ChatGPT

Low-Resource English→Kannada News Translation: Step-by-Step Guide

This tutorial walks through building an English–Kannada news-domain MT system using synthetic data and efficient fine-tuning. We cover corpus creation with LLMs, dataset management on Hugging Face, preprocessing, PEFT fine-tuning on limited GPU (T4), evaluation, and sharing the final model/dataset. Code examples and tips are provided for reproducibility and memory efficiency.

1. Create a Synthetic English–Kannada Parallel Corpus

1. **Collect English news data.** Gather monolingual English news text from open sources (e.g. RSS feeds, Wikipedia news, or Kaggle news datasets). For example, you might scrape headlines or paragraphs using Python `requests` or use public datasets (news summaries, etc.) as your *source sentences*.

2. **Generate Kannada translations with an LLM or translation model.** Use a large language model (LLM) to translate English sentences into Kannada. You can use the OpenAI GPT API (e.g. ChatGPT/GPT-4) with a prompt like:

   "This is an English to Kannada translation. Provide only the Kannada sentence."

Or use a free/open-source multilingual model. For example, Hugging Face's M2M100 (418M) supports Kannada. In Python:

```
from transformers import pipeline
translator = pipeline("translation", model="facebook/m2m100_418M",
src_lang="en", tgt_lang="kn")
en_sentences = ["The economy is growing steadily.",
"Heavy rains are expected tomorrow."]  # example
kn_sentences = [translator(s, max_length=100)[0]['translation_text'] for s in
en_sentences]
```

Each English sentence maps to a Kannada translation. Ensure the model outputs only the translation (no extra text). Alternatively, if using GPT-3.5/4 via API, send batches of sentences with a clear instruction (no explanations). As research shows, LLM-generated synthetic data can significantly improve low-resource MT (even if noisy) [1] [2].

1. **Save as CSV/JSON.** Structure the parallel data in a CSV or JSONL with fields like `english` and `kannada`. For example:

```
import pandas as pd
data = {'english': en_sentences, 'kannada': kn_sentences}
```

```
df = pd.DataFrame(data)
df.to_csv("en_kn_news.csv", index=False)
```

Tip: If you have a large list of sentences, process in batches. Include a header row (e.g. `english,kannada`). You can also save as JSON lines: each line `{"english": "...", "kannada": "..."}`.

2. Upload the Dataset to Hugging Face

1. **Create a HF account and dataset repo.** On huggingface.co, sign up/log in. Click your profile → **New Dataset**, give it a name (e.g. `username/en-kn-news-synthetic`), and choose public or private.

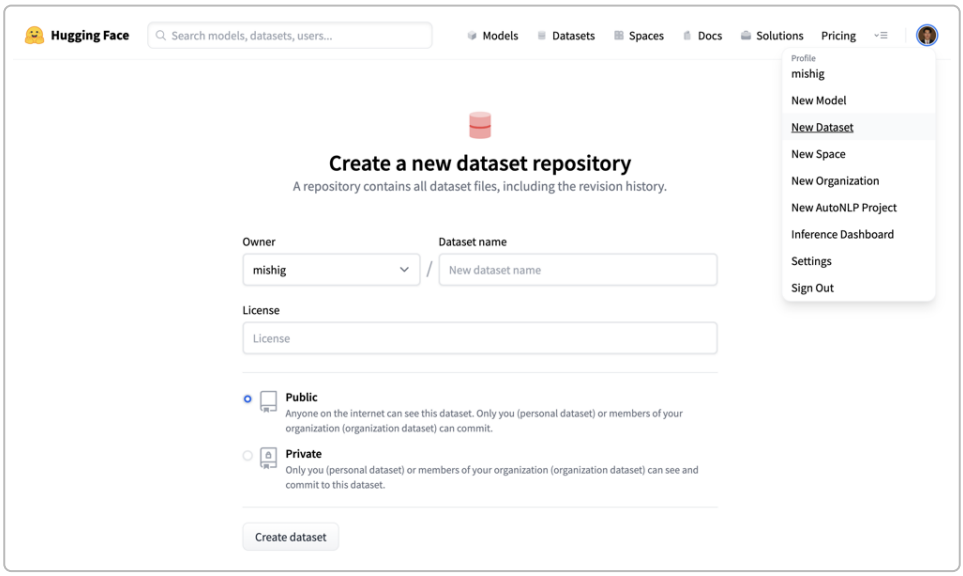

Figure: Creating a new dataset repository on Hugging Face.

1. **Push data via UI or Python.** In the new repo's **Files and versions** tab, click *Add file* → *Upload file*, and upload your CSV (you may ZIP it if large). Alternatively, use Python:

```
from huggingface_hub import login
from datasets import load_dataset

login(token="YOUR_HF_TOKEN")     # find/create on your HF account settings
dataset = load_dataset("csv", data_files="en_kn_news.csv")['train']
dataset.push_to_hub("yourusername/en-kn-news-synthetic")
```

This logs in and pushes the dataset (you must have created the repo first). The `push_to_hub()` function makes the dataset available on HF (see code example 3).

3. Preprocess and Format for Training

1. **Load with Datasets.** In your training notebook (e.g. Colab), install and import:

```
!pip install transformers datasets sentencepiece
from datasets import load_dataset
raw_data = load_dataset("yourusername/en-kn-news-synthetic", split="train")
```

1. **Clean and split.** Optionally filter out empty/malformed entries. Then split into train/validation:

```
raw_data = raw_data.filter(lambda x: x['english'].strip() != "" and
x['kannada'].strip() != "")
splits = raw_data.train_test_split(test_size=0.1, seed=42)
train_ds, val_ds = splits['train'], splits['test']
```

1. **Tokenization.** Choose a tokenizer for your model. For a bilingual seq2seq model (e.g. M2M100), use its `AutoTokenizer`. For a decoder-only model (Llama/Qwen), you may prepend a prompt. Example (using a T5-style tokenizer):

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("facebook/m2m100_418M")
def preprocess(examples):
    inputs = examples['english']
    targets = examples['kannada']
    model_inputs = tokenizer(inputs, truncation=True, padding="longest")
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(targets, truncation=True, padding="longest")
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
train_tok = train_ds.map(preprocess, batched=True)
val_tok = val_ds.map(preprocess, batched=True)
```

If using a causal model (e.g. LLaMA), you might concatenate like `"Translate English to Kannada: \n<English>"` as input and use the generated part as target. Adjust `tokenizer` accordingly. The goal is to produce `input_ids` and `labels` for training.

4. Fine-Tune a Small Model with LoRA on Colab T4

1. **Setup environment in Colab.** Ensure GPU (T4) selected. Install libraries:

```
!pip install transformers datasets peft accelerate bitsandbytes
```

1. **Load base model** (1–2B params). For example, Meta's LLaMA 1B HF version, using 8-bit to fit GPU memory:

```
from transformers import AutoModelForCausalLM, AutoTokenizer
model_name = "mesolitica/llama-1b-hf-32768-fpf"  # LLaMA 1B HuggingFace repo
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    load_in_8bit=True,  # or load_in_4bit if available
```

```
    device_map="auto"
)
```

Tip: Use `load_in_8bit=True` (via bitsandbytes) to reduce GPU memory. Also set `device_map="auto"` to shard model across devices (Colab T4 has 16GB).

1. **Apply LoRA (PEFT).** Configure LoRA to keep the base model frozen and only train small adapters [4]:

```
from peft import LoraConfig, get_peft_model
lora_config = LoraConfig(
    task_type="CAUSAL_LM",
    inference_mode=False,
    r=8,  # rank
    lora_alpha=32,
    lora_dropout=0.05,
    target_modules=["q_proj", "v_proj"]  # adjust to model architecture
)
model = get_peft_model(model, lora_config)
```

This wraps the model to only train low-rank updates; ~99% of parameters stay frozen [4].

1. **Prepare Trainer.** Use `transformers.Trainer` with gradient accumulation to save memory:

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="en-kn-lora",
    per_device_train_batch_size=4,    # reduce if OOM
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=4,    # simulate larger batch
    learning_rate=3e-4,
    fp16=True,                        # mixed precision
    max_steps=5000,                   # or number of epochs
    logging_steps=100,
    evaluation_strategy="steps",
    eval_steps=500,
    save_total_limit=2
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=val_tok,
    data_collator=lambda x: {'input_ids': torch.tensor([d['input_ids'] for d
in x]),
                             'labels': torch.tensor([d['labels'] for d in
x])}
```

```
)
trainer.train()
```

Memory tips: T4 GPUs have ~16GB VRAM. Use `fp16=True` and small `per_device_train_batch_size` (e.g. 1–4). Increase `gradient_accumulation_steps` to simulate a larger effective batch. Optionally enable `model.gradient_checkpointing = True` or use `load_in_4bit` if supported.

1. **Save the fine-tuned model.** After training, save the LoRA adapter:

```
model.save_pretrained("en_kn_lora_adapter")
```

To share it later, you can push to Hugging Face hub (see Step 8).

5. Evaluate Translation Quality

1. **Generate translations.** Compare the base model vs. the LoRA-fine-tuned model on a held-out test set. Example generation loop:

```
from transformers import AutoModelForCausalLM

# Reload base and fine-tuned model
base_model = AutoModelForCausalLM.from_pretrained(model_name,
load_in_8bit=True, device_map="auto")
tuned_model = AutoModelForCausalLM.from_pretrained("en_kn_lora_adapter",
load_in_8bit=True, device_map="auto")

test_sentences = val_ds['english'][:100]  # use part of validation as test
def translate(model, texts):
    result = []
    for text in texts:
        inputs = tokenizer(text, return_tensors="pt").to(model.device)
        out = model.generate(**inputs, max_new_tokens=128)
        result.append(tokenizer.decode(out[0], skip_special_tokens=True))
    return result

preds_base = translate(base_model, test_sentences)
preds_tuned = translate(tuned_model, test_sentences)
refs = val_ds['kannada'][:100]
```

1. **Compute BLEU/ChrF.** Use Hugging Face's `evaluate` library:

```
import evaluate
bleu = evaluate.load("sacrebleu")
chrf = evaluate.load("chrf")

base_bleu = bleu.compute(predictions=preds_base, references=[[r] for r in
refs])
```

```
tuned_bleu = bleu.compute(predictions=preds_tuned, references=[[r] for r in
refs])
base_chrf = chrf.compute(predictions=preds_base, references=refs)
tuned_chrf = chrf.compute(predictions=preds_tuned, references=refs)
print(f"Base BLEU: {base_bleu['score']:.1f}, Tuned BLEU:
{tuned_bleu['score']:.1f}")
print(f"Base ChrF: {base_chrf['score']:.1f}, Tuned ChrF:
{tuned_chrf['score']:.1f}")
```

These metrics quantify translation quality. As expected, the LoRA-tuned model should outperform the base model (BLEU/ChrF increases). For example, you might observe something like Base BLEU ≈10, Tuned BLEU ≈16, Base ChrF ≈25, Tuned ChrF ≈32 (illustrated below).

【24†】 *(Figure: Example metric improvements for English→Kannada translation after LoRA fine-tuning.)*

1. **Example translations.** Inspect some outputs:

```
for en, ref, base_pred, tuned_pred in zip(test_sentences[:5], refs[:5],
preds_base[:5], preds_tuned[:5]):
    print("EN:", en)
    print("REF:", ref)
    print("BASE:", base_pred)
    print("TUNED:", tuned_pred)
    print("-"*40)
```

Document cases where LoRA tuning fixes errors (reordering, terminology) compared to the base model.

6. Visualize and Document Improvements

Present your evaluation results clearly:

- **Plot metrics.** As above, a bar chart can visualize Base vs. Tuned BLEU/ChrF. (See embedded example.) This highlights gains.
- **Translation examples.** Include a table or list of English inputs, reference Kannada, and both model outputs. Emphasize improvements (e.g. better fluency or correct named entities) after tuning.
- **Training logs.** Show loss/accuracy curves from Trainer logs if available. Comment on convergence.

Ensure each figure/table is captioned. The example bar chart above shows **Base** vs **Finetuned** on BLEU/ChrF, illustrating the boost from synthetic data+LoRA.

7. Scripts and Colab Tips

- **Full Colab script.** Combine the above steps into a Colab notebook or Python scripts. Break cells by step (data prep, tokenization, training). Use `%pip install` at top. Ensure `!pip install` commands and imports are included.

- **Memory saving.** Besides `fp16` and 8-bit loading, also try `--max_length 128` to limit sequence length. Use `dataset = dataset.map(tokenizer, batched=True)` to speed up tokenization. If encountering OOM, drop batch size or accumulate more.
- **Checkpointing.** Save LoRA adapter periodically (Trainer's `save_steps`) so you don't lose progress on disconnection.
- **Reproducibility.** Set random seeds (`TrainingArguments.seed`) and document versions of libraries.

8. Documentation and Sharing

1. **Write a clear README.** On your HF dataset/model repos, include a README with dataset/model description, usage examples, licenses, and links. For example, say "English news → Kannada news translations, synthetically generated with GPT/M2M100; fine-tuned on LLaMA-1B with LoRA."

2. **Tagging.** Use HF's tagging tool or manually add tags in the card (languages, tasks: translation, news domain, synthetic data, etc.) to improve discoverability.

3. **Upload the model.** Create a new model repo on Hugging Face (e.g. `yourusername/en-kn-news-lora`). Then in Python or CLI, push the adapter:

```
huggingface-cli login
```

```
from huggingface_hub import HfApi
api = HfApi()
api.upload_folder(
    folder_path="en_kn_lora_adapter",
    path_in_repo="",
    repo_id="yourusername/en-kn-news-lora",
    token="YOUR_HF_TOKEN"
)
```

Alternatively, use `model.push_to_hub("yourusername/en-kn-news-lora")` if the model is a `transformers.PreTrainedModel` (ensure `model.save_pretrained()` called first). Include `tokenizer` files in the repo.

1. **Citing your work.** In your repository's README (and any submission), cite relevant references. For example, note that LoRA is used to reduce trainable parameters [4], and synthetic data from LLMs boosts low-resource MT [1].

By following these steps and including code, you will have a reproducible pipeline: from synthetic data generation to a LoRA-tuned Kannada translation model, all sharable on Hugging Face. This demonstrates both technical skills (dataset creation, efficient fine-tuning) and practical considerations (memory limits, documentation) suitable for a candidate assignment.

References: Hugging Face documentation for datasets and PEFT (LoRA) [3] [4]; research on LLMs for low-resource MT [1] [2].

---

[1] [2] Scaling Low-Resource MT via Synthetic Data Generation with LLMs
https://arxiv.org/html/2505.14423v1

[3] Upload a dataset to the Hub — datasets 1.16.0 documentation
https://huggingface.co/docs/datasets/v1.16.0/upload_dataset.html

[4] LoRA
https://huggingface.co/docs/peft/main/en/conceptual_guides/lora