```
#
======================================================================
========
# COMPLETE T5 FINE-TUNING & EVALUATION PIPELINE FOR STEM SUMMARIZATION
#
======================================================================
========
# This notebook implements:
# 1. Data preparation from ScienceQA
# 2. T5 + LoRA fine-tuning on STEM summarization
# 3. Multi-format outputs (summaries, flashcards, notes)
# 4. Comprehensive evaluation (ROUGE, BERTScore, Readability)
#
======================================================================
========

#
======================================================================
========
# SECTION 1: SETUP & INSTALLATION
#
======================================================================
========
print("□ Step 1: Installing required libraries...")
!pip install -q transformers datasets accelerate peft
!pip install -q torch torchvision rouge-score bert-score nltk textstat
!pip install -q pandas matplotlib seaborn
print("□ Libraries installed successfully!")

#
======================================================================
========
# SECTION 2: IMPORT LIBRARIES
#
======================================================================
========
print("\n□ Step 2: Importing libraries...")
import torch
from torch.utils.data import Dataset, DataLoader
import pandas as pd
import numpy as np
import os
import json
from tqdm.auto import tqdm
from datasets import load_dataset
from transformers import (
    T5ForConditionalGeneration,
    T5Tokenizer,
    get_linear_schedule_with_warmup
)
from peft import LoraConfig, get_peft_model, TaskType, PeftModel
from rouge_score import rouge_scorer
from bert_score import score as bert_score
import textstat
import matplotlib.pyplot as plt

# Set up directories
os.makedirs("training_outputs", exist_ok=True)
os.makedirs("evaluation_results", exist_ok=True)
os.makedirs("model_checkpoints", exist_ok=True)

torch.manual_seed(42)
print("□ Libraries imported successfully!")

#
======================================================================
========
# SECTION 3: DATA PREPARATION
#
======================================================================
========
print("\n□ Step 3: Preparing ScienceQA dataset...")

class STEMSummarizationDataset(Dataset):
    """Dataset for STEM text summarization"""

    def __init__(self, split='train', max_samples=500):
        print(f"Loading {split} split from ScienceQA...")

        dataset = load_dataset("derek-thomas/ScienceQA", split=split,
streaming=True)
        self.samples = []

        for idx, sample in enumerate(tqdm(dataset, desc="Loading
samples", total=max_samples)):
            if idx >= max_samples:
                break

            # Extract text content
            question = sample.get('question', '')
            hint = sample.get('hint', '')
            lecture = sample.get('lecture', '')
            solution = sample.get('solution', '')

            if not any([question, lecture, solution]):
                continue

            # Create input text
            input_parts = []
            if lecture:
                input_parts.append(f"Context: {lecture}")
            if question:
                input_parts.append(f"Question: {question}")
            if hint:
                input_parts.append(f"Hint: {hint}")
            if solution:
                input_parts.append(f"Explanation: {solution}")

            input_text = " ".join(input_parts)

            # Create summary (target)
            summary_parts = []
            if lecture:
                summary_parts.append(lecture[:150])
            if solution:
                summary_parts.append(solution[:150])
            summary = " ".join(summary_parts[:2])

            if input_text and summary:
                self.samples.append({
                    'input_text': input_text,
                    'summary': summary,
                    'question': question,
                    'solution': solution
                })

        print(f"□ Loaded {len(self.samples)} samples")

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        return self.samples[idx]

# Load datasets
train_dataset = STEMSummarizationDataset(split='train',
max_samples=500)
val_dataset = STEMSummarizationDataset(split='validation',
max_samples=100)

print(f"Training samples: {len(train_dataset)}")
print(f"Validation samples: {len(val_dataset)}")

#
======================================================================
========
# SECTION 4: TOKENIZATION
#
======================================================================
========
print("\n Step 4: Setting up tokenization...")

model_name = "t5-base"
tokenizer = T5Tokenizer.from_pretrained(model_name)

def prepare_batch(batch):
    """Prepare batch for training"""
    input_texts = [f"summarize: {item['input_text']}" for item in
batch]
    target_texts = [item['summary'] for item in batch]

    # Tokenize
    inputs = tokenizer(
        input_texts,
        max_length=512,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    targets = tokenizer(
        target_texts,
        max_length=150,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    # Replace padding with -100 for loss calculation
    targets['input_ids'][targets['input_ids'] ==
tokenizer.pad_token_id] = -100
    inputs['labels'] = targets['input_ids']

    return inputs

# Create dataloaders
BATCH_SIZE = 4
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=prepare_batch)
val_dataloader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False, collate_fn=prepare_batch)

print(" Tokenization setup complete!")

#
======================================================================
========
# SECTION 5: MODEL SETUP WITH LoRA
#
```

```
======================================================================
========
print("\n□ Step 5: Setting up T5 model with LoRA...")

# Load model
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Configure LoRA
lora_config = LoraConfig(
    task_type=TaskType.SEQ_2_SEQ_LM,
    r=8,
    lora_alpha=16,
    lora_dropout=0.1,
    target_modules=["q", "v"]
)

model = get_peft_model(model, lora_config)
model.print_trainable_parameters()

device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
print(f"□ Model ready on {device}")

#
======================================================================
========
# SECTION 6: TRAINING SETUP
#
======================================================================
========
print("\n□ Step 6: Setting up training...")

NUM_EPOCHS = 100
LEARNING_RATE = 5e-4
GRADIENT_ACCUMULATION_STEPS = 4

optimizer = torch.optim.AdamW(model.parameters(), lr=LEARNING_RATE,
weight_decay=0.01)
total_steps = len(train_dataloader) * NUM_EPOCHS
lr_scheduler = get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=100, num_training_steps=total_steps)

print("□ Training setup complete!")

#
======================================================================
========
# SECTION 7: TRAINING LOOP
#
======================================================================
========
print("\n Step 7: Starting training...")

def train_epoch(epoch):
    model.train()
    total_loss = 0
    optimizer.zero_grad()

    progress_bar = tqdm(train_dataloader, desc=f"Epoch
{epoch+1}/{NUM_EPOCHS}")

    for step, batch in enumerate(progress_bar):
        batch = {k: v.to(device) for k, v in batch.items()}

        outputs = model(**batch)
        loss = outputs.loss / GRADIENT_ACCUMULATION_STEPS
        loss.backward()

        if (step + 1) % GRADIENT_ACCUMULATION_STEPS == 0:
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()

        total_loss += loss.item() * GRADIENT_ACCUMULATION_STEPS
        progress_bar.set_postfix({'loss': loss.item() *
GRADIENT_ACCUMULATION_STEPS})

    return total_loss / len(train_dataloader)

def evaluate():
    model.eval()
    total_loss = 0

    with torch.no_grad():
        for batch in tqdm(val_dataloader, desc="Evaluating"):
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            total_loss += outputs.loss.item()

    return total_loss / len(val_dataloader)

# Training loop
train_losses = []
val_losses = []
best_val_loss = float('inf')

for epoch in range(NUM_EPOCHS):
    train_loss = train_epoch(epoch)
    val_loss = evaluate()
    train_losses.append(train_loss)
    val_losses.append(val_loss)

    print(f"\nEpoch {epoch+1}: Train Loss = {train_loss:.4f}, Val Loss
= {val_loss:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        model.save_pretrained("model_checkpoints/best_model")
        tokenizer.save_pretrained("model_checkpoints/best_model")
        print(f"□ Best model saved!")

    if torch.cuda.is_available():
        torch.cuda.empty_cache()

print("\n□ Training complete!")

# Plot training curves
plt.figure(figsize=(10, 5))
plt.plot(range(1, NUM_EPOCHS+1), train_losses, marker='o',
label='Train Loss')
plt.plot(range(1, NUM_EPOCHS+1), val_losses, marker='s', label='Val
Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Progress')
plt.legend()
plt.grid(True)
plt.savefig('evaluation_results/training_loss.png')
plt.show()

#
======================================================================
========
# SECTION 8: INFERENCE PIPELINE
#
======================================================================
========
print("\n□ Step 8: Setting up inference pipeline...")

# Load best model
inference_model =
T5ForConditionalGeneration.from_pretrained(model_name)
inference_model = PeftModel.from_pretrained(inference_model,
"model_checkpoints/best_model")
inference_model.to(device)
inference_model.eval()

def generate_summary(text, max_length=150):
    """Generate summary"""
    input_text = f"summarize: {text}"
    inputs = tokenizer(input_text, max_length=512, truncation=True,
return_tensors='pt').to(device)

    with torch.no_grad():
        outputs = inference_model.generate(
            **inputs,
            max_length=max_length,
            num_beams=4,
            temperature=0.7,
            do_sample=True,
            top_p=0.9,
            early_stopping=True
        )

    return tokenizer.decode(outputs[0], skip_special_tokens=True)

def generate_flashcard(text):
    """Generate Q&A flashcard"""
    input_text = f"create question and answer: {text}"
    inputs = tokenizer(input_text, max_length=512, truncation=True,
return_tensors='pt').to(device)

    with torch.no_grad():
        outputs = inference_model.generate(**inputs, max_length=100,
num_beams=3)

    return tokenizer.decode(outputs[0], skip_special_tokens=True)

def generate_study_notes(text):
    """Generate detailed study notes"""
    input_text = f"explain in detail: {text}"
    inputs = tokenizer(input_text, max_length=512, truncation=True,
return_tensors='pt').to(device)

    with torch.no_grad():
        outputs = inference_model.generate(**inputs, max_length=200,
num_beams=4)

    return tokenizer.decode(outputs[0], skip_special_tokens=True)

print("□ Inference pipeline ready!")

#
======================================================================
========
# SECTION 9: EVALUATION
#
======================================================================
========
print("\n□ Step 9: Comprehensive evaluation...")

# Initialize metrics
rouge = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'],
use_stemmer=True)

results = {
    'rouge1': [],
    'rouge2': [],
    'rougeL': [],
    'readability': [],
    'generated': [],
    'reference': []
}

# Evaluate on validation set
num_samples = min(50, len(val_dataset))

for idx in tqdm(range(num_samples), desc="Evaluating"):
    sample = val_dataset[idx]

    # Generate summary
    generated = generate_summary(sample['input_text'])
    reference = sample['summary']

    # Calculate ROUGE
    scores = rouge.score(reference, generated)
    results['rouge1'].append(scores['rouge1'].fmeasure)
    results['rouge2'].append(scores['rouge2'].fmeasure)
    results['rougeL'].append(scores['rougeL'].fmeasure)

    # Calculate readability
    try:
        grade_level = textstat.flesch_kincaid_grade(generated)
        results['readability'].append(grade_level)
    except:
        results['readability'].append(0)

    results['generated'].append(generated)
    results['reference'].append(reference)

    # Save example outputs
    if idx < 5:
        print(f"\n{'='*70}")
        print(f"Example {idx+1}")
        print(f"{'='*70}")
        print(f"\n□ SHORT SUMMARY:\n{generated}")
        print(f"\n★ FLASHCARD:\
n{generate_flashcard(sample['input_text'])}")
```

```
        print(f"\n□ STUDY NOTES:\
n{generate_study_notes(sample['input_text'])}")

# Calculate BERTScore
print("\nCalculating BERTScore...")
P, R, F1 = bert_score(results['generated'], results['reference'],
lang='en', verbose=False)

# Print results
print("\n" + "="*70)
print("EVALUATION RESULTS")
print("="*70)

print(f"\n□ ROUGE Scores:")
print(f"   ROUGE-1: {np.mean(results['rouge1']):.4f} ±
{np.std(results['rouge1']):.4f}")
print(f"   ROUGE-2: {np.mean(results['rouge2']):.4f} ±
{np.std(results['rouge2']):.4f}")
print(f"   ROUGE-L: {np.mean(results['rougeL']):.4f} ±
{np.std(results['rougeL']):.4f}")

print(f"\n□ BERTScore:")
print(f"   Precision: {P.mean():.4f}")
print(f"   Recall:    {R.mean():.4f}")
print(f"   F1:        {F1.mean():.4f}")

print(f"\n□ Readability:")
print(f"   Avg Grade Level: {np.mean(results['readability']):.2f}")
print(f"   (Target: 6-10 for middle/high school)")

# Save results
results_df = pd.DataFrame({
    'Generated': results['generated'],
    'Reference': results['reference'],
    'ROUGE-1': results['rouge1'],
    'ROUGE-2': results['rouge2'],
    'ROUGE-L': results['rougeL'],
    'Grade_Level': results['readability']
})
results_df.to_csv('evaluation_results/results.csv', index=False)

# Plot metrics
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# ROUGE scores
ax1.hist([results['rouge1'], results['rouge2'], results['rougeL']],
         label=['ROUGE-1', 'ROUGE-2', 'ROUGE-L'], alpha=0.7)
ax1.set_xlabel('Score')
ax1.set_ylabel('Frequency')
ax1.set_title('ROUGE Score Distribution')
```

```
ax1.legend()
ax1.grid(True, alpha=0.3)

# Readability
ax2.hist(results['readability'], bins=15, alpha=0.7, color='green')
ax2.axvline(8, color='red', linestyle='--', label='8th Grade Target')
ax2.set_xlabel('Grade Level')
ax2.set_ylabel('Frequency')
ax2.set_title('Readability Distribution')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('evaluation_results/metrics.png', dpi=150)
plt.show()

# Summary
summary = {
    'training': {
        'epochs': NUM_EPOCHS,
        'final_train_loss': train_losses[-1],
        'final_val_loss': val_losses[-1]
    },
    'evaluation': {
        'rouge1': float(np.mean(results['rouge1'])),
        'rouge2': float(np.mean(results['rouge2'])),
        'rougeL': float(np.mean(results['rougeL'])),
        'bertscore_f1': float(F1.mean()),
        'avg_grade_level': float(np.mean(results['readability']))
    }
}

with open('evaluation_results/summary.json', 'w') as f:
    json.dump(summary, f, indent=2)

print("\n□ Evaluation complete! Check 'evaluation_results/' folder.")
print("="*70)

□ Step 1: Installing required libraries...
  Preparing metadata (setup.py) ...
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.1/61.1 kB 2.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 176.4/176.4 kB 8.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 42.6 MB/s eta
0:00:00
porting libraries...
□ Libraries imported successfully!
```

```
 Step 3: Preparing ScienceQA dataset...
Loading train split from ScienceQA...

/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(

{"model_id":"2d578baf6d6848578d7797bbe7ab4d2d","version_major":2,"vers
ion_minor":0}

{"model_id":"373d9c0606a14f79a3cb32a24a09d54b","version_major":2,"vers
ion_minor":0}

 Loaded 495 samples
Loading validation split from ScienceQA...

{"model_id":"1dbedaec4ea94ec191c3e9d6365a2114","version_major":2,"vers
ion_minor":0}

 Loaded 97 samples
Training samples: 495
Validation samples: 97

 Step 4: Setting up tokenization...

{"model_id":"68267bf8bf714443b0643122342cb167","version_major":2,"vers
ion_minor":0}

{"model_id":"8862aff228794b3bb6a07ec2f35c3c03","version_major":2,"vers
ion_minor":0}

{"model_id":"44eab178ef1a45a18533696ecc22d2a9","version_major":2,"vers
ion_minor":0}

You are using the default legacy behaviour of the <class
'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is
expected, and simply means that the `legacy` (previous) behavior will
be used so nothing changes for you. If you want to use the new
behaviour, set `legacy=False`. This should only be set if you
understand what it means, and thoroughly read the reason why this was
added as explained in
https://github.com/huggingface/transformers/pull/24565
```

```
□ Tokenization setup complete!

□ Step 5: Setting up T5 model with LoRA...

{"model_id":"6a446e75db6c4ad298e15255c4356ad6","version_major":2,"vers
ion_minor":0}

{"model_id":"83bb4f72041d44b0b7837c81d5acc72b","version_major":2,"vers
ion_minor":0}

trainable params: 884,736 || all params: 223,788,288 || trainable%:
0.3953
□ Model ready on cuda

□ Step 6: Setting up training...
□ Training setup complete!

□ Step 7: Starting training...

{"model_id":"da5f9f16d06c4e7c93db9592aee21b3e","version_major":2,"vers
ion_minor":0}

{"model_id":"8af79074f3b0425292eb01aae6f06a5b","version_major":2,"vers
ion_minor":0}


Epoch 1: Train Loss = 1.3293, Val Loss = 1.1365
□ Best model saved!

{"model_id":"aa945884d9bc437e8b24915836d302ad","version_major":2,"vers
ion_minor":0}

{"model_id":"26f250b238f6436d9b7ede40eb603292","version_major":2,"vers
ion_minor":0}


Epoch 2: Train Loss = 0.8381, Val Loss = 0.5842
□ Best model saved!

{"model_id":"eeb516fb3ea344438ceed7672ec79947","version_major":2,"vers
ion_minor":0}

{"model_id":"d261f9ba011b4c1f85704c54358e7631","version_major":2,"vers
ion_minor":0}


Epoch 3: Train Loss = 0.4706, Val Loss = 0.3889
□ Best model saved!

{"model_id":"d5fd6a0e06114907931f2e1e7d7d1e42","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"b1c81f42206245bbb067a09c46a23c6e","version_major":2,"vers
ion_minor":0}


Epoch 4: Train Loss = 0.3354, Val Loss = 0.2811
□ Best model saved!

{"model_id":"7a1031f793484406aae7aaca79ebc8cd","version_major":2,"vers
ion_minor":0}

{"model_id":"981cf97769db4a969619f34a3c0b01e2","version_major":2,"vers
ion_minor":0}


Epoch 5: Train Loss = 0.2630, Val Loss = 0.2358
□ Best model saved!

{"model_id":"c29be1a0ec8143efa97ab86652058d73","version_major":2,"vers
ion_minor":0}

{"model_id":"02f3868a626640ae95841af8620a5454","version_major":2,"vers
ion_minor":0}


Epoch 6: Train Loss = 0.2206, Val Loss = 0.2049
□ Best model saved!

{"model_id":"f2723761c26e4f5eb49c24d26225efa2","version_major":2,"vers
ion_minor":0}

{"model_id":"d09f8c9e4d3c431ea7cfe2332c17d1a1","version_major":2,"vers
ion_minor":0}


Epoch 7: Train Loss = 0.1877, Val Loss = 0.1945
□ Best model saved!

{"model_id":"ccf01386189e4bc5a13e5caf85811c6e","version_major":2,"vers
ion_minor":0}

{"model_id":"80ae0273535c41a3a89275fb4c2d6016","version_major":2,"vers
ion_minor":0}


Epoch 8: Train Loss = 0.1627, Val Loss = 0.1804
□ Best model saved!

{"model_id":"1138d246f4294213b2eb0e0d8a22701d","version_major":2,"vers
ion_minor":0}

{"model_id":"ae9883066ee7412d8fcb8f8bbde1ec24","version_major":2,"vers
ion_minor":0}
```

```
Epoch 9: Train Loss = 0.1468, Val Loss = 0.1853

{"model_id":"6e28b0a6451e4108b465655ad8a7b8e7","version_major":2,"vers
ion_minor":0}

{"model_id":"1bbe6043b5d246278c37b170f632f12f","version_major":2,"vers
ion_minor":0}


Epoch 10: Train Loss = 0.1370, Val Loss = 0.1746
 Best model saved!

{"model_id":"05bb2adca67b48c19b80dae24396f1b9","version_major":2,"vers
ion_minor":0}

{"model_id":"d76de5bbbf5a404eaad1268783027e1e","version_major":2,"vers
ion_minor":0}


Epoch 11: Train Loss = 0.1283, Val Loss = 0.1587
 Best model saved!

{"model_id":"174c99cdd7d14cb992688d770466e33c","version_major":2,"vers
ion_minor":0}

{"model_id":"b2e639ba6071411e91df054e97b91cec","version_major":2,"vers
ion_minor":0}


Epoch 12: Train Loss = 0.1202, Val Loss = 0.1557
 Best model saved!

{"model_id":"ec6eb18e9d274b809abb425d19f7f47e","version_major":2,"vers
ion_minor":0}

{"model_id":"9ad4cdbe46a4404fa214c3c551ef603b","version_major":2,"vers
ion_minor":0}


Epoch 13: Train Loss = 0.1064, Val Loss = 0.1461
 Best model saved!

{"model_id":"e8f222ab943b41b59ac546ac42fbac42","version_major":2,"vers
ion_minor":0}

{"model_id":"fd3fb672034f40159d1fef42129090bd","version_major":2,"vers
ion_minor":0}


Epoch 14: Train Loss = 0.1033, Val Loss = 0.1622
```

```
{"model_id":"af7b7952a53a4c9da36c1675acdd35af","version_major":2,"vers
ion_minor":0}

{"model_id":"eb3568b4a1004c37ac391599c0f103f9","version_major":2,"vers
ion_minor":0}


Epoch 15: Train Loss = 0.1009, Val Loss = 0.1326
 Best model saved!

{"model_id":"ca67d0fdbee0420da24c9bc6036bbdf3","version_major":2,"vers
ion_minor":0}

{"model_id":"d33c88b670b64154b2a9d0dd06a65618","version_major":2,"vers
ion_minor":0}


Epoch 16: Train Loss = 0.0934, Val Loss = 0.1383

{"model_id":"a3c9e95110f5425a85c4d0cbf098b3c2","version_major":2,"vers
ion_minor":0}

{"model_id":"d6b47047c93942249185c9ca789a68ee","version_major":2,"vers
ion_minor":0}


Epoch 17: Train Loss = 0.0847, Val Loss = 0.1575

{"model_id":"bb0513e3270c4e5d93ea17187dd36d9f","version_major":2,"vers
ion_minor":0}

{"model_id":"8c314b9f33f94d72b569c92caeb45418","version_major":2,"vers
ion_minor":0}


Epoch 18: Train Loss = 0.0833, Val Loss = 0.1556

{"model_id":"4b24497e5b3346649e62b72caa0d5eb5","version_major":2,"vers
ion_minor":0}

{"model_id":"3106679580674b959d80f374cdaf47b3","version_major":2,"vers
ion_minor":0}


Epoch 19: Train Loss = 0.0784, Val Loss = 0.1641

{"model_id":"5309167b90b647939c0f3dccf2950e87","version_major":2,"vers
ion_minor":0}

{"model_id":"a68e40e1db1d4ea68e89c92229ad38a7","version_major":2,"vers
ion_minor":0}


Epoch 20: Train Loss = 0.0775, Val Loss = 0.1482
```

```
{"model_id":"24fae6b5f67047408eb772293cd0a85e","version_major":2,"version_minor":0}

{"model_id":"b3378cbf09894a5ba639f6d9965b77e9","version_major":2,"version_minor":0}


Epoch 21: Train Loss = 0.0743, Val Loss = 0.1364

{"model_id":"7b75b7e824944ef093bc96691cd7dcbb","version_major":2,"version_minor":0}

{"model_id":"62918f65c38f43a8a35ceeb1fb55c9f6","version_major":2,"version_minor":0}


Epoch 22: Train Loss = 0.0706, Val Loss = 0.1385

{"model_id":"7c2b3033fa974a9398455c4a7d7822f9","version_major":2,"version_minor":0}

{"model_id":"377d2c671a354c20aba0b28b4c4d6986","version_major":2,"version_minor":0}


Epoch 23: Train Loss = 0.0672, Val Loss = 0.1321
 Best model saved!

{"model_id":"aade2273dfa745e28380c8f2167acaf7","version_major":2,"version_minor":0}

{"model_id":"a58d397fdb674c0fb16eb4587deb553a","version_major":2,"version_minor":0}


Epoch 24: Train Loss = 0.0618, Val Loss = 0.1402

{"model_id":"5ad2b1a90f5b45f180860a05fffdeafd","version_major":2,"version_minor":0}

{"model_id":"7b0205ae86664d2cbd62cc034ccda54e","version_major":2,"version_minor":0}


Epoch 25: Train Loss = 0.0561, Val Loss = 0.1462

{"model_id":"2c3bb1a68e1f4a4a8a570980c3f933f6","version_major":2,"version_minor":0}

{"model_id":"77b56f408e1f44f6a0a83e2fd58e55e5","version_major":2,"version_minor":0}


Epoch 26: Train Loss = 0.0536, Val Loss = 0.1598
```

```
{"model_id":"c87a93c5ff6a4fc5ab7f48087c57e679","version_major":2,"vers
ion_minor":0}

{"model_id":"6af5c6c0abdf49ce9de98385ad9a166d","version_major":2,"vers
ion_minor":0}

Epoch 27: Train Loss = 0.0553, Val Loss = 0.1638

{"model_id":"7e63022c47f14742a7ccad2587a26a0b","version_major":2,"vers
ion_minor":0}

{"model_id":"b17344c8ce51482bacd52042a64b1942","version_major":2,"vers
ion_minor":0}

Epoch 28: Train Loss = 0.0532, Val Loss = 0.1331

{"model_id":"7f0639678f4c482a845496cd7c016f3b","version_major":2,"vers
ion_minor":0}

{"model_id":"a6fbda47bd3e421e9d9ecb594fe83aa5","version_major":2,"vers
ion_minor":0}

Epoch 29: Train Loss = 0.0469, Val Loss = 0.1603

{"model_id":"d4ed7155159f49cbaaf6776c32f29803","version_major":2,"vers
ion_minor":0}

{"model_id":"022c47ed36eb4901b8df00f1579d0165","version_major":2,"vers
ion_minor":0}

Epoch 30: Train Loss = 0.0513, Val Loss = 0.1533

{"model_id":"b4fafe5ca4f64845b4fc429a12108765","version_major":2,"vers
ion_minor":0}

{"model_id":"a59cc96fc77f4f9184b3e85c0ac0aee2","version_major":2,"vers
ion_minor":0}

Epoch 31: Train Loss = 0.0476, Val Loss = 0.1412

{"model_id":"80a6c95319354239b38924ab9d0f3c52","version_major":2,"vers
ion_minor":0}

{"model_id":"34bac1b5bca844e381b1b081f719999d","version_major":2,"vers
ion_minor":0}

Epoch 32: Train Loss = 0.0446, Val Loss = 0.1509
```

```
{"model_id":"1b45f83329a142c4b48da7bde1fbe341","version_major":2,"vers
ion_minor":0}

{"model_id":"eaa5558cd0aa45d6bc4a1387e184420f","version_major":2,"vers
ion_minor":0}

Epoch 33: Train Loss = 0.0438, Val Loss = 0.1548

{"model_id":"777eab304214467f8421b7aced80863b","version_major":2,"vers
ion_minor":0}

{"model_id":"302a93440d934765bf52800f098916d0","version_major":2,"vers
ion_minor":0}

Epoch 34: Train Loss = 0.0411, Val Loss = 0.1289
□ Best model saved!

{"model_id":"ec16f1af259c4ff0bd2262b15c188e17","version_major":2,"vers
ion_minor":0}

{"model_id":"84b771087fcd4b24a30a45c520fcd03d","version_major":2,"vers
ion_minor":0}

Epoch 35: Train Loss = 0.0397, Val Loss = 0.1613

{"model_id":"76554183530c4c898b76767601cc3cac","version_major":2,"vers
ion_minor":0}

{"model_id":"5c89cbbc2bee403a9ccfe8bb80ba13cb","version_major":2,"vers
ion_minor":0}

Epoch 36: Train Loss = 0.0411, Val Loss = 0.1474

{"model_id":"cf86a41729de4a6f83768f9cf04f9e2b","version_major":2,"vers
ion_minor":0}

{"model_id":"9b30b9cb928a45eda1e09348805b97a3","version_major":2,"vers
ion_minor":0}

Epoch 37: Train Loss = 0.0381, Val Loss = 0.1428

{"model_id":"01a9017ae734471fb8bd197c129669cb","version_major":2,"vers
ion_minor":0}

{"model_id":"0c8512d6be0841b19cb1511c335b0226","version_major":2,"vers
ion_minor":0}

Epoch 38: Train Loss = 0.0361, Val Loss = 0.1351
```

```
{"model_id":"84f0d1fbf71f4dfb9338662b69975e79","version_major":2,"version_minor":0}

{"model_id":"1e7a90586c584063931e3c0bc60c8d07","version_major":2,"version_minor":0}

Epoch 39: Train Loss = 0.0354, Val Loss = 0.1419

{"model_id":"894b5105c16a4dc399f468cfe9ae6aaf","version_major":2,"version_minor":0}

{"model_id":"63cfb94a78dc43c7856dceccb37cdf5b","version_major":2,"version_minor":0}

Epoch 40: Train Loss = 0.0330, Val Loss = 0.1538

{"model_id":"1065463c9bb64e7197baecfb21cb3874","version_major":2,"version_minor":0}

{"model_id":"865140c797194f98a501b18e62c3aff7","version_major":2,"version_minor":0}

Epoch 41: Train Loss = 0.0313, Val Loss = 0.1724

{"model_id":"42e7236c9642416fa2749b3fd31fce75","version_major":2,"version_minor":0}

{"model_id":"1ad03e79f03f470b817b2e9abeab7584","version_major":2,"version_minor":0}

Epoch 42: Train Loss = 0.0345, Val Loss = 0.1602

{"model_id":"22af0cdd5f504d789f769c499a07cff0","version_major":2,"version_minor":0}

{"model_id":"36d454460aea48b5a555f7c0a3cebd4e","version_major":2,"version_minor":0}

Epoch 43: Train Loss = 0.0297, Val Loss = 0.1697

{"model_id":"49da3d6509194fca867e137c1663e115","version_major":2,"version_minor":0}

{"model_id":"362e08c2770949fc8a5b5a48f7db2872","version_major":2,"version_minor":0}

Epoch 44: Train Loss = 0.0294, Val Loss = 0.1658
```

```
{"model_id":"6b8ce58ee25b4c8d97122c49ab098cda","version_major":2,"version_minor":0}

{"model_id":"3aebae8a3ff943d99e8c9284b9534347","version_major":2,"version_minor":0}


Epoch 45: Train Loss = 0.0311, Val Loss = 0.1557

{"model_id":"3649efdfa9ef405d80456d21d2b4c96e","version_major":2,"version_minor":0}

{"model_id":"6fd27db813c54943afe64caec550bba2","version_major":2,"version_minor":0}


Epoch 46: Train Loss = 0.0294, Val Loss = 0.1607

{"model_id":"b11410af69b34d56b424f025b31a3ec1","version_major":2,"version_minor":0}

{"model_id":"d2161194018d46b1a478be32d6bd6e8d","version_major":2,"version_minor":0}


Epoch 47: Train Loss = 0.0255, Val Loss = 0.1602

{"model_id":"ebec75b25b854edea7d5ce02375f4c93","version_major":2,"version_minor":0}

{"model_id":"a9ebafd230c145e8a3aecffc1f8ee918","version_major":2,"version_minor":0}


Epoch 48: Train Loss = 0.0253, Val Loss = 0.1724

{"model_id":"2b8636a8c9374da8b5c000e5f4e141b1","version_major":2,"version_minor":0}

{"model_id":"11963cca83054eff934bf91dd3eeccc2","version_major":2,"version_minor":0}


Epoch 49: Train Loss = 0.0257, Val Loss = 0.1680

{"model_id":"58a227eea7c2452ebb2a8f67c7f7fe49","version_major":2,"version_minor":0}

{"model_id":"6cbba516c2f143b6a6bfae18db08325f","version_major":2,"version_minor":0}


Epoch 50: Train Loss = 0.0245, Val Loss = 0.1692
```

```
{"model_id":"488f1ae24011488384e2a1c4ed2072f3","version_major":2,"version_minor":0}

{"model_id":"5bdb8c76880948efb49873a2bec4fcb1","version_major":2,"version_minor":0}


Epoch 51: Train Loss = 0.0238, Val Loss = 0.1598

{"model_id":"0c3f6badb4974c03afbf89ee424a3042","version_major":2,"version_minor":0}

{"model_id":"f48d613e164b423f9fef2b27ae2031bf","version_major":2,"version_minor":0}


Epoch 52: Train Loss = 0.0242, Val Loss = 0.1628

{"model_id":"3b5cbbe423444d52bba3ec85612169fb","version_major":2,"version_minor":0}

{"model_id":"813daecc9d51456bb34cc7b5efcff92f","version_major":2,"version_minor":0}


Epoch 53: Train Loss = 0.0225, Val Loss = 0.1577

{"model_id":"2c6829c4ced74374b669af49502adfb3","version_major":2,"version_minor":0}

{"model_id":"a28c871d2d55439da2960ad2150033bb","version_major":2,"version_minor":0}


Epoch 54: Train Loss = 0.0232, Val Loss = 0.1554

{"model_id":"9f6242ede04e4b698564a9b5ad428f69","version_major":2,"version_minor":0}

{"model_id":"027c0cbfd5054a9f9081d5100e498f68","version_major":2,"version_minor":0}


Epoch 55: Train Loss = 0.0223, Val Loss = 0.1612

{"model_id":"39651a52a06e4d9ca598be349d90a7a6","version_major":2,"version_minor":0}

{"model_id":"f3368eaf02d849c09c1d601fc7c1f16f","version_major":2,"version_minor":0}


Epoch 56: Train Loss = 0.0200, Val Loss = 0.1607
```

```
{"model_id":"b3e20b72136d43afa9f18462226eefd4","version_major":2,"vers
ion_minor":0}

{"model_id":"46627132c0a244a2a46019a03b50fd19","version_major":2,"vers
ion_minor":0}


Epoch 57: Train Loss = 0.0211, Val Loss = 0.1781

{"model_id":"77635b7282274e1ebc8edf52235385f7","version_major":2,"vers
ion_minor":0}

{"model_id":"862bb706412540868a4af3c1491d28af","version_major":2,"vers
ion_minor":0}


Epoch 58: Train Loss = 0.0201, Val Loss = 0.1653

{"model_id":"37cc0d4cec7f43f0b5569f49b17fa4eb","version_major":2,"vers
ion_minor":0}

{"model_id":"4cee89737a154260898fac5b3872854f","version_major":2,"vers
ion_minor":0}


Epoch 59: Train Loss = 0.0178, Val Loss = 0.1691

{"model_id":"c1b46da03ee34086a7f292f851c22204","version_major":2,"vers
ion_minor":0}

{"model_id":"30f6b2b0d8404642a076163b36817300","version_major":2,"vers
ion_minor":0}


Epoch 60: Train Loss = 0.0177, Val Loss = 0.1726

{"model_id":"cca078d70c65444f908bfd81e8eee554","version_major":2,"vers
ion_minor":0}

{"model_id":"c1eb40c5ded148d394cd363742007b3a","version_major":2,"vers
ion_minor":0}


Epoch 61: Train Loss = 0.0184, Val Loss = 0.1585

{"model_id":"70474e1621124a0d9529435db140cfe8","version_major":2,"vers
ion_minor":0}

{"model_id":"b2dae47b3c8c48fc82bb6db764c5b6c4","version_major":2,"vers
ion_minor":0}


Epoch 62: Train Loss = 0.0200, Val Loss = 0.1701
```

```
{"model_id":"ccdd4dd7e84c46419b9e95bd9f46d1b4","version_major":2,"version_minor":0}

{"model_id":"191ec598f5444ee79f69331be7de67e3","version_major":2,"version_minor":0}


Epoch 63: Train Loss = 0.0186, Val Loss = 0.1737

{"model_id":"16467bfd2a2342a7810b90729c017db1","version_major":2,"version_minor":0}

{"model_id":"35bc129eb1724aa3a8ed850de8a9478f","version_major":2,"version_minor":0}


Epoch 64: Train Loss = 0.0166, Val Loss = 0.1604

{"model_id":"5547db58a36d410892076604f6473ac4","version_major":2,"version_minor":0}

{"model_id":"80d793648edd4f3880304a371fabbde5","version_major":2,"version_minor":0}


Epoch 65: Train Loss = 0.0174, Val Loss = 0.1564

{"model_id":"bd6ad393e2d543749efa806935ba84ba","version_major":2,"version_minor":0}

{"model_id":"db6b5316851f49a4ad473a36756add82","version_major":2,"version_minor":0}


Epoch 66: Train Loss = 0.0164, Val Loss = 0.1784

{"model_id":"c89f4981500941538a471da9354c3a45","version_major":2,"version_minor":0}

{"model_id":"c20ae8f0a17a478e834e62a06e52c625","version_major":2,"version_minor":0}


Epoch 67: Train Loss = 0.0168, Val Loss = 0.1657

{"model_id":"9e17ac4bcf9a45cbbd6d53cef4d79e76","version_major":2,"version_minor":0}

{"model_id":"0003c8000269455b90cc3e2fd842efba","version_major":2,"version_minor":0}


Epoch 68: Train Loss = 0.0144, Val Loss = 0.1634
```

```
{"model_id":"b05d8c2c3c4d4a3babbbece1ab3e1ed9","version_major":2,"vers
ion_minor":0}

{"model_id":"961857f5e68c4fb8a6849ecad2d344f0","version_major":2,"vers
ion_minor":0}


Epoch 69: Train Loss = 0.0152, Val Loss = 0.1662

{"model_id":"9e4aa00db1ea47259551aea995207f84","version_major":2,"vers
ion_minor":0}

{"model_id":"1403b731a1074896a120a69b446f377e","version_major":2,"vers
ion_minor":0}


Epoch 70: Train Loss = 0.0151, Val Loss = 0.1709

{"model_id":"08c77adc22184d6bab80dc1f0e679184","version_major":2,"vers
ion_minor":0}

{"model_id":"66960b6e1b78433ca99ebf96b60bd11a","version_major":2,"vers
ion_minor":0}


Epoch 71: Train Loss = 0.0125, Val Loss = 0.1733

{"model_id":"5f6e8e5128f64d019453baa8cc08d1e2","version_major":2,"vers
ion_minor":0}

{"model_id":"6f573f239d3c4c1e90b5b28bc3db6461","version_major":2,"vers
ion_minor":0}


Epoch 72: Train Loss = 0.0145, Val Loss = 0.1670

{"model_id":"6543e4d300474701879e59b83f0b773c","version_major":2,"vers
ion_minor":0}

{"model_id":"6a1c9de33684444e972014736d0a6edd","version_major":2,"vers
ion_minor":0}


Epoch 73: Train Loss = 0.0145, Val Loss = 0.1855

{"model_id":"f5b1b9dcabd64538a76304976c7ba617","version_major":2,"vers
ion_minor":0}

{"model_id":"3db18e1855a44bcead24b9ba0959182b","version_major":2,"vers
ion_minor":0}


Epoch 74: Train Loss = 0.0131, Val Loss = 0.1736
```

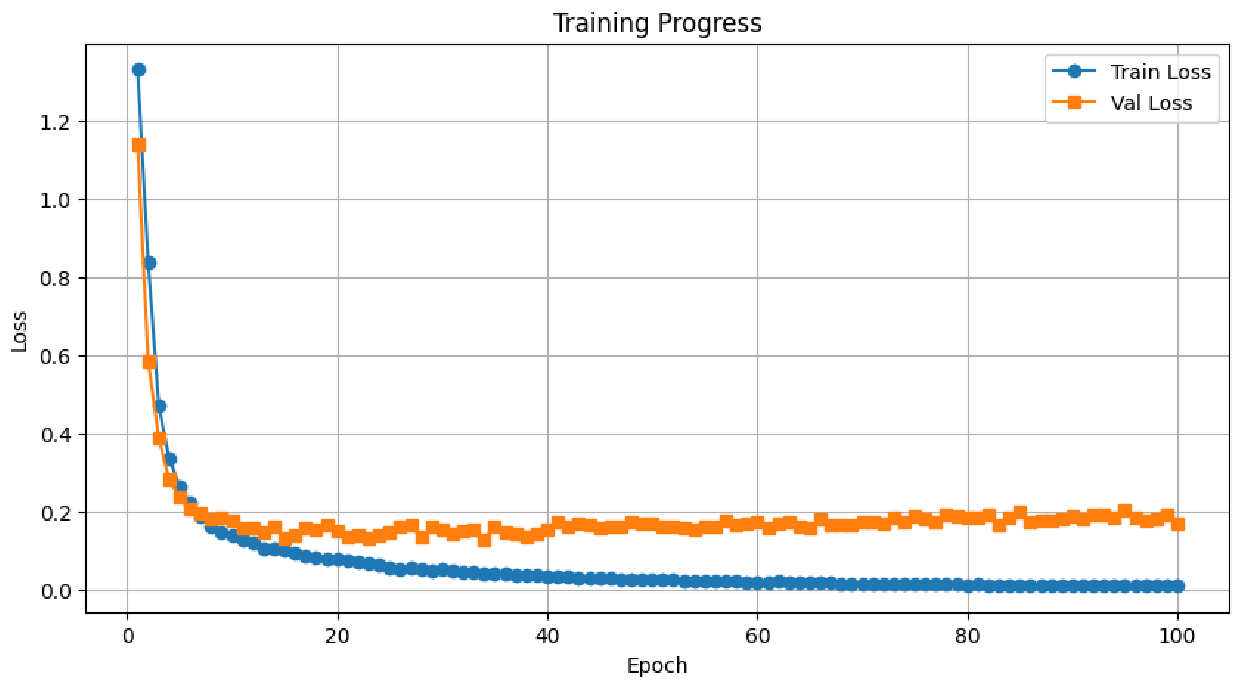
```
{"model_id":"650756b0a57446a5949deb0489d6387f","version_major":2,"version_minor":0}

{"model_id":"ac086b4536b44def9f7e56d4859cab80","version_major":2,"version_minor":0}


Epoch 75: Train Loss = 0.0143, Val Loss = 0.1863

{"model_id":"aec674d3814b49abbcdf7ea6d1bd5496","version_major":2,"version_minor":0}

{"model_id":"52d5f4218a9a4129b35474d5f83c86e0","version_major":2,"version_minor":0}


Epoch 76: Train Loss = 0.0133, Val Loss = 0.1790

{"model_id":"20562e2a7c91492eaf77fc8c35871743","version_major":2,"version_minor":0}

{"model_id":"6b785e268cda473a982f0d481d339660","version_major":2,"version_minor":0}


Epoch 77: Train Loss = 0.0129, Val Loss = 0.1714

{"model_id":"a2895c1dcb434daeb6739bc04c7c77fd","version_major":2,"version_minor":0}

{"model_id":"49e322e22adc458091ae4d642df1107c","version_major":2,"version_minor":0}


Epoch 78: Train Loss = 0.0147, Val Loss = 0.1914

{"model_id":"40ea995b93ac4a668587227310e3ac65","version_major":2,"version_minor":0}

{"model_id":"d52aeae132564e069a5db5d3b61aa595","version_major":2,"version_minor":0}


Epoch 79: Train Loss = 0.0129, Val Loss = 0.1894

{"model_id":"0d4510afa43f4573bbb222f19d07d737","version_major":2,"version_minor":0}

{"model_id":"cb900af3b7d042e49573efce761e6071","version_major":2,"version_minor":0}


Epoch 80: Train Loss = 0.0122, Val Loss = 0.1827
```

```
{"model_id":"48e70fe09f55465fb6b74dbc0786a5b1","version_major":2,"version_minor":0}

{"model_id":"7a83e885da544151a297707fad4f1db3","version_major":2,"version_minor":0}


Epoch 81: Train Loss = 0.0132, Val Loss = 0.1854

{"model_id":"43077282209949b9bae6268cd980a999","version_major":2,"version_minor":0}

{"model_id":"08f9ed114fdf4f8580dfcb9583326cf7","version_major":2,"version_minor":0}


Epoch 82: Train Loss = 0.0114, Val Loss = 0.1905

{"model_id":"9edb58e844d84665b2505754e9ce17f4","version_major":2,"version_minor":0}

{"model_id":"d53e5f138b0c4593b396bdb873dc527c","version_major":2,"version_minor":0}


Epoch 83: Train Loss = 0.0121, Val Loss = 0.1666

{"model_id":"5a29b5cc81da483d9b4f643429793bb7","version_major":2,"version_minor":0}

{"model_id":"45b168ece5884433987f99189239b685","version_major":2,"version_minor":0}


Epoch 84: Train Loss = 0.0120, Val Loss = 0.1846

{"model_id":"0cb6158a687f4f4abb45de234d0b1246","version_major":2,"version_minor":0}

{"model_id":"f677694c38eb402092ba9e3d597c0fe6","version_major":2,"version_minor":0}


Epoch 85: Train Loss = 0.0107, Val Loss = 0.1973

{"model_id":"82e305b7d01c45d3a96638775c87391e","version_major":2,"version_minor":0}

{"model_id":"00c705b15ea84b02a473089493931233","version_major":2,"version_minor":0}


Epoch 86: Train Loss = 0.0120, Val Loss = 0.1731
```

```
{"model_id":"1762a0de819b45ebb740780be40226f4","version_major":2,"version_minor":0}

{"model_id":"dee0d825621f4f30a99ecc888bff1f97","version_major":2,"version_minor":0}


Epoch 87: Train Loss = 0.0110, Val Loss = 0.1776

{"model_id":"cf15cf16291f4422a72aaec2b16f7a32","version_major":2,"version_minor":0}

{"model_id":"dd8b635bedff44408adfb600b9e457af","version_major":2,"version_minor":0}


Epoch 88: Train Loss = 0.0122, Val Loss = 0.1753

{"model_id":"d4e54b7cb987402b80de99c6b8341326","version_major":2,"version_minor":0}

{"model_id":"c964c353a8294774ae6317827f0af6c9","version_major":2,"version_minor":0}


Epoch 89: Train Loss = 0.0117, Val Loss = 0.1808

{"model_id":"8177c6ea256647e1aaab5d4e8948d220","version_major":2,"version_minor":0}

{"model_id":"8f0f7d46344f426f949dd077531c634b","version_major":2,"version_minor":0}


Epoch 90: Train Loss = 0.0107, Val Loss = 0.1893

{"model_id":"950ef502693340058535646988d849de","version_major":2,"version_minor":0}

{"model_id":"6dbb0c4e11e6488da3e248491cf052a6","version_major":2,"version_minor":0}


Epoch 91: Train Loss = 0.0103, Val Loss = 0.1809

{"model_id":"c873816f09284a5c8724d5f80b3a112e","version_major":2,"version_minor":0}

{"model_id":"750d3938a23f45bcb92b3c061060233d","version_major":2,"version_minor":0}


Epoch 92: Train Loss = 0.0121, Val Loss = 0.1931
```

```
{"model_id":"c6d31767d51745f38b12b2568c72e331","version_major":2,"version_minor":0}

{"model_id":"ba7f2fcec03a4dc391f6cb77e8557974","version_major":2,"version_minor":0}


Epoch 93: Train Loss = 0.0120, Val Loss = 0.1928

{"model_id":"255744840d784e3d987c2e90701bc2ec","version_major":2,"version_minor":0}

{"model_id":"a0eb7a2717b541ec9a6404fd66ee9b71","version_major":2,"version_minor":0}


Epoch 94: Train Loss = 0.0100, Val Loss = 0.1834

{"model_id":"1a660b32984b4bafbc07a38955bfe485","version_major":2,"version_minor":0}

{"model_id":"fc755c33363b4e589678a69169625d04","version_major":2,"version_minor":0}


Epoch 95: Train Loss = 0.0103, Val Loss = 0.2011

{"model_id":"17ee10a6fbf346728729c1bdd5692c16","version_major":2,"version_minor":0}

{"model_id":"5234fe21af60445b83cbb98f35e22178","version_major":2,"version_minor":0}


Epoch 96: Train Loss = 0.0108, Val Loss = 0.1844

{"model_id":"ac9918a7d7ec47168d135566696177cf","version_major":2,"version_minor":0}

{"model_id":"a15e10a5a44345ff887e1f296dec094d","version_major":2,"version_minor":0}


Epoch 97: Train Loss = 0.0098, Val Loss = 0.1762

{"model_id":"3a9ad82361bd4339b4e7ce73a1bb2eca","version_major":2,"version_minor":0}

{"model_id":"b59cd652d53941f2b72ce745ea803a56","version_major":2,"version_minor":0}


Epoch 98: Train Loss = 0.0106, Val Loss = 0.1810
```

```
{"model_id":"3cebfb7b245e4651bc12252b1db5e157","version_major":2,"version_minor":0}

{"model_id":"80d992b298ae4eb9a867816cc9ed1721","version_major":2,"version_minor":0}


Epoch 99: Train Loss = 0.0101, Val Loss = 0.1924

{"model_id":"7ce7628288614dd38532b89fdbdba1f0","version_major":2,"version_minor":0}

{"model_id":"6d8d1eb4b3274534a91e0a91239b09d4","version_major":2,"version_minor":0}


Epoch 100: Train Loss = 0.0097, Val Loss = 0.1695

 Training complete!
```



```

 Step 8: Setting up inference pipeline...
 Inference pipeline ready!

 Step 9: Comprehensive evaluation...

{"model_id":"7d9d80018f184afc8911db7723180cd5","version_major":2,"version_minor":0}
```

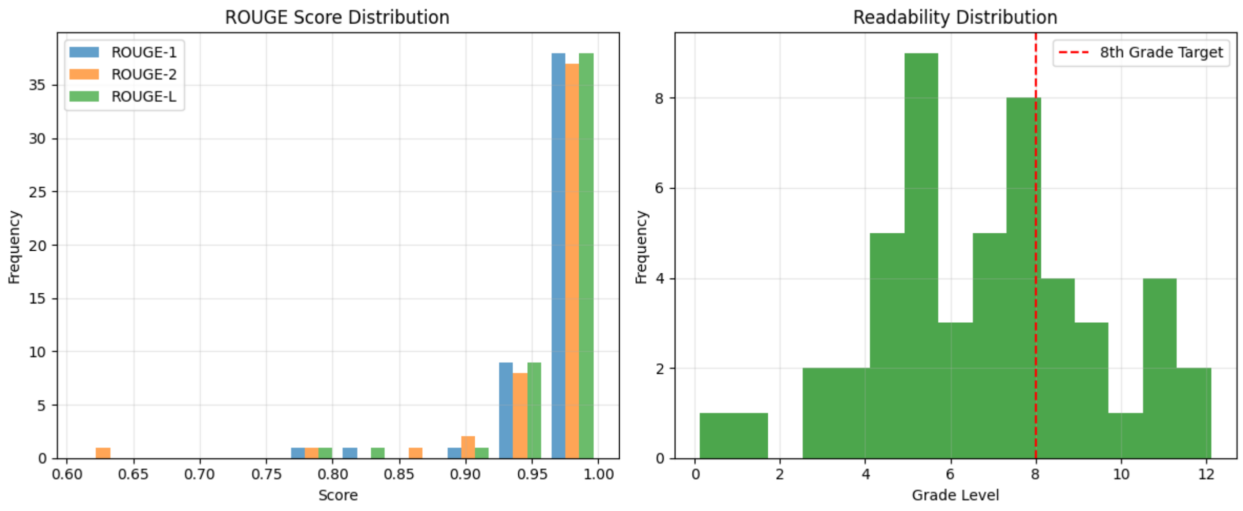
```
======================================================================
Example 1
======================================================================

□ SHORT SUMMARY:
Figures of speech are words or phrases that use language in a
nonliteral or unusual way. They can make writing more expressive.
Verbal irony involves The text uses verbal irony, which involves
saying one thing but implying something very different. As quiet as a
jackhammer suggests that the snor

★ FLASHCARD:
Figures of speech are words or phrases that use language in a
nonliteral or unusual way. They can make writing more expressive.
Verbal irony involves The text uses verbal irony, which involves
saying one thing but implying something very different. As quiet as a
jackhammer suggests that the snor

□ STUDY NOTES:
Figures of speech are words or phrases that use language in a
nonliteral or unusual way. They can make writing more expressive.
Verbal irony involves The text uses verbal irony, which involves
saying one thing but implying something very different. As quiet as a
jackhammer suggests that the snoring is loud. A jackhammer

======================================================================
Example 2
======================================================================

□ SHORT SUMMARY:
An adaptation is an inherited trait that helps an organism survive or
reproduce. Adaptations can include both body parts and behaviors. The
shape of a Look at the picture of the sturgeon. The sturgeon's mouth
is located on the underside of its head and points downward. Its mouth
is adapted for bottom f

★ FLASHCARD:
An adaptation is an inherited trait that helps an organism survive or
reproduce. Adaptations can include both body parts and behaviors. The
shape of a Look at the picture of the sturgeon. The sturgeon's mouth
is located on the underside of its head and points downward. Its mouth
is adapted for bottom feeding. The st

□ STUDY NOTES:
An adaptation is an inherited trait that helps an organism survive or
reproduce. Adaptations can include both body parts and behaviors. The
shape of a Look at the picture of the sturgeon. The sturgeon's mouth
is located on the underside of its head and points downward. Its mouth
is adapted for bottom feeding. The sturgeon
```

```
======================================================================
Example 3
======================================================================

□ SHORT SUMMARY:
A sentence is a group of words that expresses a complete thought. The
band I'm in has been rehearsing daily because we have a concert in two
weeks. A sentence fragment This is a sentence fragment. It does not
express a complete thought. During the construction of Mount Rushmore,
approximately eight hundred million pounds of rock were removed

★ FLASHCARD:
A sentence is a group of words that expresses a complete thought. The
band I'm in has been rehearsing daily because we have a concert in two
weeks. A sentence fragment is This is a sentence fragment. It does not
express a complete thought. During the construction of Mount Rushmore,
approximately eight hundred million pounds of rock were removed

□ STUDY NOTES:
A sentence is a group of words that expresses a complete thought. The
band I'm in has been rehearsing daily because This is a sentence
fragment. It does not express a complete thought. During the
construction of Mount Rushmore, approximately eight hundred million
pounds of rock were removed from the mountain to create the monument.

======================================================================
Example 4
======================================================================

□ SHORT SUMMARY:
People can use the engineering-design process to develop solutions to
problems. One step in the process is testing if a potential solution
meets the r

★ FLASHCARD:
People can use the engineering-design process to develop solutions to
problems. One step in the process is testing if a potential solution
meets the r

□ STUDY NOTES:
People can use the engineering-design process to develop solutions to
problems. One step in the process is testing if a potential solution
meets the r

======================================================================
Example 5
======================================================================

□ SHORT SUMMARY:
In a title, capitalize the first word, the last word, and every
```

```
important word in between. The Wind in the Willows James and the Giant
Peach These wor Capitalize the first word, the last word, and every
important word in between. The word of is not important, so it should
not be capitalized. The corr

🃏 FLASHCARD:
In a title, capitalize the first word, the last word, and every
important word in between. The Wind in the Willows James and the Giant
Peach These wor Capitalize the first word, the last word, and every
important word in between. The word of is not important, so it should
not be capitalized. The corr

📝 STUDY NOTES:
In a title, capitalize the first word, the last word, and every
important word in between. The Wind in the Willows James and the Giant
Peach These wor Capitalize the first word, the last word, and every
important word in between. The word of is not important, so it should
not be capitalized. The corr

Calculating BERTScore...
```

```
{"model_id":"d6150c8eb2514104853428ce855adc3b","version_major":2,"version_minor":0}

{"model_id":"d6f26d7694b747648a4581485c8e73dc","version_major":2,"version_minor":0}

{"model_id":"db629d64d4164247af7f0e242f9f722c","version_major":2,"version_minor":0}

{"model_id":"5fd3c2ccaa144811bc6892ef12f27df2","version_major":2,"version_minor":0}

{"model_id":"78ea265cee864b06a96757046b592973","version_major":2,"version_minor":0}

{"model_id":"1156b67cbdbc4f87ae893626f36f95fb","version_major":2,"version_minor":0}
```

```
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.


======================================================================
EVALUATION RESULTS
======================================================================

📊 ROUGE Scores:
   ROUGE-1: 0.9721 ± 0.0440
   ROUGE-2: 0.9671 ± 0.0630
   ROUGE-L: 0.9721 ± 0.0440

 BERTScore:
   Precision: 0.9842
   Recall:    0.9804
   F1:        0.9823

 Readability:
   Avg Grade Level: 6.82
   (Target: 6-10 for middle/high school)
```



```
 Evaluation complete! Check 'evaluation_results/' folder.
======================================================================
```