# **Evaluation and Improvement Plan for Offline Collection Status and Promise-to-Pay Models**

**Prepared by Mohak Kapoor  
June 2025**

## **Abstract**

This document outlines an internal plan to evaluate and enhance our offline collection status and promise-to-pay models for bank call audio in English, covering temporary pretrained models (using spaCy, NLTK, scikit-learn) and proprietary models under development. The models process client-provided audio calls to classify collection status, evaluate collection scoring criteria, and categorize promise-to-pay intent, operating offline on CPU. The plan focuses on internal evaluation using self-generated ground truth, generating performance reports, and improving models through iterative fine-tuning and deployment. Leveraging client audio calls and internal annotation, the approach ensures domain-specific accuracy for banking collections (e.g., loan defaults, payment reminders) while maintaining offline compatibility. Official sources, including spaCy, NLTK, scikit-learn, and Hugging Face documentation, guide the development process.

## **1. Introduction**

Our models process English bank call audio to deliver three outputs:

* **Collection Status Model**: Classifies calls into one of four types: Predues Collection, Postdue Collections Less Than 30 Days, Postdue Collections Greater Than 30 Days, or Late Collections Greater Than 60 Days.
* **Collection Scoring Criteria Model**: Evaluates 22 true/false criteria (e.g., call\_open\_timely\_manner, friendly\_confident\_tone) based on dialogue and timestamps.
* **Promise-to-Pay Model**: Classifies calls into one of four categories: Settlement, Partial Settlement, Promise Broken, Denial, and identifies two additional labels (e.g., lead identifiers like names, dates).

The temporary models use spaCy for NLP, NLTK for text processing, and scikit-learn for classification, serving as an interim solution. Proprietary models, built with transformer-based architectures akin to our transcription/translation models, are developed for long-term customization. Both operate offline on CPU, supporting banking collection processes (e.g., intent prediction for payment reminders). This internal plan addresses:

* Setting up temporary models and designing proprietary models.
* Building data pipelines for client audio calls.
* Generating ground truth for collection status, criteria, and promise-to-pay labels.
* Evaluating performance using self-generated ground truth.
* Generating downloadable performance reports.
* Iteratively fine-tuning and deploying models via a model improvement cycle.

Clients provide audio calls, while all ground truth (transcriptions, translations, labels) and evaluations are handled internally by our data annotation team.

## **2. Internal Tasks**

Our team handles the following tasks to support evaluation and improvement:

### **2.1. Model Setup and Design**

* **Temporary Models**: Configure spaCy, NLTK, and scikit-learn pipelines for offline operation, processing transcriptions for classification and criteria evaluation.
* **Proprietary Models**: Define transformer-based architectures for collection status (text classification), collection scoring criteria (multi-label classification), and promise-to-pay (multi-output classification). Prototype on pilot data and benchmark against temporary models.

### **2.2. Data Pipeline Setup**

* Develop preprocessing pipeline for audio (e.g., transcription via Whisper, normalization) and text (e.g., tokenization, code-switching handling).
* Automate data loading, splitting (training/validation/test), and annotation conversion to training formats (e.g., JSON to model inputs).
* Implement quality assurance (QA) checks for annotations (e.g., label consistency, transcription accuracy).

### **2.3. Ground Truth Generation**

* Label client audio calls for transcriptions, translations, and model-specific outputs using semi-automated tools (e.g., Whisper for transcription, manual correction). Data formats:
  + **Collection Status Model**: Audio, transcription, translation, collection status (Predues, Postdue <30, >30, >60).
  + **Collection Scoring Criteria Model**: Audio, transcription, translation, 22 true/false criteria (e.g., call\_open\_timely\_manner, friendly\_confident\_tone).
  + **Promise-to-Pay Model**: Audio, transcription, translation, promise-to-pay category (Settlement, Partial Settlement, Promise Broken, Denial), two additional labels (e.g., customer name, payment date).
* Reserve validation and test sets (e.g., 10%/10% of calls) separate from training data.
* Ensure ground truth captures banking-specific terms (e.g., “balance,” “due date”) and handles code-switching.

### **2.4. Continuous Integration and Testing**

* Set up a continuous integration (CI) pipeline with automated tests for model components (e.g., classification, criteria detection) and data pipeline (e.g., JSON parsing).
* Conduct regression testing during fine-tuning to detect performance drops.

### **2.5. Evaluation Pipeline**

* Build a Python-based pipeline to compare model outputs to ground truth, computing metrics like classification accuracy and F1-score using offline-compatible libraries (e.g., scikit-learn, nltk, spacy).

### **2.6. Performance Reporting**

* Generate downloadable reports (PDF, CSV) summarizing metrics, error analysis, and improvement recommendations.

### **2.7. Model Improvement Cycle**

* Fine-tune models iteratively, using validation sets for hyperparameter tuning and regularization to prevent overfitting.
* Optimize models for CPU deployment (e.g., quantization, pruning).
* Deploy improved models with updated code and documentation.

## **3. Evaluation Metrics**

The following metrics assess performance:

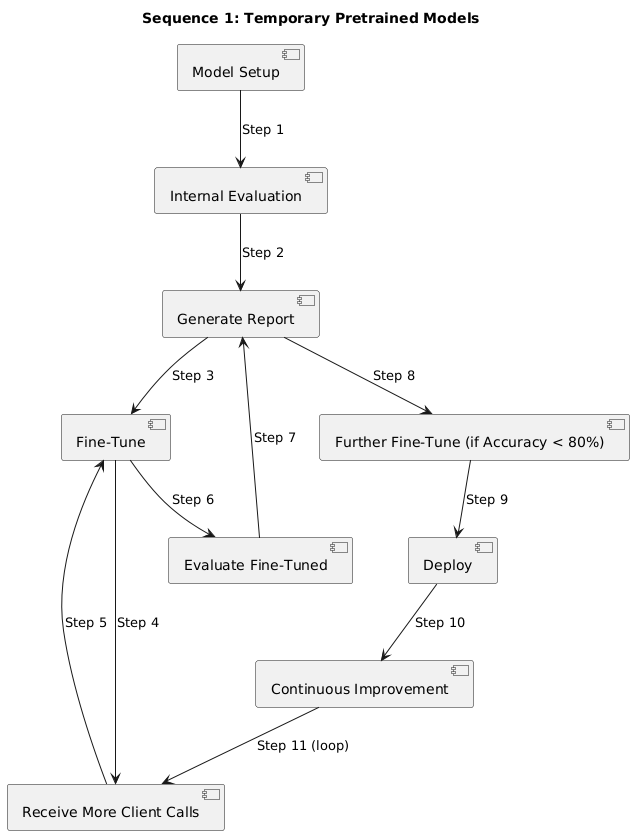
* **Collection Status Model**:
  + **Classification Accuracy**: Percentage of correctly predicted type\_of\_collection (target > 80%) [1].
  + **Confusion Matrix**: Distribution of misclassified types (e.g., Predues vs. Postdue).
  + **F1-Score**: Per-class performance for imbalanced types (e.g., Late >60).
  + **Latency**: Processing time for classification.
* **Collection Scoring Criteria Model**:
  + **Per-Criterion Accuracy**: Percentage of correctly predicted true/false values per criterion (target > 85%).
  + **Average Criteria Accuracy**: Mean accuracy across all 22 criteria.
  + **Hamming Loss**: Proportion of incorrect true/false labels.
  + **Latency**: Processing time for criteria evaluation.
* **Promise-to-Pay Model**:
  + **Classification Accuracy**: Percentage of correctly predicted type (target > 80%) [1].
  + **Confusion Matrix**: Distribution of misclassified categories (e.g., Settlement vs. Denial).
  + **Precision, Recall, F1-Score**: Per-category performance for imbalances (e.g., low recall for Denial).
  + **Label Accuracy**: Accuracy for two additional labels (e.g., customer name, payment date).
  + **Latency**: Processing time for category and label prediction.

## **4. Evaluation and Improvement Plans**

Two sequences are outlined: Sequence 1 for temporary pretrained models and Sequence 2 for proprietary models. Both incorporate a model improvement cycle, highlighted in their flow diagrams.

### **4.1. Sequence 1: Temporary Pretrained Models**

#### **Flow Diagram**



#### **Step 1: Model Setup**

* Configure spaCy for NLP, NLTK for text processing, and scikit-learn for classification, ensuring offline operation.
* Set up pipelines for collection status (e.g., logistic regression), collection scoring criteria (e.g., multi-label classifier), and promise-to-pay (e.g., random forest).

#### **Step 2: Internal Evaluation**

* Evaluate on internal dataset (e.g., 5 calls, 25 minutes) with self-generated ground truth (transcriptions, translations, labels).
* Compute metrics (accuracy, F1-score, confusion matrix) and analyze errors (e.g., misclassified Postdue, incorrect call\_open\_timely\_manner).

#### **Step 3: Generate Performance Report**

* Summarize metrics, errors, and fine-tuning recommendations.
* Note internal ground truth and temporary model usage.
* Formats: PDF (via reportlab), CSV.

#### **Step 4: Fine-Tune Model**

* Fine-tune models using internal dataset, tuning hyperparameters (e.g., learning rate, max\_depth) on validation set (e.g., 5 calls).
* Update keyword lists (e.g., “pay today” for Settlement) and fuzzy match thresholds (e.g., 0.4 for criteria).
* Apply regularization (e.g., L2) and optimize for CPU.

#### **Step 5: Receive Additional Client Calls**

* Obtain large number of calls (e.g., 50–100, 250–500 minutes).

#### **Step 6: Fine-Tune Model (Model Improvement Cycle)**

* Label new calls for training (transcriptions, translations, model-specific labels), reserving validation/test sets (e.g., 10%/10%).
* Fine-tune models on training data, using validation set.
* Conduct regression testing (e.g., compare accuracy to Step 4).

#### **Step 7: Evaluate Fine-Tuned Model (Model Improvement Cycle)**

* Evaluate on fixed test set (5 calls) with self-generated ground truth.
* Compute updated metrics and analyze errors.

#### **Step 8: Generate Performance Report (Model Improvement Cycle)**

* Summarize improvements (e.g., accuracy: 70% → 85%) and errors.
* Recommend further fine-tuning if accuracy < 80%.
* Formats: PDF, CSV.

#### **Step 9: Further Fine-Tune (Conditional, Model Improvement Cycle)**

* Fine-tune if accuracy < 80%, using validation set and regularization.
* Optimize model for CPU (e.g., pruning, quantization).

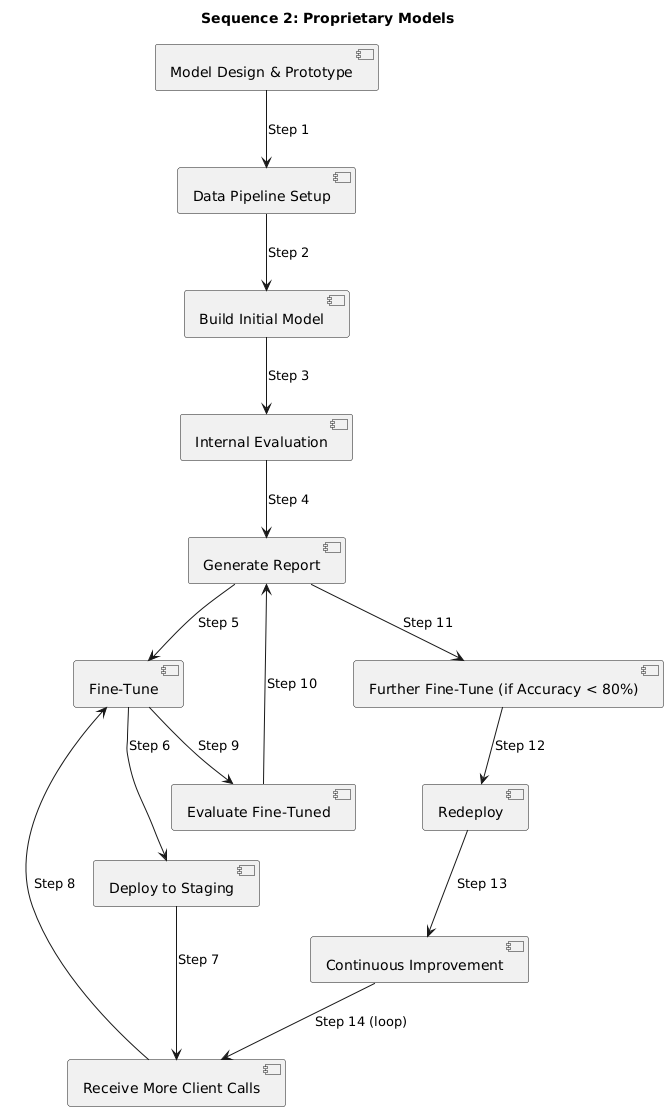
#### **Step 10: Deploy Model**

* Deploy to production with offline installation instructions.
* Document fine-tuning and evaluation results.

#### **Step 11: Continuous Improvement**

* Request more calls if accuracy < 80%.
* Re-run model improvement cycle (fine-tuning, evaluation, reporting).
* Transition to proprietary models as they mature.

### **4.2. Sequence 2: Proprietary Models**



#### **Step 1: Model Design and Prototyping**

* Define transformer-based architectures:
  + **Collection Status Model**: Text classification for type\_of\_collection (4 classes).
  + **Collection Scoring Criteria Model**: Multi-label classification for 22 true/false criteria.
  + **Promise-to-Pay Model**: Multi-output classification for category (4 classes) and two additional labels (e.g., named entity recognition for names, dates).
* Prototype on pilot dataset (e.g., 5 calls, 25 minutes) with annotations.
* Benchmark against temporary models (e.g., accuracy > 75%).
* Document architecture and results.

#### **Step 2: Data Pipeline Setup**

* Develop preprocessing scripts (e.g., transcription via Whisper, 16kHz normalization with librosa, text tokenization).
* Automate data splits (training/validation/test, e.g., 80%/10%/10%) and JSON conversion.
* Implement QA checks (e.g., verify label consistency).

#### **Step 3: Build Initial Model**

* Train on client calls (e.g., 50 calls, 250 minutes), using labeled training data:
  + **Collection Status**: Audio, transcription, translation, collection status.
  + **Collection Scoring Criteria**: Audio, transcription, translation, 22 criteria.
  + **Promise-to-Pay**: Audio, transcription, translation, category, two labels.
* Reserve validation (5 calls) and test sets (5 calls) during annotation.
* Tune hyperparameters (e.g., learning rate) on validation set, monitoring accuracy.
* Apply domain adaptation (e.g., add banking terms like “due date” to vocabulary, weighted loss for rare classes).

#### **Step 4: Internal Evaluation**

* Evaluate on fixed test set (5 calls) using self-generated ground truth.
* Compute metrics (accuracy, F1-score, confusion matrix, hamming loss) and analyze errors.

#### **Step 5: Generate Performance Report**

* Summarize metrics, per-file errors, and fine-tuning recommendations (e.g., improve accuracy > 80%).
* Note internal ground truth usage.
* Formats: PDF, CSV.

#### **Step 6: Fine-Tune Model**

* Fine-tune based on report (e.g., target low F1-score classes).
* Use validation set for hyperparameter tuning and early stopping.
* Apply regularization (e.g., dropout) and quantization (e.g., 8-bit).

#### **Step 7: Deploy to Staging**

* Deploy to internal staging environment.
* Validate on new calls (e.g., 5 calls).
* Collect informal client feedback (e.g., classification accuracy).

#### **Step 8: Receive Additional Client Calls**

* Obtain large number of calls (e.g., 50–100, 250–500 minutes).

#### **Step 9: Fine-Tune Model (Model Improvement Cycle)**

* Label new calls for training, reserving validation/test sets.
* Fine-tune on training data, tuning hyperparameters on validation set.
* Conduct regression testing (e.g., compare accuracy to Step 6).

#### **Step 10: Evaluate Fine-Tuned Model (Model Improvement Cycle)**

* Evaluate on same fixed test set (5 calls).
* Compute updated metrics and analyze errors.

#### **Step 11: Generate Performance Report (Model Improvement Cycle)**

* Summarize improvements and errors.
* Recommend further fine-tuning if accuracy < 80%.
* Formats: PDF, CSV.

#### **Step 12: Further Fine-Tune (Conditional, Model Improvement Cycle)**

* Fine-tune if accuracy < 80%, using validation set and regularization.
* Optimize model (e.g., pruning, ONNX).

#### **Step 13: Redeploy Model**

* Deploy to production with offline instructions.
* Document architecture, hyperparameters, and results.

#### **Step 14: Continuous Improvement**

* Request more calls if accuracy < 80%.
* Re-run model improvement cycle.
* Conduct knowledge transfer workshops.

## **6. Conclusion**

This internal plan ensures robust evaluation and enhancement of temporary and proprietary collection status and promise-to-pay models. Sequence 1 leverages pretrained models for immediate results, while Sequence 2 drives long-term customization with proprietary models. By using client audio calls, internal ground truth, and iterative model improvement cycles, we achieve high accuracy for banking collections while maintaining offline compatibility. Immediate next steps include setting up data pipelines and annotating new client calls to initiate both sequences.

## **References**

[1] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, 2011.  
[2] S. Bird et al., “Natural Language Processing with Python,” O’Reilly Media, 2009.  
[3] M. Honnibal et al., “spaCy: Industrial-strength Natural Language Processing in Python,” [https://spacy.io](https://spacy.io/), 2023.  
[4] Hugging Face, “Transformers Model Documentation,”<https://huggingface.co/docs/transformers>, 2023.