Scaling Up Models and Data with t5x and seqio

Abstract

Scaling up training datasets and model parameters have benefited neural network-based language models, but also present challenges like distributed compute, input data bottlenecks and reproducibility of results. We introduce two simple and scalable software libraries that simplify these issues: t5x enables training large language models at scale, while seqio enables reproducible input and evaluation pipelines. These open-source libraries have been used to train models with hundreds of billions of parameters on multi-terabyte datasets. Configurations and instructions for T5-like and GPT-like models are also provided. The libraries can be found at https://github.com/google-research/t5x and https://github.com/google/seqio.

Keywords: Large language models, data parallelism, model parallelism, data processing
1. Introduction

Scaling transformers (Vaswani et al., 2017) to hundreds of billions of parameters has shown significant improvement, but training at such scale and consistently finetuning and prompting these models for downstream usage and evaluation requires a research-friendly and scalable software framework. In this paper, we introduce t5x, an open-source library to build Transformer models at scale by leveraging Jax’s(Bradbury et al., 2018; Frostig et al., 2018) user-friendly NumPy-like (Harris et al., 2020) user interface and its powerful jax.pjit API for parallelism backed by XLA GSPMD(Xu et al., 2021).

Additionally, training at scale requires large datasets. We also introduce seqio, an open-source library for managing data pipelines and model evaluations. seqio builds on tensorflow.data, adds support for SPMD-based data parallelism and is compatible with popular modeling frameworks including JAX, TensorFlow (Abadi et al., 2015), and PyTorch(Paszke et al., 2019).

2. t5x

Figure 1: Overall structure of t5x, showing components used for principal functionalities.

Modular Design. Figure 1 illustrates the overall modular structure of t5x, highlighting the use of open-source libraries to implement different functionalities. For datasets and evaluation - t5x uses seqio to create reproducible “tasks”, which we cover in detail in Section 3. For checkpointing, we built our own library utilizing TensorStore\(^1\) as a tool for scalably reading and writing sliced tensors. This enables efficient management of checkpoints when parameters are distributed across multiple host processes. For configuration, we use Gin\(^2\) for dependency injection, allowing users to inject hyperparameters, model objects, and other components (for example, custom checkpointer) without modifying the core library. This makes t5x easily configurable, supporting fast iteration over research ideas. For model implementation, t5x leverages specialized features in the Flax (Heek et al., 2020) library, built on JAX, which are described further below. For Partitioning, we use the XLA GSPMD partitioner (Xu et al., 2021) to automatically shard the computation graph and use jax.pjit as a frontend to interact with GSPMD, providing our own simplified API to allow users to parallelize over data, parameters, and activations, described further below. The modular structure allows users to replace these components with alternative standard and custom components.

---

1. https://github.com/google/tensorstore
2. https://github.com/google/gin-config
XLA GSPMD partitioning with `jax.pjit`. `t5x` supports both data parallelism and model parallelism to scale large models by defining orthogonal axes of the physical device mesh: `model` and `data`. Data parallelism involves splitting input data and intermediate activations over along the global batch axis, either by replicating parameters and optimizer state ("1D parameter partitioning") or sharding them over `data` ("2D parameter partitioning"). Model parallelism involves partitioning parameters and intermediate activations along axes other than the batch dimension. Replicating intermediate activations over `model` is referred to as "1D activation partitioning", while sharding them is "2D activation partitioning".

These options correspond to previously described parallelism techniques: 2D parameter partitioning is also known as ZeRO-3 (Rajbhandari et al., 2020) or fully sharded data parallelism; 1D activation partitioning is also known as Megatron (Shoeybi et al., 2019) and is the default in the Mesh TensorFlow Transformer (Shazeer et al., 2018); and 2D activation partitioning is the “fully sharded” case described in Xu et al. (2021). `t5x` supports flexible partitioning configurations, including these built-in options, using the Flax APIs described in the following section.

**Model Implementation.** `t5x` is compatible with Flax-based model implementations with minor caveats. User-defined logical axis annotations via `flax.partitioning.param_with_axes` are required for parameter and activation partitioning. These logical axes group tensor dimensions that must be partitioned in the same way, for example, “batch” (across examples in a batch), “kv” (across dimensions of key-value matrices in attention layers), and “head” (across heads in multi-headed attention). While XLA GSPMD automatically selects matching partitions for intermediate activations, users can override with `flax.partitioning.with_sharding_constraint` for better memory usage and inter-device communication. At runtime, users provide a map of logical axes to hardware axes (`model` or `data`). Alternatively, logical axes can be mapped to `None` to indicate replication across all devices.

A `flax.nn.module` implemented with these annotations is wrapped in a `t5x.BaseModel` subclass defining the loss, evaluation, and inference methods to make it compatible with the core `t5x` interface. `t5x` model support is flexible—layers and modules can be written directly with Flax or using higher-level libraries like Flaxformer\(^3\). Dependency injection with Gin enables easy swapping of models. Checkpoints from other libraries can be made compatible, including legacy T5 checkpoints\(^4\) based on Mesh TensorFlow, which can be read directly by `t5x` or converted to the native `t5x` format for faster reading.

**Example Models.** We provide well-tested (validated by reproducing the T5 models from Raffel et al. (2020) originally implemented in Mesh TensorFlow) “Minimal” model implementations along with checkpoints for T5 (Raffel et al., 2020) and T5.1.1 (introduced after the paper), mT5 (Xue et al., 2021), ByT5 (Xue et al., 2022), a model configuration (without checkpoints) for a decoder-only architecture compatible with LaMDA (Thoppilan et al., 2022), and Scalable T5 - an implementation of T5.1.1 using `jax.scan` to significantly reduce compilation time and provide finer-grained control over activation memory. These use Flax with limited abstractions, closely following pedagogical Flax examples\(^5\).

---

3. https://github.com/google/flaxformer
4. https://github.com/google-research/text-to-text-transfer-transformer
**GPU Support.** We provide examples and instructions\(^6\) to run t5x on GPUs in single-node and multi-node configurations, with optimizations for better throughput. More examples can be found in the NVIDIA Rosetta repository\(^7\) which includes H100 FP8 support and performance improvements.

3. seqio

seqio is a data processing library for training, inference, and evaluation. It uses tensorflow.data for scalable pipelines, compatible with frameworks like JAX or PyTorch by easily transforming datasets to NumPy iterators. A key differentiator is the Task-based API illustrated in Figure 2, which associates data sources with preprocessing and evaluation. Feature converters transform task features into values passed to the model, making Tasks reusable across architectural variants such as encoder-decoder or decoder-only. Multiple Tasks can also be combined into a Mixture for multi-task training.

![Figure 2: Structure of a seqio Task, highlighting customizable use of APIs.](image)

4. Related Work

Previous Google-released libraries for training sequence models based on TensorFlow include Tensor2Tensor (Vaswani et al., 2018), Lingvo (Shen et al., 2019), and the Mesh TensorFlow (Shazeer et al., 2018)-based T5 (Raffel et al., 2020). Comparable projects from other research groups include model libraries like fairseq (Ott et al., 2019), large-scale parallelism libraries like FairScale (Baines et al., 2021), and libraries that include both, like DeepSpeed (Rasley et al., 2020) and Megatron (Smith et al., 2022).

Major differentiators of t5x are its use of JAX and Flax for model expression, its support for TPU (including TPU v4), and its Gin-based configuration system that allows users to modify any aspect of the model and training procedure. t5x’s native support for multi-host model parallelism allows reliably training models at massive scale. t5x doesn’t support pipeline parallelism, a major component of systems like DeepSpeed, because the inter-chip network of TPUs has performance similar to within-node GPU interconnects but scales to thousands of chips, making model and data parallelism sufficient to train efficiently at scale.

\(^6\) https://github.com/google-research/t5x/blob/main/t5x/contrib/gpu/

\(^7\) https://github.com/NVIDIA/JAX-Toolbox/tree/main/rosetta/rosetta/projects/t5x
5. Project Status and Adoption

We started the project in the fall of 2020 and open sourced the library code in October 2021. During that time, t5x and seqio achieved widespread adoption by teams across Google: t5x has been launched on TPU hundreds of thousands of times at Google, and the total number of internal t5x and seqio users exceeds 1,000. Teams are using these libraries for research projects (from small-scale research to the largest language models trained at Google) and user-facing products. External adopters include academic and commercial users of Cloud TPUs, such as portions of the the Big Science project (Wang et al., 2022).

Users of t5x and seqio cite the usability and research-friendliness of the libraries as reasons for adoption. We are continuing to actively develop both libraries, prioritizing future work based on researcher needs and feedback.

References


Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, Ruoming Pang, Noam Shazeer, Shibo Wang, Tao Wang, Yonghui Wu, and Zhifeng Chen. GSPMD:
