Algorithmic Trade Forecasting DBMS Implementation for AI and ML within Digital Research Alliance of Canada Advanced Research Computing (ARC) Platforms

Kristina Cormier
Computer Science
Okanagan College
Kelowna, Canada
0009-0004-3783-9704

Helana Jaraiseh
Computing Science
UFV
Abbotsford, BC, Canada
0009-0000-1239-2820

Justin Drenka
Computer Science
UBCO
Kelowna, Canada
0009-0008-6821-8379

Minami Sato
Computer Science
UBCO
Kelowna, Canada
0000-0000-0000-0000

Brandon Hay
Computer Science
Okanagan College
Kelowna, Canada
0009-0001-6605-6037

James Midtdal
Computer Science
Okanagan College
Kelowna, Canada
0009-0005-6312-569X

Youry Khmelevsky
Computer Science
Okanagan College
Kelowna, Canada
0000-0002-6837-3490

***Abstract*—Short-term stock market predictions using machine learning (ML) and artificial intelligence (AI) reduce the risk of human error and bias. They can also detect patterns and work with large and complex datasets. In this research paper, a new Database Management System (DBMS) implementation that uses Digital Research Alliance of Canada (DRAC) computational resources for Algorithmic Trade with Staging and Data Cleaning is presented.**

The paper explores data staging and cleaning for machine learning (ML) using the Database Management System (DBMS) and the cloud and computational resources provided by the Digital Research Alliance of Canada's Alliance Cloud Connect Pilot [1].

I. Introduction

This project explores data staging and cleaning for machine learning (ML) using the Database Management System (DBMS) and cloud and computational resources provided by the Digital Research Alliance of Canada's Alliance Cloud Connect Pilot [1]. The project is broken into several phases. The first phase involves extracting and cleaning data for the staging table. Data is extracted from Financial Modelling Prep (FMP) [2] every 5 minutes, loaded into the database, and aggregated from 5-minute intervals to 15-minute intervals in the staging table. The next phase focuses on data modelling. The final phase targets training the ML model. During this phase, 2 years of data on stocks, bonds, indexes, and commodities have been collected. Future work involves downloading and transferring 30 years' worth of data to the database. This project focuses on S&P 500 for stock data. This means that over 500 records accumulate every 5 minutes, for a total of over 6000 per hour. Data is collected for approximately 10 hours per day, resulting in over 60,000 records being inserted daily. That would be approximately 300,000 records for a 5-day work week and over 15,000,000 records per year. That is just Stocks for one year. Records for Commodities, Bonds, and Indexes are available from FMP in addition to the stocks. For that reason, ample high-performance storage is required.

The project has been qualified for the Alliance Cloud Connect Pilot through the Digital Research Alliance of Canada (DRAC), which allocates 200 core-years on the NIBI-compute system, 20.0 RGU-years on the nibi-gpu system, and 59 TB of project storage on the NIBI storage system for High Performance Computing (HPC). It also allocates 16 VCPU-years, 4 cloud instances, 32 GB of RAM, 7 volumes, 7 snapshots, 2 floating IP addresses, and 60,000 GB of cloud volume and snapshot storage for Cloud allocations on the Arbutus-persistent-cloud system. The project is provided PostgreSQL access through DRAC infrastructure.

The approach developed here emphasizes that ETL pipelines are long-lived system components whose structural and quality characteristics significantly affect reliability and operational cost. While the focus is on traditional ETL environments within data warehousing systems, the underlying concerns of workflow quality, scalability, and maintainability are directly relevant to modern data integration pipelines. In contrast to metric-driven analysis of ETL evolution, the current project emphasizes a cloud-based, ELT-oriented implementation for near real-time financial data ingestion and machine learning, providing a systems-level realization that addresses similar quality and complexity challenges through automated staging, DBMS-centric processing, and empirical validation.

The main contributions of this paper are (1) a new approach implementation environment for Big Data ML and Artificial Intelligence (AI); (2) discussion about difference between Extract Transform Load (ETL) and Extract Load Transform (ELT) process and confirming operational ELT process effectiveness in research project at post-secondary institutions; and (3) the design and testing of a new API for direct access to the DBMS from the core subsystem from technological point of view.

This work is a continuation of the previous research projects described in [3]–[18]

II. Existing Work

Collecting data to train ML models requires extremely large sets of data. The importance of data set size is discussed by Daniyal Rajput [19]. Smaller data sets may suffice, but only if the data is extremely high-quality; however this is not feasible when working with intermittent data such as stock prices.

Short-term stock market prediction using machine learning and artificial intelligence offers the potential to reduce human error, mitigate biased decision-making, and detect complex patterns in large datasets. The success of these predictive models, however, heavily depends on the quality and preprocessing of financial data. High-frequency market data, such as that extracted from Financial Modelling Prep (FMP) at 5-minute intervals, can contain missing values, duplicates, and inconsistencies, all of which can significantly impact ML performance.

Machine learning and artificial intelligence techniques have been widely applied to financial time series forecasting, particularly for short-term and intraday stock market prediction. Prior research demonstrates that the performance of ML models in financial domains is strongly influenced by data quality and preprocessing choices. Budach et al. [20] show that even minor data quality issues can significantly degrade ML performance on tabular datasets, underscoring the importance of careful data preparation before model training.

In the broader context of economic and financial applications, Nosratabadi et al. [21] present a comprehensive review of advanced ML and deep learning methods, highlighting the growing effectiveness of hybrid and ensemble approaches for economic forecasting tasks. Their findings suggest that robust preprocessing and feature engineering are essential to fully leverage these models, particularly when working with noisy and non-stationary financial data.

Several recent studies have explored advanced preprocessing and modelling techniques to enhance financial time-series prediction. For example, an IEEE publication by Chacón et al. [22] investigates the use of ensemble empirical mode decomposition combined with recurrent neural networks to improve prediction accuracy on financial time series, demonstrating the value of preprocessing steps before model training. While this work emphasizes signal decomposition techniques to improve predictive performance, it differs from the present study in its focus on model-level preprocessing rather than database-centric data staging and transformation.

Brownlee discusses practical methodologies for preparing data for ML [23], who outlines systematic approaches to data cleaning, feature selection, and data transformation. Unlike prior work that primarily emphasizes model architecture or signal processing techniques, this study focuses on designing and validating a scalable data staging and transformation framework that supports real-time financial data ingestion and subsequent ML model training.

Data inconsistencies can occur at both the schema and instance level, as discussed by Rahm and Do [24]. The problem is compounded when integrating multiple data sources, highlighting the requirement for both proper database design and thorough examination of incoming data.

Salunke and Ouda discuss PostgreSQL [25], in which they evaluate and compare its superior performance with MySQL.

Data integration flows have been identified as a significant bottleneck in business intelligence systems, underscoring the lack of systematic design methodologies and the growing complexity imposed by near-real-time and heterogeneous data requirements. These studies focus primarily on ETL-centric business intelligence architectures and conceptual design frameworks [26].

A preliminary study by Vassiliadis et al. explored the use of metrics to assess quality, complexity, and the impact of evolution in ETL ecosystems, to predict maintenance effort and understand how changes propagate through data integration workflows [27].

In 2022 Berisha et al. discussed survey work on big data analytics in cloud computing. Berisha highlights the rapid growth of large, heterogeneous datasets and the resulting need for scalable processing frameworks that can manage volume, velocity, and variety efficiently within cloud environments. These surveys also describe a shift from traditional ETL workflows to ELT paradigms in cloud data systems, where loading precedes transformation to leverage elastic storage and compute. Such findings provide helpful context for designing cloud-centred data integration pipelines and underscore why modern systems must support scalable ingestion and transformation of diverse data types [28].

Berisha's 2022 systems research demonstrated the importance of API-driven access layers for improving interoperability, scalability, and accessibility in Big Data environments, particularly in heterogeneous SaaS and multi-language settings [28]. While such approaches focus on abstracting access to distributed Big Data platforms, this work targets direct DBMS-level access from the core subsystem, optimized for near real-time ELT and machine learning workflows.

Research performed by Hellerstein has established comprehensive design principles for database system architecture, covering internal storage, query execution, transaction management, and concurrency control [29]. Such architectural understanding informs the design of system interfaces that efficiently expose DBMS capabilities, and future work could explore tighter integration with emerging DBMS internals and cloud-native database platforms.

Chaudhuri et al. argued that monolithic database systems hinder programmability, integration, and manageability in modern data-intensive environments, and instead advocated decomposing DBMS functionality into modular, RISC-style components connected through narrow, well-defined APIs. This architectural shift emphasizes predictable performance, self-tuning behaviour, and a clear separation of concerns among system components, enabling databases to act as composable services within larger software ecosystems rather than as isolated, monolithic systems. Such work establishes a strong systems-level foundation for designing APIs that expose controlled, efficient access to database functionality from core subsystems, particularly in contexts requiring scalable data ingestion and analytical processing [30].

III. The Designed and Implemented System Architecture

The illustration shown in Fig. 1 depicts the overall architecture and design of the system. The workflow follows the ETL process, with the majority of historical stock data collection and validation occurring on local (personal) machines before upload to the Fir PostgreSQL database.

A. Downloading FMP Stock Data

FMP's API has a limit on the number of rows returned: approximately 10 days' worth of data per stock per call when set to a 5-minute interval. This limitation imposed by the API requires an automation process to extract large volumes of data. The historical stock data collection script is written in Python and runs locally. The script defines a date range and a list of 503 stock symbols. A for-each loop runs on the stock symbols, and a while-loop calls the FMP API with distinct 5-day time frames (Listing 1):

Listing 1. Financial Modeling Prep API Call

```
   base_url =
       "https://financialmodelingprep.com/
   api/v3/historical-chart/5min/" + stock
       + "?"

response = requests.get(base_url + "from="
     + str(from_date) + "&" + "to=" +
     str(current_date) +
     "&extended=true&apikey=our-API-key")
```

After calling the API, the server's HTML response code is checked. Non-200 (OK) status codes trigger a console error message, notifying the operator of an error. Successful API calls are appended to a pandas DataFrame. The 5-day time frame is decremented, and the while loop terminates when the full date range has been attempted. After the while loop terminates, the DataFrame is saved as a CSV file named after the corresponding stock symbol. The program exits when all symbols have been processed.

B. Script Reliability Evaluation

Another Python script, executed on a local machine, validates the timestamps produced by the previous Historical Data Downloading Script. The script defines holidays, early-closing dates, market opening and closing times, start and end dates, and stock symbols. The stock symbols and date range match the Historical Data Downloading script. A for-each loop runs on the stock symbols, loading the corresponding CSV file and adding the timestamps to a set (Listing 2).

Listing 2. Opening CSV and Loading Timestamps

```
for stock in stocks:
    # Load CSV
    with open(stock + ".csv", newline='')
         as f:
        reader = csv.DictReader(f)
        actual_timestamps = set()
        for row in reader:
            dt =
                datetime.strptime(row['date'],
                "%Y-%m-%d %H:%M:%S")
            actual_timestamps.add(dt)
```

A while-loop is entered, and the date range is processed one day at a time until the end date is reached. Expected intervals for the day are generated, ignoring holidays and accounting for early closing dates (e.g. the day before Christmas). See Listing 3 below:

Listing 3. Generating Expected Timestamps

```
if current.weekday() < 5 and current not
     in holiday_dates:
        # Generate expected 5-minute
            intervals for this day
        t = datetime.combine(current,
            datetime.min.time()) +
            market_open
        if current in early_close_dates:
            end_dt =
                datetime.combine(current,
                datetime.min.time()) +
                early_close_time
        else:
            end_dt =
                datetime.combine(current,
                datetime.min.time()) +
                market_close
```

Another while loop is entered, and the actual 5-minute timestamps are compared with the expected timestamps.

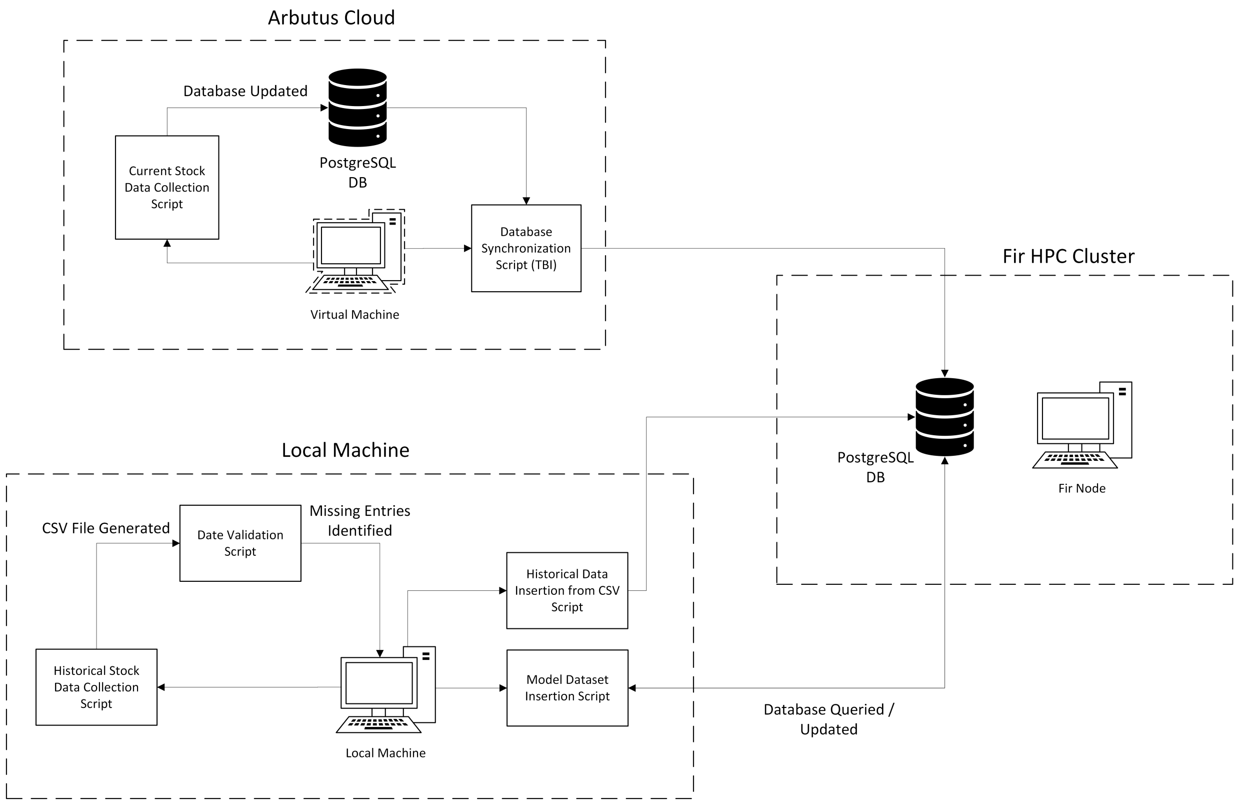
Arbutus Cloud
Database Updated
PostgreSQL DB
Current Stock Data Collection Script
Virtual Machine
Database Synchronization Script (TBI)
Fir HPC Cluster
PostgreSQL DB
Fir Node
Local Machine
CSV File Generated
Date Validation Script
Missing Entries Identified
Historical Data Insertion from CSV Script
Historical Stock Data Collection Script
Model Dataset Insertion Script
Local Machine
Database Queried / Updated

Fig. 1. DRI Staging and Data Transfer Infrastructure Design

Any missing entries are logged to the console for later verification by the operator. The total number of missing entries for that stock symbol is tracked and printed at the end of each CSV file processing loop. After all CSV files have been processed and missing entries printed to the console, a web browser is opened, and manual API calls are made to confirm the missing entries. The operator verifies that missing entries are not due to script execution and are instead caused by FMP's data collection process. A sample set of 5 CSV files (A.csv, AAPL.csv, ABBV.csv, ABNB.csv, ABT.csv) were checked in this manner to verify script integrity. The CSV files contain between 104,405 and 215,961 rows because trading occurs outside standard market hours. The test set of CSVs produced a range of missing entries per file, from 0 to 11, with 3 of the 5 files having zero missing entries. The Python script to download the files from FMP has console logging for failed HTTP responses, and no errors were noted. Therefore, the historical data downloading script's performance was deemed satisfactory, with all missing entries in the sample set manually verified; none were due to script execution. Future work on the date validator should add a second automated API call for each missing entry to eliminate the need for manual web-browser verification. With this change, timestamp entries for all 503 stocks will be verified with minimal operator input.

IV. Data Cleaning and Formatting

The pandas Python library provides an easy way to edit table data via dataframes. Dataframes act as in-memory tables that support a wide variety of functions to transform the structure of the dataframe or the data it holds.

A. Stock Table Transformation Process

Each commodity, index, and bond table is loaded from the database into separate dataframes and then combined into a single dataframe. Raw data for a single stock is fetched from the staging tables and loaded into memory. The current stock dataframe is joined on the "DATETIME" column with the combined commodity dataframe, which contains closing prices from each commodity, index, and bond. A similar process occurs with the sector data, where it is fetched from the database, loaded into a dataframe, and joined with the current stock dataframe. This process is iterated for each stock in the stock list, and each stock is appended to an output dataframe.

B. Holiday Flags

After merging all stocks, the "DATETIME" column is split into integer columns representing month, day, hour, minute, and weekday using the pandas built-in functions for the *dt* (date time) object. Boolean columns are created for pre-holiday, post-holiday, Monday morning, and Friday afternoon. A list of holidays is generated using Python's holidays library, and the integer columns are sent to the one-hot encoding function.

C. One-hot Encoding

To prevent ML models from attempting to derive a relationship between different integer values in a single column, the script uses a technique called one-hot encoding. This technique allows the script to split the original column into a set of Boolean columns for each value present in the data. For example if there is a month column that contains values 1 - 12 (representing each month) this would be broken up into 12 separate columns boolean columns for each record.

D. Verification of Column Presence

After the one-hot encoding is complete, the script needs to verify that all required columns have been created. If the transformation was performed on a small dataset, it is possible that not all possible values of the base column were represented. For example, the month may have values from 1 to 12. If the original data contained only one month, after hot encoding, the resulting dataframe will contain a boolean column for that month and will be missing the other 11 boolean month columns required by the PostgreSQL table.

This is fixed by verifying that all columns from the database table are present in the dataframe. If a column is missing, it is added, and all records are given a value of 0.

E. Type Casting

Some data retrieved from the FMP API had inconsistent data types. For some stocks, the volume column was in floating-point format, and the rest were in integer format.

To prevent these issues when working with the raw data, proper data types are required to ensure a smooth copy into the database. The script handles this by typecasting specific columns in the fully transformed dataframe to the associated types for the PostgreSQL Model Training table.

V. Database Structure

A. Staging Tables

Each stock, bond, commodity, and index has its own table in the database. The tables are named after their associated tickers, with a few additional rules. All tickers have non-alpha-numeric symbols removed when using them for table names(e.g. GC=F becomes the table GCF). The exception to this rule is that if a stock name contains a dot, it is represented with an underscore instead. (e.g. BRK.B becomes the table BRK_B). This is done to maintain a set of regularly allowed PostgreSQL table names.

Individual tables are used because the transformation process operates on one stock at a time. This also improves query speed, as smaller tables take less time to search for specific data.

Each staging table is created using the code in Listing 4.

Listing 4. Example staging table used for all tickers

```
-- Staging table format where %I holds the
    current ticker
        CREATE TABLE IF NOT EXISTS
            market.%I (
           stock_id bigserial PRIMARY KEY,
           date timestamp,
           open numeric(18,6),
           low numeric(18,6),
           high numeric(18,6),
           close numeric(18,6),
           volume bigint
        );
```

The `stock_id` column acts as a surrogate key for each table. A surrogate key is used because non-integer types can be problematic when used to identify a record.

The date column holds a timestamp representing the start time of a 5-minute window of that stock's data.

The columns open, low, high, and close are all numeric (18,6) types, as this data is both significant and highly precise.

Finally, the volume column is of type bigint because each stock can have a large number of shares issued by the company. This allows many shares to be sold at once, which may exceed the capacity of a regular integer type.

B. Sector Data Table

The Sector table is a simple lookup table in the database with only 2 columns. It has a primary key column for the stock symbol, set to a max of 10 characters, and a column for the sector in which the associated company operates (Financial Services, Industrial, Healthcare, etc).

C. Transformed Data Table

Transformed data is stored in the database in one large table. The table consists of 94 columns, including the ticker, closing price, volume sold, prices of relevant other commodities, and columns to support one-hot encoding for month, day, time, and sector.

VI. Discussion of Validation and Results

The validation results demonstrate that the proposed API-based access to the DBMS introduces minimal overhead while enabling reliable near-real-time data ingestion and aggregation. This confirms that the architectural decision to expose direct DBMS access through a controlled core subsystem interface is suitable for time-sensitive financial data pipelines.

Consistent with prior work advocating modular and API-driven database architectures, the observed system behaviour confirms that exposing database functionality through a narrow interface does not compromise correctness or performance, while significantly improving integration flexibility.

While the current evaluation focuses on a single data source and a moderate ingestion frequency, the observed stability of the staging and aggregation process suggests that the architecture can scale to additional instruments or higher polling frequencies, subject to DBMS write-throughput and network-latency constraints.

Validation also confirmed the correctness of the ELT workflow, with no observed data loss, duplication, or temporal misalignment during aggregation. This is particularly important for downstream machine learning tasks, where data consistency directly affects model performance.

The results indicate that the staged dataset produced by the proposed system is well-suited for downstream machine learning tasks, as it provides consistent, time-aligned observations at a granularity appropriate for short-term market prediction models.

This evaluation is subject to several limitations. Although the system integrates and patches data from multiple external sources to improve coverage and continuity, the current validation focuses on a limited subset of markets and instruments. In addition, stress testing under extreme ingestion rates and high levels of concurrent access was not performed. Future work will address these limitations by expanding the range of data sources and instruments, and by evaluating system performance under higher concurrency and sustained high-throughput workloads.

These results provide a strong foundation for extending the system toward full end-to-end machine learning integration, including automated feature extraction and model training directly driven by the validated DBMS access API.

VII. Future Works

Currently, this project uses 2 years of historical data and real-time data. Future work includes adding 30 years of historical data. This requires a significant amount of storage resources. Thanks to the Digital Research Alliance of Canada Pilot Program Grant, this project now has access to the resources needed to advance to the next phase.

VIII. CONCLUSION

This project demonstrates how data quality and how it is handled or transformed affect the performance of ML models for short-term stock market predictions. Specifically, it uses ETL, a traditional approach but less forgiving of staff turnover in post-secondary research. Various data sources were used to collect data. The primary source was FMP.

On DRAC's Arbutus server, scripts were used to create a PostgreSQL DB schema and tables, and extract real-time data for stocks, bonds, indexes, and commodities from FMP and load it into the Arbutus DB. There remains a disjoint between the Fir database, which contains historical data produced by the ETL process, and the real-time data collected on Arbutus Cloud.

On the Fir server, scripts were used to create a PostgreSQL DB schema and tables using DRAC, and historical data was exported from CSV on the user's local machine into the DB. Some issues were addressed, such as type casting for the volume column. Patching was also required.

A script was created to generate a table tailored to ML model requirements and populate it with appropriate data from tables containing stocks, bonds, indexes, and commodities. This includes One-hot encoding.

This project demonstrates how high-quality data, the ELT process, and resources provided by the DRAC Pilot Project Grant improve training ML models.

ACKNOWLEDGMENTS

We thank Okanagan College and the OC's Grants In Aid (GIA) Committee for funding and financial support of the applied student research projects. Additionally, we would like to thank the DRA of Canada for the DRI Champions Award.

A small portion of the documentation was reviewed and refined using AI-assisted writing tools, such as Grammarly, ChatGPT, and DeepSeek, to improve clarity and readability. The authors conducted all system design, implementation, experimentation, and analysis. AI was used to paraphrase in Existing Work and in the Discussion of Validation and Results.

References

[1] "Alliance cloud connect pilot." [Online]. Available: https://www.alliancecan.ca/en/accp

[2] "Financial Modeling Prep - FinancialModelingPrep — FMP." [Online]. Available: https://site.financialmodelingprep.com/

[3] Y. Khmelevsky, "SW Development Projects in Academia," *WCCCE 2009 - Proceedings of the 14th Western Canadian Conference on Computing Education*, vol. 1, no. 250, pp. 60–64, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1536292

[4] Y. Khmelevsky, M. Govorov, and L. Burge, "Okanagan College and Vancouver Island University educational joint projects results," in *Proceedings of the 14th Western Canadian Conference on Computing Education - WCCCE '09*, 2009, pp. 65–69. [Online]. Available: http://portal.acm.org/citation.cfm?id=1536274.1536293

[5] P. Sharma, M. Govorov, Y. Khmelevsky, and S. Dhanjal, "Oracle 9iAS Portal as a platform for Geographic Information Science distance and flexible learning at the University of the South Pacific," *WIT Transactions on Information and Communication Technologies*, vol. 31, 2004.

[6] Y. Khmelevsky and V. Voytenko, "Cloud computing infrastructure prototype for university education and research," in *Proceedings of the 15th Western Canadian Conference on Computing Education*, ser. WCCCE '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1806512.1806524

[7] L. C. Pérez, S. Cooper, E. K. Hawthorne, S. Wetzel, J. Brynielsson, A. G. Gökce, J. Impagliazzo, Y. Khmelevsky, K. Klee, M. Leary, A. Philips, N. Pohlmann, B. Taylor, and S. Upadhyaya, "Information assurance education in two- and four-year institutions," in *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Education - Working Group Reports*, ser. ITiCSE-WGR '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 39–53. [Online]. Available: https://doi.org/10.1145/2078856.2078860

[8] Y. Khmelevsky, V. Ustimenko, G. Hains, C. Kluka, E. Ozan, and D. Syrotovsky, "International collaboration in SW engineering research projects," in *Proceedings of the 16th Western Canadian Conference on Computing Education*, 2011, pp. 52–56.

[9] Y. Khmelevsky and V. Voytenko, "Strategies for teaching mobile application development," in *18th Western Canadian Conference on Computing Education*, vol. 18, 2013, pp. 8–13.

[10] ——, "Hybrid Cloud Computing Infrastructure in Academia." in *WCCCE 2015 - the 20th Western Canadian Conference on Computing Education, At May 8-9, 2015.* Vancouver Island University (VIU), Nanaimo, British Columbia, Canada., 2015.

[11] Y. Khmelevsky, "Ten Years of Capstone Projects at Okanagan College: A Retrospective Analysis," in *Proceedings of the 21st Western Canadian Conference on Computing Education*. New York, NY, USA: ACM, 2016, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2910925.2910949

[12] N. McDonald, D. Atkinson, Y. Khmelevsky, and S. McMillan, "Sport wearable biometric data encrypted emulation and storage in cloud," in *Canadian Conference on Electrical and Computer Engineering*, 2016.

[13] N. Mcdonald, D. Leader, C. K. Chiang, Y. Khmelevsky, R. Bartlett, and A. Needham, "A new online tool for gamer network performance analysis," in *2016 IEEE International Conference on Cybercrime and Computer Forensic (ICCCF)*, 2016, pp. 1–6.

[14] N. Mcdonald, D. Atkinson, C. Frank, Y. Khmelevsky, and S. McMillan, "Biometric data emulation and encryption for sport wearable devices (A case study)," in *2016 Annual IEEE Systems Conference (SysCon)*, 2016, pp. 1–6.

[15] Y. Khmelevsky, K. Chidlow, K. Sugihara, and K. Zhang, "Engaging and Motivating Students Through Programming Competitions and GIS Applied Research Projects," *Proceedings of the 22nd Western Canadian Conference on Computing Education*, 5 2017. [Online]. Available: http://dx.doi.org/10.1145/3085585.3088491

[16] B. Ward, Y. Khmelevsky, G. Hains, R. Bartlett, A. Needham, and T. Sutherland, "Gaming network delays investigation and collection of very large-scale data sets," in *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings*, 2017.

[17] M. Cocar, R. Harris, and Y. Khmelevsky, "Utilizing Minecraft bots to optimize game server performance and deployment," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017, pp. 1–5.

[18] D. Joiner, M. Clement, S. T. Chan, K. Pereira, A. Wong, Y. Khmelevsky, J. Mahony, and M. Ferri, "DW vs OLTP Performance Optimization in the Cloud on PostgreSQL (A Case Study)," in *2022 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2022, pp. 1–8.

[19] D. Rajput, W. J. Wang, and C. C. Chen, "Evaluation of a decided sample size in machine learning applications," *BMC Bioinformatics*, vol. 24, 12 2023. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://link.springer.com/content/pdf/10.1186/s12859-023-05156-9.pdf

[20] S. Mohammed, L. Budach, M. Feuerpfeil, N. Ihde, A. Nathansen, N. Noack, H. Patzlaff, F. Naumann, and H. Harmouch, "The effects of data quality on machine learning performance on tabular data," *arXiv preprint*, vol. arXiv:2207.14529, 2022, [Online]. Available: https://arxiv.org/pdf/2207.14529.pdf.

[21] S. Nosratabadi, A. Mosavi, P. Duan, P. Ghamisi, F. Filip, S. S. Band, U. Reuter, J. Gama, and A. H. Gandomi, "Data science in economics: Comprehensive review of advanced machine learning and deep learning methods," *Mathematics*, vol. 8, no. 10, 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/10/1799

[22] H. D. Chacón, E. Kesici, and P. Najafirad, "Improving financial time series prediction accuracy using ensemble empirical mode decomposition and recurrent neural networks," *IEEE Access*, vol. 8, pp. 117 133–117 145, 2020.

[23] J. Brownlee, *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Auckland, New Zealand: Machine Learning Mastery, 2020. [Online]. Available: https://books.google.ca/books?hl=en&lr=&id=uAPuDwAAQBAJ&oi=fnd&pg=PP1

[24] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *Bulletin of the Technical Committee on Data Engineering*, vol. 23, no. 4, pp. 3–13, Dec 2000.

[25] S. V. Salunke and A. Ouda, "A performance benchmark for the postgresql and mysql databases," *Future Internet*, vol. 16, no. 10, p. 382, 2024, published 19 October 2024. [Online]. Available: https://doi.org/10.3390/fi16100382

[26] U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson, "Data integration flows for business intelligence," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, ser. EDBT '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1–11. [Online]. Available: https://doi.org/10.1145/1516360.1516362

[27] P. Vassiliadis, A. Simitsis, and K. Giorgakopoulou, "Metrics for the prediction of evolution impact in etl ecosystems: A case study," *Journal on Data Semantics*, vol. 1, no. 2, pp. 75–97, 2012.

[28] B. Berisha, E. Méziu, and I. Shabani, "Big data analytics in cloud computing: an overview," *Journal of Cloud Computing*, vol. 11, no. 24, 2022. [Online]. Available: https://doi.org/10.1186/s13677-022-00301-w

[29] J. M. Hellerstein, M. Stonebraker, and J. Hamilton, "Architecture of a database system," *Foundations and Trends® in Databases*, vol. 1, no. 2, pp. 141–259, 2007.

[30] S. Chaudhuri and G. Weikum, "Rethinking database system architecture: Towards a self-tuning risc-style database system." in *VLDB*, 2000, pp. 1–10.