSTRACT

Artificial intelligence (AI) and machine learning (ML) techniques have existed for years, but new hardware trends and advances in model training and inference have radically improved their performance. With an ever increasing amount of algorithms, systems, and hardware solutions, it is challenging to identify good deployments even for experts. Researchers and industry experts have observed this challenge and have created several benchmark suites for AI and ML applications and systems. While they are helpful in comparing several aspects of AI applications, none of the existing benchmarks measures end-to-end performance of ML deployments. Many have been rigorously developed in collaboration between academia and industry, but no existing benchmark is standardized.

In this paper, we introduce the TPC Express Benchmark for Artificial Intelligence (TPCx-AI), the first industry standard benchmark for end-to-end machine learning deployments. TPCx-AI is the first AI benchmark that represents the pipelines typically found in common ML and AI workloads. TPCx-AI provides a full software kit, which includes data generator, driver, and two full workload implementations, one based on Python libraries and one based on Apache Spark. We describe the complete benchmark and show benchmark results for various scale factors. TPCx-AI's core contributions are a novel unified data set covering structured and unstructured data; a fully scalable data generator that can generate realistic data from GB up to PB scale; and a diverse and representative workload using different data types and algorithms, covering a wide range of aspects of real ML workloads such as data integration, data processing, training, and inference.

PVLDB Reference Format:
Christoph Brücke, Philipp Härtling, Rodrigo D Escobar Palacios, Hamesh Patel, and Tilmann Rabl. TPCx-AI - An Industry Standard Benchmark for Artificial Intelligence and Machine Learning Systems . PVLDB, 16(12): 3649 - 3661, 2023.
doi:10.14778/3611540.3611554

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611554

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://www.tpc.org/tpcx-ai/default5.asp.

1 INTRODUCTION

Machine learning (ML) has been shown to be beneficial in industry and academia. Tasks that were considered very complex to solve with software systems, such as image or voice recognition, can be solved with high accuracy with ML-based artificial intelligence (AI) solutions [34]. While many large companies have shown surprising results in various applications, this is hard to replicate for companies without well-funded and well-staffed AI departments. Today, improvements in accuracy and architecture come at costs that only major players can afford and that are not feasible or even reproducible for small scale deployments [11].

Contributing to this problem, current AI solutions are highly diverse and due to the great interest in the field, many new systems with different strategies and goals are developed. This creates a need for standardized methods to compare systems and solutions.

In the past, benchmarks have been pivotal for both, performance improvements in common applications and unification of interfaces for these applications. Previous database benchmarks by the Transaction Processing Performance Council (TPC) are a good example of this, which is shown by the many publications in database research that use these benchmarks as validation. Although they are a simplification of real world applications, the benchmarks serve as a common abstraction for use case scenarios and make systems and research comparable.

Researchers and practitioners have started several efforts to build similar benchmarks for AI domains. Most of these focus on the core ML training aspect, either from a micro-benchmark perspective, such as DeepBench [22] or evaluating the training of different ML models, e.g., MLBench [17]. While model training is at the heart of many AI applications, it does not reflect the various challenges AI applications face. More holistic approaches also incorporate serving[1] tasks, e.g., MLPerf [18], which allows for the evaluation of training or serving for a variety of ML models; or preprocessing and postprocessing stages, such as dcbench [7], which evaluates specific preprocessing tasks, such as data pruning.

[1]Serving is also commonly known as Inference in the AI terminology.

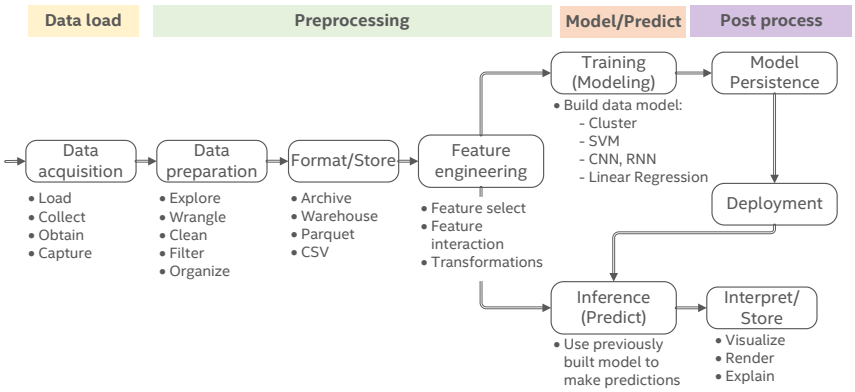

Figure 1: End-to-end ML pipeline

In this paper, we go beyond these approaches and introduce the TPC Express Benchmark for Artificial Intelligence (TPCx-AI), a benchmark for end-to-end AI systems. TPCx-AI is based on AD-ABench [29] and has recently been standardized by the TPC [4]. TPCx-AI's workload comprises 10 use cases, implemented as end-to-end AI processing pipelines that, unlike other AI benchmarks, include data ingestion, preprocessing, training, serving, and post-processing stages. Figure 1 shows the set of tasks involved in a typical end-to-end AI processing pipeline. TPCx-AI's use cases are based on diverse retail scenario tasks with various data types and ML models, both traditional and deep learning. We use a study by McKinsey Analytics [1] on potential use cases for AI and ML techniques in retail and other verticals as an inspiration for the workloads.

TPCx-AI is released with a full implementation, also known as the *toolkit* or simply the *kit*. This *kit* contains a synthetic data generator, based on the Parallel Data Generation Framework (PDGF) [30], a driver that manages the complete execution of the benchmark, as well as two reference implementations, one based on Apache Spark [38] and one based on established Python libraries. Other implementations can be added as well upon review and approval by the TPC. Despite being a young standard benchmark, there already exist 14 official benchmark submissions to the TPC, as of June 2023.

In this paper, we give an overview of TPCx-AI, the first industry standard benchmark for end-to-end AI and ML systems and make the following contributions:

- We give a detailed introduction to TPCx-AI. We describe its workload, data sets, run rules, and metrics.
- We describe the internal benchmark implementation and compare the Spark- and Python-based versions.
- We analyze the benchmark data set and workload both qualitatively and experimentally.

The rest of this paper is structured as follows. Next, we discuss related work. Section 3 gives an overview of the benchmark and describes the workload, data model, run rules, and metrics. We analyze the data set in Section 4. Section 5 presents the benchmark kit and the two workload implementations. In Section 6, we execute the benchmark and discuss the results of both implementations. We present future extensions of the benchmark in Section 7 and conclude with future work in Section 8.

2 RELATED WORK

In this section, we will give an overview of related work. For a recent survey on several machine learning, big data, and high performance computing benchmarks see also Ihde et al. [15].

Due to the high interest in ML, many benchmarks have been developed to evaluate ML systems and applications. These can be categorized in benchmark data sets, micro-benchmarks, benchmark suites, component benchmarks, and application-level benchmarks.

Benchmark data sets have a long history in ML research. Well know examples are ImageNet [5] and WordNet [20], as well as various collections of ML data sets, such as the UCI Machine Learning Repository[2], OpenML [36], and the Penn Machine Learning Benchmarks [31]. These data sets are typically used to evaluate the accuracy of ML algorithms and approaches rather than their performance. While this has helped tremendously to improve ML algorithms, it does not provide the full picture required in practical deployments, where a trade-off in accuracy and latency is always required. Several benchmark proposals are based on these data sets, e.g., MLBench [17] and MLPerf [18].

An example of a micro-benchmark is DeepBench [21, 22], which benchmarks basic deep learning operations, such as matrix multiplication, all-reduce, or convolutions. Another example is SLAB [35], a benchmark for linear algebra primitives. Like other micro-benchmarks, DeepBench and SLAB can give deep insights into performance behaviour of the underlying hardware for typical ML operations, but the results are hard to interpret for end-to-end scenarios in real world deployments.

Benchmark suites are collections of kernels, applications, tools, or other workload units, designed to represent certain workload scenarios in application domains. An example of a kernel based benchmark suite is High Performance Linpack[3] (HPL), a collection of kernels for numerical calculations to measure raw compute performance. Like many benchmark suites, HPL extensions have added ML workloads (e.g., HPL-AI[4]) to their portfolio. Another example is BigDataBench [8], which includes AI workloads in its current

[2]UCI Machine Learning Repository - https://archive-beta.ics.uci.edu/
[3]High Performance Linpack - https://www.netlib.org/benchmark/hpl/
[4]HPL-AI - https://icl.bitbucket.io/hpl-ai

version[5]. Some benchmark suites contain end-to-end application workloads, but are built to measure hardware performance rather than complete deployments.

Most research on AI has focused on ML training, which is also true for the majority of ML benchmarks. From a benchmarking perspective, this can be seen as one component (among several others) of the complete ML pipeline. Researchers have identified other tasks of ML pipelines to be relevant for end-to-end performance of ML deployments and systems [28, 37], and some benchmarks have been developed especially for those other tasks. Example of tasks are data cleaning, such as presented by CleanML [16], a benchmark for data cleaning and ML; AutoML Benchmark [10], which benchmarks automated ML. In addition, dcbench [7] is a benchmark that specifically focuses on ML pipeline stages other than training. Currently, the benchmark defines three components for data cleaning, pruning, and slicing.

Closest to TPCx-AI are MLPerf [18] and its predecessor DAWNBench [3], which both include training and serving tasks and proposed novel metrics for time to accuracy and time per serving. Both metrics are relevant for real world deployments and are included in the TPCx-AI metric. Unlike TPCx-AI, both benchmarks do not include data ingestion, preprocessing or postprocessing stages, and users can choose to evaluate only training or serving, or both. To the best of our knowledge, TPCx-AI is the only official industry standard benchmark, which covers end-to-end ML and AI workloads at an application level.

3 BENCHMARK OVERVIEW

In this section, we give an overview of the benchmark specification, which includes the workload (use case pipelines and data set), as well as the run rules and benchmark metrics.

Major design goals for the benchmark were that it is realistic, representative, and relevant for ML and AI deployments. Section 3.1 gives a brief introduction to TPC benchmarks in general. In Section 3.2 we describe the TPCx-AI data model. Section 3.3 gives a detailed description of the ten use cases and their pipelines. Section 3.4 explains run rules and the different measured and unmeasured tests of the benchmark. Finally, Section 3.5 and 3.6 define TPCx-AI's quality and performance metric *AIUCpm@SF*.

3.1 TPC Benchmarks

The Transaction Processing Performance Council (TPC) is a consortium with a history of more than 30 years developing successful benchmarks for data processing. Currently there are 10 active TPC benchmarks for different areas of data processing and data analysis. For instance, classical online analytical processing is benchmarked in TPC-H and TPC-DS [23, 24], data integration systems can be evaluated using TPC-DI [26], and TPCx-BB is a big data analytics benchmark [9], which also includes several machine learning tasks. TPC benchmarks are typically used by member companies (e.g. original equipment manufacturers or independent software vendors) to demonstrate and promote the features and capabilities of their hardware or software systems when processing certain types of workloads. Companies that submit benchmark results for official publication are also known as test sponsors.

Each TPC benchmark belongs to one of two categories: Express or Enterprise. Both Express and Enterprise benchmarks include a specification document when released. The specification document contains information about the benchmark design, run rules, as well as implementation details, among others. In addition to the specification document, Express benchmarks include a *kit* released by the TPC along with the benchmark's specification document [14]. Enterprise benchmarks, on the other hand, need to be implemented by the sponsor based on the specification document. The TPC introduced Express benchmarks in 2013 in an effort to make their benchmarks easier to adopt.

The following aspects are also integral to TPC benchmarks:

Audit requirement Before TPC benchmark results can be considered official they need be reviewed by an independent TPC certified auditor that will inspect the results thoroughly to confirm that all benchmark rules were followed and thus obtained under fair conditions to be compared to previous and future results.

System Under Test pricing In addition to reporting the performance score obtained after running the benchmark, users also need to report in detail the price of the system under test (SUT) and its components (software and hardware) used to obtain their performance score [13]. TPC benchmarks usually consolidate the price of the SUT under their price/performance metric that needs to be included in the results. This feature of TPC benchmarks allows for consumers to get a clearer perspective on the total cost of the system required to achieve a certain performance score. Recently, special clauses have been added to the pricing specification to permit running TPC benchmarks in the cloud rather than only in on-site deployments.

3.2 Data Model

The core of the TPCx-AI data model is relational, i.e., there are multiple tables with relationships to each other. Figure 2 illustrates the different schema parts and highlights the relationships between the tables as well as their usage by the individual use cases. The relational schema represents an analytical database with orders consisting of line items as common in data warehouses. Additionally, several other relational tables give more insight into customers, their financial transactions, products, and marketplace prices. Besides the structured data, there are multiple unstructured data sets in the model. This also includes text, as in product reviews, as well as, audio files and customer images. Each use case utilizes a subset of the data model as shown in Figure 2. All tables are generated scalably using the synthetic data generator PDGF [30], which we extended to enable image and audio generation.

The complete data set is scaled according to a scale factor (*SF*). Users specify the *SF* prior to running the benchmark to indicate the dataset size in GB for which they want to run the workload. Scale factors that can be used for official publications range from 1 GB up to several Petabytes in steps of 3 and 10 (i.e., 1, 3, 10, ...). The relational tables include common SQL data types, as well as text, and each table has between 2 and 12 columns. The audio files are WAV files and the images PNG files. Using synthetically generated

[5]BigDataBench Version 5.0 - https://www.benchcouncil.org/BigDataBench/files/BigDataBench5.0-User-Manual.pdf

Table 1: Use Case Overview

ID	use case	Type	Class	Data	Algorithm	DL
UC01	Customer Segmentation	Triaging	Clustering	Number	k-Means	
UC02	Call Transcription	Convert Unstr. Data	Classification	Audio	Convolutional Neural Network	X
UC03	Sales Forecasting	Forecasting	Regression	Number	Holt-Winters	
UC04	Spam Detection	Discover Anomalies	Classification	Text	Naïve Bayes	
UC05	Price Prediction	Predictive Analytics	Regression	Text	Recurrent Neural Network	X
UC06	Hardware Failure	Predictive Maintenance	Classification	Number	Support Vector Machines	
UC07	Product Rating	Hyper-Personalization	Recommendation	Number	Collaborative Filtering	
UC08	Classification of Trips	Triaging	Classification	Number	Gradient Boosted Trees	
UC09	Facial Recognition	Hyper-Personalization	Classification	Image	CNN & Logistic Regression	X
UC10	Fraud Detection	Discover Anomalies	Classification	Number	Logistic Regression	

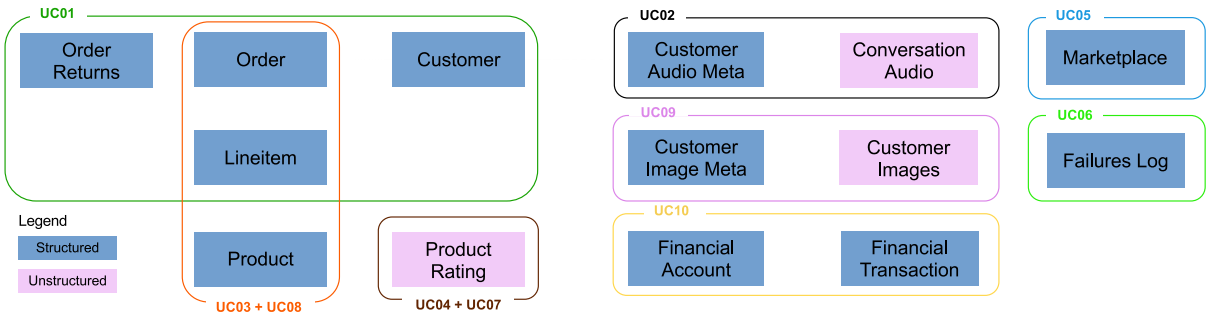

Figure 2: TPCx-AI Data Schema

data also mitigates the risks of running into ethical or privacy issues, and enables the generation of correct ground truth data. For example, the audio files are customer conversations, which are initially generated as text files and then converted to audio using a text-to-speech conversion tool. The same is true for user images, which are generated in a deterministic manner based on the user ID in different conditions using a face generator.

For the training, serving, and scoring tests of the benchmark, separate data sets are generated. This is done by facilitating the descriptive nature of a PDGF data generator. For all three data sets the same data description is used but with a different seed and scaling factor. Changing the seed will give a different data set but with the same structure and data distributions. We generate three separate data sets in order to prevent information leaks and minimize attack surface for unfair benchmark users. The training data set is used in the *Power Training Test* and contains all tables with all columns and hence all features necessary for training as well as the labels. The training data set grows proportionally to the scaling factor. The serving data set is to be used for serving and consist of all the tables and columns but *without* the labels. Because this is different from the training data set, simply memorizing the training data will not help during serving. The serving data set is used for the *Power Serving Test* as well as the *Serving Throughput Test*. The last data set is the scoring data set and is conceptually a combination of the latter two data sets. As a first step a data set containing only the features but none of the labels is generated and must be used during the *Scoring Test* but in the same environment as the *Power Serving Test*. Afterwards the labels are generated in the benchmark driver to calculate the quality metric.

The data distributions within the data sets are modeled after real data sets or carefully designed to have certain characteristics. Some attributes are highly skewed whereas others contain a lot of noise. A good example for a skewed data set is the *failures* event log. This contains log event for different hard drives, one per day, monitoring the health status of hard drive using S.M.A.R.T. data. For the training case the *failures* log also contains a flag indicating if a hard drive failed. Under real conditions a hard drive rarely fails and if so it mostly fails catastrophically, which means there is a single event out of thousands which is a failure. Examples for noisy data sets are the *financial_transactions* and *product_rating* relations. These contain fraudulent transactions and spam reviews. But they also contain transactions and reviews that share the same characteristics as the fraudulent or spammy ones but are in fact valid transactions and reviews. Adding such noise prevents the models from characterizing fraud or spam too aggressively.

3.3 Workload

TPCx-AI's workload consists of ten use cases, each covering a complete end-to-end ML pipeline as outlined by Polyzotis et al. [27], dealing with the technical challenges in ML systems [32]. An abstract ML pipeline can be seen in Figure 1. Unlike previous benchmarks, TPCx-AI's workloads cover all steps in this pipeline, although not all steps are covered in all pipelines. The use cases were inspired by real world use cases and based on an analysis by McKinsey Analytics [1]. The workload covers traditional techniques as well as deep learning models. Each use case solves a specific ML challenge. In the following, we give an overview of the ten use cases, which are also summarized in Table 1.

UC01 - Customer Segmentation. In the first use case, customers are segmented based on their shopping behavior. It uses k-means clustering on the customer, order, lineitem, and order_returns tables to identify different types of customers. In the preprocessing step, duplicates and null values are removed, then in the training, the customers are clustered unsupervised by k-means. The clusters are used while serving to assign new users to groups.

UC02 - Customer Conversation Transcription. In UC02, transcripts of user calls are generated using audio to text conversion. This is done using the Deep Speech recurrent neural network (RNN) model [12]. In the preprocessing step, the audio data is loaded and resampled to a common sample rate (16kHz). Then Mel Frequency Cepstral Coefficients (MFCC) are calculated from the raw audio and used to train the deep neural network infrastructure. During serving the same preprocessing steps are applied and the already trained model is used to generate transcriptions. The transcripts are then used to compute word error rates comparing to the ground truth data in the scoring test.

UC03 - Sales Forecasting. The objective in UC03 is to forecast the weekly sales for each department in each store given a limited history of sales data for the second to last year in the data set. The pipeline uses the tables orders, product, lineitem, and store_department. The use case utilizes an exponential smoothing technique (Holt-Winters) for time series forecasting. In the acquisition stage, the tables are loaded and joined, then aggregates for weekly sales are generated in preprocessing. The training fits the Holt-Winters model, which is used to forecast sales in the serving stage for up to one year per department per store. The scoring test checks the accuracy of the results.

UC04 - Spam Detection. In UC04, product reviews are searched for spam entries. This use case uses the product_rating text data. Our specification uses a classical Naïve Bayes model to identify spam entries. In the acquisition step, reviews are loaded and in the cleaning step, duplicates are eliminated. The text is transformed to n-grams and vectors in preprocessing and the model is fit during training. While serving, new reviews are classified as ham or spam and the resulting predictions are scored using Matthews correlation coefficient.

UC05 - Price Prediction. UC05 performs price predictions for individual retail items using a Recurrent Neural Network on the marketplace table. The prediction is based on the brand, product name, and description on an online market place. Initially, duplicates and null values are eliminated and the model is trained on the data. During serving, item prices are predicted and the predictions are scored using Root Mean Squared Log Error.

UC06 - Hardware Failure. In UC06 hardware failures of disk drives are predicted based on Self-Monitoring, Analysis, and Reporting Technology (SMART) data in the failures log data set. While this is not a pure retail use-case it is a typical infrastructure task and relevant in any larger hardware setup. The initial preprocessing performs duplicate and null value removal. A support vector machine is trained based on known data and failures. In the serving stage, impending failures are predicted and scored using the F-score metric.

UC07 - Product Rating. In UC07, cross-selling is improved by giving "next-to-buy" recommendations. Based on previously bought products, we give recommendations for products that the customer might also be interested in. The recommendation are found in the product ratings by comparing customers (by their products) and/or products (by their customers). The ratings are loaded and transformed into a numerical format without duplicates. In the training step, matrix factorization is used and an alternating least squares regression between the item and user matrix. Predictions for user item pairs are served and scored using mean absolute error.

UC08 - Classification of Trips. In a brick and mortar store setup, it is useful to classify shopping trips into trip types. Examples of trip types are: a customer may make a small daily dinner trip, a weekly large grocery trip, a trip to buy gifts for an upcoming holiday, or a seasonal trip to buy clothes. In data acquisition, the shopping history is generated by joining the tables order, lineitem, and product. Items are aggregated and categorical values in the data are binarized (after data cleaning). We train gradient boosted trees to predict the trip type in the serving stage based on active shopping sessions. The scoring computes the classification accuracy.

UC09 - Facial Recognition. In UC09, we train a classifier to recognize faces. This can be used to identify frequent customers or enable face identification systems[6]. The data set contains a set of computer generated face images for a subset of the customers. For each customer image, we also generate a foreign key to the customer table as meta data, to enable labeling and scoring. After loading the data in the acquisition step, the names are encoded and the images are aligned and resized. During training, a pre-trained embedding (a convolutional neural network) is fine-tuned for the customer images and a logistic regression model is trained on the embedding and the names of the customers. The serving step recognizes customer images and the accuracy is scored.

UC10 - Fraud Detection. In UC10, we cover a fraud detection use case. For this, there are separate financial transactions in the data set, which contain transaction time, transaction amount, sender, receiver, and transaction limits for accounts. Based on a predefined set of fraudulent transactions, a logistic regression model is trained for classification. The data is disaggregated and distributed in the tables financial_account and financial_transaction, which need to be joined and cleaned. The transaction data is normalized across the full tables before training the model. In the serving stage, transactions are classified and scored using classification accuracy.

3.4 Benchmark Tests and Run Rules

A complete benchmark run consists of several tests, some of which are timed and part of the measured result. The following tests are run in sequential order:

(1) Data Generation
(2) Load Test
(3) Power Training Test
(4) Power Serving Test I
(5) Power Serving Test II
(6) Scoring Test

[6] Given the ethical implications of facial recognition technologies UC09 will most likely be adapted to use object images instead of faces in future versions of TPCx-AI.

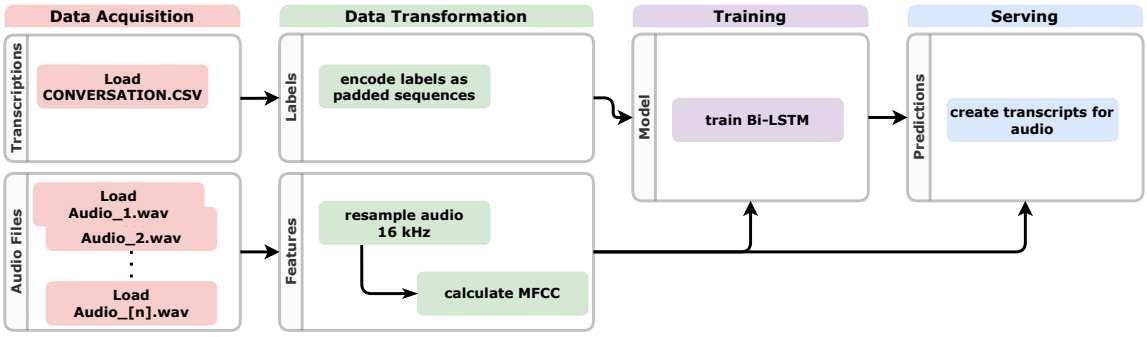

Figure 3: Process view for the UC02 Deep Learning pipeline

(7) Serving Throughput Test

During *Data Generation* all data sets are generated including training, serving, and scoring data sets. The data generation is not timed. In the *Load Test* the data is loaded into the system, which means the data is moved into the final position, where it will be used by the system. This could be just a copy operation or an ingestion into a system including data transformations, format changes, compression, encoding, etc. The *Load Test* is timed and the end-to-end elapse time is part of the final metric. The *Power Training Test* runs the training pipelines of the ten use cases sequentially, including all preprocessing, model training, and postprocessing tasks. The end-to-end elapse time of each use case is measured. In the *Power Serving Test I* and *Power Serving Test II* tests, the serving pipeline of all use cases are run sequentially and the elapse time of each use case is measured. In the *Scoring Test*, the quality of the predictions of each use case serving pipeline is measured against predefined thresholds. This test also embodies part of the model validation stage that is commonly found in production AI pipelines to decide whether a recently trained model should be deployed or discarded [37]. The time of the scoring is not measured as its purpose is to measure the quality of the models created during the *Power Training Test*. In the *Serving Throughput Test*, concurrent streams of the ten serving pipelines are defined and run. Each stream consists of the sequential execution of a permutation of the serving pipelines of the ten use cases. The end-to-end time of the execution of all streams is measured.

A complete run consists of a *Validation Run* with Scale Factor 1, followed by the actual *Benchmark Run* with the scale factor selected by the user. The purpose of the *Validation Run* is to verify the system under test is in a good state for running the benchmark. The results of the Validation Run are compared to a predefined result set, which is part of the benchmark kit. As part of the validation, the result cardinalities and data set sizes are compared to predefined results and other sanity checks are performed.

An official TPCx-AI result mandates that modifications to the kit are done only as allowed by the benchmark specification, and that all components of the system are commercially available (including software frameworks and libraries). An exception are TPCx-AI approved compute libraries. The result of an official *benchmark run* is reported to the TPC in a full disclosure report, which additionally contains all hardware and software as well as all information and configurations necessary to rerun the benchmark on the SUT.

3.5 Quality Metric

One of the acceptance criteria for a benchmark run is the quality metric. The quality metric for TPCx-AI is used to measure the quality of a trained machine learning model with previously unseen data against a ground truth data set. We call this process *SCORING* and it is embedded into to the benchmark driver. New test sponsors do not need to implement or measure the quality themselves. What is considered high or low quality is based on the particular use case and business requirements. Hence each use case has its own predefined quality metric and threshold. See Table 2 for a full reference of the different metrics, their thresholds, and an indicator showing if a certain metric should be minimized or maximized. However, since the quality metric is seen as an acceptance criteria rather than being part of the performance metric it is enough if the threshold is met, i.e., deviations are neither penalized nor rewarded. This is quite similar to the *time-to-accuracy* metric of MLPerf [18], in the sense that it acknowledges the fact that perfect accuracy is not always necessary or achievable. Slight improvements in accuracy might not warrant the cost from a business standpoint if they entail significant increases in training time. On the other hand, having a quality metric threshold also acknowledges the fact that faster trained AI models may have bad prediction quality, for instance due to their sensitivity to batch sizes and learning-rate values.

We chose the following metrics for the use cases:

word_error_rate measures how many words are correctly transcribed, the lower the better, zero is a perfect score and one and above means that every word was transcribed incorrectly.

mean_squared_log_error the Mean Squared Log Error *MSLE* is used mainly for regression problems and measures the error of the predicted value and the true value but is aware of the order of magnitude. An $MSLE <= 1.0$ means that the predicted values are of the same order as the true values on average, hence lower is better and zero is the best.

f1_score is a combined metric that takes precision and recall into consideration, i.e., how many of a certain category are found vs. how many of the predictions in this category are correct, higher is better and one would be a perfect categorization.

matthews_corrcoef The Matthews Correlation Coefficient *MCC* measures the correlation between the predicted and true values, a $MCC = 1$ would mean a perfect correlation, $MCC = 0$ would be the value of two independent random variables, and $MCC = -1$ would mean a perfect anti-correlation, hence higher is better.

median_absolute_error in contrast to the *MSLE* the Median Absolute Error *MAE* is used in cases where the predicted values are expected to be at the same order of magnitude and all deviations from the true values are treated the same. Hence the worst possible value would be the difference of the minimum and maximum value of the expected predictions $MAE = MIN - MAX$ and the best possible value would be $MAE = 0$. The direct nature of this metric makes it easy to interpret, for instance an $MAE = 3.5$ means that the predicted values are off by 3.5 on average from the true values.

accuracy_score The accuracy score is a metric used for classification problems and is probably the most intuitive one. It simply measures the amount of correct classifications. This means it is quite easy to use for multi-class problems but it is also quite vulnerable to unbalanced classes. It ranges from 0 to 1 where 0 is the worst and 1 being a perfect score.

Some metrics, such as the word_error_rate, are use-case dependent and some metrics are data dependant, such as the *MCC* or *MSLE*. The *MCC* is used for highly skewed data sets where the category we are interested in, i.e., the failure case, is under represented. Using only accuracy will not catch classifiers that only predict the majority class for instance. The *MSLE* is used in use cases where bigger variances are permitted at higher orders of magnitude. For instance, in Use Case 5 a price is predicted. A deviation of one dollar for a real price of $1000 is not as bad as the same deviation for a real price of $10. One notable exception to the automated scoring is UC01. Since it is a clustering problem there is no good metric to score the results that is not prone to outliers. Hence for this particular use case a more manual approach is used by the auditor the check if the work was actually done.

Table 2: Quality metric thresholds for each use-case

UC	Metric	Threshold	Min/Max
UC01	N/A	-	-
UC02	word_error_rate	0.50	↓
UC03	mean_squared_log_error	5.40	↓
UC04	f1_score	0.65	↑
UC05	mean_squared_log_error	0.50	↓
UC06	matthews_corrcoef	0.19	↑
UC07	median_absolute_error	1.80	↓
UC08	accuracy_score	0.65	↑
UC09	accuracy_score	0.90	↑
UC10	accuracy_score	0.70	↑

3.6 Metric

TPCx-AI defines three primary metrics, the performance metric, *AIUCpm@SF*; a price-performance metric, $/*AIUCpm@SF*; and the availability date. All three primary metrics need to be reported for an official benchmark result. *AIUCpm@SF* stands for AI Use cases-per-minute at scale factor SF, and it is a combination of the elapse times in seconds of the benchmark runs. It is designed to prevent performance improvements in only small parts of the benchmark to massively impact the overall performance metric. To this end, geometric means are used to combine the separate run times. This way, relative improvements in individual parts of the training or serving pipelines have the same influence on the overall result. In contrast, an arithmetic mean would be mainly influenced by improvements in long running stages. The use of a geometric mean is in line with previously published TPC benchmarks, such as TPC-DS and TPCx-BB [9, 23] and also recently published experiments [6]. The important parts of the metric are:

SF the user defined scale factor, which defines the size of the overall data set and approximates the data set size in GBs,

N the number of use cases (i.e., 10 in the current version),

S the user defined number of concurrent streams to run in the *Serving Throughput Test*,

T_{LD} the loading factor, which is the overall time it takes to ingest the datasets into the data store used for training and serving,

T_{PTT} the *Power Training Test* factor, defined as the geometric mean of the training times t_t of all use cases $\sqrt[N]{\prod_i^N t_t^i}$,

T_{PST} the *Power Serving Test* factor, defined as the geometric mean of the serving times t_s of all use cases $\sqrt[N]{\prod_i^N t_s^i}$, here the higher result of the two runs is taken.

T_{TT} and the *Serving Throughput Test* factor, defined as the total time spent running the throughput test divided by the number of use cases N, and the number of streams in the *Serving Throughput Test* S. Intuitively this leads to test sponsors maximizing the value of S until no gains are observed in the value of T_{TT}, indicating compute resources are saturated.

The resulting performance metric *AIUCpm@SF* is defined as:

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

Elapsed times are always reported in seconds, hence the constant value 60 is introduced to match the semantics of the performance metric. Furthermore, the performance metric is always reported for the benchmarked scale factor, e.g., *AIUCpm*@10 for scale factor 10, and results are not comparable across scaling factors. The price performance, $/*AIUCpm@SF*, is defined as the total system price divided by performance metric. The system price must include all software and hardware components of the deployment including operation, maintenance, and administration for one year. The system availability date specifies the date, from which on a customer can buy the deployment as it was used for the benchmark.

Besides the three primary metrics, a benchmark sponsor can also optionally report the energy metric, Watts/AIUCpm@SF. The energy measurement is centrally specified for all benchmarks in the TPC-Energy specification [25].

4 DATA SET CHARACTERISTICS

In this section, we give an overview of the data set scaling characteristics. One of the main contributions of TPCx-AI is its scalable data set. For ease of use each scale factor corresponds approximately to the size of the training data set in GB. Although the official data sizes are limited to scale factors 1, 3, 10, ..., 10,000, the data generator can produce any other data size in-between.

Figure 4 shows the data size at scale factors 1, 10, 100, and 1000. It can be seen that the data size is directly proportional to the scale factor and close to 1*GBxSF*. Since the data set is meant to be realistic, we do not scale all tables linearly but try to make them scale realistically in relation to the number of users and products. To

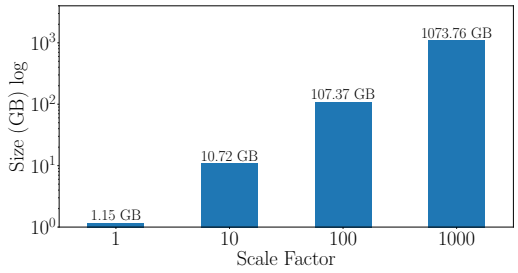

Figure 4: Data size for the all data sets per scaling factor

this end, each table has an individual scaling function with respect to an internal scale factor and a solver is used to approximate the actual size on disk for the aforementioned internal scale factor and scale it to the official scale factor.

Figure 5 illustrates the size of the individual data tables at the same scale factors. Even though the data set as a whole grows linearly and proportional to the scale factor, the tables grow at different rates. For instance, the `lineitem` table accounts for approximately 50% of the data set at scale factor $SF = 1$ whereas at a higher scale factor $SF = 1000$ it makes up only 25% of the whole data set. As can be seen in Figure 5, the number of customer entries grows linearly, but other tables grow faster, this is most notable with failures and customer images. Table 3 gives a detailed list of the individual table size and row counts.

5 BENCHMARK IMPLEMENTATIONS

TPCx-AI is an express benchmark and comes with a full reference implementation. According to TPC rules, this implementation supersedes the specification in cases of divergence. This means alternative implementations have to do equivalent work and produce the same output as the reference implementation. The reference implementation also increases the ease of use of the benchmark. For other benchmarks, such as TPC-H or TPC-C, a test sponsor has to implement a complete driver framework, which then also has to be certified by an auditor. Unlike previous TPC benchmarks, TPCx-AI includes *two* reference implementations, one for single node setups and one for cluster deployments.

Regardless of the implementation each use-case has two AI processing pipelines; one for model training on a training data set, and one for serving the trained model to make predictions on a serving data set. Each pipeline includes loading, preprocessing, training or serving, and postprocessing tasks. Figure 3 illustrates UC02's training and serving pipelines at a high level, which performs audio transcripts as described in Section 3.3. The first stage of the pipeline is data acquisition, which retrieves the data set from the data store and makes it available for further processing. In the second stage, data transformation is executed. The data set is transformed into features and potentially labels, if it is possible. The data transformation stage can include tasks such as, data cleaning, joining, aggregations, normalization, encoding, projections, or scaling. In the third stage, the model is trained. The transformed data is used to train one or several machine learning models. The last stage in the use case is the serving stage, which uses the model, as well as the transformed data set and its labels and features to make predictions.

The implemented pipelines are purposely kept simple, with a minimum number of libraries used. Furthermore, we follow best practices as outlined by the libraries and frameworks used, rather than fine tuning the use cases, in order to keep them portable and easy to maintain, as well as easy to understand as a reference. Listing 1 shows the structure of such a pipeline implementation.

Listing 1: Simplified Code Structure of a Single-Node Use Case

```
def load_data(path): # (1) ...
def pre_process(data): # (2)...
def train(training_data, labels): # (3)...
def serve(model, data): # (4) ...
def main():
    raw_data = load_data(input_path)
    (labels, data) = pre_process(raw_data)
    if stage == 'training':
        model = train(data, labels)
        dump(model, work_dir / file_name)
    if stage == 'serving':
        model = load(work_dir / file_name)
        predictions = serve(model, data)
        predictions.to_csv('predictions.csv')
```

The decision for two reference implementations was made based on the observation that machine learning pipelines come in two flavors. Many applications only require limited compute resources and all data can fully fit into main memory of a single compute unit, in these cases, single node deployments are a good choice and can often be found in practice. However, there are also pipelines and data sets, for which a single computational unit is not sufficient and the computation is distributed among several compute units. To acknowledge this fact, we include a single node implementation and a distributed implementation for the benchmark.

Workload and data set are the same for both implementations, this means, we use the same type of ML models, which are trained with the same data sets in both implementations. From a benchmark point of view, we do not focus on algorithmic improvements, but rather focus on comparability across frameworks and hardware. The model complexity stays constant across scale factors, which means higher scale factors mainly test the data throughput of systems under test. On current hardware and libraries, the single node implementation is better for small scale factors, while the distributed implementation is targeted for larger scale factors, which then are executed in shared-nothing cluster setups. The overall constraints for both implementations and any future implementations are given by the specification[7] and the driver harness.

Besides the use case implementations, the kit contains the specification and the user guide, as well as configuration files to adapt the execution to the system under test. Furthermore, the kit includes the driver, which is based on a set of scripts. The scripts execute the benchmark, perform measurements, and compute the benchmark results. The kit also generates a report, which can be used as a basis for the full disclosure report, required by the TPC for an official benchmark result. Finally, the kit includes the data generator and reference data sets for verification. Additional implementations can be included in the kit to support for instance new machine learning frameworks, as well as other libraries, execution environments, or hardware components. These have to be accredited by the TPC before being used for submission of official results.

[7] TPCx-AI - https://www.tpc.org/tpcx-ai/default5.asp

Table 3: Table sizes and row counts

Scale Factor	1		10		100		1000	
Table	size	row_count	size	row_count	size	row_count	size	row_count
conversation_audio	26.4 MB	387	134.0 MB	1,965	804.0 MB	11,787	4.3 GB	62,539
customer_images	8.0 MB	50	207.2 MB	1,287	7.5 GB	46,312	209.8 GB	1,303,712
product_rating	1.8 MB	139,292	9.8 MB	757,454	63.4 MB	4,877,735	358.0 MB	27,534,770
customer	8.5 MB	70,710	43.1 MB	358,817	258.2 MB	2,152,033	1.4 GB	11,418,023
failures	8.4 MB	50,000	215.0 MB	1,287,500	7.7 GB	46,312,500	217.7 GB	1,303,712,500
financial_account	96.9 kB	7,071	491.6 kB	35,881	2.9 MB	215,203	15.6 MB	1,141,802
financial_transactions	487.3 MB	6,247,036	4.5 GB	58,023,250	40.7 GB	522,118,524	283.2 GB	3,630,810,923
lineitem	426.4 MB	20,302,869	4.0 GB	188,575,563	35.6 GB	1,696,885,203	247.8 GB	11,800,135,500
marketplace	12.4 MB	70,710	63.2 MB	358,817	378.8 MB	2,152,033	2.0 GB	11,418,023
order	124.9 MB	3,123,518	1.2 GB	29,011,625	10.4 GB	261,059,262	72.6 GB	1,815,405,461
order_returns	16.2 MB	1,015,143	150.9 MB	9,428,778	1.4 GB	84,844,260	9.4 GB	590,006,775
product	24.0 kB	707	122.0 kB	3,588	731.7 kB	21,520	3.9 MB	114,180
Total	1.1 GB	31,027,497	10.7 GB	287,844,529	107.4 GB	2,620,696,376	1.1 TB	19,193,064,210

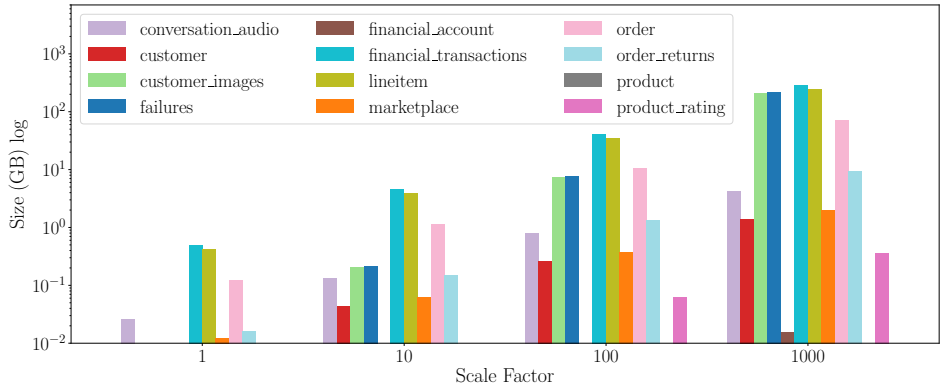

Figure 5: Data size for the all data sets per scaling factor

5.1 Single-Node Implementation

The single-node implementation is tailored towards smaller scale factors were a single data set fits into memory. This setup is typically found during the development of machine learning pipelines and for prototyping. We use Python[8] as a programming language and framework because of its widely used data science and machine learning ecosystem. Similarly, *Pandas*[9] is used for most data manipulation and data transformation. We use `DataFrames` to transfer the data between stages. For training the machine learning pipelines, we use scikit-learn[10] and Tensorflow[11] and its Keras[12] API.

Each use-case is implemented as a single command line application, which can be controlled by providing various command line arguments that specify the training or serving pipeline, the input paths, the working directory, the output directory, and hyper parameters that are allowed to be changed by the benchmark user.

The Python-based pipelines all follow a similar structure and due to the single node setup the execution does not require complex setup and tear down procedures. A single pipeline, as outlined in Listing 1, has the following steps:

(1) Load the data from one or more CSV files and merge them into a single `pandas DataFrame`,
(2) transform the `DataFrame` into a feature matrix and a labels vector (`numpy arrays` or `DataFrames` depending on the use-case),
(3) train the machine learning algorithm on the features and labels,
(4) serve the model and create prediction for features extracted from previously unseen data and write a csv file to disk.

5.2 Distributed Implementation

For the distributed implementation, we use Apache Spark[13] [38] as the data processing engine and its `DataFrame` API for data manipulation and data transformation. Apache Spark features a machine learning library[14] with distributed implementation for many popular machine learning algorithms [19]. We use it for many pipelines in our implementation. The deep learning models are similar to the single-node implementation and we also use the Keras API of

[8] Python - https://www.python.org/
[9] pandas - https://pandas.pydata.org/
[10] scikit-learn - https://scikit-learn.org
[11] Tensorflow - https://www.tensorflow.org/
[12] Keras - https://keras.io/
[13] Apache Spark - https://spark.apache.org/
[14] Apache Spark MLlib - https://spark.apache.org/docs/latest/ml-guide.html

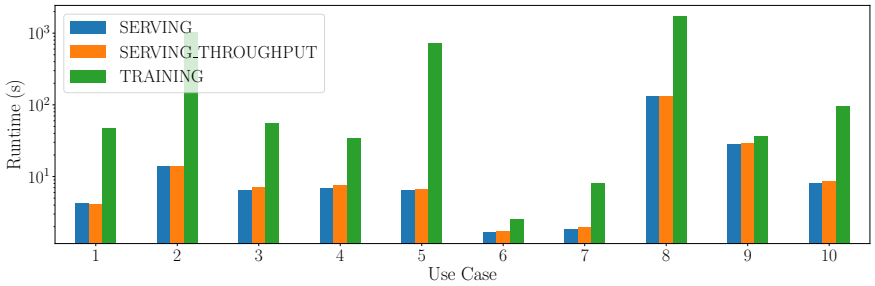

Figure 6: Runtimes of each use case in each test for the single-node implementation at $SF = 1$

Tensorflow for the distributed implementation. To train the models a data parallel approach is used, i.e., the same model is trained on different partitions of the training data set and the gradients are gathered after each step. This approach in particular is called allgather/allreduce and implemented by Horovod [33].

The code structure is similar to the single-node implementation. However, the computational model is completely different. While the execution for the single-node implementation is procedural and imperative the distributed implementation is much more declarative. We use the `DataFrame-API` of Apache Spark to create an execution graph, which is then optimized and executed by the Spark runtime. The `DataFrames` used in the context of the distributed implementation are different from the single-node version. Using `DataFrames` enables working with data sets that do not fit into memory and can be stored in a distributed file system.

The structure of the Spark pipelines is similar to the Python-based pipelines and has the same steps of loading, data transformation, model training, and serving. Unlike the Python-based execution, the Spark engine can use lazy execution within a job and does not have to materialize all intermediate steps, such as data transformations before model training.

6 EXPERIMENTS

In the following section, we present performance numbers for the single node Python-based implementation and a distributed execution using the Apache Spark implementation. While we present numbers for multiple scale factors, the results across scale factors are not comparable according to TPC rules. We perform the experiments without any special optimizations and the systems are not specifically designed for these workloads.

For all experiments, we use the same hardware setup. Our cluster has 11 nodes, each has 2 Intel® Xeon® CPUs E5-2699 V4 (with 22 cores, 2 threads each), 500 GB RAM, and 1.5 TB disk. For the Python experiments, we use a single node. For the Spark experiments, one node is configured as primary and ten servers as workers.

6.1 Single Node - Python-based

We test Scale Factors 1, 5, 10, and 15 using the Python-based kit. For the *Serving Throughput Test* we use 2 streams. We were not able to run larger scale factors in Python, because of memory constraints of our testing setup and limitations in the libraries [15] that have been used. The performance metrics of the runs can be seen in Table 5. Officially, the numbers are not comparable across scale factors, which also makes sense, since not all tables scale linearly. However, as can be seen in the table, the partial metrics across different scale factors grow with the scale factor, but not linearly. All of the individual metrics are less than a factor of 10 larger in SF 10 than in SF 1. This results in a improved primary performance metric for SF 10 compared to SF 1. This is due to the structure of the metric, which incorporates the scaling factor in the final measure. Just looking at the performance metric one could conclude that with the hardware and software being used for these experiments that the single-node implementation performs best at scaling factor $SF = 5$ or in other words on a 5GB data set.

Looking at the quality metrics (see Table 4) it can be observed that each benchmark run at each scaling factor was valid since the thresholds were met for all use cases. Furthermore, it can seen that some use cases improve at higher scaling factors, some stay more or less constant and others even worsen slightly. Even though the first case is to be desired in some cases, it might be that the model was actually overfitting at low scaling factors and is becoming more generalized with more training data.

In Figure 6 the individual run times for each use case in the Python-based execution with scale factor 1 on our server is shown. In the plot the average run time for each use case is displayed for the *Serving Throughput Test*. It can be seen that the serving time is similar for the *Power Serving Test* and the *Serving Throughput Test*. However, in the metric, we get a better value, showing that multiple streams enable more parallelism on the server. The training is always much more costly than the serving stages. Additionally, we can see that the run times of the use case differ in more than two orders of magnitude, which is to be expected in real workloads.

6.2 Multi Node - Apache Spark-based

For our multi node experiments, we use an 11 node setup (1 main node and 10 worker nodes). We execute Scale Factors 1, 10, 100, 200, 300, and 400 on our cluster and use two streams in the *Serving Throughput Test*. The performance metrics of the runs can be seen in Table 5. Interestingly, the performance numbers for smaller scale factors in the Spark-based experiments setup are slower than in the Python-based runs. This is due to the higher startup times of Spark. Additionally, we can see that the run times for Scale Factor 10 are similar to the run times of Scale Factor 1, which also shows that small scale factors are not properly utilizing the cluster. For Scale Factor 100, some of the stages seem to be performing more work and are taking considerably more time than in the smaller scale

[15] see https://github.com/pandas-dev/pandas/pull/45084 for an issue that affected uc08 and has since been fixed

Table 4: Quality metric for single- and multi-node experiments

Use Case	Metric	Single-Node (Python) SF=1	SF=5	SF=10	SF=15	Multi-Node (Apache Spark) SF=1	SF=10	SF=100	SF=200	SF=300	SF=400
UC01	N/A	-	-	-	-	-	-	-	-	-	-
UC02	WER	0.31	0.35	0.29	0.18	0.88	0.93	0.32	0.26	0.22	0.23
UC03	MSLE	4.60	3.34	3.62	3.30	5.73	3.70	3.87	4.25	4.36	4.48
UC04	F1	0.70	0.70	0.70	0.70	0.69	0.70	0.69	0.70	0.70	0.70
UC05	MSLE	0.01	0.04	0.02	0.02	0.30	0.21	0.04	0.03	0.02	0.02
UC06	MCC	0.46	0.53	0.54	0.54	0.23	0.23	0.20	0.23	0.22	0.21
UC07	MAE	0.89	0.98	1.02	1.01	1.71	1.62	1.31	1.44	1.43	1.47
UC08	Acc	0.71	0.73	0.74	0.74	0.72	0.73	0.76	0.75	0.77	0.72
UC09	Acc	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
UC10	Acc	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82

Table 5: Performance metric for single-node and multi-node experiments

	Single-Node (Python) SF=1	SF=5	SF=10	SF=15	Multi-Node (Apache Spark) SF=1	SF=10	SF=100	SF=200	SF=300	SF=400
T_{LD}	2.38	4.99	15.19	14.63	94.19	138.12	797.76	1729.97	2735.43	5993.25
T_{PTT}	91.18	352.61	905.16	1432.03	130.02	153.21	328.58	441.98	588.84	761.54
T_{PST}	9.92	23.11	52.39	65.33	58.59	64.07	77.70	91.27	96.30	105.02
T_{TT}	14.26	38.04	97.62	165.86	35.70	41.66	58.19	84.34	102.42	124.46
AIUCpm@SF	45.28	85.10	65.51	73.31	8.43	69.20	323.37	433.22	506.99	485.61

Table 6: Performance metrics of two official TPCx-AI results

	Python SF=10	Spark SF=1000
T_{LD}	4.53	4,405.47
T_{PTT}	314.80	981.90
T_{PST}	26.89	128.21
T_{TT}	4.69	110.67
AIUCpm@SF	291.35	1,205.43

factors. Overall, the Spark cluster can execute larger scale factors and get better performance numbers for these scale factors.

Similar to the performance metric the quality metric also performs worse at low scale factors than for the single-node implementation but improves at the high and very high scale factors for some use cases. An interesting observation is that for UC02 and UC03 the default parameters will not produce a valid benchmark run due to the quality metrics exceeding the thresholds, the values highlighted in red in Table 4. The reason is, especially for UC02, that the data is partitioned, a model is trained on each partition, and all partition models are then merged. When the partitions are too small then these partition-based models are not able to perform well. This can be changed by reducing the parallelism.

In Figure 7 the individual test run times of each use case for Scale Factor 10 are shown. Again, we report the average run time for the *Serving Throughput Test*. Overall, the performance characteristics are similar to the Python experiment in Figure 6. An outlier is UC06, which is the slowest use case in Spark, while it is fast in Python. However, the failures table grows faster than the scaling factor, meaning in the larger scale factor disproportionately more work needs to be done for UC06.

6.3 Official TPCx-AI Results

As of June 2023, 17 results have been published on the TPC website[16]. To give a comparison of our experiments to real benchmark results, we briefly discuss the first two results here.

The first result [17] was submitted by the *Telecommunications Technology Association (TTA)* [18] in April 2022. It is an SF 1000 run (1 TB) with with 10 streams executed on Apache Spark. The system under test consists of 3 servers with each 2x Intel® Xeon® Platinum 8380 CPU and 2,048GB memory. In total the system has 6 processors with 240 cores and 480 threads, and 6,144GB memory. The total disk space is 136TB. The primary metrics of the system are:

Performance 1,205.43 AIUCpm@1000
Price/Performance ₩378,912.09 KRW/AIUCpm@1000 ($290 USD/AIUCpm@1000)
Availability date April 18, 2022
Total System Cost ₩456,752,000 KRW ($350,000 USD)

The second result [19] was submitted by *Nettrix Information Industry* [20] in May 2022. It is an SF 10 run with 100 streams using the Python-based implementation. The system under test is a one-server system consisting of 2x Intel® Xeon® Gold 6346 CPU (2 processors with 32 cores and 64 threads) with 512GB memory and a total disk space of 12TB.

Performance 291.35 AIUCpm@10
Price/Performance $84.26 USD/AIUCpm@10
Availability date May 31, 2022
Total System Cost $24,548 USD

[16] TPCx-AI result webpage - https://www.tpc.org/tpcx-ai/results5.asp
[17] ID 12204211: TPCx-AI Result Highlights TTA KR580S2 - https://www.tpc.org/5401
[18] https://www.tta.or.kr/
[19] ID 12206151: TPCx-AI Result Highlights Nettrix R620 G40 - https://www.tpc.org/5402
[20] https://www.nettrix.com.cn

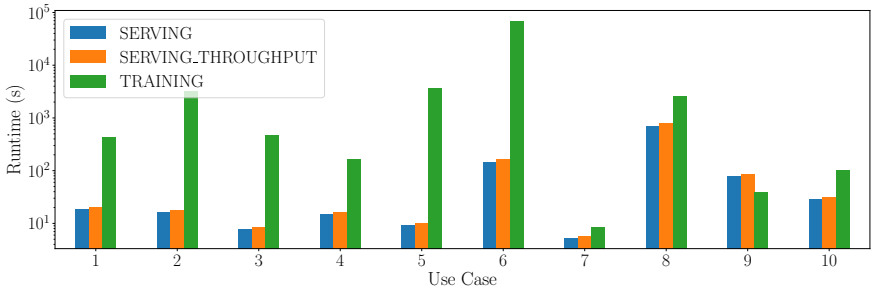

Figure 7: Run times of each use case in each test for the distributed implementation at $SF = 10$

Table 7: Future use cases

ID	use case	Class	Data	Algorithm	DL
UC11	Sentiment Analysis	Classification	Text	Random Forest	
UC12	Translate Product Descriptions	Machine Translation	Text	Transformer	X
UC13	Sentiment Analysis (DL)	Classification	Text	Recurrent Neural Network	X
UC14	Credit-Scoring for Merchants	Classification	Number	Multilayer Perceptron	X
UC15	Predict Retail Location	Regression	Number	Decision Tree	
UC16	Click-Through-Rate Prediction	Recommendation	Mixed	DLRM	X

The detailed result metrics of both benchmark runs are presented in Table 6. All details are available on the TPC website. Since then there have been additional results published by Dell Technologies, Hewlett Packard Enterprise, and Transwarp Technology in different scale factors, ranging from 3 to 3000.

7 FUTURE USE CASES

In addition to the ten published use cases, we created a set of six new use cases to be added to the workload in future versions. The list of new use cases is shown in Table 7. This set of use cases focuses more on deep learning based models. These use cases add new tables and attributes to the data set and will, therefore, change the data set scaling behavior. The use cases are not available publicly yet. In the following sections, we give a brief introduction to each new use case.

UC11/UC13 - Sentiment Analysis ML/DL In these use cases the sentiment of a product review is predicted either by using traditional ML or DL techniques, i.e., random forest model or a Recurrent Neural Network. In the preprocessing stage, the data is loaded and the classifier is trained using star ratings as labels.

UC12 - Product Description Translation In UC12, product descriptions are translated from English to German. A transformer model is used to translate the sentences in product descriptions, after it has been trained on example sentences in both languages.

UC14 - Credit Scoring UC14 predicts credit ratings for merchants based on existing loans using a multi-layer perceptron model. The preprocessing stage joins multiple tables and including new tables for merchants. The values are normalized and missing values imputed. In the serving stage, ratings for merchants applying for loans are predicted.

UC15 - Predict Retail Location In UC15, locations for new stores are predicted based on customer demographics. In the training stage, a decision tree is trained, which is then used to perform regression for new store locations.

UC16 - Click-Through-Rate Prediction UC16 predicts click conversion to purchases using a deep learning recommendation model. Click logs are used and converted in a click log model. In the serving stage, click sessions are classified by their conversion probability.

8 CONCLUSION AND FUTURE WORK

In this paper, we present TPCx-AI, the Transaction Processing Performance Council's new industry standard benchmark for artificial intelligence and machine learning systems. TPCx-AI is modeled on a retail scenario and comprises ten realistic use cases —covering both deep learning and traditional machine learning algorithms —each of which is implemented as an end-to-end AI processing pipeline. The pipelines consist of data ingestion, preprocessing, training, serving, and postprocessing as well as model update. Since TPCx-AI is a TPC *Express* benchmark, it comes with a fully implemented kit, which is ready to run on common ML/AI hardware. Besides a single node Python-based implementation, it also features a distributed Apache Spark-based implementation. TPCx-AI has already received several independent officially audited result submissions after being officially standardized in September 2021, underlining the interest in industry for the benchmark.

In future work, we want to separate preprocessing and postprocessing from the training and serving stages into independent stages and, furthermore, enable a more fine grained optimization of pipelines as well as adding continuously evolving pipelines [37]. Additionally, we hope to include the new use cases and more system implementations in future versions of the benchmark. A further extension can target other challenges found in production AI pipelines, such as model maintenance or update [2].

ACKNOWLEDGMENTS

This work was partially funded by the German Research Foundation (ref. 414984028), and the European Union's Horizon 2020 research and innovation programme (ref. 957407).

REFERENCES

[1] McKinsey Analytics. 2016. *The age of analytics: competing in a data-driven world.* Technical Report. McKinsey Global Institute.

[2] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 1387–1395. https://doi.org/10.1145/3097983.3098021

[3] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Christopher Ré, and Matei Zaharia. 2018. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *CoRR* abs/1806.01427 (2018).

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

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.

[6] Christopher Elford, Dippy Aggarwal, and Shreyas Shekhar. 2021. Revisiting Issues in Benchmark Metric Selection. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer International Publishing, Cham, 35–47.

[7] Sabri Eyuboglu, Bojan Karlaš, Christopher Ré, Ce Zhang, and James Zou. 2022. Dcbench: A Benchmark for Data-Centric AI Systems. In *Proceedings of the Sixth Workshop on Data Management for End-To-End Machine Learning*. Association for Computing Machinery, New York, NY, USA, Article 9, 4 pages.

[8] Wanling Gao, Jianfeng Zhan, Lei Wang, Chunjie Luo, Daoyi Zheng, Xu Wen, Rui Ren, Chen Zheng, Xiwen He, Hainan Ye, et al. 2018. Bigdatabench: A scalable and unified big data and ai benchmark suite. *arXiv preprint arXiv:1802.08254* (2018).

[9] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, New York, USA) *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 1197–1208. https://doi.org/10.1145/2463676.2463712

[10] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An Open Source AutoML Benchmark. *CoRR* abs/1907.00909 (2019). http://arxiv.org/abs/1907.00909

[11] B. Haibe-Kains, G.A. Adam, A. Hosny, et al. 2020. Transparency and reproducibility in artificial intelligence. *Nature* 586, E14-E16 (2020).

[12] Awni Y. 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. *CoRR* abs/1412.5567 (2014).

[13] Karl Huppler. 2011. Price and the TPC. In *Performance Evaluation, Measurement and Characterization of Complex Systems*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 73–84.

[14] Karl Huppler and Douglas Johnson. 2014. TPC Express – A New Path for TPC Benchmarks. 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, 48–60. https://doi.org/10.1007/978-3-319-04936-6_4 Series Title: Lecture Notes in Computer Science.

[15] Nina Ihde, Paula Marten, Ahmed Eleliemy, Gabrielle Poerwawinata, Pedro Silva, Ilin Tolovski, Florina M. Ciorba, and Tilmann Rabl. 2021. A Survey of Big Data, High Performance Computing, and Machine Learning Benchmarks. In *Performance Evaluation and Benchmarking: 13th TPC Technology Conference, TPCTC 2021, Copenhagen, Denmark, August 20, 2021, Revised Selected Papers* (Copenhagen, Denmark). Springer-Verlag, Berlin, Heidelberg, 98–118.

[16] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]. *CoRR* abs/1904.09483 (2019). http://arxiv.org/abs/1904.09483

[17] Yu Liu, Hantian Zhang, Luyuan Zeng, Wentao Wu, and Ce Zhang. 2018. MLBench: Benchmarking Machine Learning Services Against Human Experts. *PVLDB* 11, 10 (2018), 1220–1232.

[18] Peter Mattson, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David Patterson, Guenther Schmuelling, Hanlin Tang, Gu-Yeon Wei, and Carole-Jean Wu. 2020. MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance. *IEEE Micro* 40, 2 (2020), 8–16. https://doi.org/10.1109/MM.2020.2974843

[19] Xiangrui Meng, Joseph Bradley, Burak Yavuz, et al. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[20] George A. Miller. 1995. WordNet: A Lexical Database for English. 38, 11 (1995), 39–41.

[21] Sharan Narang. [n.d.]. DeepBench. https://svail.github.io/DeepBench/. Accessed: 2021-07-03.

[22] Sharan Narang and Greg Diamos. [n.d.]. An update to DeepBench with a focus on deep learning inference. https://svail.github.io/DeepBench-update/. Accessed: 2021-07-03.

[23] Raghunath Othayoth and Meikel Poess. 2006. The making of tpc-ds. In *Proceedings of the International Conference on Very Large Data Bases*, Vol. 32. 1049.

[24] Meikel Poess and Chris Floyd. 2000. New TPC benchmarks for decision support and web commerce. *ACM Sigmod Record* 29, 4 (2000), 64–71.

[25] Meikel Poess, Raghunath Othayoth Nambiar, Kushagra Vaid, John M Stephens Jr, Karl Huppler, and Evan Haines. 2010. Energy benchmarks: a detailed analysis. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. 131–140.

[26] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen, and Brian Caufield. 2014. TPC-DI: The First Industry Benchmark for Data Integration. *PVLDB* 7, 13 (2014), 1367–1378.

[27] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2017. Data management challenges in production machine learning. In *SIGMOD*. 1723–1726.

[28] Alexandra Posoldova. 2020. Machine Learning Pipelines: From Research to Production. *IEEE Potentials* 39, 6 (2020), 38–42. https://doi.org/10.1109/MPOT.2020.3016280

[29] Tilmann Rabl, Christoph Brücke, Philipp Härtling, Stella Stars, Rodrigo Escobar Palacios, Hamesh Patel, Satyam Srivastava, Christoph Boden, Jens Meiners, and Sebastian Schelter. 2020. ADABench - Towards an Industry Standard Benchmark for Advanced Analytics. In *Performance Evaluation and Benchmarking for the Era of Cloud(s)*, Raghunath Nambiar and Meikel Poess (Eds.). Springer International Publishing, Cham, 47–63.

[30] 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.). Springer Berlin Heidelberg, Berlin, Heidelberg, 41–56.

[31] Joseph D Romano, Trang T Le, William La Cava, John T Gregg, Daniel J Goldberg, Praneel Chakraborty, Natasha L Ray, Daniel Himmelstein, Weixuan Fu, and Jason H Moore. 2021. PMLB v1.0: an open source dataset collection for benchmarking machine learning methods. *arXiv preprint arXiv:2012.00058v2* (2021).

[32] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 2503–2511.

[33] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR* abs/1802.05799 (2018).

[34] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *ICCV*.

[35] Anthony Thomas and Arun Kumar. 2018. A Comparative Evaluation of Systems for Scalable Linear Algebra-Based Analytics. *Proc. VLDB Endow.* 11, 13 (sep 2018), 2168–2182.

[36] Joaquin Vanschoren, Jan van Rijn, Bernd Bischl, and Luís Torgo. 2013. OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15 (12 2013), 49–60. https://doi.org/10.1145/2641190.2641198

[37] Doris Xin, Hui Miao, Aditya Parameswaran, and Neoklis Polyzotis. 2021. Production Machine Learning Pipelines: Empirical Analysis and Optimization Opportunities. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2639–2652. https://doi.org/10.1145/3448016.3457566

[38] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets.. In *HotCloud*. 10–10.