Papers:

1. https://arxiv.org/pdf/2506.15309
2. https://www.manning.com/books/deep-learning-with-jax
3. https://arxiv.org/pdf/2506.09644
4. https://github.com/n2cholas/awesome-jax

```
i
```

```
   if cache_dir is None:
       return generate_fn()

   cache_path = os.path.join(cache_dir, file_name)
   if os.path.exists(cache_path):
       ds = Dataset.load_from_disk(cache_path)
       return ds
   else:
       ds = generate_fn()
       os.makedirs(cache_dir, exist_ok=True)
       try:
```

```
           ds.save_to_disk(cache_path)
       except Exception as e:
           shutil.rmtree(cache_path)
           raise e
       return ds


def tokenize_dataset(
   ds: Dataset,
   tokenizer: PreTrainedTokenizer,
   max_seq_len: int = 512,
   sequence_packing: bool = False,
   batch_size: int = 1024,
   num_proc: int = 32,
):
   n_proc = min(os.cpu_count(), num_proc)
   bos_token_id = tokenizer.bos_token_id or tokenizer.cls_token_id
   eos_token_id = tokenizer.eos_token_id or tokenizer.sep_token_id

   tokenizer_max_len = tokenizer.model_max_length
```

```
    tokenizer.model_max_length = 10_000_000

    def tokenize_fn(examples):
        tokens = tokenizer(
            examples["text"],
            truncation=False,
            padding=False,
        )["input_ids"]
        tokens = [[bos_token_id] + x + ([] if sequence_packing else [eos_token_id]) for x in tokens]
        if sequence_packing:
            tokens = np.concatenate(tokens, axis=0)
            tokens = tokens[: len(tokens) - len(tokens) % max_seq_len]
            tokens = tokens.reshape(-1, max_seq_len)
        else:
            tokens = [
                np.pad(x, (0, max_seq_len - len(x) % max_seq_len), mode="constant", constant_values=tokenizer.pad_token_id)
                for x in tokens
            ]
            tokens = [x.reshape(-1, max_seq_len) for x in tokens]
            tokens = np.concatenate(tokens, axis=0)
        return {"input_ids": tokens}


    ds = ds.map(
        tokenize_fn,
        batched=True,
        batch_size=batch_size,
        remove_columns=["text"],
        num_proc=n_proc,
    )

    tokenizer.model_max_length = tokenizer_max_len
    return ds


def default_collator(config, tokenizer, examples, text_key="text"):
    examples = [x[text_key] for x in examples]
    return tokenizer(examples, padding="max_length", truncation=True, max_length=config.model.max_seq_len, return_tensors="pt")
```

```
def pretokenized_collator(examples, pad_token_id=0, tokens_key="input_ids"):
    input_ids = np.stack([np.array(x[tokens_key]) for x in examples], axis=0)
    attn_masks = (input_ids != pad_token_id).astype(np.int32)
    input_ids = torch.from_numpy(input_ids).to(torch.long)
    attn_masks = torch.from_numpy(attn_masks).to(torch.long)
    return BatchEncoding({"input_ids": input_ids, "attention_mask": attn_masks},
tensor_type="pt", n_sequences=len(input_ids))


def subsample_collator(config, tokenizer, examples, text_key="text"):
    bos_token_id = tokenizer.bos_token_id or tokenizer.cls_token_id
    eos_token_id = tokenizer.eos_token_id or tokenizer.sep_token_id

    examples = [x[text_key] for x in examples]
    tokens = tokenizer(examples, truncation=False, return_tensors="np")
    max_length = config.model.max_seq_len
    input_ids = []
    attn_masks = []
    for i in range(len(examples)):
```

```
 toks = tokens["input_ids"][i]
        attn_mask = tokens["attention_mask"][i]
        if toks[0] != bos_token_id:
            toks = np.concatenate([[bos_token_id], toks])
            attn_mask = np.concatenate([[1], attn_mask])
        if toks[-1] != eos_token_id:
            toks = np.concatenate([toks, [eos_token_id]])
            attn_mask = np.concatenate([attn_mask, [1]])

        if len(toks) > max_length:
            overflow = len(toks) - max_length
            start_idx = np.random.randint(0, overflow +
config.data.max_add_padding)
            toks = toks[start_idx : start_idx + max_length]
            attn_mask = attn_mask[start_idx : start_idx + max_length]
        if len(toks) < max_length:
            underflow = max_length - len(toks)
            toks = np.pad(toks, (0, underflow), mode="constant",
constant_values=tokenizer.pad_token_id)
            attn_mask = np.pad(attn_mask, (0, underflow), mode="constant",
constant_values=0)
        assert len(toks) == max_length
        assert len(attn_mask) == max_length
        input_ids.append(toks)
        attn_masks.append(attn_mask)
    input_ids = torch.from_numpy(np.array(input_ids)).to(torch.long)
    attn_masks = torch.from_numpy(np.array(attn_masks)).to(torch.long)
    return BatchEncoding({"input_ids": input_ids, "attention_mask": attn_masks},
tensor_type="pt", n_sequences=len(input_ids))


def _get_dataloader(config, ds, shuffle, drop_last, batch_size,
collate_fn=None):
    if torch.distributed.is_available() and torch.distributed.is_initialized():
        sampler = DistributedSampler(ds, seed=config.training.seed,
shuffle=shuffle)
        _shuffle = False
    else:
        sampler = None
         shuffle = shuffle

    return DataLoader(
        ds,
        collate_fn=collate_fn,
        batch_size=batch_size,
        drop_last=drop_last,
        sampler=sampler,

num_workers=config.data.num_workers,
        shuffle=_shuffle,
        pin_memory=True,
        persistent_workers=True,
    )


def get_dataloaders(config, tokenizer, train_batch_size=18,
eval_batch_size=18):

    train_ds = load_dataset('DNA-LLM/experiment_one_viral_genomes_train_set_v2')
    train_ds.set_format('torch')
    train_ds = train_ds.remove_columns(['seq_len', 'id', 'name', 'host',
'species', 'organism_name', 'location_in_host', 'family', 'genus',
'molecule_type', 'decribtion_full', 'chunked_seqs', 'relative_position',
'number_of_chunks', 'percentage_position', 'sequence_quality',
'__index_level_0__'])
    print(train_ds)
    train_ds = train_ds['train']
    test_ds = load_dataset('DNA-LLM/experiment_one_viral_genomes_val_set_v2')
    test_ds.set_format('torch')
    test_ds = test_ds.remove_columns(['seq_len', 'id', 'name', 'host',
'species', 'organism_name', 'location_in_host', 'family', 'genus',
'molecule_type', 'decribtion_full', 'chunked_seqs', 'relative_position',
'number_of_chunks', 'percentage_position', 'sequence_quality',
'__index_level_0__'])
    test_ds= test_ds['train']
    train_batch_size = config.training.train_batch_size
    eval_batch_size = config.training.eval_batch_size
    train_dl = _get_dataloader(config, train_ds, shuffle=True, drop_last=False,
batch_size=train_batch_size)
    test_dl = _get_dataloader(config, test_ds, shuffle=False, drop_last=False,
batch_size=eval_batch_size)

    return train_dl, test_dl
```