Work Toward Your Project — Milestone 4

Real-Task Application & Error Log (Detailed, with CLI/UI code)

In this milestone, you will plug **your Milestone-3 model** into a tiny, real application (CLI or micro-UI), collect **20–30 edge-case inputs**, and maintain an **error log** with a short **error taxonomy**. The official brief requires: (i) an applied pipeline (e.g., sentiment dashboard, NER highlighter, translation script), (ii) 20–30 real edge cases + errors, (iii) a short taxonomy, and (iv) submission of a **demo GIF/screenshot or minimal CLI**, plus **`errors.csv`** and a **one-page error-analysis memo**.

You should reuse your Week-3 model and skills—each milestone is designed to build on the previous week.
Scope: this milestone fits the “2-hour build + ≈1 hour polish” cadence.

Deliverables (due end of Week 4)

1. **App evidence**: a **GIF/screenshot** of your UI **or** a **minimal CLI** run.
2. **`errors.csv`**: 20–30 edge-case rows with your taxonomy tags.
3. **1-page error-analysis memo**: patterns + examples + next steps.

Step 1 — Pick the pipeline you will ship

Choose **one** (match your task from earlier milestones):

- **Sentiment/Topic dashboard** (classification).
- **NER highlighter** (sequence labeling).
- **Summarizer/Translator** (seq2seq).

All three are explicitly permitted for Milestone 4.

Step 2 — Prepare your model from Milestone 3

- If you **fine-tuned** a checkpoint, load it by name/path.
- If you **prompt-engineered** a model, keep your **zero-/few-shot prompts** and any parsing helpers (e.g., JSON parsing).

This continuity preserves comparability across weeks and aligns with the project's progressive layering.

Step 3 — Implement a minimal CLI (template)

Use this if you prefer a non-UI script that processes a text file and saves predictions to CSV (so you can later annotate errors).

3.1 File layout

```
project/
  app.py
  examples.txt         # one input per line (you will curate 20-30 edge
cases)
  predictions.csv      # generated
  errors.csv           # you will fill error? / error_type later
  m3/                  # your Milestone-3 assets (checkpoint or prompts)
```

3.2 `app.py` (classification / seq2seq variants)

```
# app.py — Minimal CLI for Milestone 4
import argparse, csv, json, re, time
from pathlib import Path
from transformers import pipeline

def build_pipe(task, model_name_or_path, device=-1):
    if task in {"sentiment","topic"}:
        return pipeline("text-classification", model=model_name_or_path,
device=device, return_all_scores=False)
    elif task == "ner":
        return pipeline("token-classification", model=model_name_or_path,
device=device, aggregation_strategy="simple")
    elif task in {"summarization","translation"}:
        return pipeline(task, model=model_name_or_path, device=device)
    else:
        raise ValueError(f"Unsupported task: {task}")

# Optional: if you used prompt-engineered seq2seq in Milestone 3
def run_prompted(pipe, prompt_template, text, max_new_tokens=128):
    prompt = prompt_template.format(text=text)
    out = pipe(prompt, max_new_tokens=max_new_tokens, do_sample=False)[0]
    # Try to parse JSON if your prompt enforces a schema
    m = re.search(r"\{.*\}", out.get("generated_text", ""), flags=re.S)
    return json.loads(m.group(0)) if m else out

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--task", required=True,
choices=["sentiment","topic","ner","summarization","translation"])
```

```
    ap.add_argument("--model", required=True, help="HF hub id or local path
(use your Milestone-3 model)")
    ap.add_argument("--infile", required=True, help="txt file: one example
per line")
    ap.add_argument("--outfile", default="predictions.csv")
    ap.add_argument("--device", type=int, default=-1)  # -1 cpu, 0 gpu
    ap.add_argument("--prompt_file", default=None)     # if using prompt-
engineered seq2seq
    args = ap.parse_args()

    pipe = build_pipe(args.task, args.model, device=args.device)
    prompt_template = Path(args.prompt_file).read_text() if args.prompt_file
else None

    rows = []
    for i, raw in enumerate(Path(args.infile).read_text().splitlines()):
        raw = raw.strip()
        if not raw: continue
        t0 = time.perf_counter()
        if prompt_template:
            pred = run_prompted(pipe, prompt_template, raw)
        else:
            pred = pipe(raw)
        latency = time.perf_counter() - t0

        # Normalize outputs for CSV
        if args.task in {"sentiment","topic"}:
            label = pred[0]["label"]; score = pred[0].get("score", "")
            payload = {"label": label, "confidence": round(float(score),4)}
        elif args.task == "ner":
            payload = {"entities": json.dumps(pred, ensure_ascii=False)}
        else:  # summarization/translation
            text_out = pred[0].get("summary_text") or
pred[0].get("translation_text") or pred[0].get("generated_text","")
            payload = {"text_out": text_out}

        rows.append({
            "id": i, "raw_text": raw, "prediction": json.dumps(payload,
ensure_ascii=False),
            "confidence": payload.get("confidence",""),
            "latency_s": round(latency, 4),
            "error?": "", "error_type": "", "notes": ""
        })

    # Write CSV ready for annotation
    with open(args.outfile, "w", newline='', encoding="utf-8") as f:
        w = csv.DictWriter(f, fieldnames=rows[0].keys())
        w.writeheader(); w.writerows(rows)
    print(f"Wrote {args.outfile} with {len(rows)} rows.")

if __name__ == "__main__":
    main()
```

Run (example):

```
# CPU run, classification
```

```
python app.py --task sentiment --model m3/checkpoint --infile examples.txt --
outfile predictions.csv --device -1

# Prompt-engineered seq2seq (few-shot prompt stored in prompt.txt)
python app.py --task summarization --model google/flan-t5-small --infile
examples.txt \
              --outfile predictions.csv --prompt_file prompt.txt
```

This style (minimal CLI + CSV) satisfies the “demo (minimal CLI) + errors.csv” deliverable.

Step 4 — Implement a micro-UI (Streamlit or Gradio)

Use this if you want a one-page app to paste text, see predictions, and append rows to `errors.csv`. A screenshot/GIF of the UI fulfills the demo requirement.

4A) Streamlit (classification or NER)

```
# app_ui.py  —  streamlit run app_ui.py
import streamlit as st, json, csv, time
from pathlib import Path
from transformers import pipeline

TASK = "ner"  # "sentiment" | "ner" | "summarization"
MODEL = "m3/checkpoint"  # your Milestone-3 model path or hub id

@st.cache_resource
def load_pipe():
    if TASK == "ner":
        return pipeline("token-classification", model=MODEL,
aggregation_strategy="simple")
    elif TASK == "sentiment":
        return pipeline("text-classification", model=MODEL,
return_all_scores=False)
    else:
        return pipeline(TASK, model=MODEL)

pipe = load_pipe()
st.title("Milestone 4 — Mini NLP App")

txt = st.text_area("Paste one input:")
if st.button("Predict") and txt.strip():
    t0 = time.perf_counter()
    out = pipe(txt)
    lat = time.perf_counter() - t0

    if TASK == "ner":
        st.write(out)
        # Simple highlight
        for ent in out:
            st.markdown(f"- **{ent['word']}**  [{ent['entity_group']}]
score={ent['score']:.2f}")
```

```
        pred_json = {"entities": out}
    elif TASK == "sentiment":
        st.json(out[0])
        pred_json = {"label": out[0]["label"], "confidence": out[0]["score"]}
    else:
        text_out = out[0].get("summary_text") or
out[0].get("translation_text") or out[0].get("generated_text","")
        st.write(text_out)
        pred_json = {"text_out": text_out}

    st.caption(f"Latency: {lat:.3f}s")
    # Append to errors.csv for later review
    row = {"id": int(time.time()), "raw_text": txt, "prediction":
json.dumps(pred_json, ensure_ascii=False),
           "confidence": pred_json.get("confidence",""), "latency_s":
round(lat,3),
           "error?":"", "error_type":"", "notes":""}
    file = Path("errors.csv")
    newfile = not file.exists()
    with file.open("a", newline='', encoding="utf-8") as f:
        w = csv.DictWriter(f, fieldnames=row.keys())
        if newfile: w.writeheader()
        w.writerow(row)
    st.success("Logged to errors.csv")
```

4B) Gradio (seq2seq prompt-engineered)

```
import gradio as gr, json, re, time
from transformers import pipeline

MODEL = "google/flan-t5-small"
pipe = pipeline("text2text-generation", model=MODEL)

PROMPT = """You are a concise assistant. Summarize in ≤ 40 words.
Return ONLY JSON: {"summary": "<text>"}.
Text: {text}"""

def infer(text):
    t0 = time.perf_counter()
    out = pipe(PROMPT.format(text=text), max_new_tokens=96,
do_sample=False)[0]["generated_text"]
    lat = time.perf_counter() - t0
    m = re.search(r"\{.*\}", out, flags=re.S)
    return (m.group(0) if m else out), f"{lat:.3f}s"

gr.Interface(fn=infer, inputs="textbox", outputs=["textbox","textbox"],
             title="Milestone 4 — Prompted Summarizer").launch()
```

Step 5 — Curate 20–30 edge-case inputs and log errors

Target **coverage** (not volume). Include: noisy text (typos/emojis), long/compound sentences, domain jargon, ambiguity/sarcasm, code-switching, and out-of-domain examples. These are explicitly called for in the milestone.

`errors.csv` schema (recommended)

id	raw_text	prediction	confidence	latency_s	error?	error_type	notes

You will annotate `error?` (=1/0), `error_type`, and `notes` after reviewing outputs. The CSV is an explicit deliverable.

Step 6 — Define a short error taxonomy (5–8 tags)

Start concise and actionable (edit as patterns emerge):

- **Slang/emoji**, **Negation**, **Sarcasm**, **Long-context**, **Entity boundary** (NER), **Domain term**, **Hallucination** (seq2seq), **Ambiguous label**.
  This directly matches the brief's request to "draft a short error taxonomy."

Step 7 — Basic analysis & memo scaffold

7.1 Compute error rates by tag (example)

```
import pandas as pd
df = pd.read_csv("errors.csv")
by_tag = (df[df["error?"]==1]
          .groupby("error_type")["id"]
          .count().sort_values(ascending=False).rename("count"))
by_tag = by_tag.to_frame()
by_tag["pct_of_errors"] = 100 * by_tag["count"] / by_tag["count"].sum()
by_tag
```

7.2 Optional quick metrics (if you have gold labels)

- **Classification/NER**: accuracy/F1 (or span-level F1 for NER).
- **Summarization/Translation**: ROUGE/BLEU on a small labeled slice (10–50).
  Include 2–3 **representative examples** in your memo.

7.3 One-page memo (structure)

1. **Setup** (3–4 lines): model, task, interface (CLI/UI), data slice.
2. **Top error patterns** (5–7 bullets): ranked by frequency with short explanations.
3. **Examples** (2–3): quote, predicted vs expected, taxonomy tag.
4. **Next steps** (4–6 lines): concrete fixes (e.g., add few-shot exemplar for negation; extend max length; pre-normalize slang).

The memo is a required deliverable.

Quality checklist (self-audit)

- You used **your Milestone-3 model/prompt** in the pipeline (continuity).
- App evidence attached (GIF/screenshot or CLI run). `errors.csv` present (20–30 rows). **Memo** attached.
- Error taxonomy applied consistently.
- Optional metrics (if labels exist) support your narrative.

You now have everything needed to stand up a minimal, **real** pipeline, curate meaningful edge cases, and produce a crisp error log and memo—exactly what Milestone 4 asks for.