Work Toward Your Project — Milestone 3

Prompt–Engineer a Pre-Trained Model with Hugging Face Pipelines

Milestone focus

You will **select a suitable Hugging Face pre-trained model** for your task, **build a pipeline**, then **prompt-engineer and evaluate** zero-shot and few-shot prompts. You will **compare against your Week-2 RNN baseline** and include a brief **prompt rationale** as the interpretability artefact.

What you will deliver (end of Week 3)

- **`milestone3.ipynb`** containing: (a) model selection + pipeline, (b) zero-shot & few-shot prompts, (c) metrics vs RNN baseline, (d) latency & compliance checks, and (e) a short **prompt rationale** (interpretability artefact).

Step-by-step instructions

0) Setup (Colab)

```
!pip -q install --upgrade transformers datasets accelerate evaluate scikit-learn pandas
import time, statistics, re, json, pandas as pd
from transformers import pipeline, set_seed
set_seed(42)
DEVICE = 0  if torch.cuda.is_available() else -1
```

---

1) Define task & metrics (reuse Week-1/2 choices)

State your task (e.g., **sentiment classification**, **topic tagging**, **summary generation**) and the **metric** you will report (e.g., **Accuracy/F1** for classification; **ROUGE** for summarization). Keep **the same dev/test split** you used in earlier milestones so the comparison is fair.

---

2) Shortlist and pick a checkpoint (justify)

Make a 2-model shortlist, then pick **one** model, citing:

- **Domain fit** (e.g., Twitter-Roberta for tweets; FinBERT for finance).
- **Latency/size** constraints (DistilBERT/MiniLM for speed; larger for accuracy).
- **License** and **sequence length** needs if relevant.

Tip: "Understand → encoder; Generate → decoder; Transform → encoder-decoder." Use this rule to match task to architecture.

---

3) Build a Hugging Face pipeline (task-appropriate)

A) Classification (single-label)

```
clf = pipeline("text-classification",
               model="distilbert-base-uncased-finetuned-sst-2-english",
device=DEVICE)
clf("Battery dies fast but looks great.")
```

B) Zero-shot multi-label (if you have custom tags)

```
zero = pipeline("zero-shot-classification", model="facebook/bart-large-mnli",
device=DEVICE)
labels = ["refund request","feature request","bug report"]
zero("The app crashes when I upload a photo.", candidate_labels=labels,
multi_label=True)
```

(Use `multi_label=True` and tune thresholds.)

C) Sequence-to-sequence (e.g., summarization/extraction)

```
summ = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6",
device=DEVICE)
summ("Long paragraph ...", max_new_tokens=80, min_length=30, do_sample=False)
```

---

4) Design your prompts (zero-shot → few-shot)

You will craft prompts with the five blocks—**persona, instruction, context, exemplars, output constraints**—then iterate. Evaluate **format compliance** and **quality**; log **latency** and **token use** as part of your loop.

Zero-shot template (no exemplars)

```
[PERSONA] You are a senior NLP analyst for {domain}.
[INSTRUCTION] Given ONE input text, classify it into {labels} and return a
JSON object.
```

```
[CONTEXT] Definitions of labels: {brief, unambiguous}. If none apply, return
{"label":"none"}.
[OUTPUT CONSTRAINTS] Return ONLY valid JSON:
{"label":"<one_of_labels>","confidence":<0..1>}. No extra text.
[INPUT] {text}
```

Few-shot template (2–3 exemplars)

```
[PERSONA] You are a senior NLP analyst for {domain}.
[INSTRUCTION] Map the input to one label from {labels}. Return JSON only.
[CONTEXT] Short policy notes or tie-breakers.
[EXEMPLARS]
Input: "Delivery was early; setup was easy." →
{"label":"positive","confidence":0.98}
Input: "Crashes on upload; support is unresponsive." →
{"label":"bug_report","confidence":0.96}
[OUTPUT CONSTRAINTS] EXACT JSON schema: {"label":"...","confidence":...}. No
prose.
[INPUT] {text}
```

---

5) Run inference utilities (latency)

```
def time_call(fn, *args, **kwargs):
    t0 = time.perf_counter(); out = fn(*args, **kwargs); t1 =
time.perf_counter()
    return out, t1 - t0
```

For decoder/seq2seq prompts (e.g., FLAN-T5 or BART), call your pipeline with the **full prompt string** and check JSON parse success to enforce **strict output formatting**.

---

6) Evaluate on your dev/test split

A) Classification metrics (Accuracy/F1) and compliance

```
from sklearn.metrics import accuracy_score, f1_score

def evaluate_prompts(texts, labels, prompt_fn):
    preds, ok_flags, times = [], [], []
    for x in texts:
        y_raw, t = time_call(prompt_fn, x)
        preds.append(y_raw["label"] if isinstance(y_raw, dict) else y_raw)
        ok_flags.append(isinstance(y_raw, dict))
        times.append(t)
    results = {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro"),
        "json_ok_rate": sum(ok_flags)/len(ok_flags),
        "mean_s": statistics.mean(times),
        "p90_s":  statistics.quantiles(times, n=10)[8],
```

```
    }
    return results
```

B) Summarization or other seq2seq — compute ROUGE (or BLEU) alongside **JSON compliance** if you enforce a schema.

Your evaluation cell should log **format compliance**, **quality metric(s)**, and **latency (mean & p90)**—the exact checks recommended in the prompt-evaluation guide.

---

7) Compare to the Week-2 baseline

Produce a small table contrasting your **RNN baseline** vs **prompt-engineered transformer** on the same split. Include accuracy/F1 (or ROUGE) and one line on trade-offs (speed, stability).

---

8) Interpretability artefact — Prompt rationale (1–2 slides or 1 cell)

Briefly label how your prompt uses **persona, instruction, context, exemplars, output constraints**, and show one case where **few-shot** corrected a zero-shot error (e.g., fixed formatting, better label choice).

Final cell (copy these headings into your notebook)

Milestone 3 — Results & Reflection (≈200–250 words)

- **Model choice & rationale:** why this checkpoint (domain, size/latency, license).
- **Zero-shot → Few-shot deltas:** metric change, JSON compliance, latency notes.
- **Baseline comparison:** what improved vs RNN and by how much.
- **Next iteration:** one concrete prompt tweak you'd test next (add rule, refine exemplar, adjust decoding).

Practical tips

- Start with **small instruction-tuned** models (e.g., FLAN-T5-small) for fast iteration; switch to domain-specific encoders if your task is pure classification.
- Keep prompts **strict** about output format and **validate** with `json.loads`; iterate: **Prompt → Score → Modify one element → Re-score**.

By completing this milestone you will have a **prompt-engineered transformer baseline** that is evaluated rigorously and explained clearly—exactly what Milestone 3 asks for.