Fine-Tuning Genomic Foundation Models

Ikram Ullah, Staff Scientist

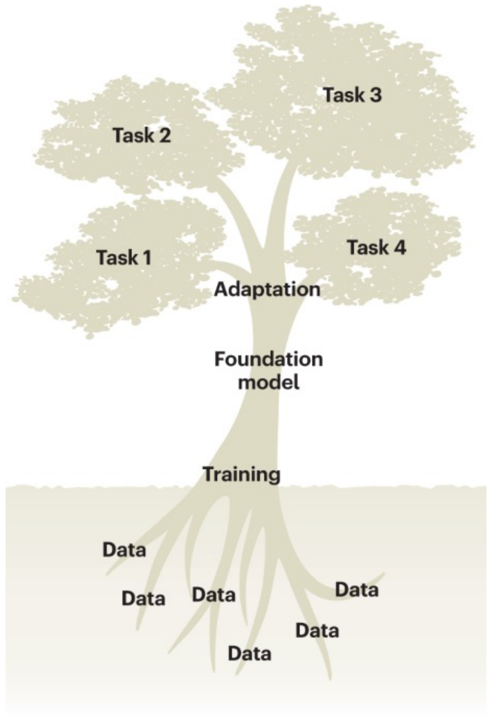


Image taken from – Tang, Lin. "Large models for genomics." *Nature Methods* 20.12 (2023): 1868-1868.

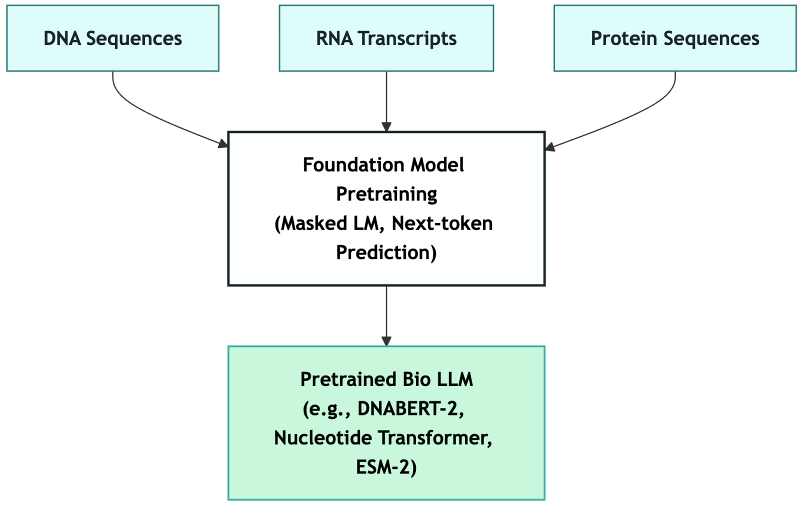
Agenda

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.

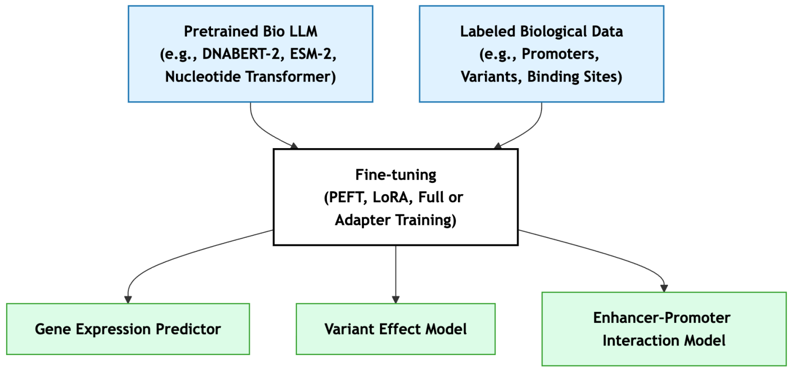
What is Pre-training?

- Building General Knowledge
  - Training on massive unlabeled genomic data to learn patterns
- Key Points:
  - Huge datasets (billions of sequences)
  - Self-supervised learning
  - General patterns learned
  - Compute-intensive (days/weeks on GPUs)
  - Foundation model as output
- Example
  - DNABERT trained on Human DNA
  - Captures motifs, regulatory syntax, dependencies between different regions


DNA Sequences
RNA Transcripts
Protein Sequences
Foundation Model Pretraining (Masked LM, Next-token Prediction)
Pretrained Bio LLM (e.g., DNABERT-2, Nucleotide Transformer, ESM-2)

What is Fine-tuning?

- Specialization for Your Task
  - Adapt a pre-trained model to specific tasks using labeled data.
- Key Points:
  - Small datasets (thousands of sequences)
  - Labeled data required
  - Adapts knowledge to your problem
  - Fast & cheap (hours on single GPU)
  - Task-specific output
- Example:
  - Use pathogenic variants to identify disease mutations
  - Understanding gene regulation
  - Synthetic element design


Pretrained Bio LLM (e.g., DNABERT-2, ESM-2, Nucleotide Transformer)
Labeled Biological Data (e.g., Promoters, Variants, Binding Sites)
Fine-tuning (PEFT, LoRA, Full or Adapter Training)
Gene Expression Predictor
Variant Effect Model
Enhancer-Promoter Interaction Model

Pre-training vs Fine-tuning

Aspect	Pre-training	Fine-tuning
Data Size	Billions of sequence tokens (e.g., genomes, proteomes)	Thousands to millions of labeled samples
Data Type	Unlabeled biological sequences (self-supervised tasks)	Labeled data for specific tasks (e.g., promoters, variants, binding sites)
Objective	Learn general biological representations	Adapt representations to a specific prediction task
Time	Weeks to months	Hours to days
Compute & GPUs	Multi-GPU or TPU clusters (e.g., 64–256 GPUs)	Single or few GPUs (e.g., LoRA on RTX 3090)
Cost Estimate	\$50k–\$500k+	\$10–\$1k
Result	Foundation (base) model	Task-adapted (specialized) model

Pretraining builds “biological intelligence,” fine-tuning channels it for specific task

Fine-Tuning: The Transfer Learning Paradigm

Data Efficiency: Learn More from Less	• Requires only a small labeled dataset to specialize for a new biological task.
Compute & Time Efficiency: Faster by Design	• Enables rapid iteration and experimentation across many tasks.
Performance Boost: Smarter Starting Point	• Achieves strong accuracy and generalization with minimal additional training.
Sustainability & Reusability: Train Once, Use Many Times	• Reuses existing foundation models instead of retraining from scratch.

Full Fine-Tuning vs. Parameter-Efficient Fine-Tuning

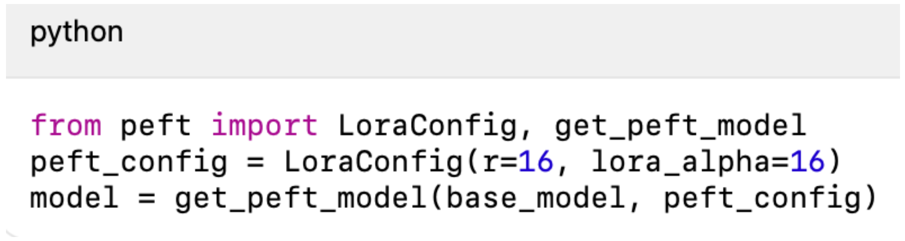
- Finetuning full model can be very heavy and complex
  - Let’s take examle of DNABERT-2 model (117 Million parameters)

Aspect	Full Fine-Tuning	Parameter-Efficient Fine-Tuning (PEFT)
Parameters Updated	All model 117M parameters	Only a small subset (e.g., LoRA or adapter weights)
GPU Memory Requirement	Very high (≈30 GB or more)	Low (≈4× less)
Training Speed	Slow — hours to days	Fast — minutes to a few hours
Risk of Catastrophic Forgetting	High — overwrites pre-trained knowledge	Low — base model remains frozen
Flexibility / Adaptability	Full flexibility — can adapt all layers	Moderate — only modifies low-rank or adapter components
Best Suited For	Large institutions or foundational model creators	Research labs and applied task fine-tuning

LoRA: From Full Fine-Tuning → Smart Parameter Updates

- Full fine-tuning updates every parameter, demanding huge GPU memory and long training.
- LoRA offers a smarter alternative — train only a few thousand parameters while keeping most knowledge intact.

- How LoRA works
  - Freeze the pre-trained model
  - Insert small trainable matrices A and B
  - Update only 1–3 % of parameters
- Outcome
  - 100× fewer trainable parameters
  - 4× less GPU memory
  - Modular adapters per task


python

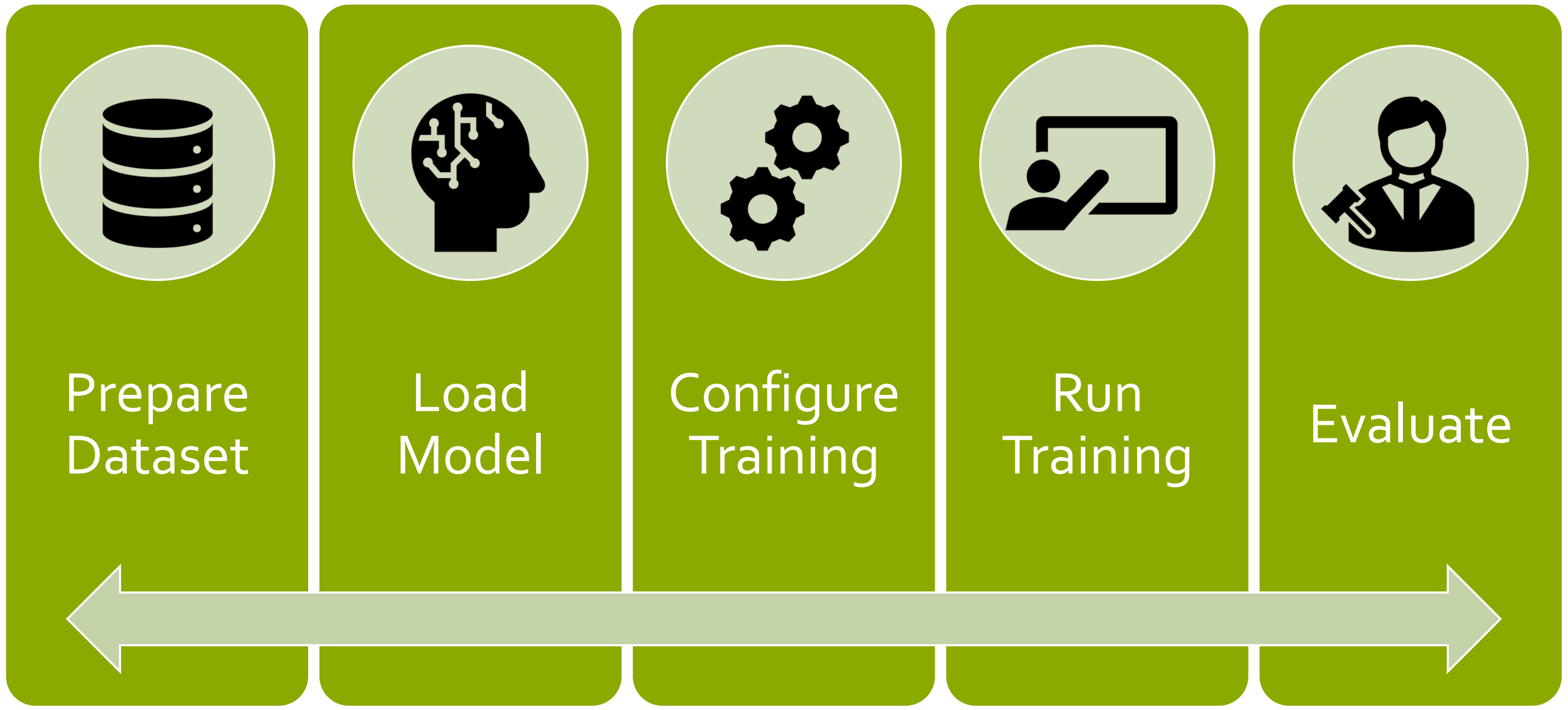
```python
from peft import LoraConfig, get_peft_model
peft_config = LoraConfig(r=16, lora_alpha=16)
model = get_peft_model(base_model, peft_config)
```

Quantization + LoRA → QLoRA

- Even LoRA models can be large → compress the frozen base further.
- Quantization:
  - ~50% memory reduction when moving from fp16 → 8-bit
  - ~75% reduction from fp16 → 4-bit
  - Accuracy largely maintained if quantization is applied correctly
- QLoRA = Quantization + LoRA:
  - Quantize the frozen base → smaller memory footprint
  - Keep LoRA adapters in higher precision for stability
  - Enables fine-tuning billion-parameter models on consumer GPUs

- LoRA shrinks what you train; quantization shrinks what you load
- Together, they achieve up to 4×–8× memory savings with relatively smaller in accuracy loss.

Overview of Fine-Tuning Pipeline



Prepare Dataset

- Get the data
  - Collect raw biological sequences (e.g., DNA, RNA, or protein) along with task-specific labels such as binding sites, promoters, or variant effects.
- Tokenize data
  - Convert sequences into numerical tokens (e.g., k-mers, BPE, character tokens) so the model can understand and process them efficiently.
- Split into Train, Validation and Test set
  - Divide the dataset to train the model, tune its parameters, and fairly evaluate its performance on unseen examples.
- Configure Data Collation
  - Define how batches are formed during training—padding, truncation, and alignment—to ensure consistent sequence lengths and efficient GPU use.

Load Model

- Load pre-trained model
  - Usually hosted on model hubs (e.g., Hugging Face Hub, GitHub, or lab repositories)
  - Can also be stored locally or on HPC
- Attach PEFT adapters (e.g., LoRA)
- Model organization
  - Architecture – the model's structure (e.g., transformer layers, attention heads).
  - Weights / Parameters: Numerical values that capture learned biological relationships.
  - Tokenizer / Vocabulary: The mapping between biological symbols (A, T, C, G, amino acids) and numerical tokens
  - Configuration File: Metadata such as model size, layer count, and training setup.

Configure Training

- Why configuration matters
  - Same model + Same data + Different configuration = Totally different outcomes
- The Restaurant Analogy
  - Same ingredients but a change in recipe may drastically change food experience

- Bad Configuration
  - Training: Crashes after 2 days
  - Memory: OOM errors
  - Accuracy: 52% (coin flip!)
  - Cost: $500 wasted
  - You: "Deep learning is broken!"

- Good Configuration
  - Training: 2 hours done
  - Memory: 60% utilized
  - Accuracy: 89% (paper-worthy!)
  - Cost: $5 total
  - You: "I'm publishing this!"

Key Training Hyperparameters

Parameter	Description	Typical Value / Range
Learning Rate	Most critical parameter: too high → divergence; too low → no learning.	1e-5 to 1e-4 (for fine-tuning)
Batch Size	Balances memory usage vs gradient quality.	8–32 (use gradient accumulation if limited)
Number of Epochs	Number of full passes through the data.	Use early stopping rather than fixed count
Warmup Steps	Gradual lr increase at start to prevent instability.	5–10% of total steps
Optimizer	Update rule; affects convergence speed and stability.	AdamW (β_1=0.9, β_2=0.999, ε=1e-8)
Dropout Rate	Randomly zeros activations to regularize; applied to embeddings/attention/MLP.	0.0–0.2 (often 0.1 in pretrained configs)

Questions &
Comments