# Internal Evaluation and Improvement Plan for Offline Call Insights Model **June 2025**

## Abstract

This document outlines an internal plan to evaluate and enhance our offline Voice AI Call Insights Model, which predicts customer intent and assesses call quality in bank call audio, covering temporary pretrained models (using NumPy, TextBlob, scikit-learn) and proprietary models under development. The model analyzes audio metrics, sentiment, quality, speech, and critical parameters to deliver actionable insights for banking interactions, 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 applications (e.g., payment issues, account inquiries) while maintaining offline compatibility. Official sources, including NumPy, TextBlob, scikit-learn, and Hugging Face documentation, guide the development process.

## 1. Introduction

Our Voice AI Call Insights Model processes English bank call audio to deliver actionable insights, comprising:

* **Audio Metrics**: Measures signal energy, loudness, and pitch variance to assess audio characteristics.
* **Sentiment Analysis**: Quantifies customer and agent sentiment/emotion (e.g., positive/negative, neutral/surprise) using NLP.
* **Quality Metrics**: Scores call quality based on criteria (e.g., timely\_greeting, polite\_tone, proper\_summary as true/false).
* **Speech Analytics**: Analyzes speech rate (words/second), pitch variance, and loudness for interaction dynamics.
* **Critical Parameters**: Evaluates verification, compliance, and customer satisfaction with pass/fail outcomes.
* **Final Scoring**: Combines weighted metrics for an overall call score (0–1).

The temporary models use NumPy for audio processing, TextBlob for sentiment analysis, and scikit-learn for classification, serving as an interim solution. Proprietary models, built with transformer-based architectures similar to our transcription/translation models, are developed for long-term customization. Both operate offline on CPU, supporting banking applications (e.g., intent prediction for payment resolutions). This internal plan addresses:

* Setting up temporary models and designing proprietary models.
* Building data pipelines for client audio calls.
* Generating ground truth for audio metrics, sentiment, quality, speech, and critical parameters.
* 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 NumPy for audio processing, TextBlob for sentiment analysis, and scikit-learn for classification, ensuring offline operation.
* **Proprietary Models**: Define transformer-based architectures for audio metrics (regression), sentiment/emotion analysis (classification), quality metrics (multi-label classification), speech analytics (regression), and critical parameters (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 with librosa) 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 format (per segment):
  + **Audio Metrics**: Energy (float), loudness (dB), pitch\_variance (float).
  + **Sentiment/Emotion Analysis**: Sentiment (POSITIVE/NEGATIVE), sentiment\_score (0–1), emotion (e.g., neutral, surprise, anger), emotion\_score (0–1).
  + **Quality Metrics**: True/false for criteria (e.g., timely\_greeting, polite\_tone, proper\_summary).
  + **Speech Metrics**: Speech\_rate (words/second), pitch\_variance (float), loudness (float).
  + **Critical Parameters**: Pass/fail for verification, compliance, customer\_satisfaction.
  + **Final Score**: Overall call score (0–1).
* Reserve validation and test sets (e.g., 10%/10% of calls) separate from training data.
* Ensure ground truth captures banking-specific terms (e.g., “account,” “payment”) and handles code-switching or accents.

### 2.4. Continuous Integration and Testing

* Set up a continuous integration (CI) pipeline with automated tests for model components (e.g., sentiment classification, quality scoring) 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 accuracy, mean absolute error (MAE), and correlation using offline-compatible libraries (e.g., NumPy, TextBlob, scikit-learn).

### 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:

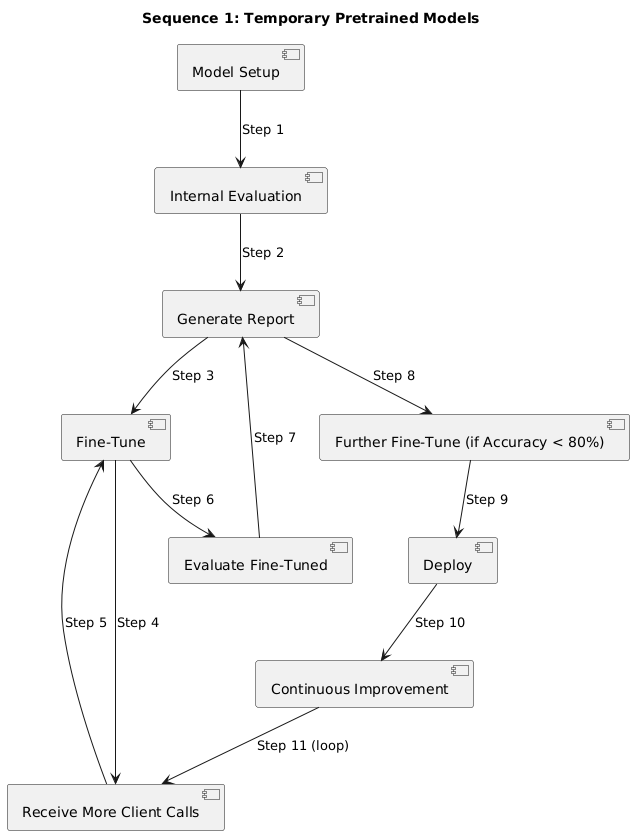
* **Audio Metrics**:
  + **Relative Error**: Percentage error for energy, loudness, and pitch\_variance predictions (target < 10%).
  + **Mean Absolute Error (MAE)**: For predicted vs. ground truth values.
* **Sentiment/Emotion Analysis**:
  + **Correlation**: Pearson correlation for sentiment/emotion scores (target > 0.8).
  + **Classification Accuracy**: Percentage of correctly predicted sentiment/emotion labels (target > 85%).
  + **MAE**: For sentiment/emotion scores (0–1).
* **Quality Metrics**:
  + **Per-Criterion Accuracy**: Percentage of correctly predicted true/false values for criteria (e.g., timely\_greeting, target > 85%).
  + **Hamming Loss**: Proportion of incorrect true/false labels.
* **Speech Analytics**:
  + **Relative Error**: Percentage error for speech\_rate, pitch\_variance, and loudness predictions (target < 10%).
  + **MAE**: For predicted vs. ground truth values.
* **Critical Parameters**:
  + **Accuracy**: Percentage of correct pass/fail predictions (target > 90%).
* **Final Score**:
  + **MAE**: Between predicted and ground truth final scores (0–1, target < 0.1).
* **Latency**: Processing time for each component (target < 1 second per call).

## 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 NumPy for audio metrics, TextBlob for sentiment/emotion analysis, and scikit-learn for classification/regression, ensuring offline operation.
* Set up pipelines for audio metrics (e.g., energy calculation), sentiment/emotion (e.g., TextBlob polarity), quality metrics (e.g., logistic regression), speech analytics (e.g., regression), and critical parameters (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, MAE, correlation, hamming loss) and analyze errors (e.g., mispredicted polite\_tone, inaccurate sentiment scores).

#### 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 TextBlob lexicon with banking terms, adjust keyword patterns for quality metrics, and recalibrate audio/speech processing.
* Apply regularization (e.g., L2) and optimize for CPU (e.g., quantization).

#### 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., quality accuracy: 75% → 90%) and errors.
* Recommend further fine-tuning if accuracy < 85%.
* Formats: PDF, CSV.

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

* Fine-tune if accuracy < 85%, using validation set and regularization.
* Optimize model for CPU (e.g., pruning, ONNX).

#### Step 10: Deploy Model

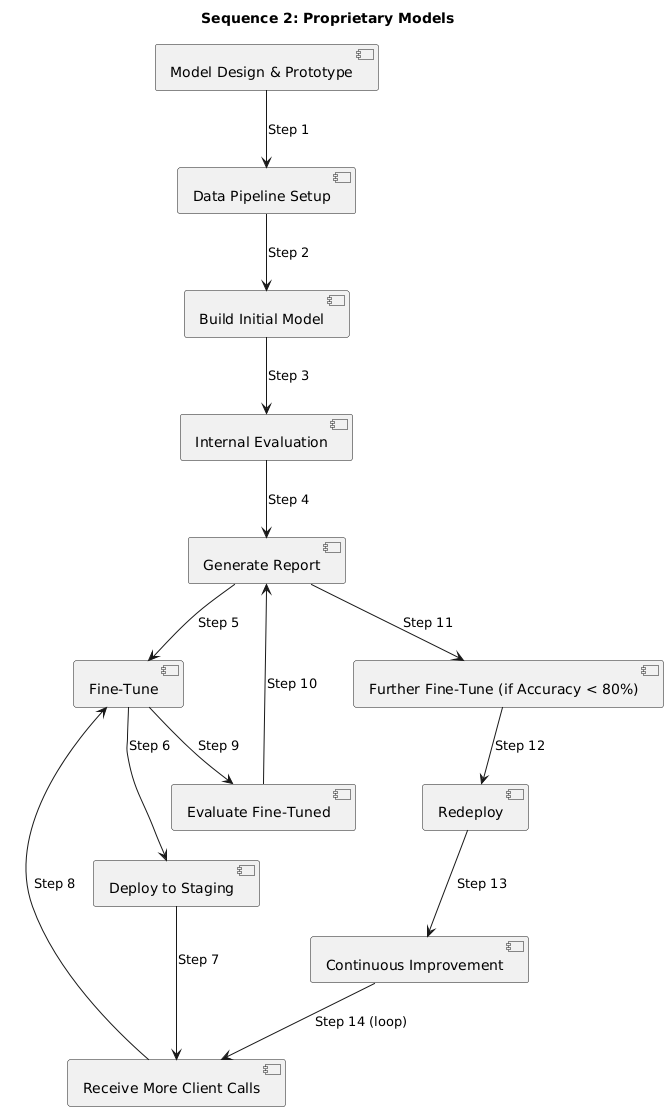
* Deploy to production with offline installation instructions.
* Document fine-tuning and evaluation results.

#### Step 11: Continuous Improvement

* Request more calls if accuracy < 85%.
* Re-run model improvement cycle (fine-tuning, evaluation, reporting).
* Transition to proprietary models as they mature.

### 4.2. Sequence 2: Proprietary Models

#### Flow Diagram



#### Step 1: Model Design and Prototyping

* Define transformer-based architectures:
  + **Audio Metrics**: Regression for energy, loudness, pitch\_variance.
  + **Sentiment/Emotion Analysis**: Classification for sentiment/emotion labels and scores.
  + **Quality Metrics**: Multi-label classification for true/false criteria.
  + **Speech Analytics**: Regression for speech\_rate, pitch\_variance, loudness.
  + **Critical Parameters**: Classification for pass/fail outcomes.
  + **Final Score**: Regression for overall score (0–1).
* Prototype on pilot dataset (e.g., 5 calls, 25 minutes) with annotations.
* Benchmark against temporary models (e.g., quality accuracy > 80%).
* 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 (audio metrics, sentiment/emotion, quality metrics, speech metrics, critical parameters, final score).
* Reserve validation (5 calls) and test sets (5 calls) during annotation.
* Tune hyperparameters (e.g., learning rate) on validation set, monitoring accuracy/MAE.
* Apply domain adaptation (e.g., add banking terms to vocabulary, weighted loss for rare emotions).

#### Step 4: Internal Evaluation

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

#### Step 5: Generate Performance Report

* Summarize metrics, per-file errors, and fine-tuning recommendations (e.g., improve quality accuracy > 85%).
* Note internal ground truth usage.
* Formats: PDF, CSV.

#### Step 6: Fine-Tune Model

* Fine-tune based on report (e.g., target low accuracy criteria, high MAE scores).
* 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., sentiment 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 < 85%.
* Formats: PDF, CSV.

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

* Fine-tune if accuracy < 85%, 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 < 85%.
* Re-run model improvement cycle.
* Conduct knowledge transfer workshops.

## 6. Conclusion

This internal plan ensures robust evaluation and enhancement of temporary and proprietary Call Insights Models for bank call audio analysis. 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 insights 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. Loria, “TextBlob: Simplified Text Processing,” [https://textblob.readthedocs.io](https://textblob.readthedocs.io/), 2023.  
[3] S. Bird et al., “Natural Language Processing with Python,” O’Reilly Media, 2009.  
[4] Hugging Face, “Transformers Model Documentation,”<https://huggingface.co/docs/transformers>, 2023.