04_Finetune_Model

December 14, 2025

1 QLoRA Fine-tuning NER

1.1 2. Imports dependencies

[ ]:
```
import os
import json
import random
import logging
import sys
import re
from pathlib import Path
from typing import Dict, List, Tuple, Optional
from dataclasses import dataclass, field
from collections import defaultdict

import torch
import numpy as np
from torch.utils.data import Dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    TrainingArguments,
    Trainer,
    BitsAndBytesConfig,
    TrainerCallback,
    EvalPrediction,
)
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
from seqeval.metrics import classification_report, f1_score, precision_score,␣
 ↪recall_score

# Add project root to path
project_root = Path.cwd().parent
if str(project_root) not in sys.path:
    sys.path.insert(0, str(project_root))

# Import custom data loader
from src.data import load_processed_data
```

```
# Import prompt builder
from src.prompt import build_prompt

# Import evaluation utilities
from src.utils.evaluation import parse_ner_response

# Configure logging
logging.basicConfig(
    format="%(asctime)s - %(levelname)s - %(message)s",
    level=logging.INFO,
    datefmt="%Y-%m-%d %H:%M:%S"
)
logger = logging.getLogger(__name__)

# Set random seeds for reproducibility
def set_seed(seed: int = 42):
    """Set random seeds for reproducibility."""
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    # Enhanced determinism for production
    os.environ['PYTHONHASHSEED'] = str(seed)
    if torch.cuda.is_available():
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False

set_seed(42)
```

1.2 3. Configuration

Security Note: Never hardcode tokens in notebooks. Use environment variables or secure credential management.

[2]:
```
@dataclass
class Config:
    """Training configuration parameters."""

    # Model configuration
    # Option 1: Ministral-3-14B (requires transformers >= 4.47.0)
    # model_name: str = "mistralai/Ministral-3-8B-Instruct-2512"

    # Option 2: Mistral-7B (works with older transformers, still excellent for␣
 ↪NER)
    model_name: str = "mistralai/Mistral-7B-Instruct-v0.3"
```

```
    max_length: int = 1536

    # Data configuration
    data_dir: Path = Path("../data/vlps_2018_ner/processed")
    train_file: str = "train.json"
    val_ratio: float = 0.1
    seed: int = 42

    # QLoRA configuration
    lora_r: int = 64  # Increased for better capacity
    lora_alpha: int = 128  # 2x lora_r is recommended
    lora_dropout: float = 0.05  # Lower dropout for better convergence
    lora_target_modules: List[str] = field(default_factory=lambda: [
        "q_proj", "k_proj", "v_proj", "o_proj",  # All attention projections
        "gate_proj", "up_proj", "down_proj"  # MLP layers for better performance
    ])

    # Training configuration
    output_dir: str = "./checkpoints/mistral-7B-Instruct-v0.3-ner-qlora"
    num_train_epochs: int = 10
    per_device_train_batch_size: int = 1
    per_device_eval_batch_size: int = 2
    gradient_accumulation_steps: int = 8  # Effective batch size = 8
    eval_accumulation_steps: int = 8  # Match training accumulation for␣
↪consistency
    learning_rate: float = 2e-4  # Standard for QLoRA
    weight_decay: float = 0.01
    warmup_ratio: float = 0.03
    lr_scheduler_type: str = "cosine"
    max_grad_norm: float = 1.0  # Standard gradient clipping (was 0.3, too␣
↪aggressive)
    optim: str = "paged_adamw_8bit"

    # Logging and checkpointing
    logging_steps: int = 10
    eval_steps: int = 100
    save_steps: int = 100
    save_total_limit: int = 3

    # Hardware configuration
    bf16: bool = True  # Use bfloat16 for better stability
    use_4bit: bool = True
    bnb_4bit_compute_dtype: str = "bfloat16"
    bnb_4bit_quant_type: str = "nf4"  # Normal Float 4-bit
    use_nested_quant: bool = True  # Double quantization for memory savings

    # HuggingFace token (read from environment)
```

```
    hf_token: Optional[str] = field(default_factory=lambda: os.
↪getenv("HF_TOKEN"))

    def __post_init__(self):
        """Validate configuration after initialization."""
        self.data_dir = Path(self.data_dir)
        self.output_dir = Path(self.output_dir)

        if not self.data_dir.exists():
            raise ValueError(f"Data directory does not exist: {self.data_dir}")

        train_path = self.data_dir / self.train_file
        if not train_path.exists():
            raise ValueError(f"Training file does not exist: {train_path}")

        # Create output directory
        self.output_dir.mkdir(parents=True, exist_ok=True)

        # Warn if HF token is not set for gated models
        if self.hf_token is None:
            logger.warning(
                "HF_TOKEN environment variable not set. "
                "If using a gated model, authentication will fail."
            )

        # Validate batch configuration
        if self.per_device_eval_batch_size < self.per_device_train_batch_size:
            logger.warning(
                f"Eval batch size ({self.per_device_eval_batch_size}) is␣
↪smaller than "
                f"train batch size ({self.per_device_train_batch_size}).␣
↪Consider increasing it."
            )

# Initialize configuration
config = Config()
logger.info(f"Configuration initialized successfully")
logger.info(f"Model: {config.model_name}")
logger.info(f"Output directory: {config.output_dir}")
logger.info(f"GPU available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    logger.info(f"GPU device: {torch.cuda.get_device_name(0)}")
    logger.info(f"GPU memory: {torch.cuda.get_device_properties(0).total_memory␣
↪/ 1e9:.2f} GB")
```

```
2025-12-14 06:29:00 - WARNING - HF_TOKEN environment variable not set. If using
a gated model, authentication will fail.
```

```
2025-12-14 06:29:00 - INFO - Configuration initialized successfully
2025-12-14 06:29:00 - INFO - Model: mistralai/Mistral-7B-Instruct-v0.3
2025-12-14 06:29:00 - INFO - Output directory:
checkpoints/mistral-7B-Instruct-v0.3-ner-qlora
2025-12-14 06:29:00 - INFO - GPU available: True
2025-12-14 06:29:00 - INFO - GPU device: NVIDIA GeForce RTX 4060 Ti
2025-12-14 06:29:00 - INFO - GPU memory: 16.71 GB
```

1.3 4. Data Processing

Define the NER instruction prompt and data loading utilities.

[ ]:
```python
def build_training_prompt(example: Dict, include_response: bool = True) -> str:
    """
    Build a formatted prompt for training.
    Uses build_prompt() from src.prompt for the base prompt,
    then adds ground truth response for training.

    Args:
        example: Dictionary containing 'text' and optionally 'ground_truth'
        include_response: Whether to include the response (for training)

    Returns:
        Formatted prompt string with optional response
    """
    # Use the imported build_prompt for consistency
    prompt = build_prompt(example['text'])

    if include_response and 'ground_truth' in example:
        gt = example.get('ground_truth', {}) or {}
        response = {
            "person": gt.get("person", []),
            "organizations": gt.get("organizations", []),
            "address": gt.get("address", []),
        }
        prompt += f"\n{json.dumps(response, ensure_ascii=False, indent=2)}"

    return prompt


def split_train_data(
    data: List[Dict],
    val_ratio: float = 0.1,
    seed: int = 42
) -> Tuple[List[Dict], List[Dict]]:
    """
    Split data into train and validation sets.
```

```
    Args:
        data: List of examples
        val_ratio: Validation set ratio
        seed: Random seed

    Returns:
        Tuple of (train_data, val_data)
    """
    random.seed(seed)
    np.random.seed(seed)

    # Shuffle data
    shuffled_data = data.copy()
    random.shuffle(shuffled_data)

    # Split
    val_size = max(1, int(len(data) * val_ratio))
    val_data = shuffled_data[:val_size]
    train_data = shuffled_data[val_size:]

    # Warn if validation set is too small
    if val_size < 50:
        logger.warning(
            f"Validation set very small ({val_size} examples). "
            f"Consider increasing val_ratio for more reliable evaluation."
        )

    return train_data, val_data


# Load and split data
logger.info("Loading data using src.data loader...")
all_data = load_processed_data("data/vlps_2018_ner/processed/train.json")
logger.info(f"Loaded {len(all_data)} examples")

train_examples, val_examples = split_train_data(all_data, val_ratio=config.
 ↪val_ratio, seed=42)
logger.info(f"Split: {len(train_examples)} train, {len(val_examples)}␣
 ↪validation")

# Display sample
logger.info("\n=== Sample Training Example ===")
sample_prompt = build_training_prompt(train_examples[0], include_response=True)
print(sample_prompt[:500] + "...\n" if len(sample_prompt) > 500 else␣
 ↪sample_prompt + "\n")
```

```
2025-12-14 06:29:00 - INFO - Loading data using src.data loader…
2025-12-14 06:29:00 - INFO - Loaded 260 examples
2025-12-14 06:29:00 - INFO - Split: 234 train, 26 validation
2025-12-14 06:29:00 - WARNING - Validation set very small (26 examples).
Consider increasing val_ratio for more reliable evaluation.
2025-12-14 06:29:00 - INFO -
=== Sample Training Example ===

### Instruction:
You are a Vietnamese Named Entity Recognition (NER) expert. Extract named
entities from the given text and classify them into three categories:
- person: Names of people
- organizations: Names of organizations, companies, institutions
- address: Location names, addresses

Return your answer as a JSON object with these three keys. Each value should be
a list of strings. If a category has no entities, return an empty list. Do not
invent entities that are not present in the text.

…
```

1.4 5. Model Setup

Load the base model with 4-bit quantization and prepare it for QLoRA training.

[4]:
```python
# Configure 4-bit quantization
bnb_config = BitsAndBytesConfig(
    load_in_4bit=config.use_4bit,
    bnb_4bit_quant_type=config.bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=getattr(torch, config.bnb_4bit_compute_dtype),
    bnb_4bit_use_double_quant=config.use_nested_quant,
)

logger.info("Loading tokenizer...")

# Ministral models may have tokenizer backend issues
# Try multiple methods with graceful fallbacks
tokenizer = None
methods_tried = []

# Method 1: Try MistralTokenizerFast (recommended for Ministral)
try:
    from transformers import MistralTokenizerFast
    logger.info("Method 1: Attempting MistralTokenizerFast...")
    tokenizer = MistralTokenizerFast.from_pretrained(
        config.model_name,
        token=config.hf_token,
```

```
    )
    logger.info("  Successfully loaded MistralTokenizerFast")
except Exception as e:
    methods_tried.append(f"MistralTokenizerFast: {str(e)[:50]}")
    logger.warning(f"MistralTokenizerFast failed: {e}")

# Method 2: Try slow MistralTokenizer
if tokenizer is None:
    try:
        from transformers import MistralTokenizer
        logger.info("Method 2: Attempting MistralTokenizer (slow)...")
        tokenizer = MistralTokenizer.from_pretrained(
            config.model_name,
            token=config.hf_token,
        )
        logger.info("  Successfully loaded MistralTokenizer")
    except Exception as e:
        methods_tried.append(f"MistralTokenizer: {str(e)[:50]}")
        logger.warning(f"MistralTokenizer failed: {e}")

# Method 3: Try PreTrainedTokenizerFast directly with Mistral-7B vocab
if tokenizer is None:
    try:
        logger.info("Method 3: Attempting standard Mistral-7B tokenizer...")
        # Use a standard Mistral model's tokenizer (compatible with Ministral)
        tokenizer = AutoTokenizer.from_pretrained(
            "mistralai/Mistral-7B-Instruct-v0.3",
            token=config.hf_token,
        )
        logger.info("  Successfully loaded Mistral-7B tokenizer (compatible␣
 ↪with Ministral)")
        logger.info("Note: Using Mistral-7B tokenizer for Ministral-3-14B (they␣
 ↪share the same vocabulary)")
    except Exception as e:
        methods_tried.append(f"Mistral-7B tokenizer: {str(e)[:50]}")
        logger.warning(f"Mistral-7B tokenizer failed: {e}")

# Method 4: AutoTokenizer without trust_remote_code
if tokenizer is None:
    try:
        logger.info("Method 4: Attempting AutoTokenizer (no trust_remote_code)..
 ↪.")
        tokenizer = AutoTokenizer.from_pretrained(
            config.model_name,
            token=config.hf_token,
            trust_remote_code=False,
            use_fast=False,
```

```
        )
        logger.info("  Successfully loaded AutoTokenizer")
    except Exception as e:
        methods_tried.append(f"AutoTokenizer: {str(e)[:50]}")
        logger.error(f"AutoTokenizer failed: {e}")

if tokenizer is None:
    error_msg = "All tokenizer loading methods failed:\n" + "\n".join(f"  -␣
 ↪{m}" for m in methods_tried)
    logger.error(error_msg)
    raise RuntimeError(error_msg)

# Set padding token (required for batching)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
    tokenizer.pad_token_id = tokenizer.eos_token_id

logger.info(f"Tokenizer vocab size: {len(tokenizer)}")
logger.info(f"Padding token: {tokenizer.pad_token}")

logger.info("Loading base model with 4-bit quantization...")
model = AutoModelForCausalLM.from_pretrained(
    config.model_name,
    quantization_config=bnb_config,
    device_map="auto",
    token=config.hf_token,
    trust_remote_code=True,
    torch_dtype=torch.bfloat16,
)

# Prepare model for k-bit training
model = prepare_model_for_kbit_training(
    model,
    use_gradient_checkpointing=True,  # Save memory at cost of speed
)

logger.info("Model loaded successfully")
logger.info(f"Model dtype: {model.dtype}")
logger.info(f"Model device: {next(model.parameters()).device}")
```

```
2025-12-14 06:29:00 - INFO - Loading tokenizer…
2025-12-14 06:29:00 - WARNING - MistralTokenizerFast failed: cannot import name
'MistralTokenizerFast' from 'transformers'
(/home/nabang1010/anaconda3/envs/a2a/lib/python3.12/site-
packages/transformers/__init__.py)
2025-12-14 06:29:00 - WARNING - MistralTokenizer failed: cannot import name
'MistralTokenizer' from 'transformers'
```

```
(/home/nabang1010/anaconda3/envs/a2a/lib/python3.12/site-
packages/transformers/__init__.py)
2025-12-14 06:29:00 - INFO - Method 3: Attempting standard Mistral-7B
tokenizer…
2025-12-14 06:29:01 - INFO -   Successfully loaded Mistral-7B tokenizer
(compatible with Ministral)
2025-12-14 06:29:01 - INFO - Note: Using Mistral-7B tokenizer for
Ministral-3-14B (they share the same vocabulary)
2025-12-14 06:29:01 - INFO - Tokenizer vocab size: 32768
2025-12-14 06:29:01 - INFO - Padding token: </s>
2025-12-14 06:29:01 - INFO - Loading base model with 4-bit quantization…
`torch_dtype` is deprecated! Use `dtype` instead!
2025-12-14 06:29:02 - INFO - We will use 90% of the memory on device 0 for
storing the model, and 10% for the buffer to avoid OOM. You can set `max_memory`
in to a higher value to use more memory (at your own risk).

Loading checkpoint shards:   0%|          | 0/3 [00:00<?, ?it/s]

2025-12-14 06:29:15 - INFO - Model loaded successfully
2025-12-14 06:29:15 - INFO - Model dtype: torch.float32
2025-12-14 06:29:15 - INFO - Model device: cuda:0
```

1.5 5.5 Verify Target Modules (Critical Check)

Before applying LoRA, verify that the target modules exist in the Ministral architecture.

[ ]:
```python
# Verify target modules exist in the model
logger.info("Verifying LoRA target modules in model architecture...")
logger.info(f"Configured target modules: {config.lora_target_modules}")

# Find all projection and gate modules
projection_modules = set()
for name, module in model.named_modules():
    if any(keyword in name.lower() for keyword in ['proj', 'gate', 'mlp',␣
 ↪'attention']):
        # Extract the layer-independent name
        parts = name.split('.')
        if len(parts) > 0:
            module_name = parts[-1]
            projection_modules.add(module_name)

logger.info(f"\nFound projection/gate modules in model:␣
 ↪{sorted(projection_modules)}")

# Check if configured modules exist
missing_modules = []
for target_module in config.lora_target_modules:
    if target_module not in projection_modules:
        missing_modules.append(target_module)
```

```
if missing_modules:
    logger.warning(f"\n  WARNING: These target modules don't exist in the model:
 ↪ {missing_modules}")
    logger.warning("LoRA will NOT be applied to these modules!")
    logger.warning("Consider updating config.lora_target_modules to match␣
 ↪actual model architecture.")
else:
    logger.info("\n All target modules verified successfully!")

# Log actual modules that will be targeted
actual_targets = [m for m in config.lora_target_modules if m in␣
 ↪projection_modules]
logger.info(f"\nModules that will have LoRA adapters: {actual_targets}")
logger.info(f"Number of adapter layers: {len(actual_targets)}")
```

```
2025-12-14 06:29:15 - INFO - Verifying LoRA target modules in model
architecture…
2025-12-14 06:29:15 - INFO - Configured target modules: ['q_proj', 'k_proj',
'v_proj', 'o_proj', 'gate_proj', 'up_proj', 'down_proj']
2025-12-14 06:29:15 - INFO -
Found projection/gate modules in model: ['act_fn', 'down_proj', 'gate_proj',
'k_proj', 'mlp', 'o_proj', 'post_attention_layernorm', 'q_proj', 'up_proj',
'v_proj']
2025-12-14 06:29:15 - INFO -
 All target modules verified successfully!
2025-12-14 06:29:15 - INFO -
Modules that will have LoRA adapters: ['q_proj', 'k_proj', 'v_proj', 'o_proj',
'gate_proj', 'up_proj', 'down_proj']
2025-12-14 06:29:15 - INFO - Number of adapter layers: 7
```

1.6 6. LoRA Configuration

Apply Low-Rank Adaptation (LoRA) to enable efficient fine-tuning.

[6]:
```
# Configure LoRA
lora_config = LoraConfig(
    r=config.lora_r,
    lora_alpha=config.lora_alpha,
    target_modules=config.lora_target_modules,
    lora_dropout=config.lora_dropout,
    bias="none",
    task_type="CAUSAL_LM",
    inference_mode=False,
)

logger.info("Applying LoRA adapters...")
```

```
model = get_peft_model(model, lora_config)

# Print trainable parameters
trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
all_params = sum(p.numel() for p in model.parameters())
trainable_percent = 100 * trainable_params / all_params

logger.info(f"\n{'='*60}")
logger.info(f"Trainable parameters: {trainable_params:,}")
logger.info(f"Total parameters: {all_params:,}")
logger.info(f"Trainable %: {trainable_percent:.2f}%")
logger.info(f"{'='*60}\n")

model.print_trainable_parameters()
```

```
2025-12-14 06:29:15 - INFO - Applying LoRA adapters…
2025-12-14 06:29:17 - INFO -
============================================================
2025-12-14 06:29:17 - INFO - Trainable parameters: 167,772,160
2025-12-14 06:29:17 - INFO - Total parameters: 3,926,134,784
2025-12-14 06:29:17 - INFO - Trainable %: 4.27%
2025-12-14 06:29:17 - INFO -
============================================================


trainable params: 167,772,160 || all params: 7,415,795,712 || trainable%: 2.2624
```

1.7 7. Prepare Datasets

[7]:
```
# Create tokenized datasets
train_dataset = NERDataset(train_examples, tokenizer, config.max_length)
val_dataset = NERDataset(val_examples, tokenizer, config.max_length)

logger.info(f"Training dataset size: {len(train_dataset)}")
logger.info(f"Validation dataset size: {len(val_dataset)}")

# Calculate training statistics
effective_batch_size = config.per_device_train_batch_size * config.
 ↪gradient_accumulation_steps
total_steps = (len(train_dataset) // effective_batch_size) * config.
 ↪num_train_epochs
eval_steps_per_epoch = len(train_dataset) // effective_batch_size // (config.
 ↪eval_steps + 1)

logger.info(f"\n{'='*60}")
logger.info(f"Effective batch size: {effective_batch_size}")
logger.info(f"Total training steps: {total_steps}")
```

```
logger.info(f"Evaluations per epoch: ~{eval_steps_per_epoch}")
logger.info(f"{'='*60}\n")
```

```
2025-12-14 06:29:17 - INFO - Created dataset with 234 examples
2025-12-14 06:29:17 - INFO - Created dataset with 26 examples
2025-12-14 06:29:17 - INFO - Training dataset size: 234
2025-12-14 06:29:17 - INFO - Validation dataset size: 26
2025-12-14 06:29:17 - INFO -
============================================================
2025-12-14 06:29:17 - INFO - Effective batch size: 8
2025-12-14 06:29:17 - INFO - Total training steps: 290
2025-12-14 06:29:17 - INFO - Evaluations per epoch: ~0
2025-12-14 06:29:17 - INFO -
============================================================
```

1.8 8.5 Evaluation Metrics with Seqeval

Add proper NER evaluation metrics: precision, recall, and F1 per entity type.

1.9 8.6 Inference Helper Function

Define the prediction function that will be used for evaluation and testing.

[8]:
```
def generate_predictions(
    model,
    tokenizer,
    text: str,
    max_new_tokens: int = 512,
    temperature: float = 0.0,  # Use greedy decoding for structured output
    top_p: float = 0.9,
) -> Dict:
    """
    Generate NER predictions for a given text with robust JSON parsing.

    Args:
        model: Fine-tuned model
        tokenizer: Tokenizer
        text: Input text
        max_new_tokens: Maximum tokens to generate
        temperature: Sampling temperature (0.0 for deterministic)
        top_p: Nucleus sampling parameter

    Returns:
        Dictionary with extracted entities
    """
    # Validate input
    if not text or not isinstance(text, str):
```

```
        logger.error("Invalid input text")
        return {"error": "Invalid input", "person": [], "organizations": [],␣
↪"address": []}

    # Build prompt without response
    prompt = build_training_prompt({"text": text}, include_response=False)

    # Tokenize
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    # Generate
    model.eval()
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            temperature=temperature,
            do_sample=temperature > 0,
            top_p=top_p if temperature > 0 else None,
            pad_token_id=tokenizer.pad_token_id,
            eos_token_id=tokenizer.eos_token_id,
        )

    # Decode
    generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Extract response
    if "### Response:" in generated_text:
        response_text = generated_text.split("### Response:")[-1].strip()

        # Use robust parsing with fallback to regex
        entities = parse_entities_from_json(response_text)
        return entities

    logger.warning("No response marker found in generated text")
    return {"error": "No response generated", "person": [], "organizations":␣
↪[], "address": []}


logger.info("Inference helper function defined")
```

```
2025-12-14 06:29:17 - INFO - Inference helper function defined
```

[9]:
```
def parse_entities_from_json(json_str: str) -> Dict[str, List[str]]:
    """
    Parse entities from JSON string with fallback to regex extraction.
```

```
    This is a wrapper around the utility function for backward compatibility.
    Uses robust parsing that handles:
    - Markdown code blocks (```)
    - Extra text before/after JSON
    - Key variations (organizations vs organization)

    Args:
        json_str: JSON string containing entities (may include markdown)

    Returns:
        Dictionary with entity categories and lists
    """
    # Use the robust utility function
    return parse_ner_response(json_str)


def compute_ner_metrics(predictions: List[Dict], references: List[Dict]) ->␣
 ↪Dict[str, float]:
    """
    Compute NER metrics using seqeval format.

    Args:
        predictions: List of predicted entity dictionaries
        references: List of ground truth entity dictionaries

    Returns:
        Dictionary with precision, recall, F1 scores
    """
    # Convert to seqeval format: list of lists of entity labels
    pred_labels = []
    true_labels = []

    for pred, ref in zip(predictions, references):
        pred_entities = []
        true_entities = []

        # Aggregate all entity types for this example
        for entity_type in ["person", "organizations", "address"]:
            pred_ents = pred.get(entity_type, [])
            true_ents = ref.get(entity_type, [])

            # Add labels in BIO format for seqeval
            pred_entities.extend([f"B-{entity_type.upper()}" for _ in␣
 ↪pred_ents])
            true_entities.extend([f"B-{entity_type.upper()}" for _ in␣
 ↪true_ents])
```

```
        # If empty, add "O" (outside)
        if not pred_entities:
            pred_entities = ["O"]
        if not true_entities:
            true_entities = ["O"]

        pred_labels.append(pred_entities)
        true_labels.append(true_entities)

    # Compute metrics
    try:
        precision = precision_score(true_labels, pred_labels)
        recall = recall_score(true_labels, pred_labels)
        f1 = f1_score(true_labels, pred_labels)

        return {
            "precision": precision,
            "recall": recall,
            "f1": f1,
        }
    except Exception as e:
        logger.warning(f"Failed to compute seqeval metrics: {e}")
        return {
            "precision": 0.0,
            "recall": 0.0,
            "f1": 0.0,
        }


def compute_metrics_callback(eval_pred: EvalPrediction) -> Dict[str, float]:
    """
    Custom compute_metrics function for Trainer.

    Note: For causal LM training, detailed entity extraction during training
    is expensive. This is a placeholder that returns loss-based metrics.
    Use the separate evaluation script for full NER metrics.

    Args:
        eval_pred: EvalPrediction object with predictions and label_ids

    Returns:
        Dictionary with metrics
    """
    # For now, we rely on eval_loss computed by Trainer
    # Full entity-level evaluation should be done separately post-training
    return {}
```

```
logger.info("Evaluation metrics functions defined")
logger.info("Note: Full entity-level F1/precision/recall will be computed␣
 ↪post-training")
```

```
2025-12-14 06:29:17 - INFO - Evaluation metrics functions defined
2025-12-14 06:29:17 - INFO - Note: Full entity-level F1/precision/recall will be
computed post-training
```

1.10 8. Training Configuration

[10]:
```
# Define training arguments
training_args = TrainingArguments(
    # Output configuration
    output_dir=str(config.output_dir),
    overwrite_output_dir=False,

    # Training configuration
    num_train_epochs=config.num_train_epochs,
    per_device_train_batch_size=config.per_device_train_batch_size,
    per_device_eval_batch_size=config.per_device_eval_batch_size,
    gradient_accumulation_steps=config.gradient_accumulation_steps,
    gradient_checkpointing=True,

    # Optimization configuration
    learning_rate=config.learning_rate,
    weight_decay=config.weight_decay,
    optim=config.optim,
    lr_scheduler_type=config.lr_scheduler_type,
    warmup_ratio=config.warmup_ratio,
    max_grad_norm=config.max_grad_norm,  # Gradient clipping (now 1.0, standard␣
 ↪value)

    # Precision configuration
    bf16=config.bf16,
    fp16=False,

    # Logging configuration
    logging_dir=str(config.output_dir / "logs"),
    logging_strategy="steps",
    logging_steps=config.logging_steps,
    logging_first_step=True,

    # Evaluation configuration (use eval_strategy instead of␣
 ↪evaluation_strategy for newer transformers)
    eval_strategy="steps",  # Changed from evaluation_strategy
    eval_steps=config.eval_steps,
```

```
    eval_accumulation_steps=config.eval_accumulation_steps,  # Consistent with␣
 ↪training

    # Checkpointing configuration
    save_strategy="steps",
    save_steps=config.save_steps,
    save_total_limit=config.save_total_limit,
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False,

    # Other configuration
    report_to="tensorboard",
    disable_tqdm=False,
    remove_unused_columns=False,
    dataloader_pin_memory=True,
    dataloader_num_workers=2,
    seed=config.seed,
)

logger.info("Training arguments configured successfully")
```

```
2025-12-14 06:29:17 - INFO - Training arguments configured successfully
```

1.11 9. Custom Callbacks

Add custom callbacks for better monitoring.

[11]:
```
class MemoryCallback(TrainerCallback):
    """Callback to log GPU memory usage."""

    def on_step_end(self, args, state, control, **kwargs):
        if state.global_step % args.logging_steps == 0 and torch.cuda.
 ↪is_available():
            memory_allocated = torch.cuda.memory_allocated() / 1e9
            memory_reserved = torch.cuda.memory_reserved() / 1e9
            logger.info(
                f"GPU Memory - Allocated: {memory_allocated:.2f} GB, "
                f"Reserved: {memory_reserved:.2f} GB"
            )


class SampleGenerationCallback(TrainerCallback):
    """Callback to generate sample predictions during training."""

    def __init__(self, tokenizer, sample_text: str, eval_every: int = 500):
        self.tokenizer = tokenizer
```

```
        self.sample_text = sample_text
        self.eval_every = eval_every

    def on_step_end(self, args, state, control, model, **kwargs):
        if state.global_step % self.eval_every == 0 and state.global_step > 0:
            logger.info("\n" + "="*60)
            logger.info("Generating sample prediction...")

            # Create prompt without response
            prompt = build_training_prompt({"text": self.sample_text},␣
 ↪include_response=False)
            inputs = self.tokenizer(prompt, return_tensors="pt").to(model.
 ↪device)

            # Generate
            model.eval()
            with torch.no_grad():
                outputs = model.generate(
                    **inputs,
                    max_new_tokens=256,
                    temperature=0.7,
                    do_sample=True,
                    top_p=0.9,
                    pad_token_id=self.tokenizer.pad_token_id,
                    eos_token_id=self.tokenizer.eos_token_id,
                )

            generated_text = self.tokenizer.decode(outputs[0],␣
 ↪skip_special_tokens=True)

            # Extract only the response part
            if "### Response:" in generated_text:
                response = generated_text.split("### Response:")[-1].strip()
                logger.info(f"Generated Response:\n{response[:500]}")

            model.train()
            logger.info("="*60 + "\n")


# Prepare sample for generation callback
sample_text = val_examples[0]['text'] if val_examples else␣
 ↪train_examples[0]['text']

callbacks = [
    MemoryCallback(),
    SampleGenerationCallback(tokenizer, sample_text, eval_every=500),
]
```

1.12 10. Initialize Trainer

[12]:
```
# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    callbacks=callbacks,
)

logger.info("Trainer initialized successfully")
logger.info(f"Ready to start training with {len(train_dataset)} examples")
```

```
2025-12-14 06:29:17 - INFO - Trainer initialized successfully
2025-12-14 06:29:17 - INFO - Ready to start training with 234 examples
```

1.13 11. Train the Model

Start the training process. Monitor the progress through logs and TensorBoard.

[13]:
```
# Start training
logger.info("\n" + "="*60)
logger.info("Starting training...")
logger.info("="*60 + "\n")

try:
    train_result = trainer.train()

    # Log training results
    logger.info("\n" + "="*60)
    logger.info("Training completed successfully!")
    logger.info(f"Training loss: {train_result.training_loss:.4f}")
    logger.info(f"Training steps: {train_result.global_step}")
    logger.info("="*60 + "\n")

    # Save metrics
    metrics = train_result.metrics
    trainer.log_metrics("train", metrics)
    trainer.save_metrics("train", metrics)

except KeyboardInterrupt:
    logger.warning("Training interrupted by user")
except Exception as e:
    logger.error(f"Training failed with error: {e}")
    raise
```

```
2025-12-14 06:29:17 - INFO -
============================================================
```

```
2025-12-14 06:29:17 - INFO - Starting training…
2025-12-14 06:29:17 - INFO -
============================================================

`use_cache=True` is incompatible with gradient checkpointing. Setting
`use_cache=False`.

<IPython.core.display.HTML object>

2025-12-14 06:34:05 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 06:38:55 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 06:43:21 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 06:48:10 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 06:52:59 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 06:57:26 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:02:14 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:07:03 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:11:30 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:16:19 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:21:36 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:26:03 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:30:52 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:35:40 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:40:07 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:44:56 - INFO - GPU Memory - Allocated: 5.38 GB, Reserved: 8.44 GB
2025-12-14 07:48:03 - WARNING - Training interrupted by user
```

1.14 12. Evaluate the Model

[14]:
```python
# Evaluate on validation set
logger.info("Evaluating model on validation set...")
eval_results = trainer.evaluate()

logger.info("\n" + "="*60)
logger.info("Evaluation Results:")
for key, value in eval_results.items():
    logger.info(f"{key}: {value:.4f}")
logger.info("="*60 + "\n")

# Save evaluation results
trainer.log_metrics("eval", eval_results)
trainer.save_metrics("eval", eval_results)
```

```
2025-12-14 07:48:08 - INFO - Evaluating model on validation set…
2025-12-14 07:48:36 - INFO -
============================================================
2025-12-14 07:48:36 - INFO - Evaluation Results:
2025-12-14 07:48:36 - INFO - eval_loss: nan
2025-12-14 07:48:36 - INFO -
```

```
============================================================


***** eval metrics *****
  eval_loss = nan
```

1.15 13. Save LoRA Adapters

[15]:
```python
# Save LoRA adapters
adapter_dir = config.output_dir / "final-adapters"
adapter_dir.mkdir(parents=True, exist_ok=True)

logger.info(f"Saving LoRA adapters to {adapter_dir}")
model.save_pretrained(adapter_dir)
tokenizer.save_pretrained(adapter_dir)

logger.info("LoRA adapters saved successfully")
logger.info(f"Adapter size: {sum(f.stat().st_size for f in adapter_dir.
 ↪rglob('*') if f.is_file()) / 1e6:.2f} MB")
```

```
2025-12-14 07:48:36 - INFO - Saving LoRA adapters to
checkpoints/mistral-7B-Instruct-v0.3-ner-qlora/final-adapters
2025-12-14 07:48:37 - INFO - LoRA adapters saved successfully
2025-12-14 07:48:37 - INFO - Adapter size: 675.56 MB
```

1.16 14. Test Inference

Test the fine-tuned model with sample inputs.

[16]:
```python
# Note: generate_predictions() function was already defined in section 8.6
# It's available for use here

# Test with validation examples
logger.info("\n" + "="*60)
logger.info("Testing inference on sample examples...")
logger.info("="*60 + "\n")

num_samples = min(3, len(val_examples))
for i in range(num_samples):
    example = val_examples[i]

    logger.info(f"\nExample {i+1}:")
    logger.info(f"Input text: {example['text'][:200]}...")

    # Generate prediction
    prediction = generate_predictions(model, tokenizer, example['text'])

    logger.info(f"\nPrediction:")
```

```
    print(json.dumps(prediction, ensure_ascii=False, indent=2))

    if 'ground_truth' in example:
        logger.info(f"\nGround Truth:")
        print(json.dumps(example['ground_truth'], ensure_ascii=False, indent=2))

    logger.info("\n" + "-"*60)
```

```
2025-12-14 07:48:37 - INFO -
============================================================
2025-12-14 07:48:37 - INFO - Testing inference on sample examples…
2025-12-14 07:48:37 - INFO -
============================================================

2025-12-14 07:48:37 - INFO -
Example 1:
2025-12-14 07:48:37 - INFO - Input text: Công nghệ Hawk-Eye hay còn được biết
đến là mắt thần sẽ được BTC giải ATP Next Gen Finals sử dụng cho tất cả các tình
huống diễn ra trên sân, qua đó loại bỏ hoàn toàn các vị trọng tài dây. Giải sẽ
đượ…
The following generation flags are not valid and may be ignored:
['temperature']. Set `TRANSFORMERS_VERBOSITY=info` for more details.
2025-12-14 07:48:59 - INFO -
Prediction:
2025-12-14 07:48:59 - INFO -
Ground Truth:
2025-12-14 07:48:59 - INFO -
------------------------------------------------------------
2025-12-14 07:48:59 - INFO -
Example 2:
2025-12-14 07:48:59 - INFO - Input text: NDĐT - Với mục tiêu bảo đảm khai thác
hiệu quả, nâng cao chất lượng cơ sở vật chất phục vụ giáo dục thể chất, trường
THPT chuyên Hà Nội -Amsterdam đã báo cáo với Sở Giáo dục và Đào tạo (GD-ĐT) Hà
Nội …
{
  "person": [],
  "organizations": [
    "Hawk-Eye"
  ],
  "address": [
    "Milan"
  ]
}
{
  "person": [],
  "organizations": [],
  "address": [
```

```
    "Milan"
  ]
}
```


```
2025-12-14 07:49:39 - INFO -
Prediction:
2025-12-14 07:49:39 - INFO -
Ground Truth:
2025-12-14 07:49:39 - INFO -
------------------------------------------------------------
2025-12-14 07:49:39 - INFO -
Example 3:
2025-12-14 07:49:39 - INFO - Input text: Làm giả giấy tờ, hợp đồng mua bán thẻ
cào để tạo lòng tin, nữ giám đốc lừa đảo nhiều người chiếm đoạt hàng trăm tỉ
đồng. Ngày 22-9, TAND TP HCM mở phiên tòa sơ thẩm xét xử bị cáo Lã Thị Thanh (44
tuổi…
```


```
{
  "person": [
    "Nguyễn Đình Vinh",
    "MINH PHƯƠNG"
  ],
  "organizations": [],
  "address": [
    "Hà Nội",
    "Amsterdam",
    "TP Hà Nội"
  ]
}
{
  "person": [
    "Nguyễn Đình Vinh",
    "MINH PHƯƠNG"
  ],
  "organizations": [
    "NDĐT"
  ],
  "address": [
    "Amsterdam",
    "thành phố Hà Nội",
    "quận Cầu Giấy",
    "TP. Hà Nội",
    "Hà Nội -Amsterdam",
    "Hà Nội",
    "Hà Nội Amsterdam",
    "TP Hà Nội",
    "Hà Nội - Amsterdam",
    "phường Trung Hòa"
  ]
```

```
}
2025-12-14 07:50:07 - INFO -
Prediction:
2025-12-14 07:50:07 - INFO -
Ground Truth:
2025-12-14 07:50:07 - INFO -
------------------------------------------------------------

{
  "person": [
    "Lã Thị Thanh",
    "Nguyễn Quang Đức",
    "Đức",
    "Quang Anh",
    "Nguyễn Quang Đức",
    "Thanh",
    "Quoc Chiến"
  ],
  "organizations": [
    "Công ty Petechland Jsc",
    "Công ty TMDV Quang Anh",
    "Viettel",
    "Công ty TNHH Khánh Linh"
  ],
  "address": [
    "TP HCM",
    "quận 1",
    "đường Trần Hưng Đạo",
    "phường Phạm Ngũ Lão",
    "Ninh Bình",
    "thành phố Hà Nội"
  ]
}
{
  "person": [
    "Lã Thị Thanh",
    "Thanh",
    "Nguyễn Quang Đức",
    "Đức",
    "Quốc Chiến",
    "Khánh Linh"
  ],
  "organizations": [
    "Công ty Petechland Jsc",
    "Công ty TMDV Quang Anh",
    "công ty Quang Anh",
    "Viettel"
  ],
```

```
  "address": [
    "quận 1",
    "TP HCM",
    "đường Trần Hưng Đạo",
    "phường Phạm Ngũ Lão",
    "Ninh Bình"
  ]
}
```