Manuale di Addestramento e Sviluppo

Sistema di Captioning SVG

Progetto di Tesi Magistrale

Ediluzio

30 marzo 2025

Indice

1 Struttura del Progetto Aggiornata

Questa è la struttura del progetto dopo le eliminazioni di file ridondanti e spostati. Si raccomanda ulteriore refactoring spostando i moduli core e dataset in `shared/`.

1.1 Architettura Decoder-Only

Listing 1: Struttura directory decoder_only

```
TESI_EDILUZIO/└──
 decoder_only/
    ├── slurm_error_*.log
    ├── slurm_output_*.log
    ├── requirements.txt          (*@\color{red}{Racc: Spostare/Unire}@*)
    ├── monitor_training.py       (*@\color{red}{Racc: Spostare}@*)
    ├── svg_direct_captioning_pipeline.py
    ├── install_dependencies.sh   (*@\color{red}{Racc: Spostare}@*)
    ├── test_direct_decoder_only.py
    ├── final_training_config.json
    ├── main_direct.py
    ├── svg_dataset/              (*@\color{red}{Racc: Spostare a livello TESI\_
    │   ├──  data_handling_slurm.py
    │   ├──  filter_svg_dataset.py
    │   ├──  main_dataset.py
    │   ├──  download_svg_dataset.py
    │   ├──  prepare_svg_dataset.py  (*@\color{blue}{MODIFICATO}@*)
    │   ├──  *.log
    ├── training_plots/
    └── svg_captioning/           (*@\color{red}{Racc: Spostare moduli core in s
        ├── __init__.py
        ├── svg_direct_tokenizer.py (*@\color{blue}{MODIFICATO}@*)
        ├── svg_vector_processor.py
        ├── nemotron_model.py       (*@\color{blue}{MODIFICATO}@*)
        ├── text_generation.py
        ├── train_model_slurm.py
        ├── evaluation.py
        ├── reward_model.py
        ├── generate_candidates.py
        ├── train_dpo.py
        ├── dpo_trainer.py
        ├── train_slurm.sh
        ├── dpo_slurm.sh
        ├── slurm_training_config.json
        └── final_training_config.json (*@\color{orange}{Duplicato?}@*)
```

1.2 Architettura Encoder-Decoder

Listing 2: Struttura directory encoder_decoder

```
TESI_EDILUZIO/└──
 encoder_decoder/
    ├── requirements.txt          (*@\color{red}{Racc: Eliminare}@*)
    ├── monitor_training.py       (*@\color{red}{Racc: Eliminare}@*)
    ├── finetune_encoder_decoder_slurm.sh
    ├── encoder_decoder_integration.py
    ├── encoder_decoder_conditioning.py
    ├── train_encoder_decoder.py
    ├── finetune_encoder_decoder.py
    ├── install_dependencies.sh   (*@\color{red}{Racc: Eliminare}@*)
    ├── encoder_decoder_slurm.sh
    ├── how_to_use.txt
    ├── config/
    │   ├──  final_training_config.json
    │   ├──  checkpoint_config.json
    │   └──  attention_config.json
    ├── utils/
    │   ├──  logger.py
    │   ├──  attention_visualization.py
    │   └──  checkpoint_manager.py
    └── svg_captioning/           (*@\color{red}{Racc: Spostare moduli core in s
        ├── __init__.py
        ├── dpo_trainer_encoder_decoder.py
        ├── train_dpo_encoder_decoder.py
        ├── train_model_slurm.py    (*@\color{orange}{Generico? Duplicato?}@*)
        └── dpo_encoder_decoder_slurm.sh (*@\color{orange}{Controllare path!}@*)
```

** Racc: Raccomandazione; MODIFICATO: File il cui contenuto è stato corretto; Duplicato?: File potenzialmente ridondante da verificare.*

1.3 Directory Condivisa

Listing 3: Struttura directory shared

```
TESI_EDILUZIO/└──
 shared/
    ├── __init__.py
    ├── requirements.txt          (*@\color{red}{Racc: Unire qui}@*)
    └── utils/                    (*@\color{red}{Racc: Unire qui da enc-dec/util
        ├── __init__.py
        ├── visualization.py
        └── logging_utils.py
        (*@\textit{...altre utils comuni...}@*)
```

(*@\color{red}{Racc: Aggiungere svg_core/ con moduli comuni}@*)

2 Workflow di Addestramento e Fine-tuning

I workflow assumono che la struttura sia stata riorganizzata come raccomandato (dataset e moduli core in posizione condivisa/accessibile). I comandi SLURM (`sbatch`) lanciano gli script specifici.

2.1 Addestramento Decoder-Only

Preparazione Ambiente e Dataset:

1. • Eseguire `bash install`$_d$*ependencies.sh*(*dallaposizionecondivisa/principale*). *Assicura*

2. **Addestramento Base (Progressivo):** Lanciare tramite SLURM. Lo script `train_slurm.sh` chiama `train_model_slurm.py`, che legge `slurm_training_config.json` o `final_training_config.json` per le fasi.

```
# Esempio per avviare la fase 3 (tutte le categorie)
cd /path/to/TESI_EDILUZIO/decoder_only/svg_captioning/
sbatch train_slurm.sh phase3_all_categories 20 16 1e-5
# Argomenti: [phase_name] [epochs] [batch_size] [learning_rate] (opzi
```

3. **Fine-tuning DPO:** Lanciare tramite SLURM. Lo script `dpo_slurm.sh` chiama `train_dpo.py`.

```
# Assicurarsi che BASE_MODEL in dpo_slurm.sh punti al modello addestr
cd /path/to/TESI_EDILUZIO/decoder_only/svg_captioning/
sbatch dpo_slurm.sh
```

4. **Monitoraggio:**

Usare `monitor_training.py` puntando al file `slurm_output_<jobid>.log` corretto. Usare TensorBoard: `tensorboard --logdir /path/to/logs/te`

2.2 Addestramento Encoder-Decoder

Preparazione Ambiente e Dataset: Come per Decoder-Only. **Addestramento Base Encoder-Decoder:** Lanciare tramite SLURM. Lo script `encoder_decoder_slurm.sh` chiama `train_model_slurm.py` (o potenzialmente `train_encoder_decoder.py` - da verificare).

2. **cd** /path/to/TESI_EDILUZIO/encoder_decoder/
   sbatch encoder_decoder_slurm.sh full 8 2 3e-5
   # Argomenti: [phase_name] [batch_size] [grad_accum] [learning_rate] (

3. **Fine-tuning Encoder-Decoder (Opzionale):** Se necessario, usare lo script specifico.

   cd /path/to/TESI_EDILUZIO/encoder_decoder/
   # Assicurarsi che PRETRAINED_MODEL punti al modello base enc-dec
   sbatch finetune_encoder_decoder_slurm.sh

4. **Fine-tuning DPO Encoder-Decoder:** Lanciare tramite SLURM. Lo script `dpo_encoder_de` chiama `train_dpo_encoder_decoder.py`.

   cd /path/to/TESI_EDILUZIO/encoder_decoder/svg_captioning/
   # !!! Aggiornare i percorsi placeholder in dpo_encoder_decoder_slurm.
   sbatch dpo_encoder_decoder_slurm.sh

5. **Monitoraggio:** Come per Decoder-Only, usando i file di log specifici dell'encoder-decoder.

3 File di Configurazione Esemplificativi

3.1 Configurazione di Addestramento (final_training_config.json)

Estratto da `decoder_only/final_training_config.json`:

Listing 4: Estratto Configurazione Addestramento Finale (Decoder-Only)

```
{
  "model_config": {
    "base_model": "nvidia/Nemotron-Mini-4B-Instruct",
    "model_save_dir": "/work/tesi_ediluzio/decoder_only/svg_dataset/model
  },
  "training_config": {
    "epochs": 50,
    "batch_size": 16,
    "learning_rate": 5e-5,
    "fp16": true,
    // ... altri parametri ...
  },
  "dataset_config": {
    "base_dir": "/work/tesi_ediluzio/decoder_only/svg_dataset/svg_dataset
```

```
    "progressive_datasets": {
      "phase1_simple_bw": {
        "train": { "path": "...", "pt": "...", "count": 190 },
        "val": { "path": "...", "pt": "...", "count": 40 },
        "test": { "path": "...", "pt": "...", "count": 20 },
        "categories": ["simple_bw"]
      },
      // ... altre fasi ...
    }
  },
  "training_phases": [
    { "name": "phase1_simple_bw", "epochs": 50, /*...*/ },
    { "name": "phase2_all_bw", "epochs": 30, /*...*/ },
    { "name": "phase3_all_categories", "epochs": 20, /*...*/ }
  ],
  "logging_config": {
    "log_dir": "/work/tesi_ediluzio/decoder_only/svg_dataset/logs",
    // ... altri parametri ...
  }
}
```

3.2 Configurazione Attention (attention_config.json)

Estratto da encoder_decoder/config/attention_config.json:

Listing 5: Estratto Configurazione Attenzione (Encoder-Decoder)

```
{
  "attention_config": {
    "num_attention_heads": 16,
    "num_conditioning_layers": 4,
    "position_embedding_type": "relative_key",
    "use_svg_structure": true,
    // ... altri parametri ...
  },
  "conditioning_config": {
    "use_gating": true,
    // ... altri parametri ...
  },
  "visualization_config": {
    "save_attention_maps": true,
    // ... altri parametri ...
  }
}
```

3.3 Configurazione Checkpoint (checkpoint_config.json)

Estratto da encoder_decoder/config/checkpoint_config.json:

Listing 6: Estratto Configurazione Checkpoint

```
{
    "checkpoint_config": {
        "max_checkpoints": 5,
        "save_frequency": 1000,
        "save_best_only": false,
        "metric_name": "loss",
        "metric_mode": "min",
        "checkpoint_dir": "checkpoints",
        // ... altri parametri ...
    }
}
```

4 Troubleshooting Avanzato

(Sezione mantenuta dal template originale, potenzialmente utile)

4.1 Errori Comuni e Soluzioni

Problema	**Soluzione Proposta**
GPU Out of Memory	1. Ridurre `batch_size`. 2. Aumentare `gradient_accumulation_steps`. 3. Abilitare *gradient checkpointing* (se supportato). 4. Usare precisione mista (`fp16` o `bf16`) se non già attiva. 5. Usare ottimizzatori a basso consumo di memoria (es. AdamW 8-bit tramite *bitsandbytes*).
NaN in loss (Not a Number)	1. Controllare la normalizzazione degli input/output. 2. Ridurre il `learning_rate`. 3. Aggiungere o ridurre il *gradient clipping* (`max_grad_norm`). 4. Provare a disattivare la precisione mista (`fp16`) per debug. 5. Verificare la presenza di divisioni per zero o logaritmi di zero/negativi nel codice custom.

Continua...

Problema	Soluzione Proposta
Training lento	1. Ottimizzare il caricamento dati: fare preprocessing offline e caricare dati già tokenizzati (come sembra tu stia già facendo con i file `.pt`). 2. Aumentare il numero di `num_workers` nel DataLoader (monitorare l'uso CPU). 3. Usare `pin_memory=True` nel DataLoader (se si usa GPU). 4. Verificare l'effettivo utilizzo della GPU (es. con `nvidia-smi`). 5. Profilare il codice Python (`cProfile`) per identificare colli di bottiglia.
Errori SLURM (es. Permessi, Moduli)	1. Verificare i percorsi assoluti negli script `.sh`. 2. Assicurarsi che i moduli (`module load ...`) siano corretti per l'ambiente HPC. 3. Controllare i permessi di scrittura nelle directory di output e temporanee (`$SLURM_TMPDIR`). 4. Usare `strace` (vedi sotto) per tracciare le chiamate di sistema dello script SLURM.
ImportError: Module not found	1. Assicurarsi che l'ambiente corretto (es. conda) sia attivato nello script SLURM. 2. Verificare che `requirements.txt` sia completo e installato nell'ambiente. 3. Impostare correttamente `PYTHONPATH` se si importano moduli locali (es. da `shared/`).

4.2 Comandi Utili per Debug

Listing 7: Comandi utili per il debug

```
# Analisi utilizzo risorse GPU (aggiorna ogni 5 secondi)
nvidia-smi --query-gpu=utilization.gpu,memory.used --format=csv -l 5

# Analisi processi su un nodo specifico (utile in SLURM)
# ssh <nome_nodo>
# top -u <tuo_username> # o htop se installato

# Profiling base del codice Python
# Eseguire su un nodo di calcolo, non sul nodo di login
# python -m cProfile -o profile.stats path/to/script_train.py [args...]

# Visualizzazione del profiling (richiede 'snakeviz' o simili)
# pip install snakeviz
# snakeviz profile.stats

# Profiling memoria (richiede 'memory-profiler')
# pip install memory-profiler
```

```
# python -m memory_profiler path/to/script_train.py [args...]

# Trace delle chiamate di sistema (utile per debug I/O o SLURM)
# Utile se uno script SLURM fallisce in modi strani
# strace -f -o slurm_trace.txt sbatch path/to/script.sh [args...]
```