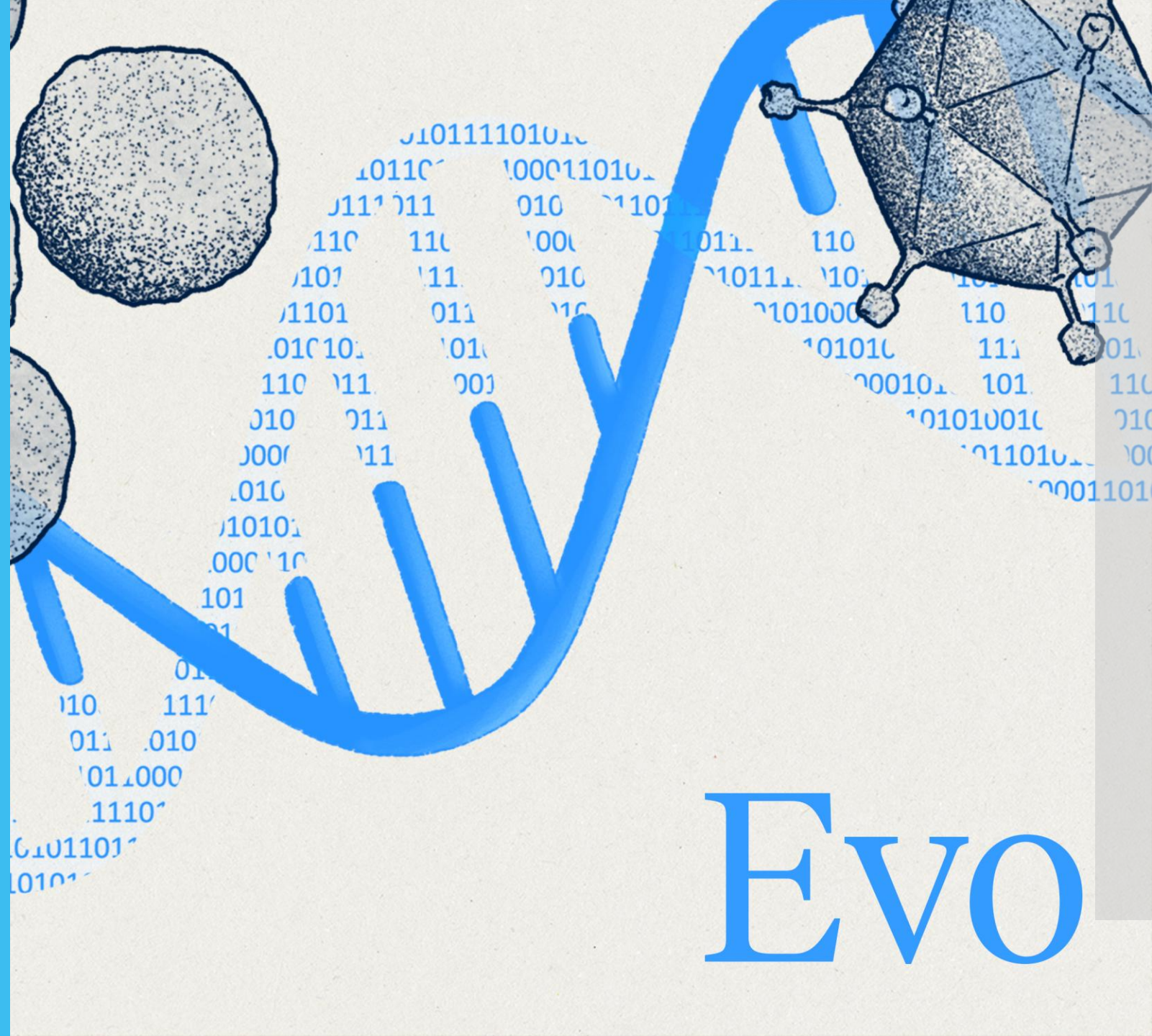


# Optimization of DNA Constructs for Gene Expression Guided by Machine Learning

João Manuel Barbosa Lima PG55701  
Mestrado em Bioinformática 2024/25

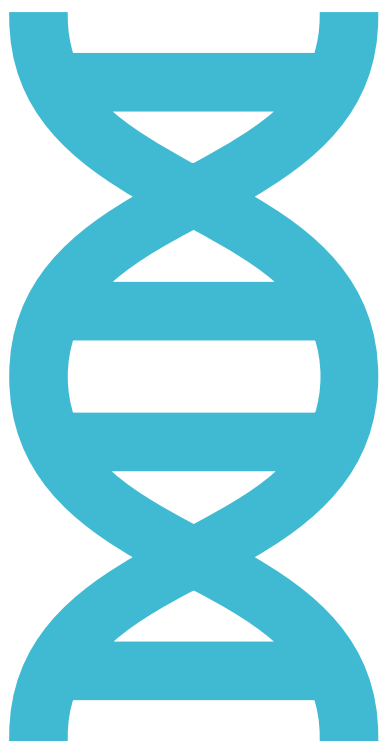


# Evo

# Objetivos

- Prever a expressão proteica com base em sequências de DNA (promotor + RBS)
- Usar o modelo EVO como extrator de features
- Treino supervisionado

EVO



## EVO

“Foundational model” treinado em 2.7 milhões de genomas bacterianos e de fagos

Processa sequências longas (131 kb)

7 mil milhões de parâmetros, tokenização de nucleótido único

Produce embeddings ricos em informação biológica

## Pontos a reter

- Padding e batching dinâmico
- Extração dos embeddings (média dos hidden states)
- embeddings: vetores densos que representam informação biológica
- Capturam motivos, contexto local/global, estrutura

Evo



# Dados

- Estudo de Kosuri et al. (2013) com 12.563 combinações promotor-RBS em *E. coli*
- 114 promotores + 111 RBS (ribossomal binding site)
- Mediram DNA, RNA e proteína para cada combinação

Evo

# Lógica Geral do Treino

Dados experimentais (Kosuri)



Filtragem e pré-  
processamento dos dados



EVO extrai representações  
numéricas (embeddings)



Modelo supervisionado faz a  
previsão da expressão proteica

EVO

# Problemas...

- VRAM insuficiente (testado em várias máquinas do IQTB)
- **Tentativa de forçar o uso da CPU** (sem sucesso, possivelmente a stripedhyena ou o FlashAttention possuem dependências das GPUs em funcionalidades específicas de GPUs NVIDIA com arquitetura Ampere )
- Máximo que foi disponibilizado até à data: VRAM Total 12 GiB
- **Peso do evo:** Cada parâmetro ocupa 2 bytes (FP16/BF16). Com  $6.45 \times 10^9$  parâmetros  $\times 2$  bytes/parâmetros  $\approx 12.9$  GB (arredondando para **~14 GB** para considerar overheads e ativações)
- Recomendado: 24 GB de VRAM

```
jlima@ssb03:~/projeto$ python3 script.py
GPU desativada programaticamente: torch.cuda.is_available() agora retorna False.
Módulo 'evo.models.Evo' importado com sucesso.
TensorFloat32 ativado para operações de matriz CUDA (se aplicável).

Usando dispositivo: cpu (forçado para CPU).
AVISO: A execução na CPU será EXTREMAMENTE LENTA para o modelo EVO-1-8k-base (7 bilhões de parâmetros).

Dados carregados com sucesso de /data/jlima/home/projeto/data/training_data.csv. Dimensão: (8794, 3)

Iniciando modelo EVO 'evo-1-8k-base'...
Passo 4.1: Instanciando o modelo Evo('evo-1-8k-base'). Isso fará o download dos pesos do modelo (aprox. 7GB).
Loading checkpoint shards: 100% | 3/3 [00:00<00:00, 55.05it/s]
Passo 4.2: Modelo Evo instanciado. Parâmetros: 6.45 Bilhões.
Passo 4.3: Tentando mover o modelo para o dispositivo 'cpu' (CPU RAM)...
Passo 4.4: Modelo EVO carregado com sucesso no dispositivo (CPU).
AVISO: O modelo está a ser executado na CPU. A inferência será EXTREMAMENTE LENTA.
A máquina tem 31GiB de RAM, o que é suficiente para o modelo EVO-1-8k-base.
Passo 4.5: Iniciando a extração de features com batch_size=1...

Extraindo features do EVO para 8794 sequências com batch_size=1...

ERRO CRÍTICO na secção 4: Falha ao carregar ou extrair features do modelo EVO: invalid argument to exchangeDevice
Verifique a sua conexão com a internet (para download do modelo) e se o 'evo-model' está instalado corretamente.
A causa mais provável deste erro é um problema inesperado na execução do modelo na CPU.
Se o problema persistir, pode indicar uma incompatibilidade fundamental da biblioteca 'evo-model' com a execução puramente em CPU, ou um problema de baixo n
ível com a instalação do PyTorch para CPU.
```

```
jlima@ssb03:~/projeto$ python3 script.py
Módulo 'evo.models.Evo' importado com sucesso.
TensorFloat32 ativado para operações de matriz CUDA (se aplicável).

Usando dispositivo: cuda:0.

Dados carregados com sucesso de /data/jlima/home/projeto/data/training_data.csv. Dimensão: (8794, 3)

Iniciando modelo EVO 'evo-1-8k-base'...
Passo 4.1: Instanciando o modelo Evo('evo-1-8k-base'). Isso fará o download dos pesos do modelo (aprox. 7GB).
Loading checkpoint shards: 100% | 3/3 [00:00<00:00, 54.91it/s]
Passo 4.2: Modelo Evo instanciado. Parâmetros: 6.45 Bilhões.
Passo 4.3: Tentando mover o modelo para o dispositivo 'cuda:0'...
Tentando carregar o modelo em half precision (FP16) para economizar VRAM...


ERRO CRÍTICO na secção 4: Falha ao carregar ou extrair features do modelo EVO: CUDA out of memory. Tried to allocate 86.00 MiB. GPU 0 has a total capacity of
11.76 GiB of which 47.00 MiB is free. Including non-PyTorch memory, this process has 11.56 GiB memory in use. Of the allocated memory 11.45 GiB is allocat
ed by PyTorch, and 16.69 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expanda
ble_segments:True to avoid fragmentation. See documentation for Memory Management (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
Verifique a sua conexão com a internet (para download do modelo) e se o 'evo-model' está instalado corretamente.
Se o problema persistir, pode haver um problema de configuração do ambiente ou de recursos inesperado.
jlima@ssb03:~/projeto(1)$
```

# Pré- Processamento dos Dados

Filtragem com flags de qualidade,  
Remoção de outliers e Normalização  
com z-score da expressão proteica



Merge das sequências:  
promotor + "AACTT" + RBS



Resultado:  
~8.700 sequências biológicas completas  
com a medição de expressão proteica



# Explicação da lógica do código

- `model.eval()` - Coloca o modelo EVO em modo de avaliação
- Processamento em Lotes - `batch_size`
- `tokenizer.tokenize(seq)` - Tokenização e Padding
- `with torch.no_grad()::` - Envolve a chamada do modelo EVO
- `logits, hidden_states = model(input_ids):` - Chamada principal do EVO
- Logits - São as saídas brutas do modelo
- `sequence_embeddings = hidden_states[0].mean(dim=1)` - calcular a média ao longo da dimensão da sequência
- `all_embeddings.append(sequence_embedding.s.cpu().numpy()):` - embeddings são movidos para a CPU e convertidos para arrays NumPy

```
# --- 3. Função para Extrair Features do EVO ---
def extract_evo_features(sequences: list, model, tokenizer, device: str, batch_size: int):
    """
    Extraí embeddings de sequência usando o modelo EVO.
    Assume que o modelo retorna (logits, hidden_states) e usa a média dos hidden_states.
    """
    print(f"\nExtraindo features do EVO para {len(sequences)} sequências com batch_size={batch_size}..")
    model.eval() # Colocar o modelo em modo de avaliação
    all_embeddings = []

    for i in range(0, len(sequences), batch_size):
        batch_sequences = sequences[i:i + batch_size]

        # Encontrar o comprimento máximo no batch para padding
        max_seq_length = max(len(seq) for seq in batch_sequences)
        input_ids_batch = []
        for seq in batch_sequences:
            tokenized_seq = tokenizer.tokenize(seq)
            # Adicionar padding ao final se a sequência for menor que a máxima do batch
            padding_needed = max_seq_length - len(tokenized_seq)
            padded_tokenized_seq = tokenized_seq + [tokenizer.pad_id] * padding_needed
            input_ids_batch.append(padded_tokenized_seq)

        input_ids = torch.tensor(input_ids_batch, dtype=torch.long).to(device)

        with torch.no_grad():
            logits, hidden_states = model(input_ids)

            # Para obter um embedding de sequência, fazemos a média ao longo da dimensão da sequência
            sequence_embeddings = hidden_states[0].mean(dim=1)

            all_embeddings.append(sequence_embeddings.cpu().numpy())

        print(f"  Processado {min(i + batch_size, len(sequences))}/{len(sequences)} sequências...")
        # Forçar a coleta de lixo para liberar memória após cada batch
        gc.collect()
        if device == 'cuda:0':
            torch.cuda.empty_cache()

    return np.vstack(all_embeddings)
```

# Explicação da lógica do código

- Carregamento de Embeddings Pré-calculados
- Inicialização do Modelo EVO
- `evo_model_instance = Evo(EVO_MODEL_NAME)` - carrega os pesos pré-treinados do EVO
- `model.half().to(device)` - Move o modelo para a GPU (se `device = cuda:0`)
- `extract_evo_features(...)` – função definida anteriormente gera os embeddings
- `np.save(...)` – salva os embeddings no disco

```
# --- 4. Carregar Modelo EVO e Extrair Features ---
evo_features = None

# Tentar carregar embeddings pré-calculados para economizar tempo
if os.path.exists(EVO_EMBEDDINGS_FILE):
    try:
        loaded_embeddings = np.load(EVO_EMBEDDINGS_FILE)
        if loaded_embeddings.shape[0] == len(sequences):
            evo_features = loaded_embeddings
            print(f"\nCarregando embeddings do EVO de {EVO_EMBEDDINGS_FILE}. Dimensão: {evo_features.shape}")
        else:
            print(f"\nAviso: Embeddings carregados ({loaded_embeddings.shape[0]} sequências) não correspondem ao dataset atual ({len(sequences)}).")
    except Exception as e:
        print(f"\nErro ao carregar embeddings de {EVO_EMBEDDINGS_FILE}: {e}. Recalculando.")

if evo_features is None:
    print(f"\nIniciando modelo EVO '{EVO_MODEL_NAME}'...")
    try:
        print(f"Passo 4.1: Instanciando o modelo Evo('{EVO_MODEL_NAME}'). Isso fará o download dos pesos do modelo (aprox. 7GB).")
        evo_model_instance = Evo(EVO_MODEL_NAME)
        model, tokenizer = evo_model_instance.model, evo_model_instance.tokenizer

        print(f"Passo 4.2: Modelo Evo instanciado. Parâmetros: {sum(p.numel() for p in model.parameters()) / 1e9:.2f} Bilhões.")
        print(f"Passo 4.3: Tentando mover o modelo para o dispositivo '{device}'...")

        # Tentar carregar o modelo em half precision (FP16) para economizar VRAM, se usando GPU.
        if device == 'cuda:0':
            print("Tentando carregar o modelo em half precision (FP16) para economizar VRAM...")
            model.half().to(device)
        else:
            model.to(device) # Mover para CPU RAM

        print("Passo 4.4: Modelo EVO carregado com sucesso no dispositivo.")
        if device == 'cpu':
            print("AVISO: O modelo está a ser executado na CPU. A inferência será mais lenta do que na GPU.")
        else:
            print("O modelo está a ser executado na GPU.")

        print(f"Passo 4.5: Iniciando a extração de features com batch_size={EVO_BATCH_SIZE}...")
        evo_features = extract_evo_features(sequences, model, tokenizer, device, batch_size=EVO_BATCH_SIZE)

        # Garantir que o diretório para salvar os embeddings exista
        os.makedirs(DATA_PATH_FOR_EMBEDDINGS, exist_ok=True)
        np.save(EVO_EMBEDDINGS_FILE, evo_features)
        print(f"\nPasso 4.6: Embeddings do EVO gerados e salvos em {EVO_EMBEDDINGS_FILE}.")

    except Exception as e:
        print(f"\nERRO CRÍTICO na secção 4: Falha ao carregar ou extrair features do modelo EVO: {e}")
        print("Verifique a sua conexão com a internet (para download do modelo) e se o 'evo-model' está instalado corretamente.")
        print("Se o problema persistir, pode haver um problema de configuração do ambiente ou de recursos inesperado.")
        sys.exit(1)
```

# Explicação da lógica do código

- X = evo\_features - variáveis de entrada
- y = prot\_z - variável alvo

```
# --- 5. Preparar Dados para o SVM ---  
X = evo_features  
y = prot_z  
  
print(f"\nDimensão de X (features do EVO): {X.shape}")  
print(f"Dimensão de y (target 'prot_z'): {y.shape}")
```

# Explicação da lógica do código

- `n_splits = 5; kf = KFold(...)` - validação cruzada de 5-fold
- `param_grid` - Define o espaço de busca
- `grid_search = GridSearchCV(svr, param_grid, cv=kf, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)`
- `grid_search.fit(X, y)` - Inicia o processo de procura
- `best_svr_model` - devolve o modelo SVM com elhor desempenho médio

```
# --- 6. Treinar o SVM (SVR - Support Vector Regressor) ---

# Implementação da Validação Cruzada (KFold)
n_splits = 5 # Validação cruzada de 5-fold
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# Otimização de Hiperparâmetros com GridSearchCV
# Ranges expandidos para aproveitar os novos recursos.
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100, 1000, 10000], # Mais opções para C
    'gamma': ['scale', 'auto', 0.00001, 0.0001, 0.001, 0.01, 0.1, 1], # Mais opções para gamma
    'kernel': ['rbf']
}

svr = SVR()

print("\nIniciando busca por hiperparâmetros com GridSearchCV (pode demorar)...")
# Usando n_jobs=-1 para usar todos os núcleos disponíveis.
grid_search = GridSearchCV(svr, param_grid, cv=kf, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)
grid_search.fit(X, y)

print(f"\nMelhores hiperparâmetros encontrados: {grid_search.best_params_}")
best_svr_model = grid_search.best_estimator_
```



# Prespetivas Futuras

- Executar o código em computador/servidor com recursos adequados
- Implementar FNN e outros algoritmos de ML
- Expansão do dataset de Kosuri et al
- Experimentar o modelo evo-1-131k-base (se houver recursos)

Evo