BUT 2 Internship Report

Research and development for machine learning using functional programming

7 April to 13 June 2025

Host organization : *Mr BEKKI Daisuke*
Reference professor : *Mr Bruno MERY*
University Year : *2024 - 2025*

Castillo Theo

Acknowledgment

I would like to sincerely thank Bekki-sensei for welcoming us into his laboratory and giving us the opportunity to complete this internship. He greeted us with great kindness and made sure we were fully integrated into the team.

I would also like to express my gratitude to Sora-sensei, who, despite a very busy schedule, was always available and responsive in supervising our internship.

A heartfelt thank you to Mr. Mery and Ms. Cartier, who go above and beyond every year to offer students such enriching experiences.

I am also deeply grateful to my parents, whose financial support was essential in making this ambitious project possible.

Finally, I would like to thank everyone in the laboratory for their warm hospitality and kindness. In particular, my thanks go to Ms. Koharu Saeki and Tanaka-san for greatly contributing to the friendly and welcoming atmosphere that made this internship so memorable.

Abstract

This internship report documents research conducted at the Bekki Lab of Ochanomizu University, exploring machine learning through functional programming using Hasktorch, the Haskell equivalent of PyTorch, under the supervision of Professor Daisuke Bekki and Ms. Sora Tagami. The work progressed from fundamental neural network implementations including the Titanic survival prediction and Word2Vec Skip-Gram models to the ambitious complete reimplementation of GPT-2 architecture in Hasktorch. This pioneering approach demonstrates the feasibility of developing state-of-the-art language models in functional programming environments.

Résumé

Ce rapport de stage documente les recherches menées au Bekki Lab de l'Université d'Ochanomizu, explorant l'apprentissage automatique à travers la programmation fonctionnelle en utilisant Hasktorch, l'équivalent Haskell de PyTorch, sous la supervision du Professeur Daisuke Bekki et de Mme Sora Tagami. Le travail a progressé depuis l'implémentation de réseaux de neurones fondamentaux incluant la prédiction de survie du Titanic et les modèles Word2Vec Skip-Gram jusqu'à la réimplémentation complète et ambitieuse de l'architecture GPT-2 en Hasktorch. Cette approche pionnière démontre la faisabilité du développement de modèles de langage de pointe dans des environnements de programmation fonctionnelle.

Contents

1 Context

1.1 Ochanomizu University

Figure 1: Ochanomizu University

Established in 1875 as the first higher education institution for women in Japan, Ochanomizu University takes its name (”water for tea”) from a historical event in which Tokugawa Hidetada, the shogun, made tea using local, pure water. Originally located in Ochanomizu, the university moved to Otsuka Bunkyo-ku during the Great Kanto Earthquake in 1923. After becoming a national university corporation in 2004, it works to empower women who experience gender inequality while also welcoming international students. With 2,807 students as of 2021, the university actively supports the goals of sustainable development while excelling in a variety of scientific fields like biology and computer science. The campus has a great restaurant serving traditional Japanese dishes and a wide variety of extracurricular activities [1].

1.2 Bekki Lab

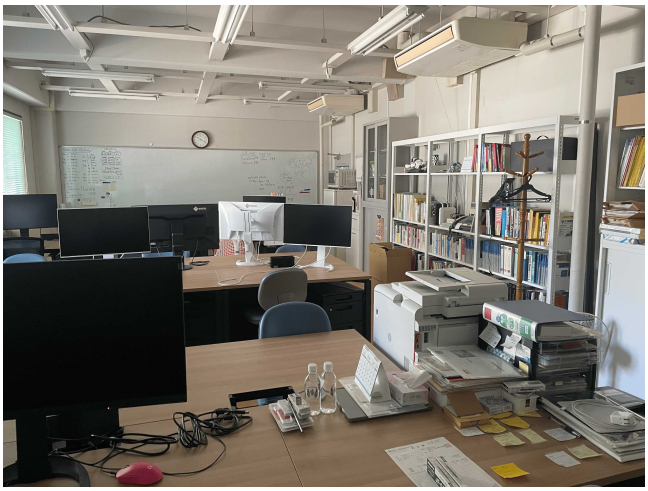


Figure 2: Bekki Lab

The Bekki Lab, directed by Daisuke Bekki who is also my tutor, specializes in mathematical linguistics, particularly Combinatory Categorial Grammar (CCG) and Dependent Type Semantics (DTS) [2]. The laboratory structure is straightforward: Mr. Bekki serves as the director, Ms. Tanaka holds the position of secretary, and Ms. Tagami Sora, a former student now working as a Software Engineer at Google Maps, supervises our internship activities thanks to her expertise in Hasktorch.
The laboratory also includes nine other female students, ranging from second-year undergraduates to second-year master's students. There are also two other Japanese first-year master's students doing the same internship as us who attend the meetings. Upon our arrival, a welcome meeting was held with all the students from the lab. We all introduced ourselves and got acquainted. Desks were assigned to us.
Our work environment is friendly and open to questions. Our presence is only required during the weekly Tuesday meeting, where we receive exercises and share our progress with Mr. Bekki, who seeks to integrate neural machine learning networks with traditional linguistic theories. These meetings take place sitting on the floor around a low table in very small groups, allowing us to easily exchange ideas and ask questions.
Unlike the other students who must take courses simultaneously, we are solely focused on our internship. We therefore see them coming in and out of the laboratory throughout the day. There is nobody at the laboratory on Wednesdays and Thursdays, so we work from home on those days. We are generally at the laboratory on Mondays and Tuesdays, and work remotely the rest of the week.
The laboratory is funded by Ochanomizu University and the Japanese govern-

ment for its research projects.

1.3 Organization

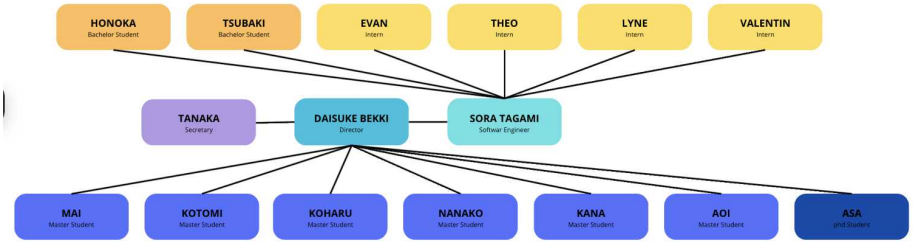


Figure 3: Bekki lab organization

1.4 Haskell and Hasktorch

The main objective of my internship is to explore and master Hasktorch[1] [3], which is a Haskell Library that offers an innovative approach to neural network development through functional composition and abstract data types.

The first significant challenge of this internship is learning the Haskell programming language, which is uncommon and relatively little-used. This language has the distinction of being a pure functional language, making its learning quite complicated as it is fundamentally different from the more commonly used imperative languages.

The second and most important challenge is learning the Hasktorch library. This library is the equivalent of PyTorch[2] but in the Haskell language, offering powerful tools for Machine Learning and neural network development. Although still in the initial phase of development, Hasktorch shows great potential for improving the efficiency of neural network coding.

To achieve this objective, we are following a precise program, detailed by week. Each week includes different missions related to Hasktorch and deep learning[3] in general in order to master the various concepts. A weekly meeting is organized during which we must report on what we have accomplished.

[1] Hasktorch is a tensor and neural network library for Haskell

[2] PyTorch is an open-source machine learning library used for deep learning applications and neural network development in Python.

[3] Deep learning is a subset of machine learning that uses artificial neural networks

Finally, to put all this learning to use, during the last two weeks, we will have the opportunity to create a project of our choice related to the Hasktorch library.

2 Activity

2.1 Why Haskell and Hasktorch?

The first question we asked Mr. Bekki concerned the choice of using Haskell for this project. Indeed, Haskell is very rarely used in the field of machine learning and therefore much less documented than languages like Python. This situation is explained by its rather steep learning curve and the paradigm shift it imposes.

Haskell's Specificities Haskell is a purely functional language, which means there are no mutable states and everything is immutable. A function with the same parameters will always return the same result because there are no side effects. This property, called referential transparency, guarantees predictable behavior and facilitates reasoning about the code.

Motivations for the Choice Several reasons justify this seemingly counter-intuitive choice:

- **Conceptual Adequacy**: It is actually simpler and more natural to implement certain ideas with a functional language, particularly for complex mathematical algorithms.
- **Proximity to Mathematics**: A functional language is closer to mathematics, which facilitates the implementation of machine learning algorithms that rely on solid mathematical foundations.
- **Training and Personal Challenge**: Working with Haskell and Hasktorch makes us better programmers. The use of generative AI is very inconclusive with these technologies because very few resources are present on the internet, which forces us to deeply understand the underlying concepts.
- **Compiler Optimizations**: The fact that Haskell is a functional language allows for performance gains, as the compiler is capable of performing advanced optimizations thanks to the absence of side effects and the purity of functions.

Although this choice represents an additional challenge, it offers a unique opportunity to explore the implementation of language models in a different paradigm, while developing a deeper understanding of the underlying algorithms.

2.2 Neural Networks

A neural network is a computational entity capable of processing input data to produce outputs. It consists of an input layer, an arbitrary number of hidden

layers, and an output layer. On each layer are several neurons, each having specific weights for each of its input connections. The output of a neuron corresponds to the weighted sum of its inputs, transformed by an activation function[4]. The objective of the neural network is to optimize its parameters (weights and biases) in order to learn to correctly map inputs to desired outputs.

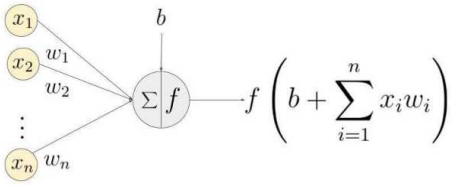


Figure 4: Computation for one neuron

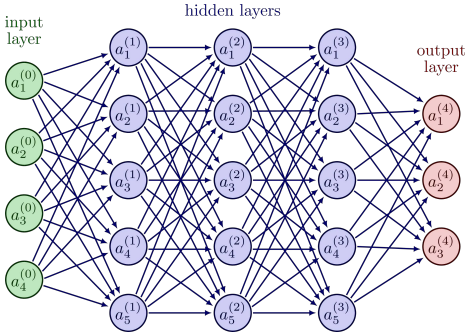


Figure 5: Neural Network Illustration

2.2.1 Activation Functions

Activation functions introduce non-linearity into neural networks, allowing the learning of complex relationships between data. They determine whether a neuron should be activated based on the weighted sum of its inputs.

The main functions used include:

- **ReLU**: $f(x) = \max(0, x)$ - Simple and effective, avoids the vanishing gradient problem
- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$ - Output between 0 and 1, used in binary classification
- **Tanh**: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ - Output between -1 and 1, faster convergence than sigmoid

[4]An activation function is a mathematical function applied to a neuron's output to introduce non-linearity

The choice of activation function directly influences the performance and learning speed of the model.

2.2.2 Loss Functions

Loss functions quantify the gap between model predictions and actual values. They guide the optimization of network parameters during training.

Commonly used functions are:

- **Mean Squared Error (MSE)**: $L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ - Regression, heavily penalizes large errors
- **Cross-Entropy**: $L = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$ - Classification, measures divergence between distributions
- **Mean Absolute Error (MAE)**: $L = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$ - Regression, less sensitive to outliers

The choice of loss function depends on the type of problem (regression, classification) and the desired sensitivity to errors.

2.2.3 Optimizer

Optimizers determine how network parameters are updated to minimize the loss function. They control the speed and direction of learning by adjusting weights according to calculated gradients.

The main optimization algorithms are:

- **SGD**: Simple and robust. Weight update: weights = weights - learning_rate × gradients
- **Adam**: Combines momentum and learning rate adaptation

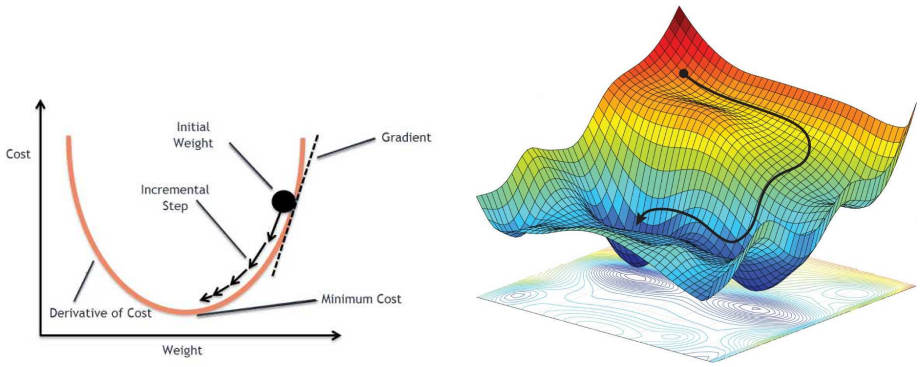
The choice of optimizer influences model convergence, training stability and final performance.

The **learning rate** α[5] is a crucial hyperparameter that controls the step size when updating parameters. A rate that is too high can cause divergence, while a rate that is too low considerably slows convergence. Techniques like scheduling allow this rate to be dynamically adapted during training.

2.3 Titanic Case Study

One of our practical projects was to train a model to predict from passenger information such as their age, sex, class, etc., whether they would survive or not [4].

[5]The learning rate is a hyperparameter that controls the size of steps taken during gradient descent optimization, determining how quickly or slowly a model learns.



(a) Learning Rate (b) Optimizer

Figure 6: Comparison of Learning Rate and Optimizer Effects

2.3.1 Data Management

The training of our prediction model relies on a dataset obtained via the Kaggle[6] community platform. Although the amount of data is important for machine learning models, the quality and relevance of the data are determining factors for the final performance of the model.

Evaluating Data Relevance A critical analysis of each variable in the dataset is essential. For example, the presence of a ”name” column in our dataset requires an evaluation of its predictive value. Proper names, being unique identifiers, generally provide no exploitable information for prediction and can introduce noise into the model.

Data Types Data present in a dataset falls into two main categories:

- **Quantitative data**: Numerical variables directly interpretable by the model (age, size, income, etc.). This data can be continuous or discrete and is naturally compatible with learning algorithms.
- **Qualitative data**: Categorical variables requiring preprocessing to be exploitable by the model. Take the example of eye color with the values ”blue”, ”green”, ”brown”. This textual data must be numerically encoded via techniques like one-hot encoding[7] or ordinal encoding depending on their nature.

[6] Kaggle is an online community platform for data scientists and machine learning practitioners, hosting competitions, datasets, and educational resources.

[7] One-hot encoding is a technique that converts categorical variables into binary vectors where only one element is 1 and all others are 0, representing each category uniquely.

Data Division The final step consists of partitioning the dataset into three distinct subsets, each having a specific role in the learning process:

- **Training set (70-80%)**: Used for learning model parameters
- **Validation set (10-15%)**: Allows hyperparameter[8] adjustment and optimal model selection
- **Test set (10-15%)**: Evaluates final model performance on unseen data during training

This separation guarantees an objective evaluation of performance and prevents overfitting[9] by maintaining data completely unknown to the model until final evaluation.

2.3.2 Training

Model training is a critical phase requiring careful parameterization and rigorous monitoring of performance metrics.

Training Configuration Training hyperparameters are selected according to best practices:

- **Optimizer**: Adam with an initial learning rate of 0.001, combining the advantages of momentum and automatic learning rate adaptation
- **Loss function**: Mean Squared Error (MSE) for regression or Cross-Entropy for classification
- **Batch size**: 32 or 64 depending on dataset size, offering a compromise between stability and computational efficiency
- **Number of epochs**[10]: Maximum of 100 epochs with early stopping[11] based on validation loss

Training Process Training follows a structured protocol:

1. **Initialization**: Random weights initialized
2. **Training loop**: For each epoch, the model processes the training set in batches, calculates loss and updates weights via backpropagation[12]

[8] Hyperparameters are configuration variables that are set before the learning process

[9] Overfitting occurs when a model learns the training data too well, including its noise and peculiarities, resulting in poor generalization to new, unseen data.

[10] An epoch represents one complete pass through the entire training dataset

[11] Early stopping is a regularization technique that stops training when the model's performance on a validation set stops improving, preventing overfitting.

[12] Backpropagation is an algorithm used to calculate gradients of the loss function with respect to the network's weights, enabling efficient training of neural networks.

3. **Validation**: Evaluation on the validation set after each epoch without weight updates
4. **Monitoring**: Tracking convergence via training and validation loss curves

Tracking Metrics Several metrics allow evaluation of training quality:

- **Training loss**: Progressive decrease indicating model learning
- **Validation loss**: Stabilization or slight increase signaling optimal stopping point
- **Accuracy**[13]: For classification tasks, percentage of correct predictions

Overfitting Management Early detection of overfitting is performed by observing the divergence between training and validation loss curves. When validation loss begins to increase while training loss continues to decrease, training is automatically stopped to preserve the best generalization performance.

2.4 Natural Language Processing

Now instead of only processing numerical data, we want to process text.

2.4.1 Word embedding

Embedding is a method to extract the meaning of a word into an n-dimensional vector that the model is then able to understand.

For this, we must first teach the model how to represent and interpret words. For this, there is a method called word2vec. During our learning, we implemented word2vec Skip Gram which consists of training the model on the entire corpus to predict from a word that we call the center, the n words found around it. N being an arbitrary parameter that we call windowSize.

To evaluate this type of model, we used a standardized test called Semantic Textual Similarity (STS) [5]. This test works as follows: we have a list of sentence pairs, and each pair has a score that indicates how similar the two sentences are in meaning. The objective is to transform each sentence into an embedding with our model, then calculate the cosine similarity[14] between the two embeddings. Finally, we compare this calculated score with the reference score: the closer they are, the better the model.

[13] Accuracy is a metric that measures the percentage of correct predictions made by the model out of all predictions.

[14] Cosine similarity: metric measuring angle between vectors to determine semantic similarity.

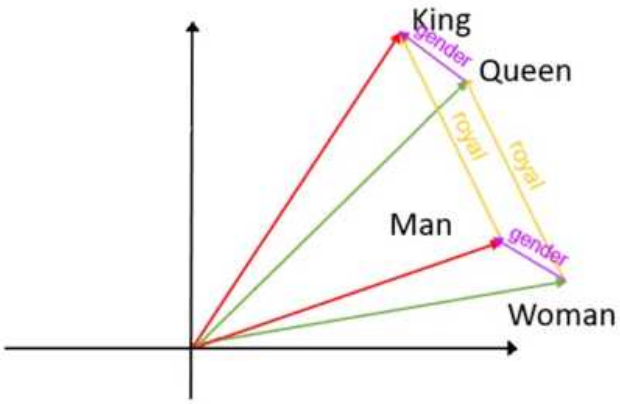


Figure 7: Word Embedding Visualization

2.5 GPT2 Implementation

For my end-of-internship project, I decided to reimplement GPT-2[15] in Hasktorch. Aware that this project was very ambitious, I decided to start it a month and a half before the end of the internship, instead of the two weeks initially planned.

The idea for this project came to me after seeing a very instructive video on YouTube by Andrej Karpathy [6], a former OpenAI employee who now makes very advanced and concrete tutorials on modern artificial intelligence. In this video, he showed step by step how to reimplement GPT-2, but in PyTorch directly based on OpenAI's research paper. So I thought that what is feasible in PyTorch must also be feasible in Hasktorch.

2.5.1 Attention Mechanism

The attention mechanism[16] is what all modern artificial intelligence relies on. It was discovered by Google in 2017 and revolutionized the world of artificial intelligence [7].

Concretely, for each word (or token) in input, we calculate an importance score for each other word in the sequence. These scores determine to what extent each word should influence the representation of the word being processed.

This process allows us to capture long-range dependencies and contextual relationships between words.

[15] GPT-2: OpenAI's large-scale transformer model for text generation.
[16] Attention: technique allowing models to focus on different input parts when producing output.

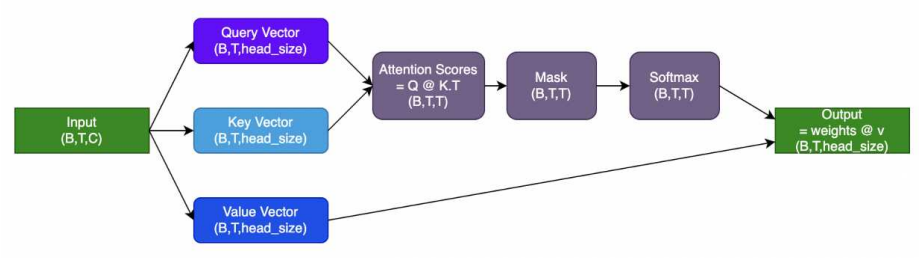


Figure 8: Self Attention

2.5.2 Forward and Backward Propagation

This step constitutes the first phase of the project. It consists of recoding all GPT2 modules in Haskell/Hasktorch and assembling them to obtain a prediction from an input token sequence.

The following modules have been implemented:

EmbeddingLayer Converts token indices into dense vectors. Used to encode input tokens and their positions in the model to extract meaning.

MultiLayerPerceptron The MLP module implements a feed-forward neural network that processes representations after attention. It uses GELU activation[17] after each layer. This structure allows the model to introduce non-linearities and enrich contextual representations.

CausalSelfAttention The CausalSelfAttention module implements the causal mask attention mechanism. It ensures that each sequence position can only access previous tokens. The implementation uses multi-head attention[18] to capture different relationships between tokens, with linear projections to transform inputs into queries, keys and values.

LayerNorm This module implements layer normalization[19], an essential technique for stabilizing learning in deep architectures. This normalization is applied before each main sub-layer of the transformer.

[17] GELU: activation function combining dropout and ReLU properties for better transformer performance.

[18] Multi-head attention: parallel attention operations capturing different representation subspaces.

[19] Layer normalization: technique normalizing inputs across features to stabilize deep network learning.

Block The Block module encapsulates a complete transformer block[20], combining all essential components in the GPT architecture. Each block sequentially applies: normalization, attention, another normalization, then MLP.

GPT The GPT module assembles all components to form the complete model. It manages the initialization and connection of embedding layers (tokens and positions), stacked transformer blocks, and the output layer. It also provides the inference and loss calculation functions necessary for training and evaluating the language model.

Regarding backward propagation, I did not need to implement it explicitly as it is automatically handled by Hasktorch. To make my modules compatible with it, I simply had to derive them with the ”Parameterized” class.

2.5.3 Data Management

Data management is a major challenge for training an LLM[21], as the amount of data to process is very large. It is not possible to load all data into memory and feed it to the model bit by bit.

Dynamic data loading is necessary. For this, I implemented a module called `LazyDataloader`, which loads text files piece by piece in arbitrary-sized blocks to avoid memory overload.

The data processing follows these steps:

1. **Block loading**: Reading X-byte blocks from the source file.
2. **Data preprocessing**: Removing unnecessary characters to clean text.
3. **Tokenization**[22]: Transforming preprocessed data into tokens using Evan's GPT2 tokenizer implementation in Haskell [9].
4. **Batch creation**: Generating training batches from tokens, placing excess tokens in a buffer.

The buffer optimizes performance by reducing time-consuming input/output operations through sufficiently large block sizes that minimize disk accesses.

2.5.4 Training

LLM training requires significant time and computational resources. Several technical challenges were addressed to optimize this process.

[20] Transformer block: fundamental unit with multi-head attention, feed-forward network, and normalization layers.
[21] LLM: Large Language Model trained on vast text data for human-like text generation.
[22] Tokenization: breaking text into smaller units (tokens) for language model processing.

CUDA Acceleration The first challenge involved modifying code to use CUDA[23] acceleration [8]. CUDA exploitation of graphics resources for matrix calculations drastically reduces computation times compared to CPU execution.

Training Monitoring Training loss alone insufficiently indicates model training state. Additional complementary metrics are necessary:

- Training data accuracy
- Validation set loss

Gradient Accumulation Gradient accumulation[24] addresses insufficient computing power for optimal batch sizes. This technique sums gradients from multiple mini-batches before optimization steps, simulating larger batch effects while respecting memory constraints.

Saving and Resuming Training Given training duration, backup systems enabling exact resumption are essential. This involves saving model weights, optimizer state, and current epoch number.

2.5.5 Results

Unfortunately, the implementation only works on small models. When scaling up, computing power proved insufficient for larger model loads. GPU memory limitations prevented training models exceeding a few million parameters, whereas GPT-2 typically contains hundreds of millions. Despite these constraints, the implementation successfully demonstrated transformer architecture feasibility within functional programming paradigms.

[23] CUDA: NVIDIA's parallel computing platform enabling GPU acceleration

[24] Gradient accumulation: summing gradients over multiple mini-batches to simulate larger batch sizes within memory constraints.

3 Ecological Impact

As part of this 2.5-month internship in Japan, an analysis of environmental and societal impacts was carried out according to a Sustainable Development and Social Responsibility approach. This study reveals a total carbon footprint of 2.94 tons CO2eq[25], significantly exceeding the French target of 2 tons per person per year set for 2050.

The environmental impact is largely dominated by international flights, representing 2.9 tons CO2eq or 98.6% of the total. The 12,000 kilometers traveled by plane constitute the major footprint of this experience.

The Japanese host structure presents contrasting environmental practices. Among the positive aspects, we note a partial telework policy allowing reduction of travel, rigorous selective sorting in accordance with Japanese standards, and the use of recent and energy-efficient computer equipment. However, certain areas for improvement have been identified, notably the unnecessary maintenance of artificial lighting during the day and sometimes excessive air conditioning. The artificial intelligence sector in which this internship is embedded also requires significant computational resources, generating a significant carbon footprint.

At the societal level, this experience has made it possible to develop valuable intercultural and linguistic skills. Immersion in a Japanese work environment revealed different managerial practices, notably a marked respect for work-life balance and regular working hours.

This analysis highlights the paradox between the undeniable professional and cultural enrichment of the international experience and its high environmental cost. For the future, several avenues for improvement can be considered: favoring longer-term internships to amortize the impact of transport, developing digital alternatives when possible, or considering compensation for emissions through carbon sequestration projects.

[25] CO2eq: metric comparing greenhouse gas emissions by global warming potential.

4 Assessment

This 2.5-month internship at Bekki Lab represented an exceptional formative experience, marked by the exploration of Hasktorch in an enriching intercultural context.

Threats The **emerging ecosystem** of Hasktorch, with its limited documentation, and the **language barrier** constitute the main difficulties. The lack of community support and tutorials made debugging particularly challenging.

Weaknesses My **initial dependence on generative AI tools**, ineffective with Haskell, and my **limited theoretical understanding** of ML mathematics require further study. **Japanese language skills** remain to be developed for better integration.

Opportunities The **pioneering positioning** on Hasktorch opens unique perspectives. The **international network** established and the **emergence of functional technologies** in ML offer opportunities for innovation and specialization in a niche but growing field.

Strengths This internship consolidated my **adaptation capacity** and **technical resilience**. Mastery of an **alternative paradigm** enriches my profile. The **in-depth understanding of algorithms** developed by implementing them from scratch constitutes a competitive advantage. This experience confirms my orientation towards AI research, favoring innovative and responsible approaches that challenge conventional methodologies.

References

[1] About Ochanomizu University. Ochanomizu University. 2025. https://www.ocha.ac.jp/en/introduction/index.html (visited on 04/07/2025).

[2] Bekki Lab. Computational Linguistics and Logic. Ochanomizu University. 2025. https://www.is.ocha.ac.jp/~bekki/.

[3] Hasktorch Team. Hasktorch: Tensors and neural networks in Haskell. GitHub Repository. 2025. https://github.com/hasktorch/hasktorch.

[4] Kaggle. Titanic - Machine Learning from Disaster. Competition Dataset. 2012. https://www.kaggle.com/competitions/titanic.

[5] D. Cer et al. "SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation". In: Proceedings of the 11th International Workshop on Semantic Evaluation (2017), pp. 1–14.

[6] Andrej Karpathy. Let's build GPT: from scratch, in code, spelled out. YouTube. Jan. 2023. https://www.youtube.com/watch?v=kCc8FmEb1nY.

[7] A. Vaswani et al. "Attention Is All You Need". In: Advances in Neural Information Processing Systems 30 (2017), pp. 5998–6008. https://arxiv.org/abs/1706.03762.

[8] NVIDIA Corporation. CUDA Toolkit Documentation. 2025. https://docs.nvidia.com/cuda/.

[9] GPT-2 Tokenizer Implementation. Hugging Face Transformers. 2025. https://huggingface.co/docs/transformers/model_doc/gpt2.

THEO CASTILLO

Computer Science Student

(+33) 6 16 18 38 92
theocastillo@yahoo.com
Bordeaux, France
tcastillo.me
theosorus
theocastillo

I am currently looking for a work-study position starting in September 2025 to apply and further develop my skills in software development.

PROJECTS (E-PORTFOLIO)

Language Model Personal

Creation and training of a French language model using PyTorch

GPT2 Haskell Internship

Implementation of the GPT-2 model in Haskell using the Hasktorch library in functional programming

Neural Network Library Personal

Development of a neural network library from scratch in Python

Graph Visualizer Studies

Development of a graph visualizer in Java using the JavaFX library

Particle Simulator Personal

Implementation of a particle system simulation in C++ using the SFML library and Verlet integration

TradeFormer Personal

Development of a Transformer-based model to predict financial data

License Plate Detection and Recognition Personal

Development of a Python pipeline with two YOLO models in sequence: the first detects the license plate in the image, the second extracts the characters from the detected area

COMMUNITY ACTIVITIES

Participation in machine learning competitions on the Kaggle platform

Publication of a Formula 1 dataset on Kaggle

HOBBIES

Chess, Formula 1, Card Games, Running, Programming

LANGUAGES

English: B2
Spanish: A2

SKILLS

Programming Languages: Java, Javascript, HTML, CSS, C, C++, C#, Haskell

Artificial Intelligence : PyTorch, scikit-learn, Matplotlib, TensorFlow, Numpy, Pandas

Web: React, Symfony, Django, Bootstrap

Software: NetBeans, WebStorm, PyCharm, Visual Studio Code, Visual Studio, IntelliJ

Databases: SQL, MySQL, pgAdmin, SQL Server Management Studio, WinDesign

Versioning: Github, Git, GitLab

EXPERIENCE

Ochanomizu University - Tokyo 2025
10-week internship at Ochanomizu University in Tokyo, in a research laboratory focused on machine learning and natural language processing.

Grands Chais de France 2024
Seasonal job for one month, where I was responsible for order palletization.

Aquilux 2019
One-week observation internship at Aquilux, a company in the electrical field.

EDUCATION

IUT of Gradignan
2024-present
BUT in Computer Science
Currently in my second year of a BUT in Computer Science

Gustave-Eiffel
2020 - 2023
Technological Baccalaureate with High Honors

Sainte Marie Bastide
2015 - 2019
Middle School Certificate with High Honors