PII Entity Recognition for Noisy STT Transcripts

Duration: 2 hours

Objective

Build a **token-level NER model** that identifies PII entities from noisy Speech-to-Text (STT) transcripts and returns character-level entity spans with PII flags.
Focus on **high precision for PII categories** under a reasonable CPU latency budget.

You will:

- Detect the following entity types:
  CREDIT_CARD, PHONE, EMAIL, PERSON_NAME, DATE, CITY, LOCATION
- Mark PII = **true** for: CREDIT_CARD, PHONE, EMAIL, PERSON_NAME, DATE
- Mark PII = **false** for: CITY, LOCATION
- Output entity spans as **character offsets** on the original transcript
- Use a **learned sequence labeler** (BERT-style token classifier, BiLSTM tagger, etc.)
- Use regex/dictionaries only as helpers, not the primary detector

---

Constraints

- **Timebox:** 2 hours (open book)
- **Data:** Generate your own train (500-1000 examples) and dev (100-200) sets reflecting noisy STT patterns
- **Model requirement:** Must be a *learned* token classification model
- **Latency target:** p95 ≤ **20 ms** per utterance (batch size 1) on CPU
- **Precision target:** PII precision ≥ **0.80** on dev is considered strong
- Precision on PII entities is more important than recall
  - Especially for: **CREDIT_CARD, PHONE, EMAIL, PERSON_NAME, DATE**

What's Provided

You receive a starter repository containing:

Data

Located in `data/`:

NOTE: You will need to add more data in your dev.jsonl

- `Train.jsonl`
- `stress.jsonl`
- `dev.jsonl`
- `test.jsonl` (no labels)

Each labeled line contains:

```
{
  "id": "utt_0012",
  "text": "my credit card number ...",
  "entities": [
    { "start": 3, "end": 19, "label": "CREDIT_CARD" },
    { "start": 63, "end": 77, "label": "PERSON_NAME" },
    { "start": 81, "end": 105, "label": "EMAIL" }
  ]
}
```

- `text`: noisy STT transcript (spelled-out numbers, “dot”, “at”, no punctuation, etc.)
- `start`/`end`: character offsets
- `label`: entity type

Starter Code (src/)

- `dataset.py` – JSONL → tokenized BIO dataset
- `labels.py` – label list + PII mapping
- `model.py` – token classification skeleton (customizable)
- `train.py` – training script
- `predict.py` – inference + span decoding
- `eval_span_f1.py` – span-level metrics + PII metrics
- `measure_latency.py` – p50/p95 latency

Requirements

Dependencies listed in `requirements.txt`.

Tasks

1. Install dependencies & run baseline (after generating more data sets)

```
pip install -r requirements.txt

python src/train.py \
  --model_name distilbert-base-uncased \
  --train data/train.jsonl \
  --dev data/dev.jsonl \
  --out_dir out

python src/predict.py \
  --model_dir out \
  --input data/dev.jsonl \
  --output out/dev_pred.json

python src/eval_span_f1.py \
  --gold data/dev.jsonl \
  --pred out/dev_pred.json

python src/predict.py \
  --model_dir out \
  --input data/stress.jsonl \
  --output out/stress_pred.json

python src/eval_span_f1.py \
  --gold data/stress.jsonl \
  --pred out/stress_pred.json
```

2. Improve the NER system within the 2-hour timebox

You may:

- Adapt or replace `model.py` with any lightweight learned sequence tagger
- Tune hyperparameters via `train.py`: epochs, batch size, learning rate, dropout, etc.
- Strengthen span decoding / post-processing to increase PII precision
- Ensure BIO → span decoding is correct and robust

3. Evaluate

Demonstrate improvements in:

- **Per-entity F1**, especially PII entities
- **PII precision**
- **Overall span F1**

4. Measure latency

Run:

```
python src/measure_latency.py \
  --model_dir out \
  --input data/dev.jsonl \
  --runs 50
```

Record:

- **p50 latency**
- **p95 latency (must be ≤ 20 ms)**

5. Explain your approach in a short Loom (~5 min)

Include:

- Final results
- Code base explanation
- Model + tokenizer used
- Key hyperparameters
- PII precision/recall/F1
- Latency results and any trade-offs