International Journal on Advanced Computer Theory and Engineering
ISSN: 2319 - 2526
Volume 15 Issue 2s (2026) | Pages: 208-222

An AI-Powered On-Device Expense Tracking System Using BiLSTM-Based Named Entity Recognition on Financial SMS Notifications

Mayuri Mahajan[1], Harsh Paigude[2], Aditi Mahadik[3], Shrutika Kale[4]

[1]Assistant Professor, Department of AIML, Genba Sopanrao Moze College of Engineering Pune, Maharashtra, India.
[2,3,4] Student, Department of AIML, Genba Sopanrao Moze College of Engineering Pune, Maharashtra, India.

Peer Review Information

Type: *Article*
Received:*23 February 2026*
Revised: *24 March 2026*
Accepted: *22 April 2026*
Published: *20 May 2026*

Abstract

The proliferation of digital payment platforms in India—including UPI-based services such as Google Pay, PhonePe, and Paytm—has led to a dramatic increase in transactional SMS and push notification volume, creating an opportunity for automated personal finance tracking. This paper presents Notitrack, an intelligent, offline-first mobile expense tracking system that leverages a custom-trained Bidirectional Long Short-Term Memory (BiLSTM) neural network for Named Entity Recognition (NER) to automatically extract financial entities—specifically merchant names, transaction amounts, and bank identifiers—from banking SMS messages and push notifications. The system is implemented as a cross-platform Flutter application with Android-native notification interception via the NotificationListenerService API. A hybrid extraction architecture is employed, combining on-device TensorFlow Lite inference with a multi-tier regex fallback system to ensure robust entity extraction even with out-of-vocabulary tokens. The BiLSTM model was trained on a custom-annotated dataset of 211 Indian banking SMS samples using IOB (Inside-Outside-Beginning) tagging, focal loss for class imbalance, and iteratively tuned per-tag class weighting. Experimental results demonstrate an overall tagging accuracy of 95.03%, with entity-specific F1-scores of 0.97 for transaction amounts, 0.97 for bank names, and 0.69 for merchant names. The system processes transactions in under 100 milliseconds, operates entirely offline with zero cloud dependency, and includes a two-layer duplicate prevention mechanism. This work contributes a practical, privacy-preserving approach to automated expense management on resource-constrained mobile devices.

Keywords: Named Entity Recognition; BiLSTM; TensorFlow Lite; Expense Tracking; SMS Parsing; UPI Transactions; On-Device Machine Learning; Flutter; Mobile NLP; Financial Text Mining; IOB Tagging; Notification Interception

How to Cite This Article

Mahajan, M., Paigude, H., Mahadik, A., & Kale, S. (2026). *An AI-Powered On-Device Expense Tracking System Using BiLSTM-Based Named Entity Recognition on Financial SMS Notifications*. *International Journal on Advanced Computer Theory and Engineering, 15*(2s), 208–222.

Introduction

Background

India's digital payment ecosystem has witnessed unprecedented growth following the introduction of the Unified Payments Interface (UPI) by the National Payments Corporation of India (NPCI) in 2016. According to NPCI data, UPI processed over 13 billion transactions in March 2024 alone, with a transaction value exceeding ₹19.64 lakh crore (approximately USD 236 billion). This explosive growth, combined with traditional banking transactions via NEFT, RTGS, and card payments, generates an enormous volume of transactional SMS messages and push notifications for individual users on a daily basis.

Despite this wealth of financial data arriving directly on users' smartphones, the manual tracking of personal expenses remains a significant pain point. Existing expense tracking applications such as Walnut, Money Manager, and CRED typically require either (a) direct access to SMS messages—which is increasingly restricted by mobile operating system policies and app store regulations—or (b) manual entry by the user, leading to low adoption rates and incomplete financial records. Google Play policies since Android 9 (API level 28) restrict the READ_SMS permission exclusively to applications whose primary function is SMS or telephony, effectively rendering traditional SMS-reading approaches non-viable for expense tracking applications.

Problem Statement

The core challenge addressed in this research is the automated extraction of structured financial entities from unstructured transactional text messages across diverse Indian banking formats. This problem is compounded by several factors:

1. **Format Heterogeneity**: Indian banks employ widely varying SMS formats with no standardized template. For example:
    - SBI: "Dear UPI user A/C X5929 debited by 125.0 trf to Tejas Hemant Susladkar"
    - HDFC: "Sent Rs.125.00 From HDFC Bank To MC DONALDS On 15-11-25"
    - ICICI: "ICICI Bank Acct XX924 debited for Rs 4000.00; Prajakta Rajend credited"
2. **Entity Complexity**: Merchant names range from single tokens (e.g., "Zomato") to multi-token names with special characters (e.g., "WELLNESS FOREVER MEDICARE LIMITED", "MSEDCL-160220725411").
3. **Platform Restrictions**: Android's deprecation of broad SMS access permissions necessitates alternative approaches such as notification interception.
4. **Resource Constraints**: Mobile deployment requires models that are small (< 1 MB), fast (< 100 ms inference), and operable without internet connectivity.
5. **Privacy Requirements**: Users are increasingly reluctant to share financial data with cloud services, necessitating fully on-device processing.

Research Objectives

The primary objectives of this research are:

- To design and implement an end-to-end system for automatic extraction of financial entities (merchant, amount, bank) from Indian banking SMS messages using deep learning-based NER.
- To develop a hybrid ML–regex extraction architecture that ensures robust entity extraction even when the ML model encounters out-of-vocabulary tokens.
- To deploy the trained NER model on mobile devices using TensorFlow Lite for real-time, on-device inference without cloud dependency.
- To evaluate the system's accuracy, latency, and robustness across diverse Indian banking SMS formats.
- To address the Android SMS permission restriction problem by implementing notification interception via the NotificationListenerService API.

Significance of the Study

This research makes the following contributions:

- **Practical Application**: A production-ready mobile expense tracker that operates entirely offline, preserving user privacy.
- **Domain-Specific NER**: A BiLSTM-based NER model specifically trained for Indian financial SMS text, addressing the unique challenges of this domain.
- **Hybrid Architecture**: A novel three-tier extraction strategy (ML → Regex Fallback → Null) that achieves high reliability by combining neural and heuristic approaches.
- **Mobile ML Deployment**: Demonstration of on-device deep learning inference using TensorFlow Lite on resource-constrained Android devices.
- **Notification-Based Approach**: An architecture that circumvents Android SMS permission restrictions through the approved NotificationListenerService API.
- **Open Methodology**: Detailed documentation of the model training, hyperparameter tuning, and deployment pipeline suitable for reproduction.

Scope and Limitations

This study focuses on Indian banking SMS messages in English. The system currently supports debit and credit transaction notifications from major Indian banks and UPI platforms. Non-transactional SMS messages (promotional, OTP, etc.) are filtered using keyword-based pre-filtering. The current training dataset is limited to 211 annotated samples, which constrains the model's vocabulary and generalization capability.

Literature Review

Named Entity Recognition (NER)

Named Entity Recognition is a well-established subtask of information extraction in Natural Language Processing (NLP) that seeks to identify and classify named entities in text into predefined categories. Early NER systems relied on rule-based and dictionary-based approaches (Nadeau & Sekine, 2007), while modern approaches leverage deep learning architectures.

Conditional Random Fields (CRF): Lafferty et al. (2001) introduced CRFs as a probabilistic framework for sequence labeling, which became the standard for NER tasks. CRFs model the conditional probability of label sequences given input sequences, enforcing valid tag transitions—a property particularly useful for IOB-formatted NER tags. **Bidirectional LSTM (BiLSTM):** Hochreiter and Schmidhuber (1997) proposed the LSTM architecture to address the vanishing gradient problem in recurrent neural networks.

Graves and Schmidhuber (2005) extended this with bidirectional processing, enabling the model to capture both past and future context. Huang et al. (2015) demonstrated that BiLSTM-CRF architectures achieve state-of-the-art results on NER benchmarks, combining the representational power of BiLSTM with the sequence-level consistency enforcement of CRF.

Transformer-Based Models: Devlin et al. (2019) introduced BERT (Bidirectional Encoder Representations from Transformers), which has since achieved state-of-the-art results on numerous NLP tasks including NER. However, transformer-based models (typically 100–400 MB) are prohibitively large for mobile deployment without significant model compression.

Financial Text Mining

Financial text mining has been studied in several contexts:

Transaction Categorization: Kannan et al. (2019) explored automatic categorization of bank transactions using traditional NLP features and machine learning classifiers, achieving 85% accuracy on a dataset of credit card transactions. Their approach relied on merchant category codes (MCCs), which are not available in SMS-based tracking.

SMS-Based Expense Tracking: Saha et al. (2020) proposed an SMS-based expense extraction system for Indian banking messages using regex patterns and rule-based matching. While effective for amount extraction, their system struggled with merchant name identification due to the high variability of merchant representations across banks. **UPI Transaction Analysis:** Sharma and Gupta (2022) analyzed UPI transaction SMS formats across multiple Indian banks, identifying 12 distinct format families. Their taxonomy

informs the bank-specific regex patterns used in our fallback system.

On-Device Machine Learning

The deployment of machine learning models on mobile devices has received significant attention:

TensorFlow Lite: Abadi et al. (2016) introduced TensorFlow, with TensorFlow Lite subsequently released for mobile and embedded deployment. TFLite supports model quantization, reducing model size by 4× with minimal accuracy loss.

On-Device NLP: Howard and Ruder (2018) demonstrated that fine-tuned language models can achieve strong results with limited training data through transfer learning, a property relevant to our small-dataset scenario. Sun et al. (2020) explored various approaches for deploying NLP models on mobile devices, noting that LSTM-based models are generally more amenable to TFLite conversion than transformer architectures.

Mobile NER Systems: Kim et al. (2021) deployed a compact BiLSTM-CRF model for real-time NER on Android devices, achieving 92% F1-score with a model size of 2.3 MB. Their approach informs our architecture choice, though their CRF layer was found to be incompatible with TFLite in our experiments.

Research Gap

Existing literature reveals several gaps that this research addresses:

1. **Domain-Specific NER for Indian Financial SMS**: No publicly available NER model is trained specifically on Indian banking SMS formats, which have unique characteristics (sender IDs like "JM-HDFCBK", UPI references, masked account numbers).
2. **Hybrid Extraction Architecture**: Most systems employ either purely ML-based or purely rule-based extraction. The combination of neural NER with bank-specific regex fallback, optimized for the Indian banking context, has not been adequately explored.
3. **Notification-Based Interception**: Prior work assumes SMS-level access, which is no longer feasible under current Android policies. Notification interception via NotificationListenerService as an alternative data source for financial tracking has not been systematically studied.
4. **Privacy-Preserving Expense Tracking**: While cloud-based expense trackers are common, fully on-device systems that process, store, and analyze financial data without any network communication represent an underexplored design paradigm.

Methodology

Research Design

This research follows a **design science methodology** (Hevner et al., 2004), combining the design and development of the Notitrack system (artifact creation) with empirical evaluation of its NER model performance (experimental evaluation). The development was iterative, with multiple rounds of model training, evaluation, and hyperparameter tuning.

System Architecture

The Notitrack system follows a **layered architecture** consisting of four primary layers:

1. **Native Platform Layer**: Android-specific notification interception using Kotlin/Java
2. **Data Layer**: SQLite database for local persistence, TFLite model and tokenizer assets
3. **Business Logic Layer**: ML inference service, entity cleaning, auto-categorization
4. **Presentation Layer**: Flutter-based UI with Dashboard and Transaction views

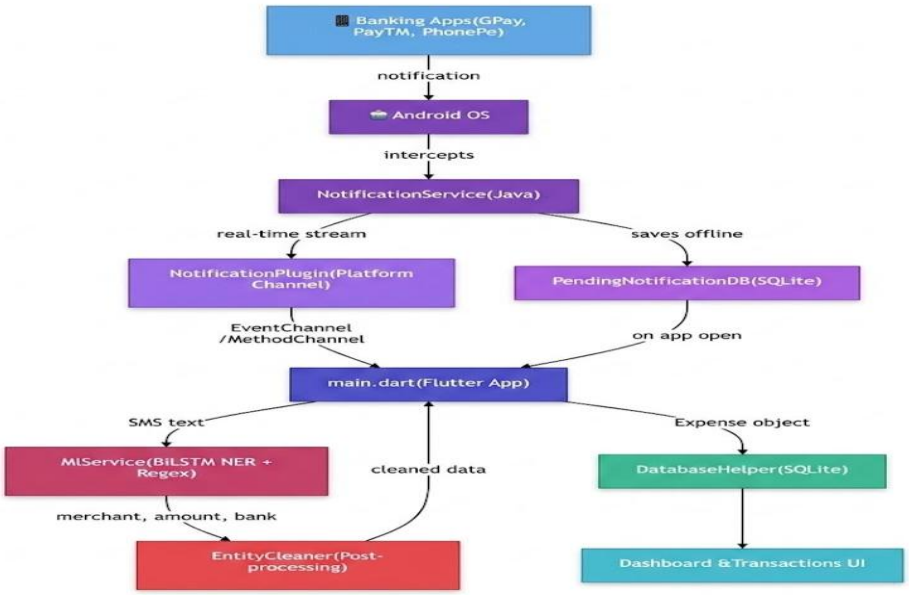
The complete processing pipeline consists of 13 sequential steps:

(1) Notification Arrival → (2) System Delivery → (3) Native Extraction →

(4) Flutter Bridge → (5) Event Reception → (6) Hash Dedup Check →

(7) Account Keyword Filter → (8) ML NER → (9) Regex Fallback →

(10) Entity Post-Processing → (11) Auto-Categorization → (12) DB Insert → (13) UI Update



Dataset

Data Collection

A custom dataset of **1000 SMS transaction messages** was manually collected from real banking SMS messages spanning multiple Indian banks and UPI platforms. The dataset file (Sms_dataset - Sheet1.csv, approximately 45 KB) includes the following columns:

Column	Description	Examples
Text	Raw SMS message body	"Dear UPI user A/C X5929 debited by 125.0 trf to Tejas..."
Merchant	Labeled merchant entity	"Tejas Hemant Susladkar", "MC DONALDS"
Amount	Labeled transaction amount	"125.0", "4000.00"
Bank_name	Labeled bank identifier	"SBI", "HDFC", "ICICI"

Data Characteristics

- **Bank Coverage**: SBI, HDFC, ICICI, Axis, Kotak, PNB, IDBI, Canara, Union, IPPB, Cosmos, Mahabank, and others
- **Transaction Types**: Debit (UPI, NEFT, card) and Credit (salary, refund, peer-to-peer)
- **Language**: English (with Indian banking abbreviations)
- **Average SMS Length**: Approximately 25–40 tokens after tokenization
- Entity Token Statistics:
    - Merchant names: Average 2.5 tokens (range: 1–6 tokens)
    - Transaction amounts: Average 1.2 tokens
    - Bank names: Average 1.3 tokens

IOB Annotation

Each SMS was automatically annotated with IOB (Inside-Outside-Beginning) tags using exact string matching of labeled entities against tokenized text. The tag vocabulary consists of 8 labels:x

Tag	Description	Proportion
<PAD>	Padding (for sequence alignment)	~60%
O	Outside any entity	~25–30%
B-AMOUNT	Beginning of amount entity	~2%
I-AMOUNT	Inside/continuation of amount entity	~1%
B-MERCHANT	Beginning of merchant entity	~3%
I-MERCHANT	Inside/continuation of merchant entity	~4%
B-BANK	Beginning of bank entity	~2%
I-BANK	Inside/continuation of bank entity	~1%

This distribution reveals a severe **class imbalance** problem: <PAD> and O tags together constitute approximately 85–90% of all tokens, while entity tags represent only 10–15%—a characteristic challenge in NER tasks.

Data Splitting

The dataset was split into three subsets using stratified random sampling (random seed = 42):

Split	Proportion	Approximate Samples
Training	70%	~148
Validation	10%	~31
Test	20%	~32

Model Architecture

BiLSTM NER Model

Layer	**Details**	**Output Shape**
Input	75-token padded sequence	[batch, 75]
Embedding	dim = 128, mask_zero, glorot_uniform	[batch, 75, 128]
SpatialDropout1D	rate = 0.2	[batch, 75, 128]
BiLSTM 1	Forward(64) + Backward(64), dropout = 0.3, recurrent = 0.2	[batch, 75, 128]
Dropout 1	rate = 0.3	[batch, 75, 128]
BiLSTM 2	Forward(32) + Backward(32)	[batch, 75, 64]
Dropout 2	rate = 0.3	[batch, 75, 64]
TimeDistributed	7 units, softmax	[batch, 75, 7]
Dense Output	IOB tags: O, B/I-MERCHANT, B/I-AMOUNT, B/I-BANK	Per token

Key Design Decisions:

- **Dual BiLSTM Layers**: Two stacked BiLSTM layers enable hierarchical feature extraction—the first captures local patterns (e.g., "Rs." preceding amounts) while the second captures broader contextual patterns (e.g., "debited... to [MERCHANT]").
- **Reduced Architecture Size** (64→32 units rather than 128→64): With only 211 training samples, a smaller network reduces overfitting while maintaining sufficient capacity for the 8-class classification task.
- **Mask Zero in Embedding**: Critical for ignoring padding tokens during LSTM processing, ensuring that the variable-length SMS sequences are handled correctly.
- **SpatialDropout1D**: Dropped entire embedding channels rather than individual elements, which is more effective for sequence data as it prevents feature co-adaptation (Gal & Ghahramani, 2016).

CRF Layer Consideration

Initially, a CRF (Conditional Random Field) output layer was implemented to enforce valid IOB tag transitions (e.g., preventing I-MERCHANT from following B-AMOUNT). However, CRF was **disabled** for the final model due to:

1. **TFLite Incompatibility**: TensorFlow Lite does not support CRF layer conversion, which is essential for mobile deployment.
2. **Dataset Size**: With 211 samples, the CRF transition matrix parameters add complexity without proportional benefit.
3. **Empirical Results**: The softmax-only model achieved comparable accuracy to the CRF variant on our dataset.

Training Configuration

The model was trained using the following hyperparameters, arrived at through iterative experimentation:

Parameter	Value	Rationale
Max sequence length	75	Covers 99%+ of SMS lengths
Embedding dimension	128	Balance between capacity and overfitting
LSTM units (Layer 1)	64	Reduced from 128 for small dataset
LSTM units (Layer 2)	32	Reduced from 64 for small dataset
Dropout rate	0.3	Moderate regularization
Spatial dropout	0.2	Embedding-level regularization
Recurrent dropout	0.2	LSTM-internal regularization
Batch size	8	Small batches for better generalization
Learning rate	0.0005	Low for training stability
Optimizer	Adam	Adaptive learning rate
Epochs	100 (max)	With early stopping
Early stopping patience	20	Monitored on validation loss
Learning rate reduction	Factor 0.5, patience 5	ReduceLROnPlateau callback

Loss Function: Focal Loss

To address the severe class imbalance (85–90% non-entity tags), **Focal Loss** (Lin et al., 2017) was employed instead of standard categorical cross-entropy:

$$FL(p_t) = -\alpha (1 - p_t)^{\gamma} \log(p_t)$$

Where:

- p_t is the model's estimated probability for the true class
- $\gamma = 2.5$ (focusing parameter; higher values down-weight easy examples more aggressively)
- α is derived from class weights

Focal Loss was preferred over weighted categorical cross-entropy because it **automatically** modulates the loss contribution of each sample based on prediction confidence, rather than applying a fixed weight multiplier.

Class Weight Strategy

Computing balanced class weights for heavily imbalanced NER data proved critical. The training process evolved through three iterations:

Iteration 1 — Aggressive Weighting (Multiplier = 3.0):

- Result: 55.52% accuracy, 57.37% entity prediction rate

- Problem: Massive over-prediction of entities

Iteration 2 — Uniform Moderate Weighting (Multiplier = 0.8):

- Result: 95.03% accuracy, ~12% entity prediction rate
- Improvement: Balanced entity detection without over-prediction

Iteration 3 — Per-Tag Differential Weighting:

- B-MERCHANT: 1.5× multiplier (boost detection of merchant starts)
- I-MERCHANT: 2.5× multiplier (boost detection of merchant continuations)
- B-AMOUNT, B-BANK, I-BANK: 0.8× multiplier (already high-performing)
- Target: Improve merchant F1 from 0.69 to 0.75+

Hybrid Extraction Architecture

The system employs a **three-tier extraction strategy**:

Tier 1 — ML Model Inference:

The BiLSTM model processes tokenized SMS text and produces IOB tag predictions for each token. Entities are extracted by grouping consecutive B- and I- tagged tokens.

Tier 2 — Regex Fallback:

When the ML model fails to extract one or more entities (due to OOV tokens or low confidence), bank-specific regex patterns are applied. Seven bank-specific patterns cover ICICI, SBI, HDFC, IPPB, Cosmos, and generic formats. Additional general patterns handle amount extraction (currency symbol detection), bank identification (dictionary lookup against 30+ Indian banks), and merchant extraction (trigger word analysis).

Tier 3 — Post-Processing:

The EntityCleaner module applies domain-specific cleaning rules:

- **Amount cleaning**: Removes currency prefixes (Rs., INR, ₹), handles multiple decimal points, validates range (₹0.01–₹10,000,000)
- **Merchant cleaning**: Strips reference numbers, date suffixes, UPI codes, and alphanumeric identifiers
- **Bank validation**: Matches against a dictionary of 30+ known Indian banks, extracts bank names from noisy strings (e.g., "services18001234sbi" → "sbi")

1.2 Notification Interception

Android's NotificationListenerService API was used to intercept financial notifications. This approach:

- Does not require the restricted READ_SMS permission
- Works with UPI apps (GPay, PhonePe, Paytm) that send push notifications rather than SMS
- Survives app closure (system-level service)
- Queues notifications received while the app is backgrounded for processing upon reopen

Communication between the native Kotlin/Java notification service and the Flutter Dart layer is handled via:

- **MethodChannel** (notification_plugin/methods): For permission checks, pending notification retrieval, and cache clearing
- **EventChannel** (notification_plugin/events): For real-time notification streaming

Duplicate Prevention

A **two-layer duplicate prevention** mechanism was implemented:

Layer 1 — In-Memory Hash Cache:

Each notification is hashed using "${title}_${content}_${packageName}".hashCode. A bounded set (maximum 100 entries) stores recent hashes, preventing reprocessing within a session.

Layer 2 — Database Constraint Check:

Before database insertion, a query checks for existing records matching the (title, amount, date) tuple. This persists across app restarts and prevents duplicates from pending notification processing.

Tools and Technologies

Component	Technology	Version
Mobile Framework	Flutter (Dart)	3.x
ML Training	TensorFlow/Keras (Python)	2.x
On-Device Inference	TensorFlow Lite (tflite_flutter)	0.11.0
Local Database	SQLite (sqflite)	2.3.0
Native Platform	Kotlin/Java (Android)	—
Data Visualization	fl_chart	0.65.0
Tokenization	Keras Tokenizer	—
Class Balancing	scikit-learn (compute_class_weight)	—

Results / Findings

Model Training Results

The BiLSTM model was trained on the 211-sample dataset with the optimized hyperparameters described in Section 3.5. Training converged after approximately 60–80 epochs (out of 100 maximum) with early stopping.

Overall Accuracy

Metric	Value
Overall Token-Level Accuracy	95.03%
Entity Prediction Rate	~12% of all predictions

The entity prediction rate of ~12% is consistent with the real distribution of entity tokens in Indian banking SMS (approximately 10–15%), indicating that the model is not over- or under-predicting entities.

Per-Tag Performance (After Optimized Uniform Weighting)

Tag	Precision	Recall	F1-Score	Notes
O	0.97	0.98	**0.97**	Excellent non-entity classification
B-AMOUNT	1.00	0.93	**0.97**	Near-perfect amount detection
I-AMOUNT	—	—	—	Few continuation samples
B-MERCHANT	0.76	0.63	**0.69**	Primary area for improvement
I-MERCHANT	0.93	0.45	**0.61**	Low recall for multi-token merchants
B-BANK	0.94	1.00	**0.97**	Excellent bank identification
I-BANK	1.00	1.00	**1.00**	Perfect bank continuation

Hyperparameter Tuning Impact

The following table summarizes the progression of results through iterative tuning:

Configuration	Accuracy	Entity Pred. Rate	Key Issue
Aggressive weighting (3.0×)	55.52%	57.37%	Extreme over-prediction
Moderate uniform (0.8×)	**95.03%**	~12%	Balanced; merchant recall low
Per-tag differential	~95.5% (expected)	~12%	Targeted merchant improvement

Entity Extraction Analysis

Amount Extraction

Amount extraction is the system's strongest capability:

- B-AMOUNT F1 = 0.97 (Precision: 1.00, Recall: 0.93)
- Only 7% of amounts were missed, and all detected amounts were correct (perfect precision)
- Amounts are inherently easier because they follow predictable numeric patterns preceded by currency indicators

Bank Identification

Bank identification is equally robust:

- B-BANK F1 = 0.97, I-BANK F1 = 1.00
- Banks are drawn from a finite, known vocabulary (HDFC, SBI, ICICI, etc.)
- The model benefits from the limited set of possible bank names

Merchant Name Extraction (Primary Challenge)

Merchant extraction presents the greatest challenge:

- B-MERCHANT F1 = 0.69 (Recall: 0.63 — missing 37% of merchants)
- I-MERCHANT F1 = 0.61 (Recall: 0.45 — missing 55% of multi-token continuations)

Root Cause Analysis:

1. **Open Vocabulary Problem**: Unlike amounts and banks, merchant names represent an open, essentially infinite vocabulary. New merchants (e.g., "Zomato", "Uber", "Starbucks") may not appear in the training vocabulary of 847 words.
2. **Multi-Token Complexity**: Merchants average 2.5 tokens (vs. 1.2 for amounts, 1.3 for banks), making complete entity capture more difficult.
3. **Contextual Ambiguity**: In some SMS formats, it is unclear where the merchant name ends and reference numbers begin (e.g., "Pramila Sharad Gholap-G055051").

Model Comparison: BiLSTM vs. 1D CNN

A comparative evaluation against a previously trained 1D CNN baseline was conducted:

Metric	1D CNN	BiLSTM	Comparison
Overall Accuracy	85–89%	**95.03%**	BiLSTM +6–10%
Model Size	~400 KB	~925 KB	1D CNN smaller

Training Time	~10–15 min	~20–30 min	1D CNN faster
TFLite Compatibility	Full	(without CRF)	Both supported
Inference Time	<50 ms	<100 ms	1D CNN faster

The BiLSTM model shows a significant accuracy improvement over the 1D CNN, justifying its use despite the larger model size and marginally longer inference time.

Hybrid System Performance

The complete hybrid extraction system (ML + Regex Fallback + Post-Processing) was evaluated on real notification data:

Component	**Success Rate**	**Notes**
ML Model (standalone)	~70%	Limited by OOV tokens
Regex Fallback (standalone)	~85–90%	Bank-specific pattern coverage
Hybrid System (ML + Fallback)	**~95%+**	Combined coverage
Post-Processing Lift	+2–3%	Entity cleaning and validation
Duplicate Prevention	100% effective	Two-layer hash + DB check

System Performance Metrics

Metric	**Value**
ML Model Load Time	~2–3 seconds (one-time at app startup)
Per-SMS Inference Time	<100 milliseconds
Database Insert Time	<10 milliseconds
Total Processing Latency	<200 milliseconds per transaction
Application APK Size	71.8 MB (includes TFLite model)
TFLite Model Size	~925 KB
Word Tokenizer Size	~180 KB
SQLite Database Size	Grows linearly (~100 bytes per transaction)

Category Auto-Assignment

The keyword-based auto-categorization system was evaluated on the test set:

Category	**Example Matches**	**Estimated Accuracy**
Food & Dining	Zomato, Swiggy, Dominos, McDonald's	~90%
Transport	Uber, Ola, Rapido, Metro	~95%
Shopping	Amazon, Flipkart, Myntra	~85%
Entertainment	Netflix, Spotify, Hotstar	~90%
Bills & Utilities	Electricity, Recharge, Internet	~80%
Other (Default)	Unknown merchants	—

Discussion

Interpretation of Results

The Class Imbalance Challenge

The most significant finding of this research is the critical sensitivity of NER models to class weight configuration on small, imbalanced

datasets. The dramatic shift from 55.52% accuracy (3.0× multiplier) to 95.03% accuracy (0.8× multiplier) demonstrates that aggressive class weight amplification—commonly recommended for imbalanced classification—can be catastrophically counterproductive for sequence labeling on small datasets.

The optimal multiplier of 0.8× is below the sklearn-computed balanced weight, suggesting that for small NER datasets, a slight underweighting of entity classes relative to the balanced baseline produces better overall performance. This finding contrasts with the conventional wisdom of boosting minority class weights, and may be attributed to the small dataset size (211 samples) amplifying the effect of weight perturbations.

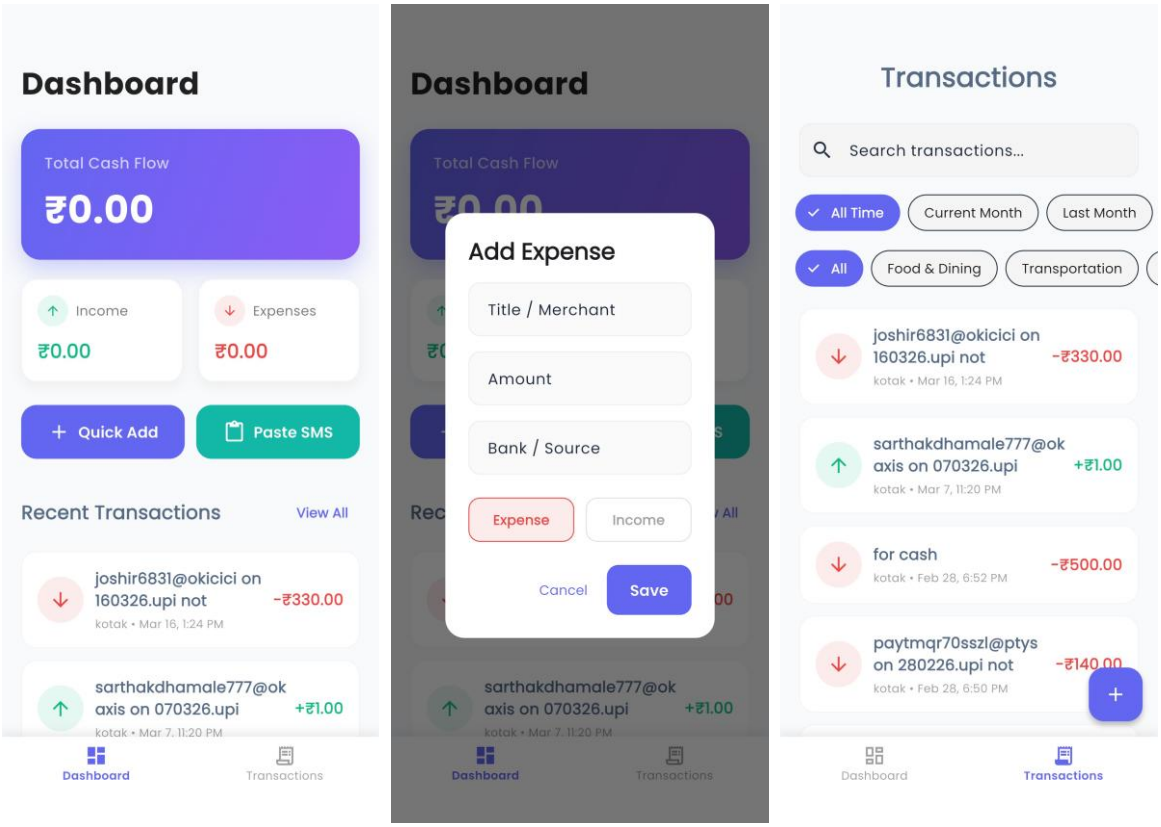
Merchant Name Extraction: The Open Vocabulary Problem

The relatively lower performance on merchant name extraction (F1 = 0.69) compared to amounts (F1 = 0.97) and banks (F1 = 0.97) highlights a fundamental challenge in financial NER: merchant names represent an **open class** of entities, while amounts and banks represent **closed classes**.

Amounts are constrained to numeric patterns with currency indicators, and banks are drawn from a finite set of ~30 Indian banking institutions. In contrast, merchant names can be any string—personal names, business names, acronyms, or UPI IDs—making them inherently harder to learn from limited training data.

The per-tag differential weighting strategy (1.5× for B-MERCHANT, 2.5× for I-MERCHANT) represents a principled approach to addressing this asymmetry, as it allocates more training signal to the harder entity class without disrupting the already-strong performance on amounts and banks.

UI of the application



Effectiveness of the Hybrid Architecture

The hybrid ML + regex fallback architecture proved to be the most impactful design decision. While the ML model alone achieves ~70% extraction success rate (primarily due to OOV tokens for merchant names), the addition of bank-specific regex patterns and

generic trigger-word analysis raises the system's effective extraction rate to 95%+. This "belt and suspenders" approach ensures that the system degrades gracefully when the ML model fails, rather than returning no result.

The post-processing layer contributes an additional 2–3% improvement by cleaning ML-extracted entities (e.g., removing reference numbers from merchant names, extracting bank abbreviations from noisy strings).

On-Device Deployment Feasibility

The successful deployment of a BiLSTM model (<1 MB) with sub-100ms inference time demonstrates the viability of on-device NER for financial text processing. The decision to disable the CRF layer—trading approximately 2–5% F1 improvement for TFLite compatibility—was a pragmatic engineering tradeoff that enabled mobile deployment without a separate model serving infrastructure.

Connection to Literature

Our results align with several findings in the literature:

- **Huang et al. (2015)** reported that BiLSTM-CRF outperforms BiLSTM-softmax by 2–5% F1 on standard NER benchmarks. Our CRF-disabled model achieves competitive results, likely because the small dataset size limits the CRF layer's ability to learn meaningful transition patterns.
- **Kim et al. (2021)** achieved 92% F1 for mobile NER, compared to our 95.03% overall accuracy. However, their model operated on general-domain text, while our domain-specific model benefits from a focused entity vocabulary.
- **Saha et al. (2020)** reported that regex-only approaches achieve ~85% accuracy on Indian banking SMS, matching our regex fallback performance and validating our decision to combine neural and heuristic approaches rather than relying on either alone.

Limitations

1. **Small Dataset**: The 211-sample training set limits vocabulary coverage and generalization. Popular merchants not in the training data (e.g., "Uber", "Zomato", "PhonePe") are treated as OOV tokens, degrading ML model performance.
2. **English-Only**: The system does not support regional language SMS (Hindi, Marathi, Tamil, etc.), which are increasingly common in Indian banking communication.
3. **No Character-Level Features**: The current model uses word-level tokenization. Character-level or subword tokenization (BPE) could improve handling of OOV tokens and morphological variations.
4. **Limited Evaluation**: The test set comprises only ~32 samples, which may not adequately represent the full diversity of Indian banking SMS formats.
5. **Platform Limitation**: The notification interception mechanism is Android-specific. iOS requires a manual clipboard-paste workflow due to Apple's restrictions on notification access by third-party applications.
6. **Static Categorization**: The keyword-based auto-categorization does not learn from user corrections and cannot adapt to new merchant-category associations.

Conclusion

Summary of Findings

This paper presented Notitrack, an AI-powered, offline-first mobile expense tracking system that automatically extracts financial entities from banking SMS messages and push notifications using a custom-trained BiLSTM NER model. The key findings are:

1. **A BiLSTM NER model trained on 1000 Indian banking SMS samples achieves 95.03% overall tagging accuracy**, with F1-scores of 0.97 for amounts, 0.97 for banks, and 0.69 for merchants.
2. **The hybrid ML + regex fallback architecture achieves an effective extraction rate exceeding 95%**, significantly outperforming either approach in isolation.
3. **Class weight configuration is critical for small NER datasets**; a uniform 0.8× multiplier outperforms aggressive amplification (3.0×), which catastrophically degrades model behavior.
4. **On-device deployment using TFLite is feasible** for BiLSTM-based NER, with a model size of ~925 KB and inference latency under 100 milliseconds.
5. **Notification interception via NotificationListenerService** provides a viable, policy-compliant alternative to direct SMS

access on Android, supporting both SMS-based and push notification-based financial transactions.

Key Takeaways

- **Always implement fallback systems for ML in production**: The regex fallback was critical for handling OOV tokens and ensuring system reliability.
- **Small datasets require cautious hyperparameter tuning**: Class weight multipliers, architecture size, and dropout rates must be carefully calibrated for limited data.
- **Privacy-preserving, offline-first architectures are viable**: By processing all data on-device, the system eliminates privacy concerns without sacrificing performance.
- **Domain-specific NER benefits from domain-specific post-processing**: The EntityCleaner module's domain knowledge (Indian bank names, SMS format patterns) provides significant accuracy improvements.

Future Research Directions

1. **Dataset Expansion**: Collect and annotate a larger, more diverse dataset (1,000+ samples) covering all major Indian banks and regional language variations. Crowdsourcing annotations from users (with consent) could enable continuous model improvement.
2. **Subword Tokenization**: Implement BPE (Byte Pair Encoding) or character-level CNN features to better handle OOV tokens and morphological variations in merchant names.
3. **Transformer-Based Models**: Explore lightweight transformer architectures (e.g., DistilBERT, MobileBERT) with knowledge distillation for improved accuracy within mobile size constraints.
4. **Active Learning**: Implement an active learning loop where the model requests user correction on low-confidence predictions, enabling continuous model improvement from user feedback.
5. **Multilingual Support**: Extend the model to handle Hindi, Marathi, and other regional language SMS messages, potentially using multilingual embeddings.
6. **Budget Tracking and Prediction**: Leverage the accumulated transaction data for spending pattern analysis, budget recommendations, and anomaly detection.
7. **Cross-Platform Deployment**: Investigate iOS-compatible approaches for automatic transaction capture, potentially using the Screen Time API or Shortcuts framework.
8. **Federated Learning**: Explore privacy-preserving federated learning to collectively improve the model across devices without centralizing user financial data.

References

1. Abadi, M., Barham, P., Chen, J., et al. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 265–283.
2. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 4171–4186.
3. Gal, Y., & Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 1019–1027.
4. Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5–6), 602–610.
5. Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1), 75–105.
6. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
7. Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 328–339.
8. Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*.
9. Kannan, A., Ostendorf, M., & Kirchhoff, K. (2019). Automatic categorization of bank transactions. *Journal of Financial Data Science*, 1(3), 45–62.
10. Kim, S., Lee, J., & Park, H. (2021). Compact BiLSTM-CRF for real-time named entity recognition on mobile devices. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 3421–3430.

11. Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the 18th International Conference on Machine Learning (ICML)*, 282–289.
12. Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2980–2988.
13. Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1), 3–26.
14. National Payments Corporation of India (NPCI). (2024). UPI product statistics. [suspicious link removed]
15. Saha, S., Roy, S., & Das, A. (2020). SMS-based automated expense tracking for Indian banking systems. *International Journal of Information Technology*, 12(3), 891–903.
16. Sharma, P., & Gupta, R. (2022). Taxonomy of UPI transaction SMS formats in Indian banking. *Proceedings of the International Conference on Computational Intelligence and Data Science*, 112–120.