End-to-end NLP Pipelines in Rust

Guillaume Becquin
guillaume.becquin@gmail.com

Abstract

The recent progress in natural language processing research has been supported by the development of a rich open source ecosystem in Python. Libraries allowing NLP practitioners but also non-specialists to leverage state-of-the-art models have been instrumental in the democratization of this technology. The maturity of the open-source NLP ecosystem however varies between languages. This work proposes a new open-source library aimed at bringing state-of-the-art NLP to Rust. Rust is a systems programming language for which the foundations required to build machine learning applications are available but still lacks ready-to-use, end-to-end NLP libraries. The proposed library, rust-bert, implements modern language models and ready-to-use pipelines (for example translation or summarization). This allows further development by the Rust community from both NLP experts and non-specialists. It is hoped that this library will accelerate the development of the NLP ecosystem in Rust. The library is under active development and available at https://github.com/guillaume-be/rust-bert.

1 Introduction

Natural language processing (NLP) has undergone a rapid transformation over the last few years. Modern architectures based on the Transformers (Vaswani et al., 2017), leveraging efficiently the large amount of data available for unsupervised pre-training, have enabled significant progress for a variety of tasks including sentiment analysis, question answering, summarization or translation. These research efforts have been accompanied by the development of a rich Python ecosystem enabling a democratization of these technologies for both practitioners and users, from tokenization to deep learning architectures. The Transformers library (Wolf et al., 2019) is an example of a library proposing APIs at various levels to either promote further development of NLP or their integration in higher level applications.

The adoption of these technologies in other programming languages has unfortunately not been as fast, for example in Rust. Rust (Klabnik and Nichols, 2018) is a promising modern static, strongly typed language that offers execution speeds similar to C. Its built-in memory safety design makes it an attractive alternative to C++ for the development of productive machine learning systems. Rust does not include a garbage collector but instead relies on strict ownership rules for the variables, dropping them when going out of scope. Its modern implementation of the strings data model that complies with UTF-8 standards is especially relevant to NLP applications. Finally, Rust includes a powerful utility called *cargo* to manage external dependencies. This allows the development of open-source ecosystems, similar to Python's *PyPI (Python Packaging Authority, 2000)* or Java's *Maven (Miller et al., 2010)*.

Rust is a modern programming language for which the foundations of a machine learning ecosystem are still being built. A number of initiatives including array manipulation (rust-ndarray Team, 2011), low-level CUDA libraries and deep learning framework bindings for Tensorflow (Tensorflow Project, 2016) or Torch (Mazare, 2019) are now maturing. However, there is still a lack of end-to-end, ready to use libraries leveraging state-of-the-art NLP models. The proposed library aims at filling this gap and exposes both Transformers-based architectures to NLP practitioners in Rust and pipelines that are ready for integration in Rust-based back-ends. The proposed library, *rust-bert*, is available at https://github.com/guillaume-be/rust-bert or https://crates.io/crates/rust-bert and is shared under Apache 2.0 license.

Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS), pages 20–25

2 Related Work

This work leverages the rich open-source resources available in Python. Especially relevant is the Transformers library (Wolf et al., 2019), of which large sections of the proposed Rust library were ported from. The model architectures and layers naming have been aligned with the Transformers implementation, and Rust-compatible pre-trained weights are available in Hugging Face's Model Hub (Hugging Face, 2019). The general API for the high-level and ready-to-use pipelines has been strongly inspired by the SpaCy library (Honnibal and Montani, 2017).

A number of specialized libraries have been developed for Rust, including high performance tokenizers ((Hugging Face, 2020) or (Becquin, 2020)) and language detection libraries (Potapov, 2016).

3 Architecture Design

The library exposes three main features:

- Language models implementation, covering state-of-the-art architectures including for example BERT (Devlin et al., 2019) or GPT2 (Radford et al., 2019).
- Ready-to-use pipelines, combining these models with pre-and post-processing routines.
- Utilities to load external resources, including a converter from PyTorch (Paszke et al., 2019) pickled model files to a C-array format.

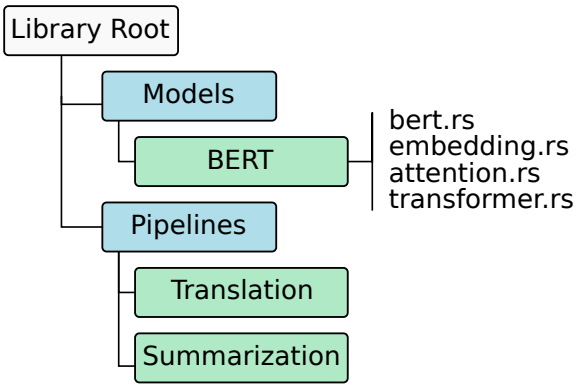

Figure 1: High-level library structure.

The language models and the pipelines are separated in different modules. Within the models, a sub-module is defined for each model (for example, BERT) with individual files for the major model components (for example, its attention mechanism). This promotes readability and modularity of the code base (Figure 1).

An important design aspect of the library is related to the choice of abstractions. Rust does not implement the concept of classes and inheritance in a similar way to Python. Rather, data is arranged in *structs* that may implement associated methods in an *impl* block or shared behaviour via *traits*. As opposed to Python, layers do not inherit from a shared *nn.Module* because Rust requires a strict definition of the names and types of the inputs and outputs (those may differ significantly from model to model). As a consequence the registration of the model parameters in the variable store is done manually:

```
pub struct ModelA {
  dense: nn::Linear
}
impl ModelA {
  pub fn new(p: &VarStore) -> Self {
    // manual varstore registration
  }
  pub fn forward(&self,
                 arg1: Tensor,
                 arg2: bool)
    -> OutputType {
    // forward pass
  }
}
```

While the model architectures have been generally ported from the Python Transformers' library, the proposed work is innovative in its handling of shared behavior. Models and configurations share capabilities using *Traits*. This includes for example the possibility for a model to be used as a conditional text generator by implementing the *LanguageGenerator* trait. A given model implements the trait by providing model-specific methods (e.g. *prepare_inputs* or *reorder_cache*). The complex text generation post-processing steps (beam search, sampling, non-repetition rules...) and the generation routine can then be readily leveraged by this model.

```
trait PrivateLanguageGenerator {
  fn prepare_inputs() {}
  fn reorder_cache() {}
  fn top_k_top_p_filtering() {...}
  fn generate_beam_search() {...}
}
pub trait LanguageGenerator:
  PrivateLanguageGenerator
 {
  fn generate() {...}
}

impl PrivateLanguageGenerator
  for ModelA {
    fn prepare_inputs() {...}
    fn reorder_cache() {...}
  }
impl LanguageGenerator for ModelA {}
```

Shared behavior is also required for the ready-to-use pipelines that implement logic valid for a wide range of language models. Here the mechanism instead relies on *Enums* wrapping specific models in a shared abstraction. A given pipeline takes a Model Enum, a Tokenizer Enum and a Configuration Enum as inputs. The pipeline calls generic functions that are implemented by the enum (for example a forward pass). Each variant of the enum defines how the forward method is implemented. Note that this allows defining a common interface to models expecting a different set of inputs.

```
pub enum ClassifierModel {
  BERT(BertModel),
  XLNet(XLNetModel)
}
impl ClassifierModel{
  pub fn forward(x, y) {
    match *self {
      BERT(mdl)=>{mdl.forward(x)},
      XLNet(mdl)=>{mdl.forward(x, y)}
    }
  }
}
pub struct Classifier {
  pub model: ClassifierModel
}
impl Classifier{
  pub fn predict(text: &str) {
    self.model.forward(x, y)
  }
}
```

This pattern is similar to dependency injection (while the traits are closer to inheritance) and has benefits of a greater flexibility in the interface for model loading and forward methods and reduced coupling between the model and the pipelines.

4 Capabilities Overview

The library exposes an API at two different levels: the language models themselves, allowing to build NLP pipelines from scratch, and end-to-end pipelines that can readily be integrated in higher level applications.

A rust implementation for a wide range of language models has been implemented, including BERT (Devlin et al., 2019), DistilBERT (Sanh et al., 2019), RoBERTa (Liu et al., 2019), GPT (Radford, 2018), GPT2 (Radford et al., 2019), ALBERT (Lan et al., 2019), BART (Lewis et al., 2020), Marian (Junczys-Dowmunt et al., 2018), XLM-RoBERTa (Conneau et al., 2020) and T5 (Raffel et al., 2019). For each of these models, pre-trained weights have been converted to a C-array format and are hosted alongside the Python version on Hugging Face's model hub (Hugging Face, 2019).

A large user base of NLP technologies also benefits from the availability of state of the art, end-to-end pipelines requiring little to no familiarity with NLP to be integrated in higher level applications. To answer these needs of the Rust community, the following capabilities have been implemented:

- Translation between 8 language pairs using either Marian (Junczys-Dowmunt et al., 2018) or T5 (Raffel et al., 2019) models.
- Summarization using a BART (Lewis et al., 2020) model trained on the CNN / Daily Mail summarization dataset (See et al., 2017).
- Conversational model using DialoGPT (Zhang et al., 2020).
- Question Answering using a DistilBERT (Sanh et al., 2019) model trained on the SQuAD dataset (Rajpurkar et al., 2016).
- Sentiment Analysis using a DistilBERT model trained on the SST-2 dataset (Socher et al., 2013)
- Named Entity Recognition for English, German, Spanish and Dutch trained on CoNLL03 (Tjong Kim Sang and De Meulder, 2003) and CoNLL02 (Tjong Kim Sang, 2002) datasets

These pipelines can be created and used in a few lines of code without prior knowledge in NLP.

```
let model = TranslationModel::
  new(translation_config)?;
let input = ["Hello, world!"];
model.translate(&input);
```

While the implementation of the language models is a prerequisite, the availability of powerful end-to-end pipelines is key to a broader adoption of NLP technology in Rust. These pipelines can easily be integrated with server back-ends running Rust with queuing and batching of incoming requests (Walsh, 2020).

5 Benchmarks

This library was developed with the primary goal of making state of the art NLP capabilities available to the Rust community rather than speeding up inference. Nevertheless, Rust is a high performance language with execution speeds matching C or C++. Efficient predictions using NLP systems has become a key subject of research and engineering development over the past few months. Several methods have been investigated to improve the

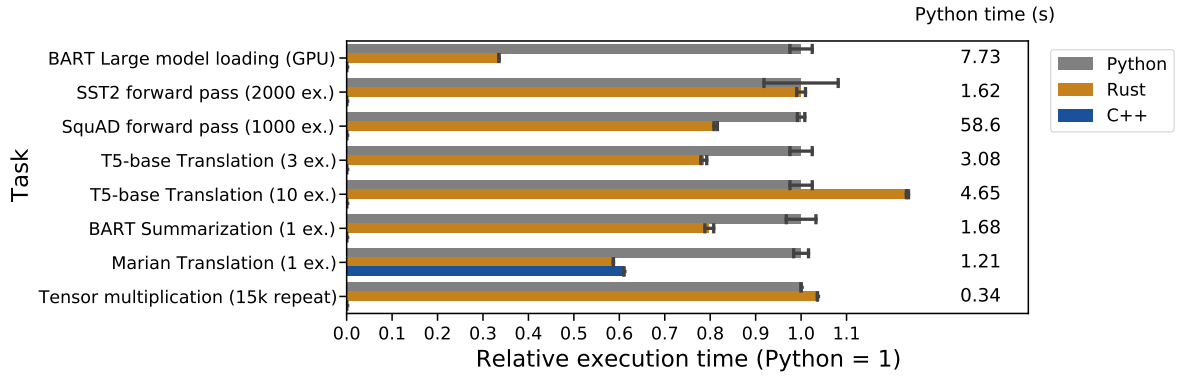

Figure 2: Rust-Python benchmark results.

model predictions performance, including for example pruning, quantization and Huffman Coding ((Han et al., 2016), (Shen et al., 2020)), distillation (Sanh et al., 2019), graph optimizations and layer fusing (Nvidia, 2020) or optimized runtimes such as ONNX (Bai et al., 2019). The high performance of the state of the art models usually comes with a significant computational cost.

It should be noted that the proposed library is based on bindings (Mazare, 2019) to LibTorch (Paszke et al., 2019), and therefore limited benefits can be expected from the tensor operations. These are executed in the CUDA layer that is effectively shared with the Python-based models. The following investigates if these high performance features of the language translate into benefits for the proposed NLP pipelines.

Benchmarks between Python and Rust are shown in Figure 2 using a Turing RTX2070 GPU with a AMD 2700X CPU. For all experiments the average time relative to Python is reported with the standard deviation. For all prediction tasks, the Transformers (Wolf et al., 2019) library (v3.2.0) is used as a reference. All experiments are run for 10 iterations, with various number of samples (provided in brackets). For reference, the Python absolute execution time per iteration is provided.

The loading benchmarks represent the average time required to load models into the GPU buffer. Significant benefits can be observed for Rust. This is probably caused by the simpler serialization format based on C-arrays for Rust, and may be advantageous for event-driven applications loading models on a per-request basis (short warm-up time).

The forward pass results vary between applications. As expected, pipelines with very simple pre- and post-processing steps offer virtually identical performance (for example sentiment analysis). Significant benefits can be observed for question answering, coming entirely from the tokenization process (At the time this document was prepared, the Transformers' (Wolf et al., 2019) question answering pipeline did not leverage Rust-based tokenizers yet). The performance of pipelines involving complex post-processing steps (text generation with sampling and beam search) can show significant benefits. Marian-based translation models (Tiedemann and Thottingal, 2020) exhibit a 40% speedup (in line with the native C++ implementation (Junczys-Dowmunt et al., 2018)). The T5 implementation is faster for small effective batch sizes (with a beam size of 6) but slower for larger batches, indicating optimization potential remains. In general it was observed that the actual model forward pass (tensor operations) is comparable albeit slightly slower in Rust than in Python. A last experiment (large matrix multiplication) shows the Rust LibTorch bindings seem to be 1 to 2% slower than the PyTorch equivalent.

6 Conclusion

Rust is a promising language for the development of NLP systems. Its concurrency capabilities, memory safety features and modern strings data model make it a good alternative to C++ for production systems. While evolving quickly, the Rust NLP open-source ecosystem still lags behind Python rich set of libraries. Complementing the availability of high performance tokenizers, *rust-bert* makes state-of-the-art language models and end-to-end NLP pipelines available to the Rust community.

7 Acknowledgments

The list of contributors to the *rust-bert* project is available on the project repository.

References

Junjie Bai, Fang Lu, Ke Zhang, et al. 2019. Onnx: Open neural network exchange. https://github.com/onnx/onnx.

Guillaume Becquin. 2020. rust-tokenizers. https://github.com/guillaume-be/rust-tokenizers.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Hugging Face. 2019. Hugging face model hub.

Hugging Face. 2020. tokenizers. https://github.com/huggingface/tokenizers.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Steve Klabnik and Carol Nichols. 2018. *The Rust Programming Language*. No Starch Press, USA.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Laurent Mazare. 2019. tch-rs. https://github.com/LaurentMazare/tch-rs.

Frederic P Miller, Agnes F Vandome, and John McBrewster. 2010. *Apache Maven*. Alpha Press.

Nvidia. 2020. Optimizing the performance of tensorrt.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Sergey Potapov. 2016. whatlang-rs. https://github.com/greyblake/whatlang-rs.

Python Packaging Authority. 2000. Python package index (pypi).

Alec Radford. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

rust-ndarray Team. 2011. rust-ndarray. https://github.com/rust-ndarray/ndarray.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *AAAI*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Tensorflow Project. 2016. Tensorflow rust. https://github.com/tensorflow/rust.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.

Erik F. Tjong Kim Sang. 2002. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Evan Pete Walsh. 2020. rust-dl-server. https://github.com/epwalsh/rust-dl-webserver.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. 2020. DIALOGPT : Large-scale generative pre-training for conversational response generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 270–278, Online. Association for Computational Linguistics.