

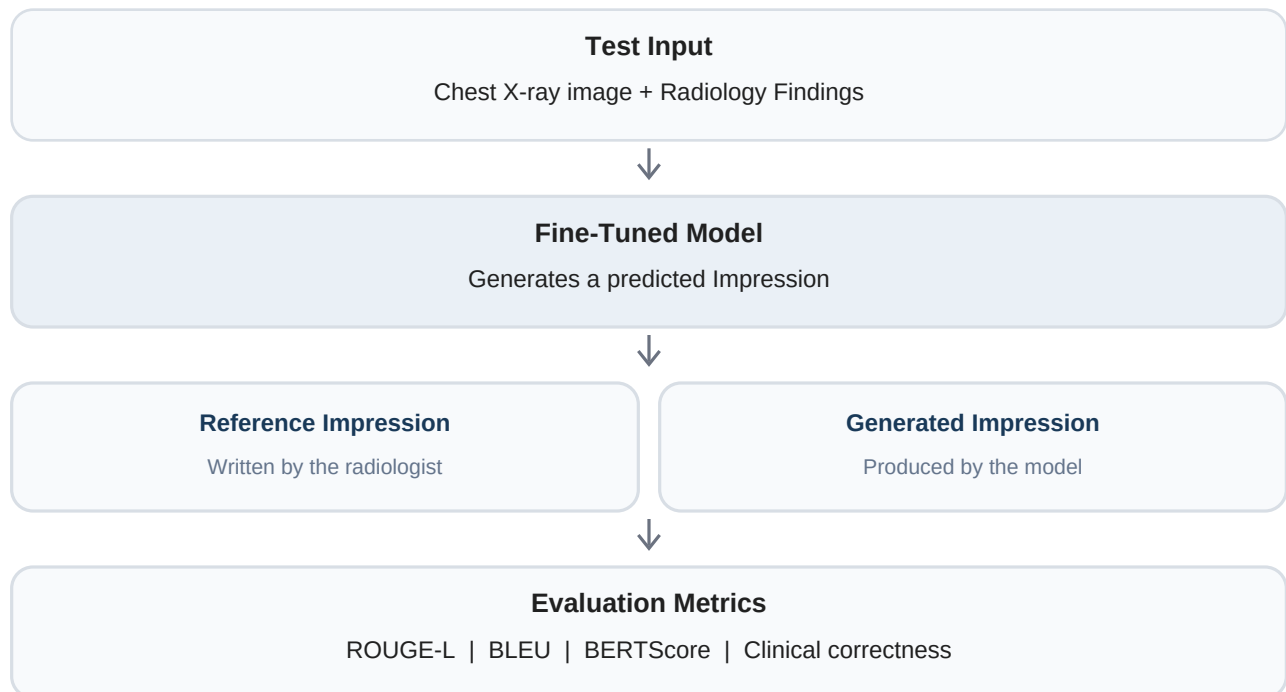
Testing a Fine-Tuned Radiology Model: Text Similarity and Evaluation Metrics

These instructions explain how to evaluate a fine-tuned generative AI model that produces a radiology Impression from a chest X-ray image and its associated Findings.

The goal is to determine how closely the model's generated Impression matches the original radiologist-written Impression in a held-out test dataset.

1. Understanding the testing task

For a dataset such as ReXGradient-160K, the evaluation workflow is:



Important: A generated Impression does not need to use exactly the same words as the radiologist's Impression to be correct. However, it must preserve the relevant clinical meaning.

Reference: No acute cardiopulmonary abnormality.

Generated: No evidence of acute heart or lung disease.

These statements have similar clinical meaning despite using different wording. By contrast, changing *no pleural effusion* to *small pleural effusion* is a clinically important error even though the sentences share many words.

2. Recommended evaluation metrics

Metric	What it measures	Best use
ROUGE-L	Longest matching word sequence between reference and generated text	Text overlap
BLEU	Matching word sequences (n-grams), with a brevity penalty	Wording similarity
BERTScore	Semantic similarity using contextual language embeddings	Similar meaning with different wording
Clinical accuracy	Agreement on abnormalities, negation, laterality, severity, etc.	Medical correctness

Recommended starting approach: Use ROUGE-L and BERTScore as your primary text-similarity metrics, optionally report BLEU, and include a clinical correctness evaluation for a meaningful assessment of radiology performance. Text similarity alone is not sufficient to establish clinical safety or diagnostic accuracy.

3. Prepare a test dataset

Create a separate test set that was not used for training or model selection. A reasonable starting point is 100-200 examples for initial testing, followed by a larger evaluation set for more reliable results.

Each example should contain the chest X-ray image, Findings, and the original Impression. For the test data, keep the radiologist's Impression separate from the model input. Otherwise, the model may simply reproduce the answer rather than generate it.

Example test.jsonl

```
{ "id": "001", "image": "images/xray001.png", "findings": "The lungs are clear. No pleural effusion or pneumothorax.", "impression": "No acute cardiopulmonary abnormality." }
{ "id": "002", "image": "images/xray002.png", "findings": "Mild enlargement of the cardiac silhouette. No pulmonary edema.", "impression": "Mild cardiomegaly without pulmonary edema." }
{ "id": "003", "image": "images/xray003.png", "findings": "Small left pleural effusion with adjacent basilar opacity.", "impression": "Small left pleural effusion and associated basilar opacity." }
```

Illustrative examples, not actual ReXGradient records. This JSONL format is for your evaluation script; adapt the request format to the model API you are using.

For studies with multiple images, preserve the study-level grouping. Do not split images from the same study across training and testing. Prefer patient-level separation when patient identifiers are available.

4. Generate predictions using the fine-tuned model

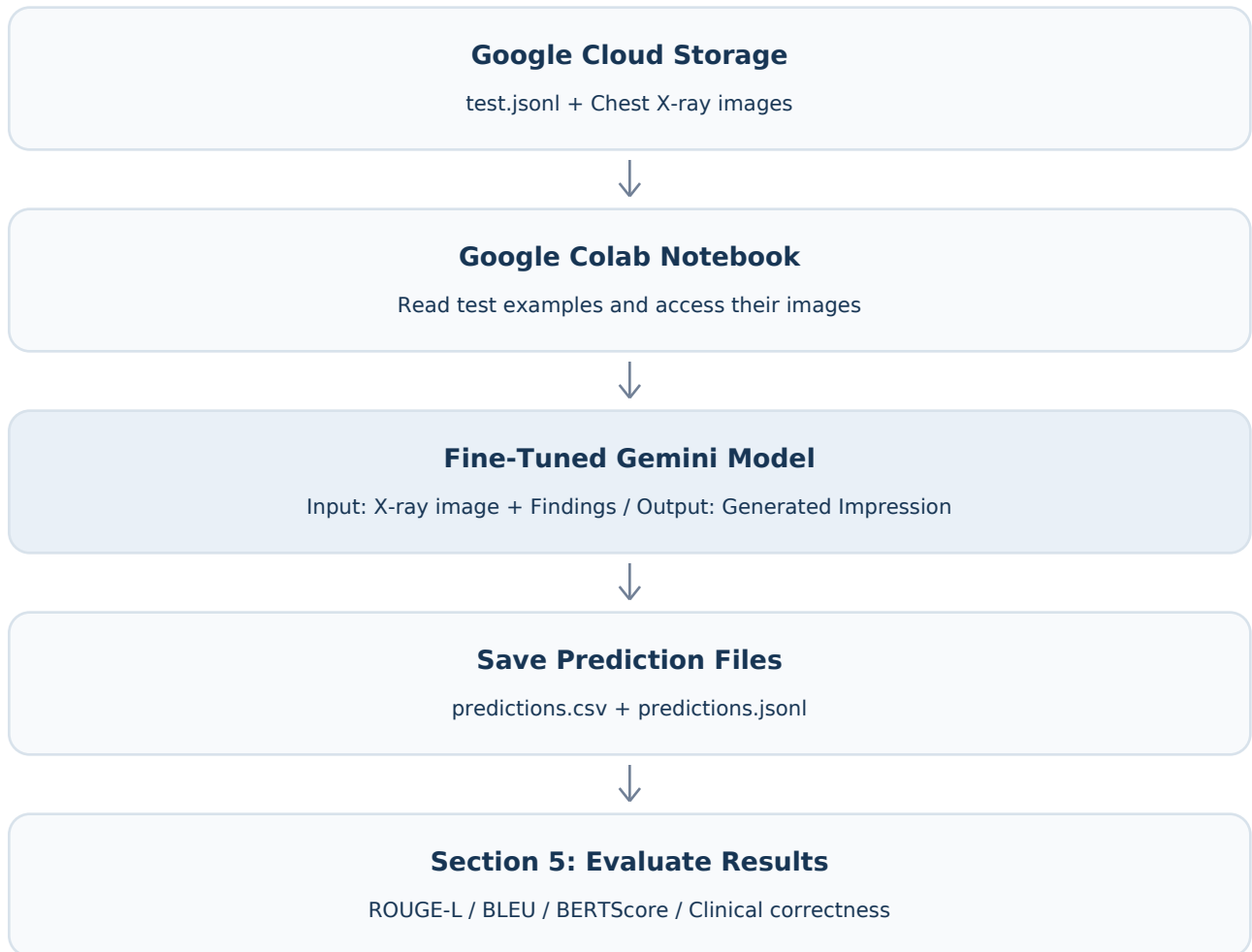
For every test example: (1) Read the image and Findings. (2) Send them to the fine-tuned model using the same prompt structure used during training. (3) Capture the generated Impression. (4) Save the generated Impression and original reference Impression together.

Your resulting file could be named predictions.csv:

4. Expanded: Generate Predictions Using Google Cloud Storage + Google Colab

This section assumes you have already fine-tuned your model using radiology data, and that both test.jsonl and the associated chest X-ray images are stored in Google Cloud Storage (GCS). The objective is to run each held-out test example through the tuned model, collect its generated Impression, and produce the files needed for the Section 5 metrics.

4.1. Overall testing workflow



4.2. Organize the files in Google Cloud Storage

```
gs://my-radiology-bucket/  
  test/  
    test.jsonl  
  images/  
    xray001.png  
    xray002.png  
    xray003.png  
  results/
```

The results/ location stores predictions and evaluation outputs. An illustrative test.jsonl could contain:

```
{
  "id": "001",
  "image": "gs://my-radiology-bucket/images/xray001.png",
  "findings": "The lungs are clear. No pleural effusion or pneumothorax.",
  "impression": "No acute cardiopulmonary abnormality."
},
{
  "id": "002",
  "image": "gs://my-radiology-bucket/images/xray002.png",
  "findings": "Mild enlargement of the cardiac silhouette. No pulmonary edema.",
  "impression": "Mild cardiomegaly without pulmonary edema."
},
{
  "id": "003",
  "image": "gs://my-radiology-bucket/images/xray003.png",
  "findings": "Small left pleural effusion with adjacent basilar opacity.",
  "impression": "Small left pleural effusion and associated basilar opacity."
}
```

These are illustrative examples, not actual ReXGradient records. If the file uses native Gemini multimodal JSONL, extract the image URI, Findings and reference Impression from its nested structure. Never include the reference Impression in the model input.

4.3. Open Google Colab and authenticate

```
!pip install -q --upgrade google-genai google-cloud-storage pandas
```

```
from google.colab import auth
auth.authenticate_user()
import google.auth
credentials, project_id = google.auth.default()
print("Authenticated project:", project_id)
```

Your authenticated account must have permission to read the Cloud Storage objects and invoke the tuned model.

4.4. Configure your project, bucket, and model

```
from google import genai
from google.genai import types
from google.cloud import storage

PROJECT_ID = "your-google-cloud-project-id"
LOCATION = "us-central1"
BUCKET_NAME = "my-radiology-bucket"
TEST_FILE = "test/test.jsonl"
TUNED_MODEL = "your-tuned-model-id"

client = genai.Client(vertexai=True, project=PROJECT_ID,
                      location=LOCATION)
storage_client = storage.Client(project=PROJECT_ID)
bucket = storage_client.bucket(BUCKET_NAME)
```

Use the tuned model resource or endpoint identifier provided by Vertex AI, not automatically the base model name.

4.5. Read the test examples from Cloud Storage

```
import json
blob = bucket.blob(TEST_FILE)
test_data = [json.loads(line)
              for line in blob.download_as_text().splitlines()
              if line.strip()]
print("Number of test examples:", len(test_data))
print("First example:", test_data[0])
```

The test data is now loaded into Colab, but no predictions have yet been generated.

4.6. Run a prediction for one test example

Test one sample first to confirm that the endpoint and image URI work.

```
sample = test_data[0]
prompt = f"""You are generating the Impression section of a
radiology report. Use the chest X-ray image and Findings to
produce a concise Impression.
Findings: {sample['findings']}
Return only the Impression."""

response = client.models.generate_content(
    model=TUNED_MODEL,
    contents=[
        types.Part.from_uri(file_uri=sample["image"],
                           mime_type="image/png"),
        prompt
    ],
    config=types.GenerateContentConfig(
        temperature=0, max_output_tokens=256
    )
)
print("Reference Impression:")
print(sample["impression"])
print("\nGenerated Impression:")
print(response.text)
```

This assumes PNG images; for JPEG use image/jpeg. The reference is displayed only for comparison and must not appear in the request.

4.7. Run predictions on the entire test dataset

```
import pandas as pd
import time
results = []
for i, sample in enumerate(test_data):
    prompt = f"""Generate a concise radiology Impression
    from the supplied chest X-ray image and Findings.
    Findings: {sample['findings']}
    Return only the Impression."""
    try:
        response = client.models.generate_content(
            model=TUNED_MODEL,
            contents=[
                types.Part.from_uri(file_uri=sample["image"],
                                    mime_type="image/png"),
                prompt
            ],
            config=types.GenerateContentConfig(
                temperature=0, max_output_tokens=256
            )
        )
        prediction = response.text or ""
        status = "success" if prediction.strip() else "empty"
    except Exception as e:
        prediction = ""
        status = "error"
        print(f"Error on example {sample['id']}: {e}")
    results.append({
        "id": sample["id"], "image": sample["image"],
        "findings": sample["findings"],
        "reference": sample["impression"],
        "prediction": prediction, "status": status
    })
    print(f"Processed {i+1}/{len(test_data)} - {status}")
    time.sleep(1)
df = pd.DataFrame(results)
print(df.head())
```

For large test sets, add exponential-backoff retries and checkpointing. A one-second delay does not guarantee that API quotas will be respected.

4.8. Create the output files for evaluation

predictions.csv is the main file used for ROUGE-L, BLEU and BERTScore. predictions.jsonl contains the same results in JSONL format.

id	reference	prediction	status
001	No acute cardiopulmonary abnormality.	No acute heart or lung abnormality.	success
002	Mild cardiomegaly without pulmonary edema.	Mild cardiac enlargement. No pulmonary edema.	success
003	Small left pleural effusion and associated basilar opacity.	Small left pleural effusion with basilar opacity.	success

```
df.to_csv("predictions.csv", index=False)
df.to_json("predictions.jsonl", orient="records",
           lines=True, force_ascii=False)
```

```

bucket.blob("results/predictions.csv").upload_from_filename(
    "predictions.csv")
bucket.blob("results/predictions.jsonl").upload_from_filename(
    "predictions.jsonl")
print("Prediction files saved to Cloud Storage.")

```

The GCS results/ folder now contains predictions.csv and predictions.jsonl.

4.9. Prepare the predictions for Section 5 metric analysis

```

valid_df = df[
    (df["status"] == "success") &
    df["reference"].notna() &
    df["prediction"].notna() &
    (df["prediction"].str.strip() != "")
].copy()
valid_df.to_csv("predictions_for_metrics.csv", index=False)
bucket.blob(
    "results/predictions_for_metrics.csv"
).upload_from_filename("predictions_for_metrics.csv")
print("Total examples:", len(df))
print("Valid predictions:", len(valid_df))
print("Failed or empty:", len(df) - len(valid_df))

```

Only successful nonempty predictions should be used in text-similarity calculations. Report the total number of cases and all failures; do not silently omit failed cases.

4.10. Final output files and their purpose

File	Purpose
test.jsonl	Original held-out examples and reference Impressions
predictions.csv	All predictions including failed cases
predictions.jsonl	Same results in machine-readable JSONL
predictions_for_metrics.csv	Successful predictions for Section 5
evaluation_results.csv	Per-example ROUGE-L and BERTScore from Section 5

Important considerations: Adapt the code to your actual JSONL schema and image MIME types. Ensure the tuned model supports image-plus-text input and has permission to read GCS images. Use consistent inference settings, preserve failed cases, and never send the reference Impression as model input.

This workflow connects Google Cloud fine-tuning, Colab inference, Cloud Storage output files and Section 5 evaluation.

id	reference	prediction
001	No acute cardiopulmonary abnormality.	No acute heart or lung abnormality.
002	Mild cardiomegaly without pulmonary edema.	Mild cardiac enlargement. No pulmonary edema.
003	Small left pleural effusion and associated basilar opacity.	Small left pleural effusion with basilar opacity.

For a fair comparison, keep the prompt, generation temperature, and other inference settings consistent across all test examples.

5. Calculate text-similarity metrics in Google Colab

You can perform this testing in a separate Google Colab notebook after the model has generated its predictions.

Step A - Install evaluation libraries

```
!pip install -q evaluate rouge_score bert_score sacrebleu pandas
```

Step B - Upload your prediction file

```
from google.colab import files
import pandas as pd

uploaded = files.upload()
df = pd.read_csv("predictions.csv")
print(df.head())
print("Number of examples:", len(df))
```

Step C - Calculate ROUGE-L, BLEU and BERTScore

```
import evaluate

# Load metrics
rouge = evaluate.load("rouge")
bleu = evaluate.load("bleu")
bertscore = evaluate.load("bertscore")

# Remove rows with missing text
df = df.dropna(subset=["reference", "prediction"]).copy()
references = df["reference"].astype(str).tolist()
predictions = df["prediction"].astype(str).tolist()

# ROUGE-L
rouge_result = rouge.compute(
    predictions=predictions,
    references=references
)

# BLEU (corpus-level)
bleu_result = bleu.compute(
    predictions=predictions,
    references=[[r] for r in references]
)

# BERTScore
bert_result = bertscore.compute(
    predictions=predictions,
```

```

    references=references,
    lang="en"
)

# Display results
print("ROUGE-L:", round(rouge_result["rougeL"], 4))
print("BLEU:", round(bleu_result["bleu"], 4))
print("BERTScore F1:",
      round(sum(bert_result["f1"]) / len(bert_result["f1"]), 4))

```

The Hugging Face Evaluate library supports loading and computing these metrics using its standard API.

Step D - Save individual results

BERTScore provides individual scores for each example. ROUGE-L can also be calculated separately for each pair.

```

df["BERTScore_F1"] = bert_result["f1"]

df["ROUGE_L"] = [
    rouge.compute(
        predictions=[pred],
        references=[ref]
    )["rougeL"]
    for pred, ref in zip(predictions, references)
]

df.to_csv("evaluation_results.csv", index=False)

print(df[[
    "id", "reference", "prediction",
    "ROUGE_L", "BERTScore_F1"
]].head())

```

Note that BLEU is best reported across the full dataset rather than interpreted as a reliable score for each very short Impression.

6. Interpret the results

Suppose you obtain these hypothetical overall results:

Metric	Hypothetical result
ROUGE-L	0.71
BLEU	0.46
BERTScore F1	0.91

Higher scores generally indicate greater similarity to the reference. However, there are no universal cutoffs such as "BERTScore above 0.90 means clinically correct." Scores depend on the dataset, metric implementation, reference wording, and evaluation configuration.

The best use of these metrics is comparing different models on exactly the same test examples. For example, you could evaluate a base model and a fine-tuned model to determine whether fine-tuning improved performance.

7. Add clinical correctness evaluation

Because these are radiology reports, text similarity should not be the only evaluation. A model could produce a highly similar sentence while making a clinically important mistake.

Reference	Generated	Problem
No pneumothorax	Small pneumothorax	Incorrect negation
Right pleural effusion	Left pleural effusion	Incorrect laterality
Mild cardiomegaly	Severe cardiomegaly	Incorrect severity
No focal consolidation	No acute abnormality	Potentially acceptable paraphrase depending on the full case

For a more advanced evaluation, consider:

RadGraph F1: Compares clinical entities and relationships extracted from reports.

CheXbert: Compares chest X-ray observation labels, including positive, negative, and uncertain findings.

Radiologist review: Identifies clinically significant errors that automated metrics might miss.

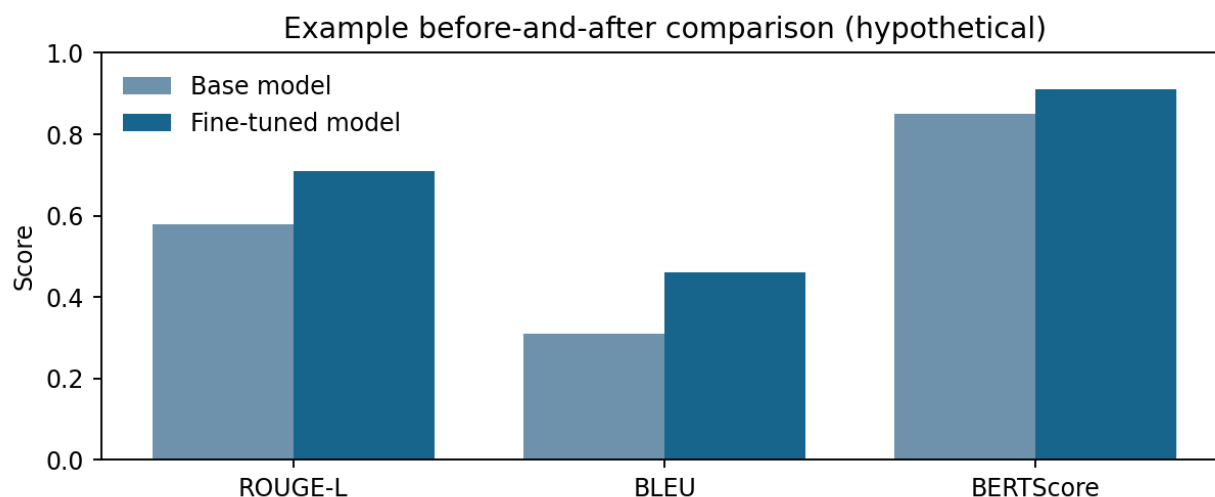
Research on radiology report generation has found that RadGraph F1 and BERTScore align relatively well with radiologist assessments, although neither replaces expert review.

8. Compare the base model and fine-tuned model

Ideally, evaluate both models using the same test dataset and inference settings.

Metric	Base model	Fine-tuned model
ROUGE-L	0.58	0.71
BLEU	0.31	0.46
BERTScore F1	0.85	0.91
Clinically significant error rate	12%	7%

Illustrative results only. The clinical error rate requires a defined annotation protocol and appropriate review.



This comparison helps answer whether fine-tuning actually improved the model's performance rather than simply producing plausible-looking responses.

9. What to include in your testing report

Test dataset: Source, number of examples, and how it was separated from training.

Model configuration: Base and fine-tuned model identifiers, prompt, and inference parameters.

Text similarity: ROUGE-L, BLEU, and BERTScore F1.

Example predictions: At least five reference/generated Impression comparisons, including errors.

Analysis: What improved, what failed, and whether any clinically important information was lost.

Conclusion: Whether the fine-tuned model improved over the base model and what further testing is needed.

Special note: This evaluation approach also applies to image question-answering tasks, where you compare the model's generated answer against a reference answer. However, the appropriate metrics depend on the task. Exact-match accuracy may be useful for short, constrained answers, whereas semantic similarity may be more appropriate for open-ended answers.

For the radiology Findings + Image to Impression task, start with ROUGE-L + BERTScore F1, add BLEU for comparison, and incorporate RadGraph F1 or a structured clinical error review when feasible.

Resources: <https://huggingface.co/datasets/rajpurkarlab/ReXGradient-160K> | <https://huggingface.co/docs/evaluate/index>