Neurocomputing 646 (2025) 130432

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Original Software Publication

Accelerated Weka: GPU Machine Learning with Weka Workbench

Guilherme Weigert Cassales *, Justin Jia Liu, Albert Bifet

AI Institute, University of Waikato, Hamilton, New Zealand

ARTICLE INFO

Communicated by Z. Wang

Keywords:
GPU
Machine learning software
Classification
Ensemble methods
Open-source software

ABSTRACT

Recent advancements in Machine Learning (ML) have driven increased interest and demand in the field. However, the complexity and mathematical background required for many techniques can be challenging for newcomers and enthusiasts. For many years, Weka has been a cornerstone in introductory ML courses worldwide, offering a user-friendly interface that facilitates understanding of various methods. As an open-source software with an active community, Weka has continued to evolve with new attributes. With the growing size of datasets, users are increasingly looking for ways to accelerate computations. GPUs have become a preferred solution for efficiently building and deploying ML models. This paper presents Accelerated Weka, a software framework that integrates GPU-accelerated methods within Weka's intuitive graphical user interface, significantly reducing execution times for large datasets while preserving Weka's accessibility. Benchmark results indicate that Accelerated Weka can achieve 2,198x speedup on an A100 chip. The framework retains Weka's GPL 3.0 license and offers a straightforward installation process through the Conda environment.

1. Introduction

In recent years, there has been a surge in the development and adoption of Machine Learning (ML) tools. Open source tools such as Weka[1] and scikit-learn[2] have become standard in the field, each with years of development and widespread usage. Weka in particular, has become a staple in introductory ML education due to its user-friendly Graphical User Interface (GUI) that allows beginners to easily explore various ML algorithms

Technological advancements, such as faster GPUs and libraries designed to exploit GPU computation, have revolutionized the field by significantly accelerating compute-intensive models. However, adapting mature software with years of development to harness these new technologies is a complex, difficult, and time-consuming task. Many methods need to be completely reformulated to exploit GPU-based data-parallelism.

Despite Weka's extensive attributes, its reliance on CPU processing limits its performance with large-scale datasets, an increasing issue as models become more compute-intensive and the data sphere keeps growing. It is, therefore, important to provide as much GPU acceleration as possible while maintaining Weka's user-friendly GUI. This improvement can greatly benefit users by combining the power of GPU computation with the accessibility and educational value of Weka's interface.

This paper introduces Accelerated Weka,[3] a software framework designed to simplify the installation, configuration, and utilization of GPU-accelerated ML within Weka. At the framework's core is wekaRAPIDS,[4] a new integration developed by the authors to connect Weka with the RAPIDS suite of GPU-accelerated libraries [1]. Accelerated Weka also incorporates existing packages like WekaDeeplearning4j (WDL4J) [2] and wekaPython, providing a seamless environment with minimal setup for exploring GPU-accelerated ML algorithms through Weka's intuitive GUI.

2. Accelerated weka

The framework is designed to support users with limited experience in system configuration, offering an straightforward setup process and GUI-based configuration for executing ML tasks. By bundling essential GPU-accelerated libraries within a single installation package, our solution eliminates the complexity typically associated with setting up GPU-based ML environments.

* Corresponding author.
E-mail addresses: guilherme.cassales@waikato.ac.nz (G.W. Cassales), justin.l@waikato.ac.nz (J.J. Liu), abifet@waikato.ac.nz (A. Bifet).

[1] In this document, Weka refers to the open-source Weka Workbench, available at: https://ml.cms.waikato.ac.nz/weka/. It should not be confused with the similarly named commercial cloud data platform.

[2] https://scikit-learn.org/stable/.

[3] https://github.com/Waikato/acceleratedWeka.

[4] https://github.com/Waikato/wekaRAPIDS.

[5] https://docs.conda.io/projects/conda/en/latest/index.html.

https://doi.org/10.1016/j.neucom.2025.130432
Received 30 July 2024; Received in revised form 28 April 2025; Accepted 2 May 2025
Available online 17 May 2025

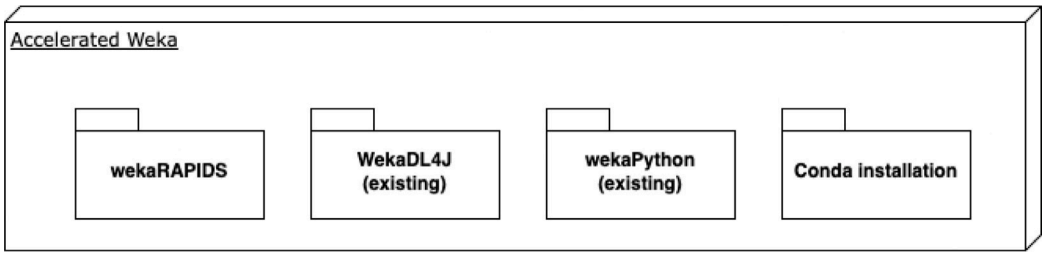

Fig. 1. Architecture of packages.

The framework relies on conda[5] to automate environment setup with a single command. In addition, it also introduces compatibility with the RAPIDS, an API suite of GPU-accelerated data science and AI libraries that mirror popular open-source tools such as pandas [3] and scikit-learn [4]. Key RAPIDS components include cuDF [5] for GPU-accelerated DataFrame processing and cuML [6] for GPU-accelerated ML algorithms.

2.1. Architecture

The architecture of Accelerated Weka includes WDL4J, which integrates deep learning capabilities via the Deeplearning4j (DL4J) library, and wekaRAPIDS, which extends Weka's support to traditional ML methods like classification and regression through RAPIDS integration. While WDL4J handles deep learning, wekaRAPIDS covers a broad range of classical ML methods. Thus, both packages have a synergetic nature rather than a competitive one. To facilitate future updates, wekaRAPIDS employs a base wrapper class that implements all supported RAPIDS algorithms, allowing easy access and parameter configuration through Weka's GUI. Supported RAPIDS methods include LogisticRegression, Ridge, Lasso, ElasticNet, MBSGDClassifier, MBSGDRegressor, MultinomialNB, BernoulliNB, GaussianNB, RandomForestClassifier, RandomForestRegressor, SVC, SVR, LinearSVC, KNeighborsClassifier, and KNeighborsRegressor.

Fig. 1 illustrates the package architecture, showing both new and existing components. The figure also portrays how Accelerated Weka provides a combined package that includes existing packages WekaDL4J and wekaPython, as well as the newly developed wekaRAPIDS, under an easy installation setup that uses Conda.

While the supported RAPIDS methods were carefully selected, they are not exhaustive, and future versions of RAPIDS may include additional relevant algorithms. To accommodate this, wekaRAPIDS was designed with extensibility in mind. New learners can be integrated by adding their configuration to the `CuMLClassifier` class. Once the parameter setup is adjusted with the appropriate identifying flags, the system automatically instantiates the new classes and handles execution.

For Python integration within Weka, the original `wekaPython` package established communication with Python libraries such as scikit-learn and XGBoost[6] by creating a server and exchanging data through sockets. This setup allowed Weka users to run Python code directly within the Weka Workbench. Initially, wekaRAPIDS used a similar approach for integrating RAPIDS. However, to improve performance, it was later upgraded to leverage Apache Arrow and GPU memory sharing, significantly reducing the overhead associated with data transfer between Java and Python. Fig. 2 illustrates the workflow of Accelerated Weka, highlighting the efficient data exchange facilitated by GPU memory sharing and Apache Arrow.

Apache Arrow provides a cross-language, in-memory columnar data format optimized for analytical workloads. Its structure allows applications in different languages (such as Java and Python) to share data

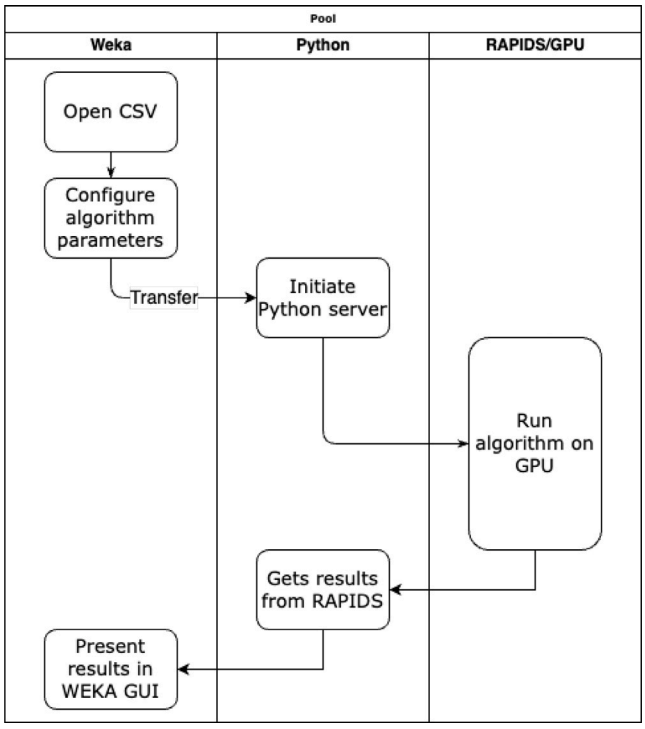

Fig. 2. Workflow of accelerated Weka.

with minimal serialization and deserialization overhead. In wekaRAPIDS, Apache Arrow is used to represent Weka Instances as Arrow Tables, enabling zero-copy data transfers between Java and Python environments. The `sendInstancesArrow` function within the `RapidsUtils` class converts Weka `Instances` objects into the Apache Arrow format. Each attribute is transformed into an Arrow object called `ColumnVector`, which are then combined into an `ArrowTable`. The table is serialized using Apache Arrow's Inter-Process Communication (IPC) protocol and transmitted via an `OutputStream` to the Python environment, where it can be directly consumed by RAPIDS algorithms. This approach greatly reduces latency compared to traditional socket-based transmission, where data must be serialized, sent over the network, and deserialized on the receiving side.

The, alternative, GPU memory sharing method enables different processes or applications to access data stored directly in GPU memory, without the need to first transfer it back to CPU memory. In the context of wekaRAPIDS, this means that once data is prepared for GPU-accelerated processing (e.g., by RAPIDS libraries), it remains on the GPU throughout the computation. The `sendInstancesCuda` function is used to provide a lower-level, more direct, transfer by flattening the dataset into a floating-point array and transferring the data directly from host to GPU memory. This process allocates memory on the GPU via CUDA APIs (i.e., `cudf`), which allows the Python process to access the shared GPU memory without any serialization or copying between

[6] https://xgboost.readthedocs.io/en/stable/python/python_intro.html.

CPU and GPU. By avoiding the expensive step of repeatedly moving data between CPU and GPU memory, GPU memory sharing improves memory efficiency and significantly accelerates processing times.

Together, Apache Arrow and GPU memory sharing create an efficient, streamlined data transfer pipeline within wekaRAPIDS. These optimizations not only lower runtime overhead but also enable the processing of large datasets that would otherwise suffer significant delays with conventional socket-based communication.

More broadly, the design choices in Accelerated Weka ensure that the system remains future-ready. As GPU architectures, IPC technologies, and machine learning libraries continue to evolve, Accelerated Weka's flexible infrastructure allows users to benefit from these advancements with minimal disruption. Faster data handling, smarter integration, and scalability are at the core of Accelerated Weka's architecture — preparing Weka users for the next generation of machine learning workflows.

2.2. Installation and usage

Accelerated Weka is easily installed using Conda, enabling users to set up all necessary dependencies with a single command. For example, to install on a Linux system, open a terminal and run the following command:

```
conda create --solver=libmamba -n accelweka -c rapidsai -c nvidia -c conda-forge -c waikato weka
```

After Conda environment creation, activate it:

```
conda activate accelweka
```

Once activated, the Weka GUI can be launched with the command:

```
weka
```

Additionally, users can run experiments from the command line, with examples provided in the documentation.[7]

3. Benchmarks

We evaluated the performance of Accelerated Weka by comparing the execution time of standard Weka algorithms (CPU) with their GPU-accelerated counterparts from wekaRAPIDS. To demonstrate performance across a range of systems, we ran benchmarks on both consumer-grade and high-performance hardware: a Ryzen 9 5950X with an RTX 4090, and a DGX Station with EPYC 7742 CPUs and up to four A100 GPUs. Unless stated otherwise, all experiments use a single GPU.

We used a mix of synthetic and real-world datasets. Synthetic datasets were chosen for reproducibility and control over attributes and instance counts (see Section 4). Two key generators were used: Random Decision Generator (RDG) and Random Radial Basis Function (RandomRBF).

RDG creates data by generating instances from a list of rules. Each rule can have several boolean tests on input attributes, controlled by `minRuleSize` and `maxRuleSize`, which we set to 5 and 20 (default: 1 and 10). All other parameters were left as default, except the number of instances and attributes. We generated datasets with 1, 2, 5, and 10 million instances (each with 10 attributes), and one with 5 million instances and 20 attributes.

RandomRBF defines class centers randomly, assigning each a weight, position, and standard deviation. Instances are generated by sampling around these centers, with class labels derived from the selected center. This generator produces only numeric attributes.

Dataset names encode their configurations: the number of attributes appears after an underscore (_), followed by 'a' (e.g., `_a10`); the number of instances appears after '_n_' (e.g., `_n5m` for 5 million instances).

While synthetic datasets help explore performance under controlled conditions, real-world data shows where Accelerated Weka can offer practical benefits. Next we briefly introduce them. The **covertype** [7] dataset contains 581 K instances and 54 attributes. It maps ecological and cartographic data (e.g., elevation, soil type) to one of seven forest cover classes. Useful for testing multiclass classification, mixed attribute types, and imbalanced data. The **Bar Crawl** [8] dataset has time series accelerometer data (14M instances, 15 attributes) collected during a bar crawl, with a regression target of Transdermal Alcohol Concentration (TAC). Can be reformulated as binary classification via a TAC threshold. **PAMAP2** [9] dataset collected human-activity data (3.8M instances, 54 attributes) from 9 participants performing 18 physical activities. It is suitable for activity recognition, segmentation, and classification tasks. Lastly, the **Physical** [10] 6-XOR_64 bit dataset simulates Physical Unclonable Functions (PUFs), it contains 2.4M instances and 65 attributes. It is used in hardware security research.

3.1. Results

Table 1 presents the average running time for executing the Random Forest algorithm on the Ryzen 9 5950X CPU and the RTX 4090 GPU, while Table 2 presents the average running time of the kNN algorithm on the EPYC 7742 CPU and the A100 GPUs. Both tables present a column with the average transfer time among all experiments, which measures how much of the execution was spent transferring the dataset between the JVM and the python server. Furthermore, rows are arranged in ascending order of speedup to facilitate clearer comparisons across datasets.

3.2. Discussion

As a general observation, the benchmarks show that Accelerated Weka is most beneficial for compute-intensive tasks, though this benefit strongly depends on the dataset size and algorithm characteristics.

Notably, in Table 1 the smallest datasets (e.g., RandomRBF variants with 100, 1000, or 5000 instances) often perform worse on the GPU than on the CPU. This is due to the fixed overheads of initializing GPU libraries and transferring data to GPU memory, which are not offset by speedups when the dataset is too small.

Another interesting aspect of the benchmark is that the second experiment consistently achieves much higher speedups than the first. We attribute this to two factors: (i) the use of more specialized GPU hardware, and (ii) the nature of kNN, which involves extensive distance calculations that map well to GPU data-parallelism. The results in Table 2, indicate that GPU acceleration begins to show tangible gains around 100,000 instances. For instance, RandomRBF datasets of this size achieve speedups of 2.8× and 3.98×, which, while modest, mark the threshold where GPU acceleration starts becoming beneficial. Substantially larger gains are observed for datasets exceeding one million instances – such as RBF_a50_n1 m, PAMAP2, and Physical – where speedups range from 271.84× to 2198.16×.

Interestingly, adding up to three additional GPUs provides no further improvement on these large datasets. Although more investigation is needed, we hypothesize that inter-GPU communication overhead may be limiting scalability in these cases.

3.2.1. Deeper insights

Table 1 includes the number of instances and features for each dataset. A clear trend emerges: increasing the number of instances helps *alleviate* the overhead associated with data transfer and GPU library initialization.

A particularly interesting comparison arises between RDG1_5m_20a and RDG1_10 m. Although both datasets contain roughly the same

[7] https://waikato.github.io/acceleratedWEKA/user_guide/getting_started/.

Table 1
Execution time of experiments with Random Forest using cross-validation comparing the baseline CPU (Ryzen 9 5950X) execution time with the Accelerated Weka execution time while sharing GPU memory on a RTX 4090. All times (i.e., CPU, GPU, and Transfer) in seconds.

Dataset	Instances	Features	CPU	GPU	Transfer	Speedup
RBFa5k	100	5000	4.095	27.99	10.90	0.15
RBFa5kn1k	1,000	5000	9.43	32.33	12.55	0.29
RBFa5kn5k	5,000	5000	36.05	50.11	19.91	0.72
RDG1_1m	1,000,000	10	520.92	57.74	9.98	9.02
RDG1_2m	2,000,000	10	1251.37	96.08	13.47	13.02
RDG1_5m	5,000,000	10	3598.81	233.20	25.88	15.43
RDG1_10m	10,000,000	10	7772.89	490.66	52.00	15.84
RDG1_5m_20a	5,000,000	20	5240.75	302.97	45.77	17.30
Bar_Crawl	14,000,000	15	45,654.22	1610.07	148.42	28.36
PAMAP2	3,850,000	54	51,599.36	741.22	66.09	69.61
Physical	2,400,000	65	23,173.77	275.21	66.25	84.20

Table 2
Execution time of experiments with KNN without using cross-validation comparing the baseline CPU (EPYC 7742) execution time with the accelerated Weka execution on 1x and 4x NVIDIA A100 GPU. All runtimes in seconds.

Dataset	CPU	1x A100	Speedup	Transfer	4x A100	Speedup	Transfer
RBFa50n10k	2.31	42.24	0.05	10.94	37.33	0.06	5.23
RBFa500n10k	2.40	44.43	0.05	6.44	39.80	0.06	5.93
RBFa5kn5k	6.58	59.94	0.11	10.68	56.21	0.12	10.42
RBFa5kn10k	11.54	62.98	0.18	13.88	59.82	0.19	14.23
RBFa500n100k	182.97	65.36	2.80	12.39	45.97	3.98	13.34
RBFa50n100k	177.34	43.37	4.09	5.90	37.94	4.67	5.87
covertype	3,755.80	67.05	56.01	9.86	42.42	88.54	10.23
RBFa50n1m	21,021.74	77.33	271.84	12.36	46.00	456.99	12.39
PAMAP2	197,494.26	178.64	1105.54	31.02	181.37	1088.90	30.40
Physical	313,260.23	142.51	2198.16	32.15	145.62	2151.22	31.51

total data volume (instances × features), the former has fewer instances but twice as many features. Despite this, RDG1_5m_20a exhibits faster transfer and runtime on both CPU and GPU, suggesting that a higher feature count may contribute positively to performance in this context.

A similar pattern is observed when comparing Bar_Crawl and PAMAP. Both datasets contain a comparable number of data points – approximately 210 million and 208 million floats, respectively – yet PAMAP demonstrates notably faster transfer and GPU execution times. One possible explanation is that Bar_Crawl's larger feature count leads to the creation and traversal of deeper decision trees, especially given Weka's default setting of no height limit in Random Forest. This effect may introduce additional computational complexity that offsets the GPU's parallelism advantage.

Finally, the Physical dataset reinforces this bias toward high-dimensional data benefiting more from GPU acceleration. With its relatively large number of features, it achieves the highest speedup among all datasets in this experiment.

4. Documentation

The documentation for Accelerated Weka can be found at https://waikato.github.io/acceleratedWeka/. It includes a GUI tutorial, a CLI tutorial, references for other packages that have been used, and a template bash script[8] with an example on how to easily run benchmarks using Accelerated Weka.

5. Conclusion

Accelerated Weka allows users to exploit GPU-accelerated methods in Weka through the wekaRAPIDS and WDL4J packages, while maintaining the easy-to-use GUI. The installation process is simplified using the Conda environment, making it straightforward to use our solution from the beginning. Finally, Accelerated Weka and the newly developed wekaRAPIDS keep the same GPU3.0 license as Weka, with the easiness of extending support of newly launched RAPIDS algorithms into wekaRAPIDS.

[8] https://github.com/Waikato/wekaRAPIDS/blob/main/test.sh.

CRediT authorship contribution statement

Guilherme Weigert Cassales: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Conceptualization. **Justin Jia Liu:** Writing – review & editing, Validation, Software, Methodology, Investigation, Data curation. **Albert Bifet:** Writing – review & editing, Supervision, Resources, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported by the TAIAO project CONT-64517-SSIFDS-UOW (Time-Evolving Data Science/Artificial Intelligence for Advanced Open Environmental Science), funded by the New Zealand Ministry of Business, Innovation, and Employment (MBIE). For more information, see https://taiao.ai/

Data availability

Data will be made available on request.

References

[1] NVIDIA, Rapids: Data science at the speed of thought, 2024, https://rapids.ai/learn-more/. (Accessed 28 July 2024).

[2] S. Lang, F. Bravo-Marquez, C. Beckham, M. Hall, E. Frank, WekaDeeplearning4j: A deep learning package for weka based on deeplearning4j, Knowl.-Based Syst. 178 (2019) 48–50, http://dx.doi.org/10.1016/j.knosys.2019.04.013, URL http://www.sciencedirect.com/science/article/pii/S0950705119301789.

[3] Wes McKinney, Data structures for statistical computing in python, in: Stéfan van der Walt, Jarrod Millman (Eds.), Proceedings of the 9th Python in Science Conference, 2010, pp. 56–61, http://dx.doi.org/10.25080/Majora-92bf1922-00a.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[5] NVIDIA, Rapids: Cudf - gpu dataframes, 2024, https://docs.rapids.ai/api/cudf/stable/. (Accessed 28 July 2024).
[6] NVIDIA, RAPIDS: cuML - GPU machine learning algorithms, 2024, https://docs.rapids.ai/api/cuml/stable/. (Accessed 28 July 2024).
[7] J. Blackard, Covertype, UCI machine learning repository, 1998, http://dx.doi.org/10.24432/C50K5N.
[8] Bar crawl: Detecting heavy drinking, UCI machine learning repository, 2020, http://dx.doi.org/10.24432/C5TK6G.
[9] A. Reiss, PAMAP2 physical activity monitoring, UCI machine learning repository, 2012, http://dx.doi.org/10.24432/C5NW2H.
[10] A. Aseeri, M. Alkatheiri, Y. Zhuang, Physical unclonable functions, UCI machine learning repository, 2018, http://dx.doi.org/10.24432/C5D03R.

Guilherme Weigert Cassales is a postdoctoral researcher at the AI Institute, University of Waikato. His research focuses on Machine Learning for Data Streams, including change detection, anomaly detection, semi-supervised learning, handling label delays, and improving efficiency of streaming algorithms. He is also interested in explainable AI (XAI) and time series forecasting.

Justin Jia Liu is currently pursuing a Ph.D. in Artificial Intelligence at The University of Waikato, focusing on anomaly detection in streaming data. His research addresses the challenges of detecting unusual patterns in dynamic and evolving data streams. Jia's work explores innovative algorithms and techniques to improve the accuracy and efficiency of real-time anomaly detection, adapting to concept drift and evolving data distributions.

Albert Bifet is a Professor of AI, Director of the Te Ipu o te Mahara AI Institute at the University of Waikato and Co-chair of the Artificial Intelligence Researchers Association (AIRA). His research focuses on Artificial Intelligence, Big Data Science, and Machine Learning for Data Streams. He is leading the TAIAO Environmental Data Science project and co-leading the open source project MOA Massive On-line Analysis. He is the co-author of a book on Machine Learning from Data Streams published at MIT Press. He is one of the winners of the best paper award at the ACM Conference on Fairness, Accountability, and Transparency (ACM FAccT) 2023, and he is the general co-chair of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD) 2024Te Ipu o te Mahara AI Institute: https://ai.waikato.ac.nz/.