```
#
================================================================================
========
# 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 = 10
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 3.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 239.2/239.2 kB 21.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 69.4 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":"73431286bbad430fa94123c302831197","version_major":2,"vers
ion_minor":0}

{"model_id":"d9f1f20e23f14213b6de16f142a32f3c","version_major":2,"vers
ion_minor":0}

 Loaded 495 samples
Loading validation split from ScienceQA...

{"model_id":"55f81e4960934815870baa34c9a3da1b","version_major":2,"vers
ion_minor":0}

 Loaded 97 samples
Training samples: 495
Validation samples: 97

 Step 4: Setting up tokenization...

{"model_id":"ce1fb29d24d54061bc4a8472663a8428","version_major":2,"vers
ion_minor":0}

{"model_id":"6ffae414429f43d0b7edf7dc975ac476","version_major":2,"vers
ion_minor":0}

{"model_id":"124a0937e6134fcd905fa827f11fa7e8","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":"76f16da11b59412b8adda101fd1e1689","version_major":2,"version_minor":0}

{"model_id":"5828e8f4130a4371a6e28d62ee59b978","version_major":2,"version_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":"04090d2146794cec979c4b09350fa4dc","version_major":2,"version_minor":0}

{"model_id":"594b44a29f3047b9bfbc69bf0dc5c991","version_major":2,"version_minor":0}

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

{"model_id":"1efb63d273f84086bd68527a021850c5","version_major":2,"version_minor":0}

{"model_id":"79a71faf81b842d3910635d17c23031a","version_major":2,"version_minor":0}

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

{"model_id":"fbd3e347bb27479fb2cd6682fd020fb9","version_major":2,"version_minor":0}

{"model_id":"e0d7625e9c644f149e5cb660b7887c56","version_major":2,"version_minor":0}

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

{"model_id":"c5cf173691cb4d29bb3e156985533c9f","version_major":2,"version_minor":0}

{"model_id":"c7f979f763fe446599e9fb14c9555f5e","version_major":2,"version_minor":0}

```
Epoch 4: Train Loss = 0.3355, Val Loss = 0.2816
 Best model saved!
```

{"model_id":"5d091a2237904b9796a02551e947201a","version_major":2,"version_minor":0}

{"model_id":"abe3a4f68f3c49e9ac410c0e423a7668","version_major":2,"version_minor":0}

```
Epoch 5: Train Loss = 0.2637, Val Loss = 0.2365
 Best model saved!
```

{"model_id":"f820e50fb1d14da88ad03f33d51e449d","version_major":2,"version_minor":0}

{"model_id":"bb7190c7a394419ab48c32f1d3356e59","version_major":2,"version_minor":0}

```
Epoch 6: Train Loss = 0.2222, Val Loss = 0.2066
 Best model saved!
```

{"model_id":"63c21e5220d945cabd1b936f1863e0cd","version_major":2,"version_minor":0}

{"model_id":"c53ef35ae694400a89e4fc227e24384b","version_major":2,"version_minor":0}

```
Epoch 7: Train Loss = 0.1894, Val Loss = 0.1984
 Best model saved!
```

{"model_id":"f2c24429ec464f1d9f581694be0ffdaf","version_major":2,"version_minor":0}

{"model_id":"723d8e42624c43adb170eda17d5e08a1","version_major":2,"version_minor":0}

```
Epoch 8: Train Loss = 0.1652, Val Loss = 0.1823
 Best model saved!
```

{"model_id":"2659cac138184e999ef7dd6493b6a722","version_major":2,"version_minor":0}

{"model_id":"aed4e88d4dbc4dc591480bfed736f66e","version_major":2,"version_minor":0}

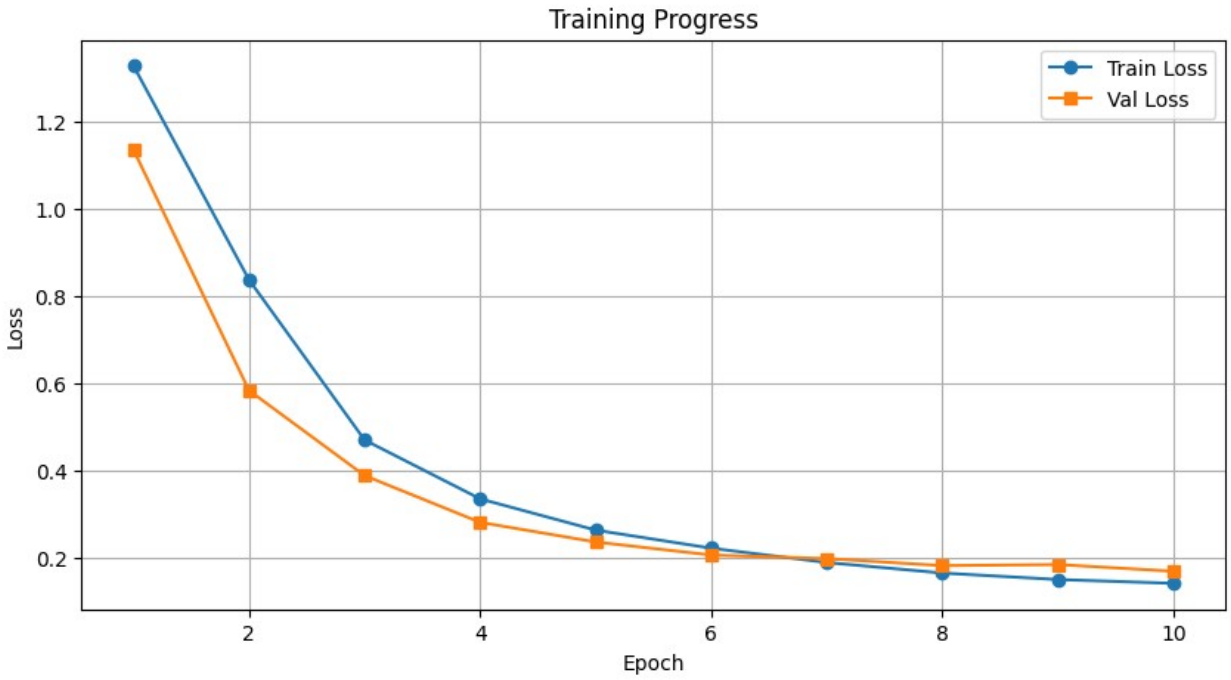
```
Epoch 9: Train Loss = 0.1504, Val Loss = 0.1847
```

{"model_id":"29d623f8fe074368afde862b1e7c692f","version_major":2,"version_minor":0}

{"model_id":"61dde19f811e424e838f5697b4e74c52","version_major":2,"version_minor":0}

```
Epoch 10: Train Loss = 0.1416, Val Loss = 0.1693
 Best model saved!

 Training complete!
```




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

 Step 9: Comprehensive evaluation...
```

{"model_id":"ae38fb13b4c24e408d94bffef5435462","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 saying one thing but implying something very different. People often use verbal irony in 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

 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 saying one thing but implying something very different. As quiet as a jackhammer suggests that the snoring is loud. A jackhammer is not quiet,

 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 saying one thing but implying something very different. As quiet as a jackhammer suggests that the snoring is loud. A jackhammer is not quiet,

==========
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 an animal's 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 uses

 FLASHCARD:
An adaptation is an inherited trait that helps an organism survive or reproduce. Adaptations can include both body parts and behaviors. 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 uses its mouth to find food hidden

 STUDY NOTES:
An adaptation is an inherited trait that helps an organism survive or reproduce. Adaptations can include both body parts and behaviors. 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 uses its mouth to find food hidden in the sediment

==========
Example 3

==========

 SHORT SUMMARY:
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 from the mountain to create the monument. Here is one way to fix the sentence fragment

 FLASHCARD:
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 from the mountain to create the monument. Here is one way to fix the sentence fragment

 STUDY NOTES:
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 correct title is A Breath of Fresh Air.

🃏 FLASHCARD:
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 correct title is A Breath of Fresh Air.

📝 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 correct title is A Breath of Fresh Air.

Calculating BERTScore...

{"model_id":"0a04f2f9f9a14b9aa76aa8a10e6598a6","version_major":2,"version_minor":0}

{"model_id":"8d53f88671a040ed81f2bac6d0178785","version_major":2,"version_minor":0}

{"model_id":"7b4ef156028449e5b0056eeb3d13bff1","version_major":2,"version_minor":0}

{"model_id":"209e685101a749918abfbd00c391abdf","version_major":2,"version_minor":0}

{"model_id":"1c83dd16a77942b69df3a80a31929492","version_major":2,"version_minor":0}

{"model_id":"c5ae8653adf14d6689d6d611a462dbc2","version_major":2,"version_minor":0}

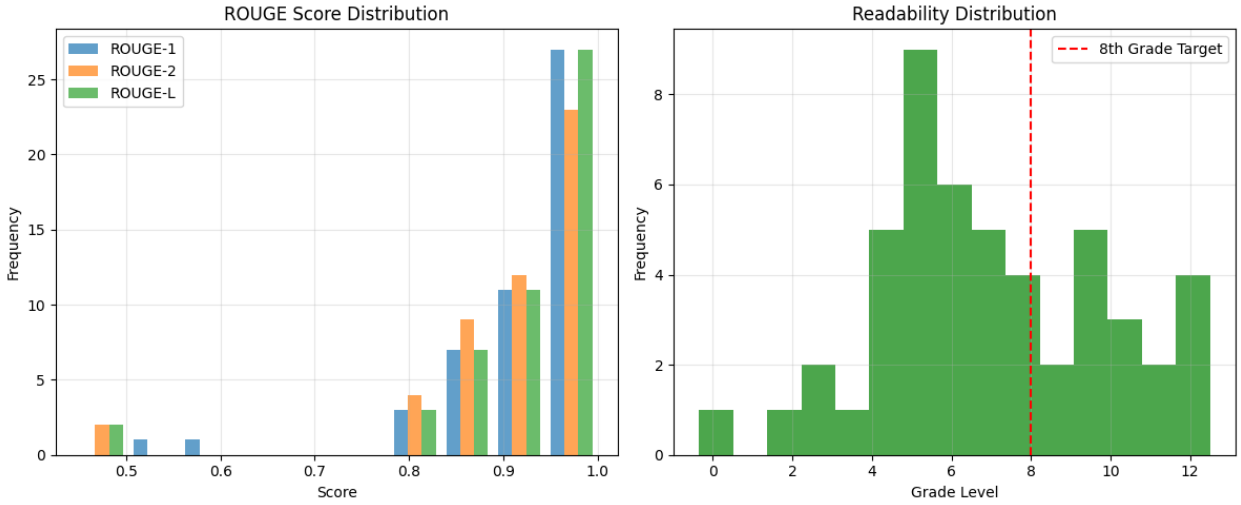
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

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

📊 ROUGE Scores:
   ROUGE-1: 0.9246 ± 0.0935
   ROUGE-2: 0.9120 ± 0.1121
   ROUGE-L: 0.9213 ± 0.1056

🎯 BERTScore:
   Precision: 0.9730
   Recall: 0.9712
   F1:   0.9720

```
📖 Readability:
   Avg Grade Level: 6.99
   (Target: 6-10 for middle/high school)
```




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

!pip install peft

Collecting peft
  Obtaining dependency information for peft from
https://files.pythonhosted.org/packages/49/fe/a2da1627aa9cb6310b603459
8363bd26ac301c4a99d21f415b1b2855891e/peft-0.17.1-py3-none-
any.whl.metadata
  Downloading peft-0.17.1-py3-none-any.whl.metadata (14 kB)
Requirement already satisfied: numpy>=1.17 in c:\users\om\anaconda3\
lib\site-packages (from peft) (1.24.3)
Requirement already satisfied: packaging>=20.0 in c:\users\om\
anaconda3\lib\site-packages (from peft) (23.1)
Requirement already satisfied: psutil in c:\users\om\appdata\roaming\
python\python311\site-packages (from peft) (5.9.7)
Requirement already satisfied: pyyaml in c:\users\om\anaconda3\lib\
site-packages (from peft) (6.0)
Requirement already satisfied: torch>=1.13.0 in c:\users\om\anaconda3\
lib\site-packages (from peft) (2.6.0)
Requirement already satisfied: transformers in c:\users\om\anaconda3\
lib\site-packages (from peft) (4.50.0)
Requirement already satisfied: tqdm in c:\users\om\anaconda3\lib\site-
packages (from peft) (4.65.0)
Requirement already satisfied: accelerate>=0.21.0 in c:\users\om\
anaconda3\lib\site-packages (from peft) (1.5.2)
Requirement already satisfied: safetensors in c:\users\om\anaconda3\
```

```
lib\site-packages (from peft) (0.5.2)
Requirement already satisfied: huggingface_hub>=0.25.0 in c:\users\om\
anaconda3\lib\site-packages (from peft) (0.28.1)
Requirement already satisfied: filelock in c:\users\om\anaconda3\lib\
site-packages (from huggingface_hub>=0.25.0->peft) (3.13.1)
Requirement already satisfied: fsspec>=2023.5.0 in c:\users\om\
anaconda3\lib\site-packages (from huggingface_hub>=0.25.0->peft)
(2025.2.0)
Requirement already satisfied: requests in c:\users\om\anaconda3\lib\
site-packages (from huggingface_hub>=0.25.0->peft) (2.31.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\
om\anaconda3\lib\site-packages (from huggingface_hub>=0.25.0->peft)
(4.12.2)
Requirement already satisfied: networkx in c:\users\om\anaconda3\lib\
site-packages (from torch>=1.13.0->peft) (3.1)
Requirement already satisfied: jinja2 in c:\users\om\anaconda3\lib\
site-packages (from torch>=1.13.0->peft) (3.1.2)
Requirement already satisfied: sympy==1.13.1 in c:\users\om\anaconda3\
lib\site-packages (from torch>=1.13.0->peft) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\om\
anaconda3\lib\site-packages (from sympy==1.13.1->torch>=1.13.0->peft)
(1.3.0)
Requirement already satisfied: colorama in c:\users\om\anaconda3\lib\
site-packages (from tqdm->peft) (0.4.6)
Requirement already satisfied: regex!=2019.12.17 in c:\users\om\
anaconda3\lib\site-packages (from transformers->peft) (2022.7.9)
Requirement already satisfied: tokenizers<0.22,>=0.21 in c:\users\om\
anaconda3\lib\site-packages (from transformers->peft) (0.21.1)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\om\
anaconda3\lib\site-packages (from jinja2->torch>=1.13.0->peft) (2.1.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\
om\anaconda3\lib\site-packages (from requests-
>huggingface_hub>=0.25.0->peft) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\om\anaconda3\
lib\site-packages (from requests->huggingface_hub>=0.25.0->peft) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\om\
anaconda3\lib\site-packages (from requests->huggingface_hub>=0.25.0-
>peft) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\om\
anaconda3\lib\site-packages (from requests->huggingface_hub>=0.25.0-
>peft) (2023.11.17)
Downloading peft-0.17.1-py3-none-any.whl (504 kB)
   ---------------------------------------- 0.0/504.9 kB ? eta -:--:--
   ---------------------------------------- 0.0/504.9 kB ? eta -:--:--
    --------------------------------------- 10.2/504.9 kB ? eta
-:--:--
   -- ------------------------------------- 30.7/504.9 kB 660.6 kB/s
eta 0:00:01
   ---- ----------------------------------- 61.4/504.9 kB 409.6 kB/s
```

```
eta 0:00:02
   ----- ---------------------------------- 71.7/504.9 kB 393.8 kB/s
eta 0:00:02
   ---------- ----------------------------- 143.4/504.9 kB 607.9 kB/s
eta 0:00:01
   --------------- ---------------------- 204.8/504.9 kB 734.2 kB/s
eta 0:00:01
   ------------------- ------------------ 256.0/504.9 kB 787.7 kB/s
eta 0:00:01
   --------------------- ---------------- 286.7/504.9 kB 770.1 kB/s
eta 0:00:01
   --------------------- ---------------- 286.7/504.9 kB 770.1 kB/s
eta 0:00:01
   ----------------------- -------------- 317.4/504.9 kB 703.0 kB/s
eta 0:00:01
   ------------------------- ------------ 337.9/504.9 kB 698.7 kB/s
eta 0:00:01
   -------------------------- ----------- 358.4/504.9 kB 676.0 kB/s
eta 0:00:01
   ---------------------------- ------- 399.4/504.9 kB 655.1 kB/s
eta 0:00:01
   ------------------------------ ----- 430.1/504.9 kB 671.7 kB/s
eta 0:00:01
   -------------------------------- --- 460.8/504.9 kB 686.8 kB/s
eta 0:00:01
   ---------------------------------  501.8/504.9 kB 683.6 kB/s
eta 0:00:01
   ---------------------------------- 504.9/504.9 kB 646.6 kB/s
eta 0:00:00
Installing collected packages: peft
Successfully installed peft-0.17.1

#
================================================================================
========
# T5 MODEL TRAINING & SAVING SCRIPT
#
================================================================================
========

import torch
from torch.utils.data import Dataset, DataLoader
import os
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
import json
```

```
#
================================================================================
========
# CONFIGURATION
#
================================================================================
========
CONFIG = {
    'model_name': 't5-base',
    'max_train_samples': 500,
    'max_val_samples': 100,
    'batch_size': 4,
    'num_epochs': 10,
    'learning_rate': 5e-4,
    'gradient_accumulation_steps': 4,
    'max_input_length': 512,
    'max_target_length': 150,
    'save_dir': './trained_model'  # Model will be saved here
}

#
================================================================================
========
# DATASET PREPARATION
#
================================================================================
========
class STEMSummarizationDataset(Dataset):
    def __init__(self, split='train', max_samples=500):
        print(f"Loading {split} split from ScienceQA...")

        # Load dataset without streaming
        try:
            dataset = load_dataset("derek-thomas/ScienceQA",
split=split)
        except:
            print("⚠ Trying alternative loading method...")
            dataset = load_dataset("derek-thomas/ScienceQA")[split]

        self.samples = []

        # Limit samples
        total_samples = min(max_samples, len(dataset))

        for idx in tqdm(range(total_samples), desc="Processing
samples"):
            sample = dataset[idx]

            question = sample.get('question', '')
```

```
            hint = sample.get('hint', '')
            lecture = sample.get('lecture', '')
            solution = sample.get('solution', '')

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

            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)

            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
                })

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

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

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

#
================================================================================
========
# TRAINING FUNCTION
#
================================================================================
========
def train_model():
    print("="*70)
    print("🚀 STARTING T5 MODEL TRAINING")
```

```
    print("="*70)

    # Load datasets
    train_dataset = STEMSummarizationDataset('train',
CONFIG['max_train_samples'])
    val_dataset = STEMSummarizationDataset('validation',
CONFIG['max_val_samples'])

    # Initialize tokenizer
    tokenizer = T5Tokenizer.from_pretrained(CONFIG['model_name'])

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

        inputs = tokenizer(
            input_texts,
            max_length=CONFIG['max_input_length'],
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        targets = tokenizer(
            target_texts,
            max_length=CONFIG['max_target_length'],
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        targets['input_ids'][targets['input_ids'] ==
tokenizer.pad_token_id] = -100
        inputs['labels'] = targets['input_ids']
        return inputs

    # Create dataloaders
    train_loader = DataLoader(train_dataset,
batch_size=CONFIG['batch_size'],
                              shuffle=True, collate_fn=prepare_batch)
    val_loader = DataLoader(val_dataset,
batch_size=CONFIG['batch_size'],
                            shuffle=False, collate_fn=prepare_batch)

    # Initialize model with LoRA
    model =
T5ForConditionalGeneration.from_pretrained(CONFIG['model_name'])
```

```
    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"🚀 Using device: {device}")

    # Optimizer and scheduler
    optimizer = torch.optim.AdamW(model.parameters(),
lr=CONFIG['learning_rate'], weight_decay=0.01)
    total_steps = len(train_loader) * CONFIG['num_epochs']
    scheduler = get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=100, num_training_steps=total_steps)

    # Training loop
    best_val_loss = float('inf')
    history = {'train_loss': [], 'val_loss': []}

    for epoch in range(CONFIG['num_epochs']):
        # Training
        model.train()
        total_loss = 0
        optimizer.zero_grad()

        pbar = tqdm(train_loader, desc=f"Epoch
{epoch+1}/{CONFIG['num_epochs']}")
        for step, batch in enumerate(pbar):
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            loss = outputs.loss /
CONFIG['gradient_accumulation_steps']
            loss.backward()

            if (step + 1) % CONFIG['gradient_accumulation_steps'] ==
0:
                torch.nn.utils.clip_grad_norm_(model.parameters(),
1.0)
                optimizer.step()
                scheduler.step()
                optimizer.zero_grad()
```

```
            total_loss += loss.item() *
CONFIG['gradient_accumulation_steps']
            pbar.set_postfix({'loss': f"{loss.item() *
CONFIG['gradient_accumulation_steps']:.4f}"})

        train_loss = total_loss / len(train_loader)

        # Validation
        model.eval()
        val_loss = 0
        with torch.no_grad():
            for batch in val_loader:
                batch = {k: v.to(device) for k, v in batch.items()}
                outputs = model(**batch)
                val_loss += outputs.loss.item()

        val_loss = val_loss / len(val_loader)

        history['train_loss'].append(train_loss)
        history['val_loss'].append(val_loss)

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

        # Save best model
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            print(f" New best model! Saving...")
            save_model(model, tokenizer, CONFIG['save_dir'])

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

    print("\n" + "="*70)
    print(" TRAINING COMPLETE!")
    print(f" Model saved to: {CONFIG['save_dir']}")
    print("="*70)

    # Save training history
    with open(os.path.join(CONFIG['save_dir'],
'training_history.json'), 'w') as f:
        json.dump(history, f, indent=2)

    return model, tokenizer

#
==========================================================================
========
# MODEL SAVING FUNCTION
```

```
#
================================================================================
========
def save_model(model, tokenizer, save_dir):
    """Save the trained model and tokenizer"""
    os.makedirs(save_dir, exist_ok=True)

    # Save LoRA adapter weights
    model.save_pretrained(save_dir)

    # Save tokenizer
    tokenizer.save_pretrained(save_dir)

    # Save config
    with open(os.path.join(save_dir, 'config.json'), 'w') as f:
        json.dump(CONFIG, f, indent=2)

    print(f"□ Model saved to {save_dir}")

#
================================================================================
========
# MAIN EXECUTION
#
================================================================================
========
if __name__ == "__main__":
    trained_model, tokenizer = train_model()

    print("\n□ To use the trained model later:")
    print(f"""
    from transformers import T5ForConditionalGeneration, T5Tokenizer
    from peft import PeftModel

    # Load base model
    base_model = T5ForConditionalGeneration.from_pretrained('t5-base')

    # Load LoRA weights
    model = PeftModel.from_pretrained(base_model,
'{CONFIG['save_dir']}')
    tokenizer = T5Tokenizer.from_pretrained('{CONFIG['save_dir']}')

    # Generate summary
    text = "Your STEM content here..."
    inputs = tokenizer(f"summarize: {{text}}", return_tensors='pt',
max_length=512, truncation=True)
    outputs = model.generate(**inputs, max_length=150, num_beams=4)
    summary = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

```
    print(summary)
    """)
```

```
================================================================================
🚀 STARTING T5 MODEL TRAINING
================================================================================
Loading train split from ScienceQA...
Downloading and preparing dataset None/None to
file://C:/Users/OM/.cache/huggingface/datasets/derek-
thomas___parquet/derek-thomas--ScienceQA-
ca4903a3b5795914/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b9344
92fdef7d8d6f236ec...
```


```
{"model_id":"8f5dc7ac00e44b49ba4a1dbd4a4a266f","version_major":2,"vers
ion_minor":0}

{"model_id":"b8a9be7b98184c6fbb40afa0c4452f99","version_major":2,"vers
ion_minor":0}

{"model_id":"304e1df8edef4fdca492c8e8fa8e3794","version_major":2,"vers
ion_minor":0}

{"model_id":"9688d9624f734133a6a1dfc9426daf29","version_major":2,"vers
ion_minor":0}

{"model_id":"e2e45fe6a5644d46988c900853c28cdd","version_major":2,"vers
ion_minor":0}

{"model_id":"cc8926cd756e497c9d79b41ed877a5f7","version_major":2,"vers
ion_minor":0}

{"model_id":"c08f265b652949fb80e41dcc22c4222a","version_major":2,"vers
ion_minor":0}
```


```
The Kernel crashed while executing code in the current cell or a
previous cell.

Please review the code in the cell(s) to identify a possible cause of
the failure.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for
more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further
details.
```