Report Tecnico: Implementazione delle Richieste per il Progetto di Captioning SVG

1. Gestione del Gradient Accumulation

Stato: Implementato

Dettagli tecnici:

- Il gradient accumulation è una tecnica che permette di simulare batch più grandi accumulando i gradienti su più forward pass prima di eseguire un backward pass
- Nei test precedenti (Test 1 & 2), il parametro gradient_accumulation_steps era impostato a 2 per il multi-GPU e 16 per il single-GPU
- Nel nuovo test (Test 3), abbiamo aumentato il gradient_accumulation_steps a 4 per compensare la riduzione del batch size a 2
- Questo permette di mantenere un batch effettivo di 8 esempi (2 batch size × 4 accumulation steps)

Implementazione:

```
// Test 3 (Semplificato)
"per_device_train_batch_size": 2,
"gradient_accumulation_steps": 4,

// Test 2 (Multi-GPU)
"per_device_train_batch_size": 4,
"gradient_accumulation_steps": 2,
```

2. Mitigazione dell'Overfitting

Stato: Implementato

Dettagli tecnici:

- Abbiamo implementato un sistema robusto di early stopping sulla loss di validazione
- Il parametro patience è impostato a 100 step, fermando il training se non ci sono miglioramenti per 100 step consecutivi
- Il parametro min_delta è impostato a 0.0001, richiedendo un miglioramento minimo di questo valore
- Il sistema salva sempre il checkpoint con la loss di validazione più bassa (load_best_model_at_end=True)

- Abbiamo aumentato il numero di checkpoint salvati a 5 (save_total_limit=5)

Implementazione:

```
# Configurazione dell'early stopping
early_stopping_callback = EarlyStoppingCallback(
    patience=args.patience,  # 100 step
    min_delta=args.min_delta  # 0.0001
)

# Configurazione del salvataggio del miglior modello
training_args = TrainingArguments(
    # ...
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False,
    # ...
)
```

3. Aumento del Rango di LoRA

Stato: Implementato

Dettagli tecnici:

- Il rango di LoRA (r) determina la dimensionalità della matrice di basso rango utilizzata per approssimare gli aggiornamenti dei pesi
- Abbiamo aumentato il rango da 4 (Test 1 & 2) a 16 (Test 3)
- Abbiamo aumentato alpha da 8 (Test 1 & 2) a 32 (Test 3)
- Abbiamo esteso i target modules per includere tutti i componenti dell'attenzione (q_proj, k_proj, v_proj, o_proj)
- Questo aumenta la capacità del modello di apprendere pattern complessi, a costo di maggiore memoria e tempo di calcolo

Implementazione:

```
# Test 3 (Semplificato)
lora_config = LoraConfig(
    r=16,  # Aumentato da 4
    lora_alpha=32,  # Aumentato da 8
    target_modules=["q_proj", "v_proj", "k_proj", "o_proj"],  # Esteso
    lora_dropout=0.05,
    bias="none",
    task_type=TaskType.CAUSAL_LM
)

# Test 1 & 2
```

```
lora_config = LoraConfig(
    r=4,
    lora_alpha=8,
    target_modules=["q_proj", "v_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type=TaskType.CAUSAL_LM
)
```

4. Funzione per il Conteggio dei Parametri

Stato: Implementato

Dettagli tecnici:

- Abbiamo implementato una funzione `calculate_trainable_parameters` che calcola:
  i. Numero di parametri trainabili
  ii. Numero totale di parametri
  iii. Percentuale di parametri trainabili rispetto al totale
- La funzione è integrata nello script di training e viene chiamata dopo l'applicazione di LoRA
- I risultati vengono loggati sia nella console che su Weights & Biands

Implementazione:

```
def calculate_trainable_parameters(model) -> Dict[str, Any]:
    """
    Calcola il numero di parametri trainabili, il numero totale di parametri e la perc

    Args:
        model: Il modello da analizzare

    Returns:
        Un dizionario con le seguenti chiavi:
        - trainable_params: Numero di parametri trainabili
        - total_params: Numero totale di parametri
        - trainable_percentage: Percentuale di parametri trainabili rispetto al totale
    """
    trainable_params = 0
    total_params = 0

    for _, param in model.named_parameters():
        total_params += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()

    trainable_percentage = 100 * trainable_params / total_params if total_params > 0 e

    return {
```

```
        "trainable_params": trainable_params,
        "total_params": total_params,
        "trainable_percentage": trainable_percentage
    }
```

5. Riduzione del Learning Rate

Stato: Implementato

Dettagli tecnici:

- Abbiamo ridotto il learning rate a 1e-5 per entrambi i modelli
- Un learning rate più basso aiuta a:
  i. Migliorare la stabilità del training
  ii. Ridurre il rischio di overfitting
  iii. Permettere una convergenza più precisa, anche se potenzialmente più lenta
- Abbiamo mantenuto lo scheduler cosine per il decadimento graduale del learning rate

Implementazione:

```
// Test 3 (Semplificato)
"learning_rate": 1e-5,
"lr_scheduler_type": "cosine",
"warmup_ratio": 0.05,

// Test 1 & 2
"learning_rate": 2e-4,
"lr_scheduler_type": "cosine",
"warmup_ratio": 0.03,
```

6. Gestione dei Checkpoint

Stato: Implementato

Dettagli tecnici:

- Abbiamo aumentato il numero massimo di checkpoint salvati a 5 (save_total_limit=5)
- I checkpoint vengono salvati ogni 100 step (save_steps=100)
- La valutazione avviene ogni 50 step (eval_steps=50)
- Il parametro load_best_model_at_end è impostato a true per caricare automaticamente il miglior modello alla fine del training
- Implementato un sistema che preserva sempre il checkpoint con la loss di validazione più bassa

Implementazione:

```
"save_steps": 100,
"save_total_limit": 5,
"evaluation_strategy": "steps",
"eval_steps": 50,
"save_strategy": "steps",
"load_best_model_at_end": true,
"metric_for_best_model": "eval_loss",
"greater_is_better": false
```

7. Training su Dataset Completo

Stato: Già implementato

Dettagli tecnici:

- Il training utilizza il dataset completo senza divisioni per complessità
- File di training: `/work/tesi_ediluzio/data/processed/xml_format/train_set_final_xml.json`
- File di validazione: `/work/tesi_ediluzio/data/processed/xml_format/test_set_final_2k_xml.json`
- Non sono state apportate modifiche specifiche poiché il sistema già utilizzava il dataset completo

Implementazione:

```
"data_file": "/work/tesi_ediluzio/data/processed/xml_format/train_set_final_xml.json",
"val_file": "/work/tesi_ediluzio/data/processed/xml_format/test_set_final_2k_xml.json"
```

8. Early Stopping sulla Validazione

Stato: Implementato

Dettagli tecnici:

- L'early stopping è configurato per monitorare la loss di validazione
- Patience impostata a 100 step (si interrompe se non ci sono miglioramenti per 100 step consecutivi)
- Delta minimo impostato a 0.0001 (un miglioramento deve essere almeno di questo valore per essere considerato)
- Questo previene l'overfitting interrompendo il training quando la performance sul set di validazione smette di migliorare

Implementazione:

```
# Nello script SLURM
EARLY_STOPPING="--early_stopping"
PATIENCE=100
MIN_DELTA=0.0001

# Nella riga di comando
$PYTHON_ENV /work/tesi_ediluzio/scripts/training/train_lora_multi_gpu_simple.py \
    # ... altri parametri ...
    $EARLY_STOPPING \
    --patience "$PATIENCE" \
    --min_delta "$MIN_DELTA" \
    # ... altri parametri ...
```

9. Captioner Esterni e CLIP

Stato: Implementato

Dettagli tecnici:

- Abbiamo implementato un sistema per testare tre captioner esterni:
  i. BLIP: Salesforce/blip-image-captioning-large
  ii. ViT-GPT2: nlpconnect/vit-gpt2-image-captioning
  iii. CogVLM: THUDM/cogvlm-chat-hf
- Il sistema pre-renderizza gli SVG in PNG per l'input ai captioner
- Implementata la valutazione con CLIP score utilizzando il modello `openai/clip-vit-base-patch32`
- Il CLIP score misura la similarità semantica tra immagini e caption
- Il job `ext_capt` (2578232) è attualmente in esecuzione e sta valutando i captioner esterni

Implementazione:

```
def calculate_clip_score(image_path, caption, clip_model, clip_processor):
    """Calcola il CLIP score tra un'immagine e una didascalia."""
    try:
        # Carica l'immagine
        image = Image.open(image_path).convert("RGB")

        # Prepara gli input per CLIP
        inputs = clip_processor(
            text=[caption],
            images=image,
            return_tensors="pt",
            padding=True
        ).to(device)
```

```
        # Calcola gli embedding
        with torch.no_grad():
            outputs = clip_model(**inputs)

        # Calcola la similarità coseno
        logits_per_image = outputs.logits_per_image
        clip_score = logits_per_image.item()

        return clip_score
    except Exception as e:
        print(f"Errore nel calcolo del CLIP score: {e}")
        return 0.0
```

Conclusioni

Tutte le richieste sono state implementate con successo, con particolare attenzione a:

1. **Ottimizzazione del training**:
   - Riduzione del learning rate per migliorare la stabilità
   - Aumento del rango di LoRA per maggiore capacità di apprendimento
   - Implementazione di un sistema robusto per la gestione dei checkpoint
2. **Monitoraggio e analisi**:
   - Funzione per il conteggio dei parametri
   - Early stopping sulla loss di validazione
   - Integrazione con Weights & Biands per il tracking degli esperimenti
3. **Valutazione esterna**:
   - Implementazione di tre captioner esterni
   - Valutazione con CLIP score
   - Creazione di report comparativi

I job sono stati riavviati con le nuove configurazioni:

- `test_svg` (2578252): In esecuzione, test di rendering SVG
- `ext_capt` (2578232): In esecuzione, valutazione captioner esterni
- `llama_test3` (2578328): In attesa, training LoRA semplificato per Llama
- `gemma_test3` (2578329): In attesa, training LoRA semplificato per Gemma

Documentazione Script SVG Captioning

Script di Training

1. Training Multi-GPU

- **File**: `scripts/slurm/run_llama_test3_multi_gpu.slurm`
- **File**: `scripts/slurm/run_gemma_test3_multi_gpu.slurm`
- **Funzionalità**:
  - Training LoRA su 2 GPU
  - Batch size: 4 per GPU (totale 8)
  - Early stopping con patience=100
  - Checkpoint ogni 100 step
  - Logging su W&B

2. Valutazione Captioner Esterni

- **File**: `scripts/slurm/run_external_captioner_evaluation.slurm`
- **Funzionalità**:
  - Valutazione BLIP, ViT-GPT2, CogVLM
  - Calcolo metriche standard (BLEU, METEOR, CIDEr)
  - Calcolo CLIP Score
  - Generazione report HTML

3. Calcolo CLIP Score

- **File**: `scripts/slurm/run_clip_evaluation_updated.slurm`
- **Funzionalità**:
  - Calcolo CLIP Score per tutte le didascalie
  - Supporto CPU e GPU
  - Salvataggio risultati in JSON

Script di Analisi

1. Generazione Grafici

- **File**: `generate_metrics_radar_chart.py`
- **Funzionalità**:
  - Generazione grafici radar comparativi
  - Inclusione CLIP Score
  - Personalizzazione stile e legenda

2. Report Qualitativi

- **File**: `experiments/xml_direct_input/generate_qualitative_report.py`
- **Funzionalità**:
  - Generazione report HTML
  - Visualizzazione SVG e didascalie
  - Confronto tra modelli

3. Analisi Parametri

- **File**: `scripts/slurm/run_count_parameters.slurm`
- **Funzionalità**:
  - Conteggio parametri totali e trainabili
  - Calcolo percentuale parametri LoRA
  - Analisi efficienza memoria

Script di Utilità

1. Preprocessing

- **File**: `scripts/data_processing/process_svg_to_xml.py`
- **Funzionalità**:
  - Conversione SVG in formato XML
  - Preparazione dataset per training
  - Validazione formato input

2. Monitoraggio

- **File**: `scripts/utils/monitor_training.py`
- **Funzionalità**:
  - Monitoraggio loss e metriche
  - Rilevamento early stopping
  - Logging su W&B

3. Valutazione

- **File**: `scripts/evaluation/evaluate_external_captioners.py`
- **Funzionalità**:
  - Calcolo metriche standard
  - Integrazione CLIP Score
  - Generazione report dettagliati

Configurazione Hardware

Requisiti

- GPU: 2x L40S 48GB o A40 48GB
- Memoria: 48GB
- CPU: 8 core

Ottimizzazioni

- Quantizzazione 4-bit
- Gradient checkpointing
- Batch size ottimizzato
- Multi-GPU training

Note di Implementazione

Training

- Learning rate: 1e-5
- Weight decay: 0.01
- Warmup ratio: 0.05
- LR scheduler: cosine

Valutazione

- Metriche: BLEU, METEOR, CIDEr, CLIP Score
- Batch size: ottimizzato per memoria
- Supporto multi-GPU

Monitoraggio

- Logging ogni 10 step
- Checkpoint ogni 100 step
- Early stopping con patience=100