GFM in Practice: Matching Models to Research Goals

Ikram Ullah, Staff Scientist

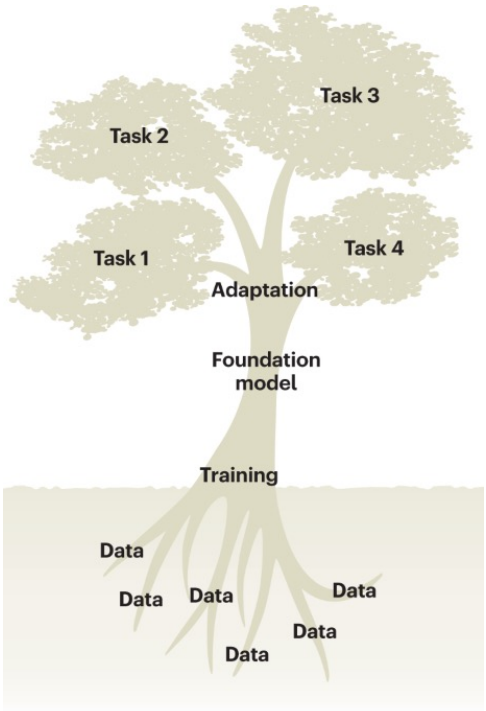


Image taken from – Tang, Lin. "Large models for genomics." *Nature Methods* 20.12 (2023): 1868-1868.

Agenda

- Framing the genomic problem
- Model selection criteria
- Data preparation & tokenization
- Fine-tuning workflow (Hugging Face)
- Evaluation & best practices
- Case studies (Human & Microbial)

UMAP visualization of the pretrained scGPT cell embeddings (emb; a random 10% subset), colored by major cell types

Cui, Haotian, et al. "scGPT: toward building a foundation model for single-cell multi-omics using generative AI." *Nature Methods* 21.8 (2024): 1470-1480.

What's Special with Using GFMs?

Pretraining Benefits

- Captures biological syntax. Transfer learning for downstream tasks

Task Versatility

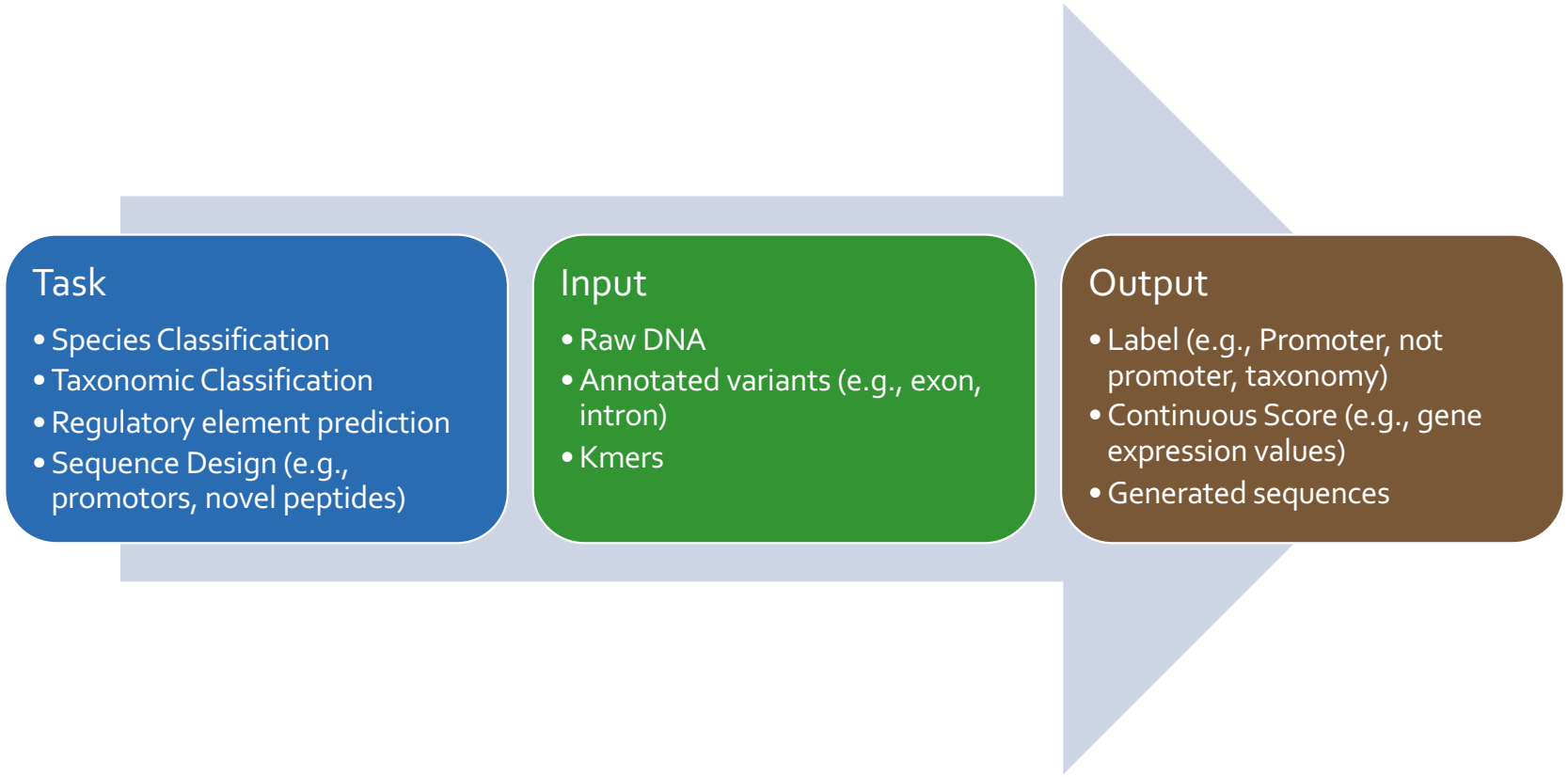
- Same pretrained model can be used for classification, regression, sequence generation

Data Efficiency

- Strong performance with limited labeled data

Key Takeaway: GFMs accelerate genomic tasks learning by leveraging large-scale pretraining

Framing Your Genomic Problem for Finetuning



Criteria for Selecting a GFM

Training Corpus

- Single Species, Multispecies

Species Domain

- Human/Mouse, Plant, Microbial, non-model organism

Model Architecture & Size

- Small (10–100 M) for quick fine-tuning vs. large (100 M– few B) for generation

Tokenization Scheme

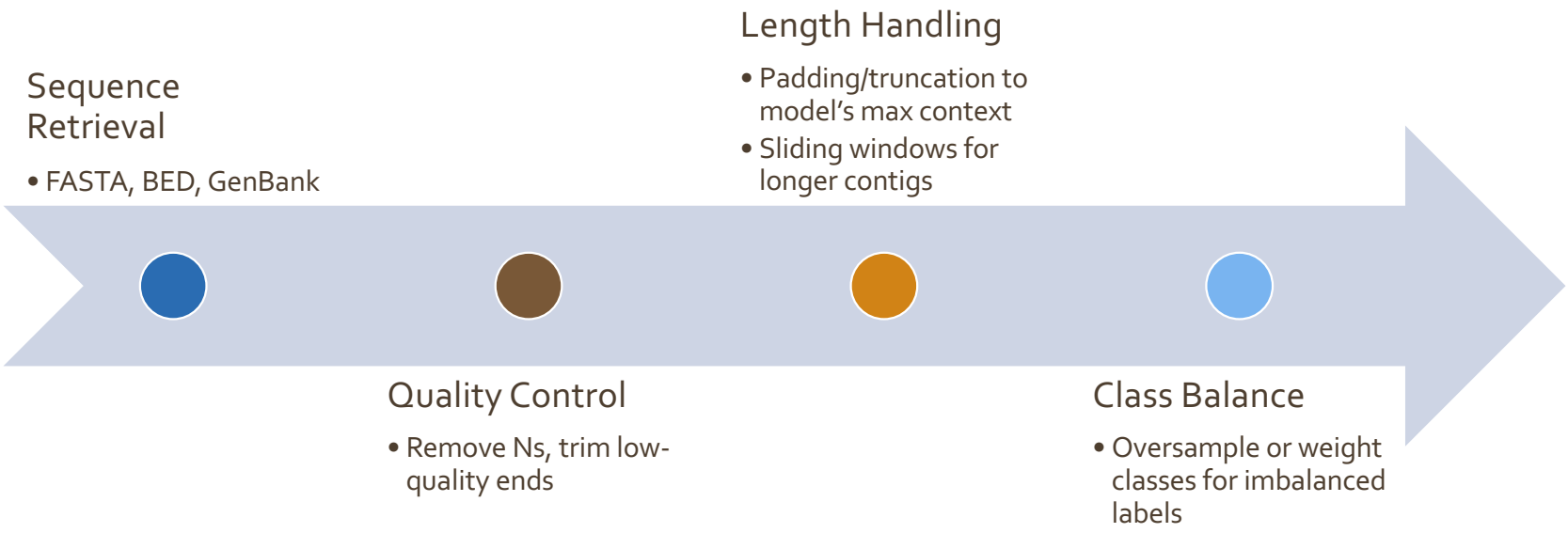
- k-mers (fixed-length) vs. BPE vs. variant-aware tokens

Accessibility, Licensing, Deployment aspects

- Hugging Face availability, MIT/Apache-2.0 licenses, Deployment complexity

Key Takeaway: Align model choice with species, compute resources, and task complexity

Data Preparation & Cleaning



Key Takeaway: Preprocessing bridges raw sequences to model inputs

Dataset Assembly

- Dataset creation
  - Convert from sequence format to Hugging Face Dataset format
  - Hugging Face Dataset with fields input_ids, attention_mask, labels
- Collation
  - Use DataCollatorWithPadding to batch variable-length inputs
- Tip
  - Precompute tokenization if training speed is critical

- **Create Dataset:**

```
import pandas as pd
from datasets import Dataset
# Example: load CSV/TSV with columns 'sequence' and 'label'
df = pd.read_csv("data/sequences.csv")
dataset = Dataset.from_pandas(df)
# Optionally split into train/validation
dataset = dataset.train_test_split(test_size=0.2)
```

- **Define Tokenization Function:**

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("DNABERT-2")
def tokenize_fn(example):
    # Tokenize the DNA sequence, truncating/padding to model max length
    return tokenizer(example["sequence"], truncation=True, padding="max_len
```

- **Apply to Dataset:**

```
tokenized_dataset = dataset.map(tokenize_fn, batched=True)
```

Fine-Tuning Workflow (Hugging Face)

- Tokenization
  - Convert DNA to tokens taking care of truncation and padding as per model max allowed length
- Load selected model with weights
  - This will be pretrained on data from your species of interest
  - Add classification/regression head
- Setup training
  - Define main training arguments like batch size, learning rate, number of epochs, stopping criteria etc
- Optimize training for large models (>100M)
  - LoRA (via Hugging Face library) allows updating only a small subset of parameters, saving memory and compute

1. **Model & Tokenizer:**

```
from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.from_pretrained(
    "Mistral-DNA-v1-17M-hg38", num_labels=2
)
```

2. **TrainingArguments:**
   - lr = 5e-5, batch_size = 16, epochs = 3
   - `evaluation_strategy="epoch"`, `save_total_limit=2`
3. **Trainer Instantiation:**

```
from transformers import Trainer
trainer = Trainer(
    model, args, train_dataset, eval_dataset, tokenizer=tokenizer
)
trainer.train()
```

Evaluation & Validation Best Practices

- Hold-Out & Cross-Validation
  - Ensure no overlap of genomic windows
- Metrics:
  - Classification: accuracy, precision/recall, ROC-AUC
  - Regression: SSE, RMSE, R^2
  - Generation: perplexity, BLAST similarity
- Biological Sanity Checks:
  - BLAST generated sequences against GenBank
  - Attention maps to highlight important motifs

Key Takeaway: Combine ML metrics with biological validation

Hands on using Human and Microbial Use Cases

Questions & Comments