ISSN: 3068-2525 Volume 4, Issue 3

FEEDFORWARD

Jul-Sep. 2025 MAGAZINE

From the Editor's Desk

Dear Readers,

Greetings! I'm happy to report two important milestones for our chapter. First, our chapter has been recognized with the Outstanding Chapter Award 2024 among all other chapters in the world. Second, Feedforward now has an ISSN.

The prestigious global honor celebrates chapters that deliver a **world-class membership experience**, and I couldn't be prouder of the work.

Over the years, I introduced several new initiatives that transformed engagement and enriched our community, including:

- **Feedforward**—our quarterly magazine https://r6.ieee.org/scv-cs/magazines/ (ISSN 3068-2525)
- **AMA Series**—engaging Q&A sessions with renowned experts https://www.youtube.com/watch?v=x72De1pdkFY&list=PLLsxQYv4DdJlnmTJPcaPW2MLY3XV4bkZp
- **Podcast Series**—spotlighting industry leaders https://podcasts.apple.com/us/podcast/ieee-computer-society-santa-clara-valley-chapter/id1708127425
- **Annual Awards**—recognizing excellence https://r6.ieee.org/scv-cs/?s=awards
- **ICADS**— International Conference on Applied Data Science featuring speakers from all over the world https://www.youtube.com/@vspendyala/search?query=icads
- **IEEE Xplore indexed Conference Sponsorship**—boosting technical visibility https://ieeexplore.ieee.org/xpl/conhome/10164830/proceeding
- **Social Media Pages**
  https://www.linkedin.com/company/ieee-computer-society-scv-chapter/
  https://www.facebook.com/IEEEComputerSocSCVchapter
- **Live-streamed Monthly Events**—connecting with a global audience https://www.youtube.com/watch?v=ACj1GW1Zd9I&list=PLLsxQYv4DdJlYcGPwqUJsnHmfqMtB3eSJ

Now, the time has come to **pass the torch** to the next visionary leader who will take our chapter to even greater heights! The call for nominations will be out soon. Please feel free to nominate or self-nominate for the various officer roles.

We are always looking for more genuinely committed volunteers to help in non-officer roles as well. You can help as a reviewer of the articles, papers, be a guest editor for issues, help organize events, help with the publicity for our events, and more. Please consider being part of the success story by signing up here: https://r6.ieee.org/scv-cs/. With the onset of a new page in the chapter's history, let's continue to Feedforward the chapter to a bright new future. Hope you are all with me in my efforts. Happy reading and happiness always!

With every best wish,
Wednesday, August 7, 2025

Vishnu S. Pendyala
San Jose, California, USA

Article Type: Original

A Deep Learning Aided Approach for Interpretation of ECG Signals and Pathology Detection

Aushim Nagarkatti, *Electrical and Computer Engineering*
Carnegie Mellon University
Pittsburgh, PA 15213

Sahana Ramu, *Electrical and Computer Engineering*
Carnegie Mellon University
Pittsburgh, PA 15213

Sanjana Kandi, *Electrical and Computer Engineering*
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract—With the increased prevalence of AI in disease prediction, we are at an excellent position for applying Deep Learning technologies for predicting cardiovascular abnormalities. In this project, disease classification is performed on the PhysioNet Challenge dataset from 12 lead ECGs for predicting 12 common heart conditions. The aim is to demonstrate through DNNs an ability to achieve excellent disease classification performance in a dataset with heavy imbalance towards Normal Sinus Rhythms (normal heartbeats). Additionally, the paper proposes an efficient and accurate segmentation pipeline to detect the onset and offset of different heartbeat waveforms (P-waves, QRS complexes and T-waves), producing ECG statistics that can be used in a downstream interpretable Machine Learning model. The proposed model also incorporates peak detection as part of the segmentation representations which enhances its interpretability over traditional ECG segmentation models which only segment beat boundaries.

Analysis of electrocardiogram (ECG) signals is one of the most important steps in the diagnosis of cardiac disorders. One of the most common ways for physicians/cardiologists to analyse ECG waveforms is through visual examination of these recordings. However, in most cases, it is difficult and extremely time consuming to analyse such huge amount of data. In order to interpret ECGs, the morphology of its three most important component waveforms, namely, the P-wave, QRS complex, and T-wave, are to be assessed to help diagnose different heart diseases. Therefore, in order to achieve high diagnostic accuracy, the ECG analysis tools/software require the knowledge about the location and morphology of different segment waveforms (P-QRS-T) in ECG records. ECG segmentation can be performed by classical mathematical methods, including such algorithms as heuristic rules built on classical signal processing and wavelet transforms. However, these approaches struggle to distinguish between beat-like artifacts and ECG segments. Deep learning models, especially Artificial Neural Networks (ANN), are currently the most promising way to overcome these limitations. 1-D Convolutional Neural Networks (CNNs) perform an excellent job in capturing short term morphologies which is augmented by recent research that show the benefit of adding memory networks in CNN which preserve morphological information of an ECG signal, thereby distinguishing between QRS like artifacts and P-QRS-T segments.

Digital Object Identifier 10.1109/XXX.0000.0000000

LITERATURE REVIEW

In [1], the authors propose a simple 4-layer CNN model for the classification of 5 typical kinds of arrhythmia signals, i.e., normal, left bundle branch block, right bundle branch block, atrial premature contraction and ventricular premature contraction. The experimental results on the public MIT-BIH arrhythmia database show that the proposed method achieves an accuracy of 97.5 In [2], the authors use different combinations of signal processing methods such as Non-Linear transformations, Non-Linear Principal Component Analysis (PCA), heuristic formulae and simple Multi Layer Perceptron (MLP) networks for performing QRS/PVC classification or ischemia detection purposes. They also explore RBFN for modelling ECG signals for differentiating between normal and abnormal beats, but we do not explore this work too much in this project. In [3], the authors mention a few pre processing methods of which we specifically focus on ECG de- noising. For our work, we seek to remove environmental noise from ECG signals such as baseline drift, and power frequency interference by using three-scale discrete wavelet transform and 2 median filters. These are standard preprocessing steps suggested by [3]. Using DWT for initial denoising followed by median filtering enhances signal quality by combining frequency-specific noise removal with robust spike suppression. In [4], [5] and [6], the authors propose 3 different segmentation architectures, DENS-ECG (combination of CNN and LSTM), 2-input 1-D CNN and Bidirectional RNN with LSTM layers for identifying P, QRS, T segments. The results of these models have been tabulated in Table 1.

TABLE 1: Comparison of Segmentation Architectures

Authors	ANN Architecture	ECG DB	Sensitivity
Peimankar, A., Puthussery-pady, S.	DENS-ECG	QTDB	P: 96.53%, QRS: 99.70%, T: 96.81%
Xiang Y., Lin Z., Meng J.	2-input 1-D CNN	MITBIH	99.86%
Abrishami H., Han C., Zhou X., Campbell M., Czosek R.	Bidirectional RNN with LSTM layers	QTDB	P: 92.00%, QRS: 94.00%, T: 90.00%

All mentioned architectures evaluate their models based on F1-Score. We intend to evaluate our models using the same metrics and additionally showcase confusion matrices for both beat annotations and disease classifications. We want to perform finer segmentation evaluation by further dividing P, QRS and T annotations into peak-start and peak-end statistics for each beat annotation. We will additionally perform statistical tests like McNamer's test to determine the significance of our classification results.

DATASET DESCRIPTION

PhysioNet Database: Includes data from multiple sources: CPSC Database and CPSC-Extra Database, INCART Database, PTB and PTB-XL Database, The Georgia 12-lead ECG Challenge (G12EC) Database, Undisclosed Database These training sets contain 12-lead ECGs of lengths ranging from 6s to 30s and varying sampling frequencies. We use the following datasets from PhysioNet for classification and segmentation tasks. We followed an 80:20 split for the train, test sets for both databases.

PTB Dataset [9]: Initially, we would be using one of the six datasets used in the PhysioNet Challenge 2020 to build the basic pipeline for the classification task. The PTB Diagnostic ECG Database contains 549 records from 290 subjects (aged 17 to 87, mean 57.2; 209 men, mean age 55.5, and 81 women, mean age 61.6). The data used 16 input channels, out of which 14 were for ECG's, 1 for respiration and 1 for line voltage. We would be using 12 conventional ECG leads (i, ii, iii, avr, avl, avf, v1, v2, v3, v4, v5, v6) for this project.

QTDB [10]: Consists of over 100 fifteen-minute two-lead ECG recordings, with onset, peak, and end markers for P, QRS, T, and (where present) U waves from 30 to 50 selected beats in each recording. We would be using this dataset for the segmentation task.

DATA PIPELINE

The PhysioNet data that is available to us is formatted in WFDB format. Each ECG recording uses a binary MATLAB v4 file for the ECG signal data and a plain text file in WFDB header format for the recording and patient attributes, including the diagnosis, i.e., the labels for the recording.

Data Processing: In order to read these signals, we use the Python waveform-database (WFDB) package which is a library of tools for reading, writing, and processing WFDB signals and annotations. From the header (.hea) file of each patient, we extract information such as recording number, sampling frequency which are then used to extract the signal data from the MATLAB (.mat) file for the corresponding recording number.

1) PTB Dataset The sampling frequency for all the

ECG signals is 1000 Hz, which implies that there are 1000 samples for every second. As shown in [11], a low sampling rate might introduce jitter, and a 500-2000Hz sampling frequency is suitable for long-term time domain experiments. Since the ECG signals across all the patients vary from <5 seconds to >60 seconds, we standardize the length of signals in our dataset by splitting signals that are >10 secs. After performing a binary search across the hyperparameter space, it was determined that a 10s window yielded the best results for segmentation. There might be some signals in our dataset that are <10 secs and are hence padded with 0 to overcome this issue.

Finally, to get our training and validation dataset, it is important to make sure that the dataset is not skewed more towards one label since it would make our dataset biased. In a classification setting, we need to ensure that the train and test sets have approximately the same percentage of samples of each target class as the complete set. This can be achieved by performing stratified split on our complete dataset by giving priority to the labels that occur less frequently in our distribution.

2) QTDB Dataset For preprocessing this dataset we do not perform bandpass filtering as we wanted to compare the unfiltered results with those of the DENS-ECG paper [4]. This dataset consists of 7 annotation markers, out of which we use .q1c annotation markers to find P, QRS, T peaks and their boundaries.

Initially, boundaries for waves that did not have annotations were removed. Even after this removal, there were parts of the wave that did not have labels. We used a sliding window approach to discard the window if 30% of the signal or more belonged to one label. This means that the missing labels contributed to one class, overpowering the other.

For the parts of signal that were padded with 0's, the corresponding labels were assigned to the class 7, i.e., T wave end to P wave start. Similarly, for the 4 class problem, these labels were also assigned as above. Finally, in order for the segmentation model to generalize better, the dataset was normalized using min-max normalization.

Data Labelling: The disease names, which are a part of the meta data (present in the .hea file) represent our 12 labels. To make this compatible for computation, these names are converted into their respective SNOMED CT Code as shown in Table 2. In the challenge, the list of SNOMED CT codes and diagnoses has been split into two CSV files: one list for diagnoses that are included in the new scoring function ("scored"), and another list for diagnoses that are ignored during scoring ("unscored"). The scored diagnoses were chosen based on prevalence of the diagnoses in the training data, the severity of the diagnoses, and the ability to determine the diagnoses from ECG recordings. For our baseline model, we used the "scored" set of labels.

SNOMED CT (Systematized Nomenclature of Medicine – Clinical Terms) is a standardized, multilingual vocabulary of clinical terminology that is used by physicians and other health care providers for the electronic exchange of clinical health information.

Since these labels cannot be directly used as inputs to our model, we convert them into output labels in the range (0,11) as shown in Table 2. Since each ECG recording has one or more labels that describe cardiac abnormalities (and/or a normal sinus rhythm), we can use these recordings to perform multilabel classification.

TABLE 2: Cardiac abnormalities to label mapping

Dx	SNOMED CT Code	Label
1st degree av block	270492004	0
atrial fibrillation	164889003	1
bradycardia	426627000	2
complete right bundle branch block	713427006	3
incomplete right bundle branch block	713426002	4
premature atrial contraction	284470004	5
premature ventricular contractions	427172004	6
left bundle branch block	164909002	7
sinus arrhythmia	427393009	8
sinus rhythm	426783006	9
sinus tachycardia	427084000	10
supraventricular premature beats	63593006	11

Baseline Models

Task: Classification

We intend to iterate through all of the proposed models mentioned in our literature review. We start by designing traditional disease classification models as proposed in [1] and [2]. We also intend to explore models such as 1-D ResNet - 18 for performing disease classification. We will be using only 12 of the most important diseases for our baseline implementation.

1) ResNet-18 Implementation: The proposed model can extract multiple features of the ECG data from the same input, which results in efficient representation of the characteristics of the ECG data, thus improving the classification accuracy [7]. The ResNet-18 model being used is an improved version of the existing basic ResNet-18 architecture in order to extract the required

characteristics of the ECG signals being fed which is of length 10000.

For our architecture, the number of input channels in our first layer is 12, since we are using ECG signals acquired from 12 leads. For low-frequency, low-sampling signals such as ECGs, using a kernel of size 3 leads to difficulties in forming a meaningful waveform change. ECG signals have slow-varying, low-frequency waveforms. A small kernel size (like 3) typically observes only a very narrow context (just 3 milliseconds at 1000 Hz), which is insufficient to capture significant waveform characteristics. In addition, these signals are highly susceptible to noise interference, which could have a significant negative impact on feature learning and, in severe cases, can even cause ineffective learning. Therefore, the use of a large convolution kernel size of 33 is proposed here for the effective alleviation of this problem.

In order to achieve better performance, a batch norm is added before the classical ResNet-18 structure to accelerate the training of the neural network, increase the convergence speed, and maintain the stability of the algorithm. We also introduce a dropout layer before the final fully connected layer for reducing over fitting of our model by preventing complex co-adaptations on training data [8].

Finally, the fully connected layer consisting of 12 neurons outputs a [12x1] vector corresponding to the 12 classes. This vector is then passed on to BCELoss() function to output a [12x1] vector with each value corresponding to a probability value for each of the 12 classes. It is important to note here that each signal could have multiple labels, so the architecture presented accounts for multi-label classification paradigms [13].

The hyperparameters used to train our model are:

- Learning Rate = 0.1
- Weight Decay = $5x10^{-5}$
- Momentum = 0.9
- Optimizer = SGD
- Criterion = BCELoss

Task: Segmentation DENS-ECG Implementation: The DENS-ECG model is used for segmentation of P, QRS and T segments from a ECG waveform [15]. This model uses two well-known Deep Learning architectures, CNN and LSTM. The primary task of extracting high level abstract features from ECG signals is done using three one dimensional convolutional layers. These features are then passed into two deep LSTM layers, followed by a dense layer to get posterior probabilities corresponding to four classes, ie, P, QRS, T and NW (No Wave).

For our architecture, the number of input channels in our first layer is 1. We take only 1 channel out of the 2 since there is a very small variation between the two. We use the parameters from the DENS-ECG paper for our CNN layers, with kernel size 3, padding 1 and stride 1. This is immediately followed by 2 LSTM layers with number of layers as 1. The dropout probability in the dropout layer is set to 0.2. The dense layer has 4 hidden units and a softmax function is used as an activation function, which assigns a value between 0 and 1 to each sample of the input ECG signals.

In additional to this, our proposed model also incorporates peak detection as part of the segmentation which enhances its interpretability over traditional ECG segmentation models which only segment beat boundaries [12]. Thus, we predict eight labels, in contrast to DENS-ECG, which only predicts four.

The labels are assigned as follows:

TABLE 3: ECG segments to label mapping

ECG segment	Label
P start to P wave peak	0
P peak to P end	1
P end to QRS start	2
QRS start to QRS peak	3
QRS peak to QRS end	4
QRS end to T wave peak	5
T peak to T wave end	6
T wave end to P wave start	7

The hyperparameters used to train our model are:

- Learning Rate = 0.001
- Weight Decay = $1x10^{-6}$
- Optimizer = Adam
- Criterion = CrossEntropyLoss

PROPOSED MODELS

U-Net: The U-Net architecture is a U shaped one, which consists of a contracting path and an expansive path. The contractive path is generally composed of several CNN layers, followed by a ReLU activation function and max-pooling operation. Here, the spatial information is reduced while feature information is increased. The expansive pathway combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path [14]. Figure 1 illustrates the U-Net architecture used in this work, including both the downsampling and upsampling stages. The configuration in Table 4 details the composition of each block, with skip connections implemented through concatenation to preserve high-resolution features during reconstruction.

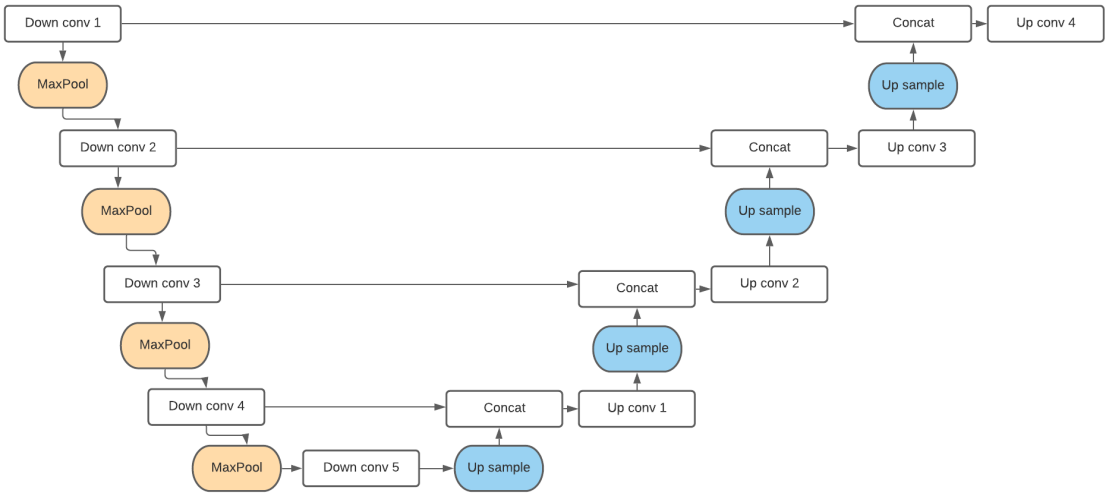

FIGURE 1: U-Net model architecture block diagram

TABLE 4: U-Net architecture: Layer-wise configuration

Block	Layer Type	Parameters	Output Channels
General	Dropout	p=0.1, p=0.2	-
General	MaxPool1D	kernel_size=2	-
Block 1	Conv1D → ReLU	in=1, out=8, kernel=40, stride=1, padding='same'	8
Block 1	Conv1D → ReLU	in=8, out=8, kernel=40, stride=1, padding='same'	8
Block 1	BatchNorm1D	num_features=8 (twice)	8
Block 2	Conv1D → ReLU	in=8, out=16, kernel=20, stride=1, padding='same'	16
Block 2	Conv1D → ReLU	in=16, out=16, kernel=20, stride=1, padding='same'	16
Block 2	BatchNorm1D	num_features=16 (twice)	16
Block 3	Conv1D → ReLU	in=16, out=32, kernel=9, stride=1, padding='same'	32
Block 3	Conv1D → ReLU	in=32, out=32, kernel=9, stride=1, padding='same'	32
Block 3	BatchNorm1D	num_features=32 (twice)	32
Block 4	Conv1D → ReLU	in=32, out=64, kernel=9, stride=1, padding='same'	64
Block 4	Conv1D → ReLU	in=64, out=64, kernel=9, stride=1, padding='same'	64
Block 4	BatchNorm1D	num_features=64 (twice)	64
Block 5	Conv1D → ReLU	in=64, out=128, kernel=9, stride=1, padding='same'	128
Block 5	Conv1D → ReLU	in=128, out=128, kernel=9, stride=1, padding='same'	128
Block 5	BatchNorm1D	num_features=128 (twice)	128
Block 6	ConvTranspose1D	in=128, out=64, kernel=8, stride=2	64
Block 6	Conv1D → ReLU	in=64, out=128, kernel=9, padding='same'	128
Block 6	Conv1D → ReLU	in=128, out=64, kernel=9, padding='same'	64
Block 6	BatchNorm1D	128, 64	64
Block 7	ConvTranspose1D	in=64, out=32, kernel=8, stride=2	32
Block 7	Conv1D → ReLU	in=32, out=64, kernel=9, padding='same'	64
Block 7	Conv1D → ReLU	in=64, out=32, kernel=9, padding='same'	32
Block 7	BatchNorm1D	64, 32	32
Block 8	ConvTranspose1D	in=32, out=16, kernel=8, stride=2	16
Block 8	Conv1D → ReLU	in=16, out=32, kernel=9, padding='same'	32
Block 8	Conv1D → ReLU	in=32, out=16, kernel=9, padding='same'	16
Block 8	BatchNorm1D	32, 16	16
Block 9	ConvTranspose1D	in=16, out=8, kernel=8, stride=2	8
Block 9	Conv1D → ReLU	in=8, out=16, kernel=9, padding='same'	16
Block 9	Conv1D → ReLU	in=16, out=8, kernel=9, padding='same'	8
Block 9	BatchNorm1D	16, 8	8
Final	Conv1D + Softmax	in=8, out=dim_out, kernel=1	dim_out
Output	Linear	in=5082, out=1000	1000

For the down conv section of the proposed U-Net model, we use Conv1D layers with stride 1 and 'same' padding for all layers, kernel size 40 for the first down conv layer, kernel size 20 for the second down conv layer, and kernel size 9 for the rest of the down conv layers. We use a ReLU activation function after every convolution layer, followed by batchnorm layer. An additional dropout layer with dropout value of 0.2 is used for all down conv layers, only for training. This is then followed by another set of convolution, ReLU and batchnorm layer. For the second half of the U-Net structure, we perform upsampling, with kernel size set to 9 for all up conv layers. The output of the unsampling layer is concatenated with the output of the corresponding down conv layer. This is again followed by the same set of layers as discussed above.

The hyperparameters used to train our model are:

- Learning Rate = 0.0005
- Weight Decay = $1x10^{-6}$
- Optimizer = Adam
- Criterion = CrossEntropyLoss
- Scheduler = ReduceLROnPlateau

U-Net + LSTM model: We propose a Deep Learning based approach for the segmentation problem, which is based on combination of U-Net and LSTM architectures. The output of our U-Net is passed through a softmax layer, followed by a linear layer. This is then sent to our LSTM layer with number of layers set to 3 and hidden size set to 64. Post this, the block of Linear-ReLU-Dropout is applied twice. The skeleton architecture of the entire network can be seen in Figure 2.

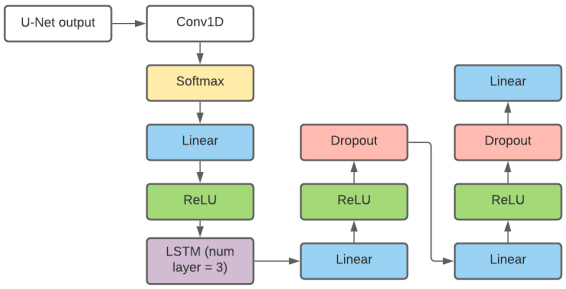

FIGURE 2: Skeleton U-Net + LSTM architecture

The hyperparameters used to train our model are:

- Learning Rate = 0.001
- Weight Decay = $1x10^{-6}$
- Optimizer = Adam
- Criterion = CrossEntropyLoss
- Scheduler = ReduceLROnPlateau

Results and Inference

Classification Results: For the ResNet-18 classification model, we measure the model's performance on 20% of the PTB dataset using a confusion matrix on the validation dataset to display Accuracy and Misclassification Rate as shown in Table 5.

TABLE 5: Accuracy and Misclassification Results

Class	Disease Name	Accuracy	Misclassification rate
1	1st degree av block	0.95	0.04
2	atrial fibrillation	0.92	0.07
3	bradycardia	0.92	0.07
4	complete right bundle branch block	0.99	0.009
5	incomplete right bundle branch block	0.97	0.02
6	premature atrial contraction	0.94	0.05
7	premature ventricular contractions	0.98	0.01
8	left bundle branch block	0.99	0.006
9	sinus arrhythmia	0.97	0.02
10	sinus rhythm	0.96	0.03
11	sinus tachycardia	0.831	0.16
12	supraventricular premature beats	0.69	0.30

$$\text{Accuracy} = \frac{TP}{TP + FP}$$

$$\text{Misclassification Rate} = \frac{FP + FN}{TP + FP + TN + FN}$$

where

- TP: True Positives
- FP: False Positives
- TN: True Negatives
- FN: False Negatives

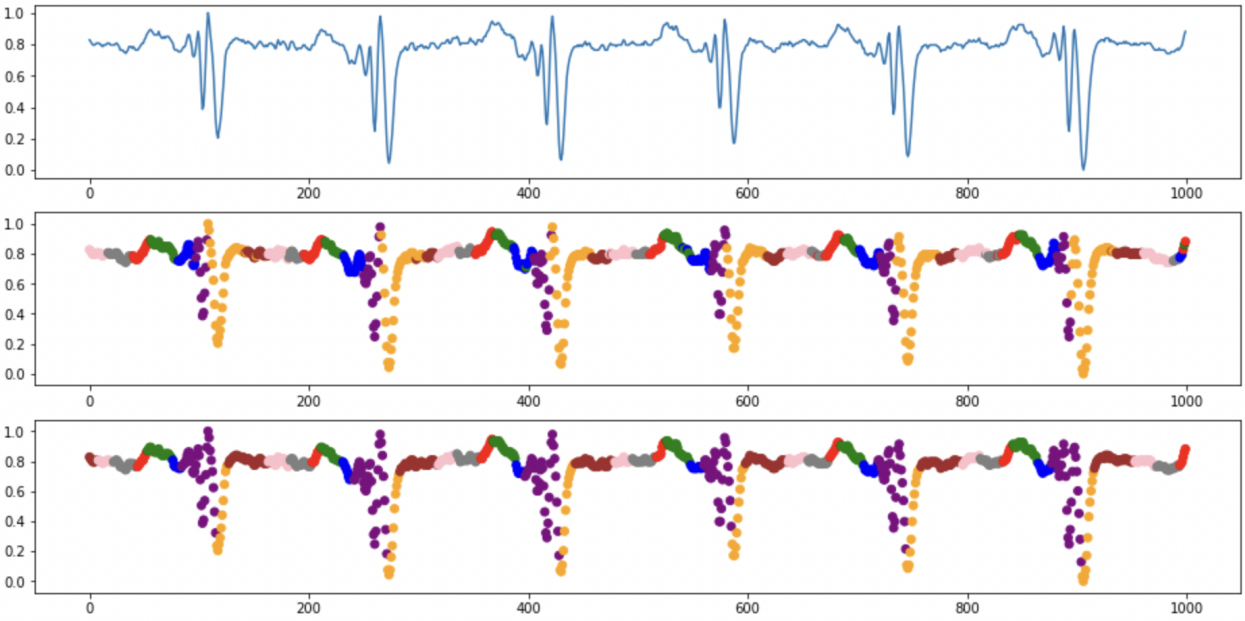

FIGURE 3: Segmentation outputs from U-Net model

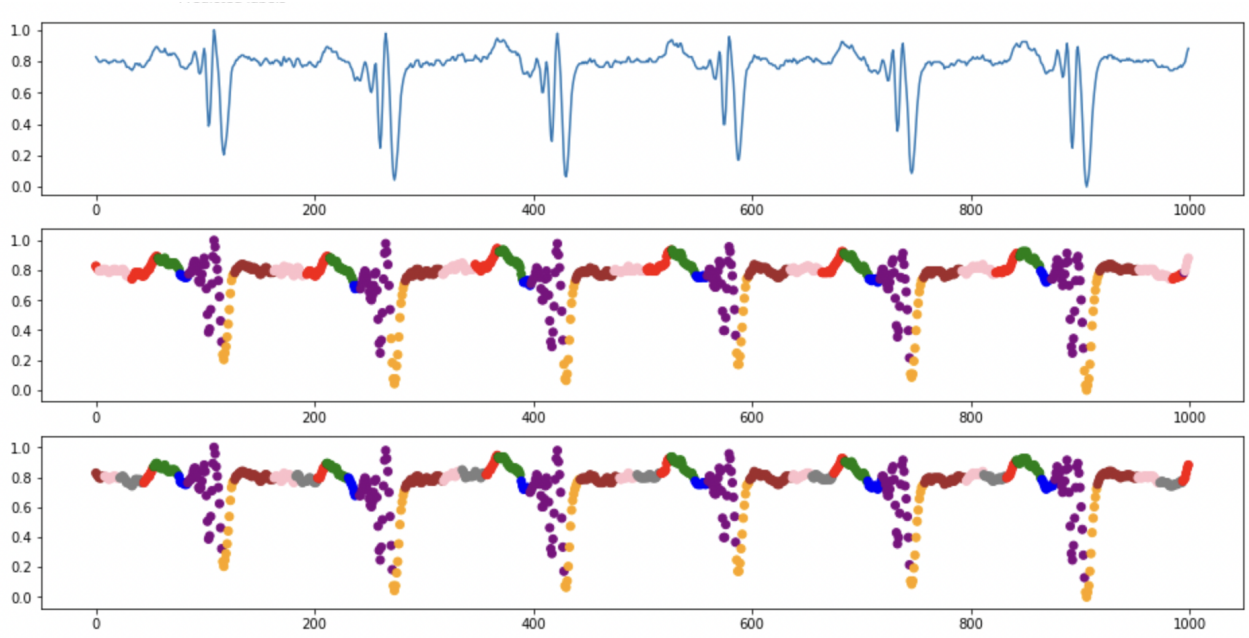

FIGURE 4: Segmentation outputs from DENS ECG model

Segmentation Results For the segmentation task, we compare the F1 scores of the baseline DENS-ECG model with the U-Net + LSTM model proposed for both 4-class and 8-class problem. For this purpose, we use 20% data from Physionet's QTDB dataset. In addition to this, the results are compared by visualizing the waveforms after segmentation. The segments are assigned different colors to differentiate them better.

TABLE 6: F1 Score: DENS-ECG vs U-Net + LSTM for 4 class problem

Model architecture	P	QRS	T	NW
DENS-ECG	0.81	0.90	0.86	0.89
U-Net + LSTM	0.87	0.83	0.94	0.98

While the F1 scores of some of the labels predicted by the U-Net and U-Net + LSTM are lower than that of the baseline DENS-ECG, the overall performance is still comparatively better when we visualize the predicted segments of the waveform for these two U-net based models.

The graphs shown in Figure 3. and Figure 4. compare the original ECG waveform, predicted segmented waveform and the actual segmented waveform. In the case of DENS-ECG, the graph shows that the labels predicted are in the incorrect order when compared to the actual labels. On the other hand, the U-Net model predicts the labels in the same order as the actual labels.

TABLE 7: F1 Score: DENS-ECG vs U-Net vs U-Net + LSTM for 8 class problem

Model architecture	0	1	2	3	4	5	6	7
DENS-ECG	0.68	0.71	0.74	0.86	0.74	0.88	0.74	0.82
U-Net	0.64	0.69	0.76	0.81	0.70	0.86	0.74	0.81
U-Net + LSTM	0.70	0.66	0.74	0.77	0.74	0.89	0.70	0.90

REFERENCES

1. D. Li, J. Zhang, Q. Zhang and X. Wei, "Classification of ECG signals based on 1D convolution neural network," 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom), 2017, pp. 1–6, doi: 10.1109/HealthCom.2017.8210784.
2. N. Maglaveras, T. Stamkapoulos, K. Diamantaras, C. Pappas, M. Strintzis, "ECG pattern recognition and classification using nonlinear transformations and neural networks: a review," Int. J. Med. Inform., vol. 52, pp. 191–208, 1998.
3. J. Li, Y.J. Si, T. Xu, S.B. Jiang, "Deep convolutional neural network based ECG classification system using information fusion and one-hot encoding techniques," Math. Probl. Eng., vol. 2018, Article ID 1387979, 2018.
4. A. Peimankar and S. Puthusserypady, "DENS-ECG: A deep learning approach for ECG signal delineation," Expert Syst. Appl., vol. 165, Article ID 113911, 2021.
5. Y. Xiang, L. Zhitao and M. Jianyi, "Automatic QRS complex detection using two-level convolutional neural network," Biomed. Eng. Online, vol. 17, no. 13, Jan. 2018.
6. H. Abrishami, C. Han, X. Zhou, et al., "Supervised ECG Interval Segmentation Using LSTM Neural Network," in Proc. BIOCOMP Conf., 2018, pp. 71–77.
7. E. Jing, et al., "ECG Heartbeat Classification Based on an Improved ResNet-18 Model," Comput. Math. Methods Med., vol. 2021, Article ID 6649970, Apr. 2021, doi: 10.1155/2021/6649970.
8. G. E. Hinton et al., "Improving neural networks by preventing co-adaptation of feature detectors," arXiv preprint arXiv:1207.0580, 2012.
9. A. L. Goldberger et al., "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," Circulation, vol. 101, no. 23, pp. e215–e220, 2000.
10. P. Laguna, R. G. Mark, A. Goldberg, and G. B. Moody, "A database for evaluation of algorithms for measurement of QT and other waveform intervals in the ECG," Comput. Cardiol., vol. 24, pp. 673–676, 1997, doi: 10.1109/CIC.1997.648140.
11. Task Force of the ESC and NASPE, "Heart rate variability: standards of measurement, physiological interpretation and clinical use," Circulation, vol. 93, no. 5, pp. 1043–1065, 1996.
12. A. M. M. Khandoker, S. Palaniswami, and C. Karmakar, "Deep Learning and Electrocardiography: Systematic Review of Current Advancements," Biomed. Eng. Online, vol. 24, no. 1, Mar. 2025.
13. K. M. Morshed, M. N. Uddin, and T. Rahman, "Enhanced ECG Signal Classification with CNN-LSTM Networks Using Aquila Optimization," Eng. Technol. Appl. Sci. Res., vol. 15, no. 3, pp. 10492–10500, Jun. 2025.
14. X. Yang, L. Zhang, and Y. Liu, "Deep Learning Hybrid Model ECG Classification Using AlexNet and Dual Branch Fusion Network," Sci. Rep., vol. 14, Article 78028, Nov. 2024, doi: 10.1038/s41598-024-78028-8.
15. S. Abedi, R. Gharaviri, and M. Mohebbi, "Robust 12-Lead ECG Classification with Lightweight ResNet," Electronics, vol. 14, no. 10, p. 1941, May 2024, doi: 10.3390/electronics14101941.

Optimized Techniques for Scalable Parallel Processing of Dynamic Graphs: Advances and Real-World Applications

Shubham Malhotra, *Rochester Institute of Technology, Rochester, NY, USA, 14623*

Dr. Meenu Gupta, *Chandigarh University, Punjab, India, 140413*

Fnu Yashu, *Stony Brook University, Stony Brook, NY, USA*

Abstract—Dynamic graphs, whose boundaries and nodes change all the time, are proving more useful in real life for things like social networks, finding fraud, and transport systems. To work with these enormous, changing datasets, we need strategies that can grow, work quickly, and change as needed. This study addresses an innovative approaches in the parallel processing for changing graphs, highlighting maximizing a efficiency strategies that will also boost a speed, minimize latency, and retain then a accuracy. We give a complete assessment of partitioning systems, incremental processing approaches, and parallel frameworks designed for dynamic situations. We also offer an efficient hybrid model integrating message-passing and shared-memory concepts to achieve great adaptability across heterogeneous infrastructures. Real-world case studies, which includes social media analytics and real-time traffic monitoring, are utilized to verify the suggested methodologies. Our results emphasize considerable gains in estimation time and resource use, establishing a stable platform for future breakthroughs in dynamic graph evaluating.

INTRODUCTION

Dynamic illustrations are data establishes that change constantly over time by clarifying, removing, or altering nodes and edges. These sorts of graphs are becoming widespread in sectors such as social networks, deception recognition systems, dialog structures, and transit networks. The continually altering nature of dynamic graphs offers new computing hurdles, especially when analyzing them in tangible time at scale. [1] [2] Conventional static graph processing approaches typically fail to effectively manage the flexible nature of changing datasets. As a consequence, dynamic graph neural networks (DGNNs) and parallel regulations have evolved to solve these challenges. Contemporary work, such as the Decoupled Graph Neural Network (DGNN) architecture, offers flexible solutions for huge dynamic datasets, while approaches like the Recurrent Structurereinforced Graph Transformer (RSGT) represent edge time frame states for higher correctness. [3],[4],[5] Survey studies in this dependent have classified flexible graph analyzing prototypes by how they manage time and structure, and they give useful perspectives into present limits and future research objectives. [6] A important area of progress has been in perpendicular evolving graph computing, where models like DyComPar have proven shared-memory analogy to find communities in enormous networks. [7] Other improvements include GPU-based exciting graph interpreting platforms, which allows an fast connectivity computing while resolving memory management difficulties. [8] Additionally, regulations such as SciDG have developed objectives for assessing dynamic plot archives in scientific uses, while systems like PiPAD allow effective identical coaching of DGNNs on GPU clusters. [9] These methods underscore the need for mixed models that interact the capabilities of shared-memory and messagepassing tactics to manage diverse systems. This study adds to the area by offering an optimal hybrid model for perpendicular absorbing of dynamic graphs. The simulation is assessed via practical applications use cases, ranging from social structure evaluation and real period traffic surveillance, exhibiting gains in execution, flexibility, and asset effectiveness.

LITERATURE REVIEW

PiPAD is a transmitted and adjacent DGNN training framework aimed to boost end-to-end efficiency on GPUs. In their work, the authors presented a system that integrates fast identical multi-snapshot analyzing together with runtime-level canal orchestration. This

technique tries to overcome many fundamental difficulties in DGNN technology, including unnecessary data transfer, unused parallelism, and recollection access inefficiencies. Initial assessments reveal that PiPAD outperforms previous DGNN structures delivering improvements stretching from 1.22× to 9.57×. [10] Xin et al. stated GraphX, a dispersed regulations that integrates data-parallel and graph-parallel execution. It solves ineffectiveness in standard graph lines that depend on exterior systems. GraphX proposes an unifying data emulate managing graphs and accumulations as superior objects. It employs relational joins and compilations for graph processing and incorporates database methods like gauge scans. The strategy delivers profitability equivalent to specialist graph programs while keeping flexibility. Manufacturing implementation indicated an accelerates of improvement to a couple orders of magnitude. [11] Chakaravarthy et al. completed one of the early research concentrating on the capacity of training dynamic Graph Neural Networks (GNNs). They presented a graph-difference focused approach to minimize CPU-toGPU movement duration and established a snapshotbased strategy for distribution. Their approach uses periodic features of dynamic graphs to boost training efficiency. The authors also highlighted potential research topics, including hybrid partitioning for big snapshots and spanning computing with communication. Additionally, they explored the difficulty of scaling Continuous-Time Dynamic Graphs (CTDGs). The work gives essential insights into rapid fluid GNN activation at scale. [12] Dhulipala et al. established that theoreticallyefficient simultaneous algorithmic graphs may achieve great speed and capacity on real-world datasets. The researchers parsed the biggest accessible actual graph utilizing only one shared-memory machine with 1TB RAM. They developed software tools and algorithms designed to allow massive graph computation. Their approaches outperformed distributed-memory systems while utilizing fewer amenities. Notably, efficiency per core was greatly enhanced. This work verifies the practical feasibility of shared-memory computations in parallel for substantial graph insights. [13] Teixeira et al. proposed Arabesque, a distributed graph mining system meant to an address the scalability issues of the transforming centralization procedures into a dispersed solutions. Unlike previously the TLV or TLP methodologies, Arabesque was created from bottom up to facilitate scalable graph extraction. It provides an straightforward and intuitive API, allowing novices to construct the distributed mining applications. influenced by the frameworks like MapReduce and Pregel, it aspires to an reduce barriers to large-scale graph analysis. The system demonstrated outstanding performance and scalability across varied workloads. This paper underlines the significance of reconsidering graph mining for distributed systems. [14]

METHODOLOGY

The proposed research proposes an integrated approach to tackle the issues presented by perpetually changing graph structures, we suggest an integrated parallel computation design aimed at real-time scalability and adaptability. This approach is organized into three fundamental contributions:

Adaptive Structure Partitioning:

The input dynamic net has been split into accessible subgraphs with a reactive split algorithm. This approach monitors node/edge changes and redirects payloads to guarantee balanced computing across worker nodes. Partition boundaries are dynamically modified to prevent load imbalances caused by rapid topology alterations.

Progressive Processing:

Alternative to updating the whole graph's layout following any of an alteration, the system executes updates just on the impacted sections. This is a achieved utilizing methods like delta arithmetic and it also influenced by the events activates which drastically decrease time required for processing. Volatile networks, defined by continually growing nodes and connections, offer new problems that typical static graph machines unable successfully manage. To address these issues, we present a hybrid perpendicular computation system that incorporates adaptive vertex splitting, incremental computing, and a dual-model concurrency approach combining message-passing mixed shared-memory approaches. Our solution starts with adaptable graph separation, where the continually changing graph is separated into numerous divisions that may be adjusted on-the-fly to meet changes in topological and demand. This decreases load imbalance concerns frequently found with static division strategies when the frequent graph changes occur. By concentrating on impacted subgraphs via continuous processing, we avoid an expensive recomputation of the whole graph, instead performing delta computations prompted by event-driven modifications such as node or edge inserting tasks and removal. The primary improvement is in a hybrid concurrent model building design, depicted in a system diagram below. The framework consists of an Administrator Node, which operates as a centralised coordinator capable of the managing job shipment, checking system health, and coordinating update dispersion. The Worker Nodes work in parallel, each

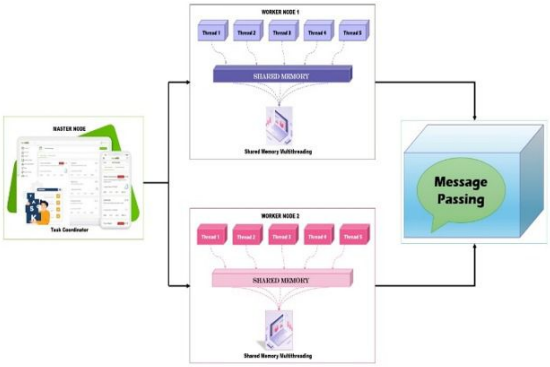

FIGURE 1. Working Diagram for Proposed Method

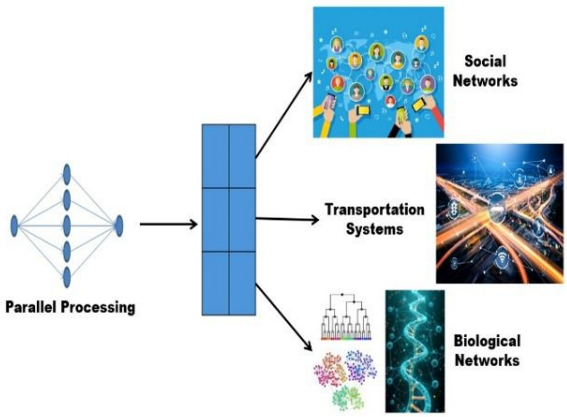

FIGURE 2. Architecture Diagram for Proposed Methodology

handling an division of the graph. Within every single worker node, shared-memory multiple processors is implemented to provide effective intra-node concurrent calculations in a local section data. This multitasking method saves overhead and uses multi-core systems well.

Figure 1 shows the working model for proposed methodology. The design employs a Master Node to organize tasks and handle graph splitting. Personnel which the Nodes execute graph fragments in parallel using shared-memory multithreading. Message Passing provides the dialogue among the nodes for an continuous updates. This blended technique offers rapid, scalable, and accurate dynamic graph synthesis.

The figure 2 depicts the underlying notion of the continuously processing an huge, changing networks utilizing an in parallel processing architecture. At the top, the changing graph is depicted with linked nodes and an emphasized red edge, indicating a real-time change (which could be an added or amended connection). These dynamical changes are continually happening in everyday systems like social media or transportation networks.

Experimental Setup:

The experimental assessment was conducted on a machine equipped with an AMD Ryzen 9 5900X CPU featuring 12 cores, 64 GB of RAM, and an NVIDIA RTX 3080 GPU running Ubuntu 22.04. The implementation in the employed PyTorch and the Deep Graph Library (DGL) for graph neural network operations, in conjunction with MPI for message-passing and custom CUDA kernels to optimize shared-memory processing. Two large-scale dynamic datasets were utilized to validate the system: the SNAP Twitter Social Network dataset, encompassing roughly 41.6 million vertices and 20.4 million nodes, and the NYC Open Traffic dataset, which records around 6.3 million updates each day across 450,000 nodes. The system was evaluated on several graph activities including community recognition using the Louvain method, shortest path updates via a Dijkstra-based approach, and frequent edge insertion and deletion events in to the averaging 5,000 alterations per second. This arrangement facilitated a rigorous evaluation of the proposed model's efficiency and scalability in managing real-time dynamic graph workloads.

RESULT

The recommended hybrid parallel processor architecture it was thoroughly examined utilizing immediate data sets from the two essential domains: community insights and also an real-time traffic monitoring. These categories were chosen from the owing to their extremely flexible and heavy on data character, which make them excellent for assessing the performance and adaptability of fluid graph mining systems. The design was examined based on essential performance criteria such as the processed latency as well as efficiency, flexibility, and accuracy in computations including finding an communities and path revisions. The proposed mixed strategy integrating messagepassing and shared-memory techniques provided a 3.4× accelerate over stationary systems. It lowered delay from 320ms to 95ms and boosted bandwidth to 1340 updates/sec with 94.8derive from flexible segmentation and delta-based updates, avoiding entire graphing recomputation. The simulator displayed stable and scalable effectiveness across varied datasets.

CONCLUSION AND FUTURE ENHANCEMENT

Volatile graphs provide substantial hurdles for typical static approaches, particularly in the real-time settings needing quick, fast processing. This study provides an hybrid parallelism model that utilizes shared-memory and message-passing methods with flexible partitions and incremental updates, yielding enhanced the latency, scalability, and also an accuracy. Potential updates that include incorporating edge computing, artificial intelligence for predictive dividing, support for heterogeneous graphs, including an energy-efficient analysis. Adding real-time visualization and enhancing tolerance for failure and cryptography will further increase usability and dependability, make the framework well-suited for convoluted, vital applications.

REFERENCES

1. C. C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," ACM Computing Surveys (CSUR), vol. 47, no. 1, pp. 1–36, 2014
2. N. K. Ahmed, J. Neville, and R. Kompella, "Network sampling: From static to streaming graphs," ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 8, no. 2, pp. 1–56, 2014
3. A. Longa et al., "Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities," arXiv preprint arXiv:2302.01018, 2023.
4. L. Yang, S. Adam, and C. Chatelain, "Dynamic graph representation learning with neural networks: A survey," arXiv preprint arXiv:2304.05729, 2023.
5. S. Hu et al., "Dynamic graph representation via edge temporal states modeling and structure-reinforced transformer," arXiv preprint arXiv:2304.10079, 2023.
6. B. Chakraborty et al., "Dynamic graph structure estimation using spiking neural networks," arXiv preprint arXiv:2504.01246, 2025.
7. S. Malhotra et al., "Efficient algorithms for parallel dynamic graph processing," International Journal of Computer Network and Information Security (IJCNIS), vol. 15, no. 2, pp. 519–534, 2023.
8. H. Gao et al., "A survey on dynamic graph processing on GPUs," Frontiers of Computer Science, vol. 18, article 184106, 2024.
9. C. Zeng et al., "SciDG: Benchmarking scientific dynamic graph queries," in Proceedings of the 2023 International Conference on Scientific and Statistical Database Management (SSDBM 2023), 2023.
10. C. Wang, D. Sun, and Y. Bai, "PiPAD: Pipelined and parallel dynamic GNN training on GPUs," in Proceedings of the ACM Symposium on Principles and Practice of Parallel Programming (PPoPP), 2023.
11. . R. S. Xin, D. Crankshaw, A. Dave, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "GraphX: Unifying data-aparallel and graph-parallel analytics," unpublished manuscript
12. V. T. Chakaravarthy et al., "Efficient scaling of dynamic graph neural networks," arXiv preprint arXiv:2109.07893, 2021.
13. L. Dhulipala et al., "Theoretically efficient parallel graph algorithms can be fast and scalable," arXiv preprint arXiv:1805.05208, 2018.
14. C. H. C. Teixeira et al., "Arabesque: A system for distributed graph mining," in Proceedings of the 25th Symposium on Operating Systems Principles (SOSP'15), 2015.

Shubham Malhotra is a seasoned software engineer and technology innovator with extensive expertise in cloud computing, distributed systems, performance engineering and optimization, DevOps, and full-stack development. He has driven the design and deployment of scalable, secure systems at leading organizations like AmazonAWS, Microsoft Azure, where his work has streamlined operations and enhanced real-time data processing. Shubham's innovative approach has led him to develop distributed automation tools and API solutions that integrate cutting-edge technologies with robust cloud infrastructures. His passion for leveraging technology to solve complex challenges is further evident in his pioneering projects, such as an AI powered sales simulation platform and intelligent data analysis systems. In addition to his technical contributions, Shubham is a dedicated thought leader - actively participating in elite tech communities, reinforcing his commitment to shaping the future of modern software solutions. Contact him at shubham.malhotra28@gmail.com.

Dr. Meenu Gupta is with the Department of Computer Science & Engineering at Chandigarh University, Punjab, India. Her current research interests include artificial intelligence, deep learning, and computer vision. Contact her at gupta.meenu5@gmail.com.

Fnu Yashu is a forward-thinking technology strategist and Senior Member of Technical Staff at Broadcom, renowned for her expertise in bridging software-defined data centers with modern cloud computing services. With years of industry experience, she has spearheaded initiatives that deploy advanced cloud agents to transform traditional infrastructures into agile, scalable, and secure ecosystems. Yashu's career spans influential roles at VMware and Informatica, where she consistently delivered innovative microservices architectures through robust CI/CD pipelines.

Her pioneering work, underscored by filed and approved patents, highlights her commitment to driving technological evolution in data center automation. An alumnus of Stony Brook University and Thapar Institute of Engineering Technology. Yashu continues to push the boundaries of software engineering, solidifying her reputation as a key influencer in the digital transformation landscape using cloud computing.

An LLM-Powered API Navigator: Building an Intelligent Assistant for API Specification Understanding

Ahmed Aziz Ben Aissa, *Aivancity, La Grande Ecole De L'intelligence Artificielle Et De La Data, 57 Av. du Président Wilson, 94230 Cachan, France*

Thibaut Goutorbe, *Aivancity, La Grande Ecole De L'intelligence Artificielle Et De La Data, 57 Av. du Président Wilson, 94230 Cachan, France*

Ravi Prakash V, *FIDE (formerly Beckn Foundation), Bangalore, India*

Anuradha Kar, *Aivancity, La Grande Ecole De L'intelligence Artificielle Et De La Data, 57 Av. du Président Wilson, 94230 Cachan, France*

Abstract—Developers often find it challenging to understand the intricacies of complex and constantly changing API specifications, both when dealing with broad standards like OpenAPI and also specialized protocols in certain domains. To address this, we have created an open source AI-powered tool to explore APIs, designed to make navigating API documentation much easier using advanced large language models. Our Python-based system leverages semantic search, vector embeddings, and AI-driven query processing to provide precise, contextually relevant answers straight from API documentation. We use LlamaIndex for data ingestion and indexing, MistralAI for creating embeddings, and ChromaDB for storing these embeddings long-term. LangGraph manages the query process, enabling us to select relevant documents dynamically, rewrite queries for better understanding, and retrieve information semantically. The user interface, built with FastAPI, allows for an interactive experience where developers can explore API structures and definitions in real time. Testing with OpenAPI and another domain-specific protocol shows that the tool is highly effective at handling both common and unique API formats, ultimately improving developer access to complicated technical information through intelligent AI assistance.

Introduction

In today's world of software development which requires APIs for communication, integration, and data exchange, understanding complex API specifications has become a critical skill for developers, technical writers, and system architects. Examples of API standards include OpenAPI and domain-specific ones like the Beckn protocol as these provide structured definitions of endpoints, request and response formats, and schema relationships. However, navigating these documents, most of which are written in YAML or markdown can be difficult and error-prone, especially for users who are not familiar with their structure or syntax.

The requirement for intelligent tools that support natural language interaction with API documentation is therefore growing [1]. Traditional keyword based search tools offer limited support for dynamic, semantically rich exploration. In contrast, recent advances in retrieval-augmented generation (RAG), vector embeddings, and large language models (LLMs) open new possibilities to improve how users interact with and understand API specifications[2 3].

Recent advances in large language models (LLMs) have enabled a variety of developer-assistive tools[4 5 6 7]. Notable examples include GitHub Copilot and OpenAI Codex 8 9, which generate code snippets and pro vide autocomplete capabilities within integrated development environments (IDEs). Other systems have explored retrieval-augmented generation (RAG) pipelines for code search 10 and natural language interfaces to query databases and APIs[11 12].

However, these tools primarily target generalpurpose code generation or question answering. Few

studies have addressed the challenge of navigating and transforming highly structured API specifications—especially those defined in machine-readable formats such as OpenAPI YAML and domain-specific protocols like Beckn. Our work is novel in combining semantic retrieval, query reformulation, and transformation code generation within a unified system tailored specifically to API specification comprehension and mapping. This approach supports both general REST APIs and specialized protocols, bridging a gap between LLM-powered assistants and structured interoperability standards.

In this article we present an intelligent API documentation explorer assistant that enables users to query and comprehend complex API specifications using natural language. Our system integrates semantic indexing, document embeddings, and dynamic workflow routing to deliver context-aware, precise answers from technical API documents. It supports both general-purpose APIs like OpenAPI and domainspecific standards such as Beckn protocol, and is designed to assist users in identifying endpoints, understanding schema definitions, and comparing structural elements without requiring deep familiarity with the source format.

By combining LlamaIndex for semantic indexing, MistralAI embeddings for vector representation, and LangGraph for flexible query routing and processing, we demonstrate a robust approach to augmenting technical documentation with AI. Our assistant is deployed via a FastAPI based web interface, providing an interactive method for understanding API specifications. This work contributes to the growing field of AI assisted developer tools, showcasing how LLM based retrieval systems can improve accessibility, efficiency, and comprehension in technical documentation workflows.

In this article,we first present the challenges faced by developers in dealing with complex API specifications, followed by description of our LLM based API explorer, its workflow and finally discussing the challenges faced by LLM based API interpretation as concluding remarks.

CHALLENGES OF COMPLEX API INTERPRETATION

Interpreting and effectively using complex APIs comes with a number of challenges. These are especially evident when working with large-scale, poorly documented, or highly abstracted APIs. Typical problems faced by developers include 1) Complex structures: Complex APIs often involve hierarchies and abstract classes, making it difficult for developers to estimate the correct way to use them without extensive digging 2) Poor or incomplete documentation: Many APIs sometimes have missing, outdated, or unclear documentation, that miss critical details 3) Hidden setup and dependencies: Many APIs rely on implicit context (e.g., initialization, environment settings) that is not obvious from the interface, leading to confusion and runtime errors. 4) Unclear use patterns and error handling: Finding out the correct order of method calls, state transitions, or properly handling errors can be difficult, especially when examples or guidelines are lacking. 5) Version mismatches and unexpected side effects: Developers may run into problems due to changes in API versions or mainly when the documentation doesn't include the latest updates.

AN LLM POWERED INTELLIGENT API EXPLORER

We developed an intelligent assistant to help users query and interpret complex API specifications, with a focus for generic and domain specific protocols. The system uses semantic search, vector embeddings, and large language models (LLMs) to generate accurate, context-aware responses directly from API documentation.

This work draws on two primary sources of API documentation:

1) **OpenAPI** (https://www.openapis.org/): OpenAPI is a specification[13] for building APIs that allows developers to describe the structure, endpoints, and behavior of RESTful web services in a standardized, machine-readable format. It is a widely used standard for documenting RESTful APIs. We used the official OpenAPI 3.1.0 Markdown specification to capture core elements such as endpoints, parameters, and request bodies. The structure and components of OpenAPI v2.0 and v3.0 are shown in Figure 1.
2) **Beckn API** (https://becknprotocol.io/): The Beckn protocol[14] is an open, domain-specific standard designed for decentralized digital commerce. Its modular, layered architecture presents a challenging and rich test case for API interpretation. The Beckn protocol architecture is shown in Figure 2, whereas the Beckn ecosystem outlining developer resources is shown in Figure 3.

For this study, we used a collection of OpenAPI-compliant YAML files covering key Beckn domains such as transaction, registry, and metadata, along with their corresponding schema index files and component

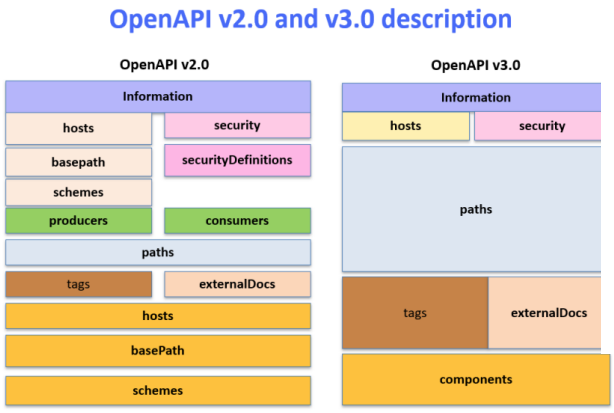

FIGURE 1. Overview of the OpenAPI specification structure, that shows the hierarchical organization of components such as paths, operations, hosts, request bodies, responses, and reusable schemas defined in the components section.)

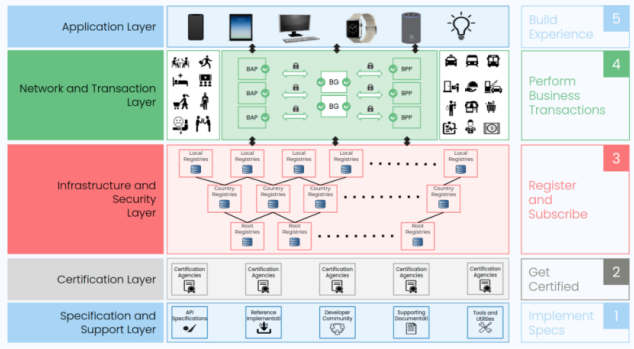

FIGURE 2. Overview of the Beckn architecture which prescribes multiple layers stacked on top of one another, each with clearly defined roles and functions. (Adapted from https://developers.becknprotocol.io/ with permission.

definitions, organized under /api/ and /schema/ directories. All files were parsed, semantically structured into document chunks, embedded using LLM-based embedding models, and stored in a vector database for efficient semantic search and retrieval. Our system is designed with a modular architecture that integrates several key components:

- **MistralAI**[15] supports both document embedding and question answering, leveraging the `mistral-small` and `mistral-medium` models.
- **LlamaIndex**[16] is used to build and query semantic indices from the ingested API documentation.

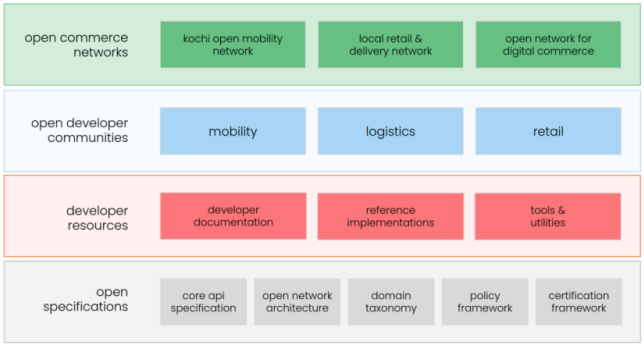

FIGURE 3. A schematic diagram of various components of the Beckn ecosystem or its "building blocks". (Adapted from https://developers.becknprotocol.io/ with permission)

- **LangGraph**[17] functions as a custom graph-based flow controller, managing the sequence of operations including classification, rewriting, querying, and retrieval.
- **ChromaDB**[18] acts as the persistent vector store for storing and retrieving embedded document vectors.
- **FastAPI**[19] serves as the lightweight web framework, powering the application and providing a form-based user interface.

STEP-BY-STEP WORKFLOW

The first major stage of our system involves building a semantic search foundation by transforming raw API documentation into vectorized, searchable representations. This process, which we call an ingestion pipeline, also ensures that natural language queries can later be matched accurately to relevant sections of the API specs. The pipeline consists of the following key components: document loading, text splitting, semantic embedding, and persistent storage. The steps are illustrated in Figure 4.

Step 1: Ingestion Pipeline and Vector Store Document loading: This initial step handles the extraction and parsing of API specification files from multiple formats and structures. The system supports both general-purpose API standards and domainspecific protocols. For example, OpenAPI documentation is ingested in Markdown or YAML format (openapi.yaml). In the case of the Beckn protocol, we process OpenAPI-compliant YAML files corresponding to specific domains like transaction.yaml, registry.yaml, and meta.yaml, along with their associated schema files. Each document is parsed and converted to structured objects. For YAML files, both endpoint details and

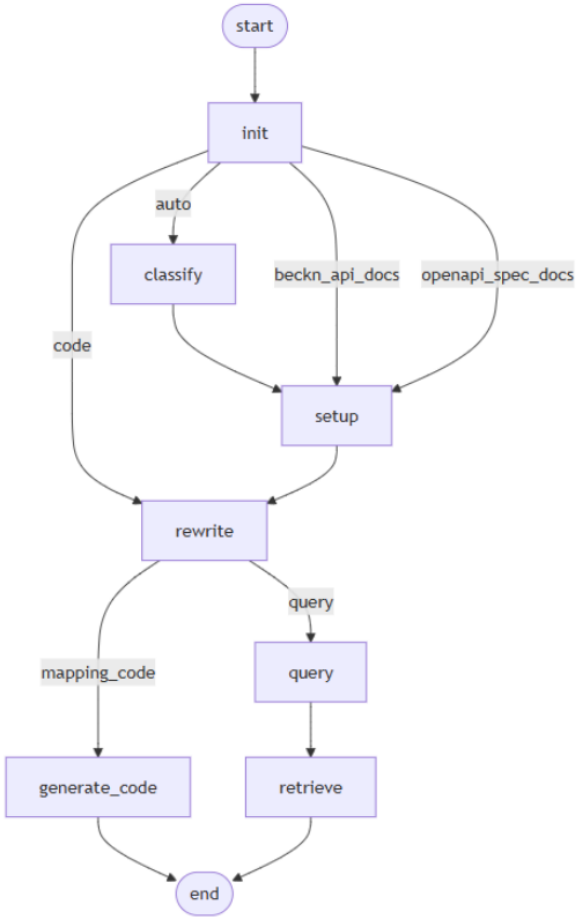

FIGURE 4. Functional components of the Query interpretation and response generation workflow (Step 2)

nested schema definitions are extracted. This deep parsing enables the creation of richly indexed documents that support cross-referencing between endpoints and data schemas.

Text Splitting and Embedding: In this step, once documents are loaded, they are broken down into semantically coherent chunks to ensure meaningful retrieval, while ensuring the contextual integrity across sections. Each chunk is then embedded using a custom wrapper that interfaces with Mistral's embedding API. The wrapper includes a throttling mechanism to manage API rate limits. The resulting embeddings are high-dimensional vector representations that encode the semantic content of the documentation segments, making them suitable for similarity-based retrieval.

We chose Mistral AI embeddings in combination with LlamaIndex and ChromaDB for several reasons. First, Mistral provides lightweight embedding models (mistral-embed) with 768-dimensional vectors that strike a balance between computational efficiency and retrieval performance. In contrast, larger embeddings (e.g., OpenAI Ada or Cohere) incur higher latency and cost, especially when indexing large YAML files. Second, Mistral models integrate seamlessly with LlamaIndex pipelines and support high-throughput ingestion with effective throttling and batching. Finally, Mistral's licensing and deployment options are favorable for research prototypes and low-resource environments. Although LlamaIndex supports multiple embedding providers, our experiments showed Mistral embeddings maintained consistent retrieval quality while reducing memory footprint and response times in the vector store. This makes the stack particularly wellsuited for interactive API exploration tools.

Text Splitting and Embedding Configuration During ingestion, API documentation was split into semantically coherent units using the SentenceSplitter transformation. The main hyperparameters were:

1) **Chunk overlap:** `chunk_overlap = 0`. This means no sentences were repeated between adjacent chunks, avoiding redundancy.
2) **Chunk size:** The `SentenceSplitter` was used in sentence-based mode, relying on default behavior that groups sentences into chunks with an approximate maximum size of 500 tokens, depending on sentence boundaries.
3) **Embedding configuration:** We used the `ThrottledMistralAIEmbedding` wrapper configured with the following:
   - **Embedding model:** `mistral-embed`
   - **Embedding dimensionality:** 768
   - **Distance metric:** Cosine similarity
4) **Throttling mechanism:** Enabled to ensure compliance with API rate limits. This configuration was selected to balance embedding throughput with retrieval precision.

Vector storage: The final step involves storing semantic vectors in ChromaDB which is a persistent vector database optimized for high-performance similarity search. We structure the data into distinct collections based on the source and type of documentation. These include embeddings from the official OpenAPI Markdown specification, vectors generated from user-provided OpenAPI YAML files, and domain-specific Beckn API definitions and schema embeddings. To minimize redundant processing and reduce API calls, the ingestion pipeline first checks ChromaDB for existing vector IDs before embedding new content. This ensures that documents are embedded only

once unless modifications are detected. This ingestion pipeline forms the backbone of our intelligent API documentation assistant, enabling fast, accurate, and semantically rich query responses across complex API specifications.

Step 2: Query Interpretation and Response Generation: Once the API documentation is ingested, user queries are handled through a modular and intelligent workflow powered by LangGraph. The workflow is visualized as a conditional state graph that enables flexible routing based on the nature of the query. The steps of query interpretation and response generation consist of the following:

Initialization and Routing: The process begins with checking whether the user has selected a specific document source. If the source is set to auto, the query is routed to the classify node.

Classification: Using the Mistral small model, the system classifies the query based on its semantics. For example it can classify Beckn specific terms, generic OpenAPI concepts and mapping code if the goal is code transformation between APIs. The classification result determines the document collection to be used for retrieval.

Setup and Indexing: The setup node loads the correct ChromaDB collection, wraps it in a ChromaVectorStore, and builds a semantic index via LlamaIndex. A query engine and retriever are initialized for semantic search.

Query Reformulation: At the rewrite node, the user query is reformulated using the LLM to enhance clarity and search performance. The reformulated query is more likely to match relevant documentation chunks.

Conditional execution: If the goal is mapping code, the workflow branches to generate code, which retrieves relevant documentation from both OpenAPI and Beckn collections, builds a mapping prompt, and invokes the LLM to return a mapping guide and a Python transformation function. Otherwise, the query is passed to the query node, where the query engine performs semantic retrieval and generates an answer.

Final Retrieval and Output: From the query node, the flow continues to retrieve, which extracts the top-k document chunks and their scores. The final output includes: a) The rewritten query b) LLM-generated answer c) Retrieved documents d) Confidence score (if available). These results are rendered in the FastAPI interface and delivered back to the user. This structured and conditional architecture supports a broad range of use cases from basic documentation lookup to advanced code generation tasks.

Summary of workflow: Relationship between

Tech Stack

Component	Tech Details
LLMs	MistralAI (Small for rewriting/classification, Medium for answers/code)
Embeddings	`mistral-embed` (with rate throttling)
Vector DB	ChromaDB
Framework	FastAPI
Logic Flow	LangGraph
Parsing	LlamaIndex, YAML
Frontend	Jinja2 + HTML
Docs Format	OpenAPI 3.1.0 YAML & Markdown

FIGURE 5. The technical components of our LLM powered API explorer assistant

Ingestion Pipeline and LangGraph Workflow To summarize the workflow, the ingestion pipeline transforms raw API documentation into semantic embeddings stored in ChromaDB. It includes document loading, sentence splitting (chunk overlap=0), embedding with MistralAI, and persistent vector storage. This process is performed once per API dataset to create a searchable vector index. This is followed by query interpretation and Response Generation. The Figure 4 exclusively illustrates this stage. It shows the LangGraph-based workflow that processes user queries dynamically. Nodes in Figure 4 correspond to:

- **init:** Session initialization and context setup.
- **classify:** Query classification to select relevant API sources.
- **setup:** Retrieval configuration for ChromaDB collections.
- **rewrite:** Query reformulation to improve retrieval accuracy.
- **query:** LLM-based semantic search.
- **retrieve:** Top-*k* document chunk extraction.
- **generate code:** Code mapping (if applicable).

This separation clarifies that ingestion and indexing are decoupled from the runtime query flow depicted in Figure 4.

All these steps as seen from the user point of view are shown in Figures 5 and 6.

EVALUATION AND TESTING

To validate the system's performance, we manually tested the query pipeline with various types of questions targeting both OpenAPI and Beckn specifications.

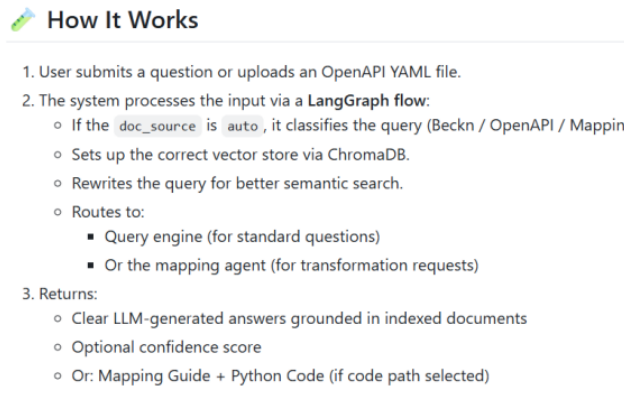

FIGURE 6. Sequential overview of the input and output stages of the API explorer

These tests aimed to assess the accuracy, relevance, and formatting of the LLM-generated answers, as well as the quality of the retrieved documents.

We evaluated 30 representative queries across three categories:

- 10 Beckn-specific queries
- 10 general OpenAPI queries
- 10 mapping queries (transforming OpenAPI structures to Beckn equivalents)

Each query was assessed using four criteria:

1) **Answer Precision** – accuracy and completeness.
2) **Confidence** – retrieval confidence more than 85 percent.
3) **YAML Retrieval** – correctness of schema extraction.
4) **Latency** – response time under 3 seconds.

The results demonstrate consistently high performance on Beckn and OpenAPI queries, while mapping queries—being inherently more complex—showed modestly lower success rates in YAML extraction and answer precision.

Example queries included: OpenAPI:

- What is the difference between `parameters` and `requestBody`?
- How are `enum`s represented in OpenAPI 3.1?

Beckn Protocol:

- What is the purpose of the `on_search` endpoint?
- Which fields are mandatory in a `cancel` request?

For each query, we evaluated the following:

Queries	Number	Precision of answer	Confidence >85%	YAML Retrieval	Latency <3 seconds
Beckn	10	8/10	10/10	9/10	10/10
OpenAPI	10	9/10	9/10	9/10	9/10
Mapping Queries	10	6/10	8/10	6/10	9/10

FIGURE 7. Performance estimates of the query pipeline

1) **Answer precision:** Whether the LLM provided an accurate and complete answer.
2) **Confidence score:** Returned by the query engine, typically above 85 percent for well-structured queries.
3) **YAML retrieval:** Whether the raw structure was correctly extracted and displayed.
4) **Responsiveness:** System latency was acceptable (<3 seconds) even with multiple LangGraph steps.

The performance evaluation results are presented in Figure 7. Overall, the assistant performed well on most queries. Its performance was particularly effective for queries involving known schema terms and standard OpenAPI elements.

CHALLENGES AND CONCLUSION

This article outlines the development of a new and open source intelligent assistant powered by a fine-tuned LLM designed to automate the interpretation of API documentation. Built using the Mistral LLM, the system is trained to read API specifications, generate developer-friendly resources, and promote interoperability across sectors. While it performs well on standard schema and OpenAPI elements, challenges remain particularly in handling ambiguous user queries, which often lack clarity or specificity. Additionally, without real-time access to the latest API updates, the model risks producing outdated suggestions in rapidly evolving API domains. Real-time performance is a concern, because deploying large models in environments like IDEs requires low-latency inference that may not be feasible on constrained hardware. In addition, evaluating the effectiveness of the assistant remains difficult due to the lack of standardized benchmarks that reflect real-world development tasks. Despite these challenges, our LLM-driven framework bridges the gap between complex API documentation and practical API exploration. Future work in this direction includes expanding support for additional API specification formats, integrating multimodal capabilities (e.g., code

+ visual schema parsing), and improving real-time performance.

SUPPLEMENTARY MATERIAL

The repository describing the implementation of the LLM-based API explorer can be found here: https://github.com/zaizou1003/AI$_C$*LINIC*.

REFERENCES

1. R. Lehmann, "Towards Interoperability of APIs - an LLM-based approach," *Proc. 25th Int. Middleware Conf.: Demos, Posters and Doctoral Symposium*, Hong Kong, Hong Kong, pp. 29–30, 2024.
2. D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu and B. Myers, "Using an LLM to Help with Code Understanding," *2024 IEEE/ACM 46th Int. Conf. Software Engineering (ICSE)*, Lisbon, Portugal, pp. 1184–1196, 2024. doi: 10.1145/3597503.3639187
3. M. Kim, T. Stennett, D. Shah, S. Sinha, and A. Orso, "Leveraging Large Language Models to Improve REST API Testing," *Proc. 2024 ACM/IEEE 44th Int. Conf. Software Engineering: New Ideas and Emerging Results*, Lisbon, Portugal, pp. 37–41, 2024.
4. J. Chen et al., "When LLMs meet API documentation: Can retrieval augmentation aid code generation just as it helps developers?," *arXiv [cs.SE]*, 2025.
5. J. Liu, Y. Yang, K. Chen, and M. Lin, "Generating API parameter security rules with LLM for API misuse detection," *arXiv [cs.CR]*, 2024.
6. H. Liu, J. Liao, D. Feng, K. Xu, and H. Wang, "AutoFeedback: An LLM-based framework for efficient and accurate API request generation," *arXiv [cs.SE]*, 2024.
7. Y. Wu, P. He, Z. Wang, S. Wang, Y. Tian, and T.-H. Chen, "A comprehensive framework for evaluating API-oriented code generation in large language models," *arXiv [cs.SE]*, 2024.
8. GitHub, "GitHub Copilot." [Online]. Available: https://github.com/features/copilot
9. M. Chen et al., "Evaluating Large Language Models Trained on Code," *ArXiv*, vol. abs/2107.03374, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2107.03374
10. P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," *arXiv [cs.CL]*, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2005.11401
11. S. Zhou, U. Alon, F. F. Xu, Z. Wang, Z. Jiang, and G. Neubig, "DocPrompting: Generating code by retrieving the docs," *arXiv [cs.CL]*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2207.05987
12. O. Khattab and M. Zaharia, "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT," *Proc. 43rd Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, Virtual Event, China, pp. 39–48, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2004.12832
13. OpenAPI Initiative, "OpenAPI Specification v3.1.0," 2021. [Online]. Available: https://spec.openapis.org/oas/v3.1.0
14. Beckn Protocol, "Beckn Protocol Documentation," 2023. [Online]. Available: https://becknprotocol.io/
15. Mistral AI, "Mistral API Documentation," 2024. [Online]. Available: https://docs.mistral.ai/
16. LlamaIndex, "LlamaIndex Documentation," 2024. [Online]. Available: https://docs.llamaindex.ai/
17. LangGraph, "LangGraph Framework," 2024. [Online]. Available: https://www.langgraph.dev/
18. ChromaDB, "Chroma Vector Database," 2024. [Online]. Available: https://docs.trychroma.com/
19. FastAPI, "FastAPI Framework," 2024. [Online]. Available: https://fastapi.tiangolo.com/

- **Ahmed Aziz Ben Aissa** is a Master's student in AI and Data Science at Aivancity School in Paris with interests in LLMs, Agentic AI, and software engineering.
- **Thibaut Goutourbe** is a Master's student in AI and Data Science at Aivancity School in Paris, with interest in LLMs, computer vision, and real-time AI applications.
- **Ravi Prakash V** is the Head of Architecture and Technology Ecosystem at FIDE (previously Beckn Foundation). He is the genesis co-author of the Beckn protocol specification.
- **Anuradha Kar** is an Associate Professor in AI and Robotics at Aivancity Paris Cachan, France, where she teaches and mentors graduate students in deep learning, AI for health, computer vision, explainable AI, and human-computer interactions. Contact her at `kar@aivancity.ai`.

A Scalable Cloud-Based System for Real-Time Deepfake Detection

Shubham Malhotra, *Rochester Institute of Technology, Rochester, NY, USA, 14623*

Dr. Meenu Gupta, *Chandigarh University, Punjab, India, 140413*

Muhammad Saqib, *Texas Tech University, Lubbock, TX, USA, 79409*

Abdul Muqtadir Mohammed, *University at Buffalo Getzville, NY, USA, 14068*

Abstract—In the era of digital media, the rapid development of AI-based manipulation techniques like deepfakes and audio-video lip-sync changes has raised concerns about the truthfulness of video footage. These technologies are increasingly being used by malicious actors to spread false information, slander people, or produce convincingly fraudulent videos. To solve this important problem, we suggest that can reliably identify videos as authentic or fraudulent based on audio-visual coherence. The hybrid deep learning (DL) framework used in our method integrates Long Short-Term Memory (LSTM) networks for periodic sequence prediction with ResNet-based Convolutional Neural Networks (CNNs) for spatial feature extraction. The suggested model illustrates enhanced performance in comparison to traditional deep learning techniques with a highest test accuracy of 99.67testing by using confusion matrices, pair plots, heatmaps, and accuracy comparison tests confirms the accuracy and stability of the model. Streamlit is used to enable simple installation of the system as a web application, hence increasing its accessibility and usability by enabling users to provide videos and obtain real-time lip-sync authenticity analysis in a fast manner. The project seeks to provide a trustworthy tool for the detection of tampered video content, hence benefiting journalism, media forensics, and digital content verification.

INTRODUCTION

In the past decade, there has been an unprecedented increase in the efficiency of visual speech recognition software, due to the breakthroughs in deep learning techniques and the affordability of big data sets[1].

Lip comprehension may be described as the capacity to interpret what individuals communicate via their perception of lip movement. Reading one's lips is a tough skill for humans since lip motions pertaining to distinct letters are outwardly identical (e.g., b and p or d and t)[2].

To properly capitalize on the data obtained, we constructed a deep neural system which takes the characteristic of the voice as input and produces the 3D vertex distortion. The voice component is extracted using a voice recognition model trained, and the pixel displacement is utilized to drive 3D mesh simulations to produce realistic face motions coordinated with the input vocal sound. The proposed network combines a one-dimensional convolution and LSTM (long-short-term memory) and is able to generate realistic, smooth, and natural facial animations [3].

This technology combines artificial intelligence, computer vision, and natural language processing to create a lifelike digital representation of a person, delivering speech realistically and expressively. This paper presents an SLR to provide insight into the advances, limitations, and gaps in facial expression and lip movement synchronization of an audio track. Initially [4], researchers heavily depended on the Hidden Markov Model (HMM) for facial animation, with the aim of capturing the dynamics of video and speech sequences [5].

To achieve faithful speaking face generation, two critical issues need to be considered: the high fidelity of the subject's appearance details and the synchronization of lip movement with the audio input. To generate lip-synced talking videos, prior methods directly model the mapping between the audio signal and visual content [6].

Standard video counterfeiting includes painstak-

ing frame-level changes using programs like Adobe Premiere, frequently leaving obvious spatial artifacts. However, Deepfakes employ computational models to make lifelike alterations. Contemporary detection relies on temporal irregularities and cognitive discrepancies, such as intermittent blinking and odd head movements [7,8,9,10].

Lip-sync deepfake scams utilize two distinct technologies. First, a celebrity's voice undergoes cloning from genuine audio samples. Previously requiring hours of recordings for realistic voice replication, modern methods achieve this with mere minutes of authentic audio. Deepfake denotes deployment of deep learning algorithms to fabricate manipulated digital content— video or audio—frequently employed maliciously for fraud, defamation, and spreading disinformation or propaganda. Small- and large-scale fraud, and to produce dis-information designed to disrupt democratic elections and sow civil unrest.

LITERATURE REVIEW

Recent advancements in audio-visual deepfake detection have shown significant promise in lip-sync analysis using hybrid neural architectures. For example, the LSTM module utilizes a 0.5 dropout probability to iteratively evaluate frame sequencing efficiently, allowing end-to-end model training requiring explicit loss coefficients. Since manipulations may happen at any moment of deepfake production, fixed-length uninterrupted frame sections served as input, providing a classification efficiency of 97% on footage segments under 2 seconds [11].

Researchers have also made a lot of progress in lipreading by using deep learning systems including 3D convex networks, MouthNet, Bi-LSTM layers, and CTC goal inside an functions. The model worked well, as shown by high classification accuracies 96.2% and also 93.8% on the Oulu-VS2 and GRID information sets, correspondingly [12].

Moreover, the precision in identifying deepfake movies altered by Faceswap and FSGAN increased from 0.91 to 0.98 and 0.96, respectively, by supplying extra details to the model, so illustrating the efficacy of multisensory fusion [13].

In a separate procedure, the integration of spatial convolutions with residual and Bi-LSTM nets has demonstrated stable performance on real-life data sets. On the Lipreading In-The-Wild benchmark, a tough sample of 500 target words, an average proficiency of 83% was attained—surpassing prior standards by 6.8% [14].

Nevertheless even powerful APIs like those of the two giants may be misled. It has been revealed that 78of deepfake material successfully misled through the Microsoft Azure testing API, underlining the demand for more robust detection mechanisms [15].

In the case of Faceswap Wav2Lip, FSGAN Wav2Lip, and related test sets, detection accuracy reached 98%, 97%, 96%, 94%, and 94% respectively. Nonetheless, the performance declined slightly on certain test sets like RTVC, due to specific challenges associated with those datasets [16].

Another study reported an impressive 92.3% testing accuracy, 94.3% recall, and 93.3% F1 score on the LRW dataset. The use of band-pass filters and dual-resolution image streams enhanced performance, with peak accuracies of 94%, indicating effective detection of forged video content [17]. Moreover, average detection accuracy exceeded 95.3% in spotting lip-syncing videos, outperforming baseline methods. Even in real-world scenarios like WeChat video calls, the model achieved up to 90.2accuracy, showcasing robustness across varying environment [18]. A face-swapping-based approach combining InceptionV3 (CNN) and LSTM was proposed, where CNN extracts frame-level features and LSTM constructs sequential descriptors. This model achieved over 97accuracy in distinguishing between pristine and manipulated videos [19].

Common CNN architectures like ResNet, InceptionV3, VGG-16, and MobileNet have also been employed as lip feature extractors. Despite noise in the data, these models demonstrated strong real-world applicability, achieving 74–75% across all performance metrics [20].

Finally, temporal dynamics modelled through LSTM and their fusion using Bi-LSTM have shown consistent improvements. In particular, this approach reported a 9.7% absolute improvement on the OuluVS2 dataset and 1.5% on the CUAVE dataset, confirming the value of sequence modelling in visual speech recognition [21].

The literature study presents a complete basis for understanding current breakthroughs and also limits in the audio-visual deepfake and lip-syncs detection. By reviewing the major studies that employ CNNs, LSTMs, BiLSTMs, and also the hybrid models, it shows the continued search of accurate and real-time detection approaches. This research together stresses the relevance of spatiotemporal features, multimodal inconsistencies, and resilient model designs in the attaining high detection of the performance. The literature also underlines how deepfake material may trick even commercial APIs, thus emphasizing the importance of accurate detection methods. The selection of relevant research gives insight into datasets, experimental set-

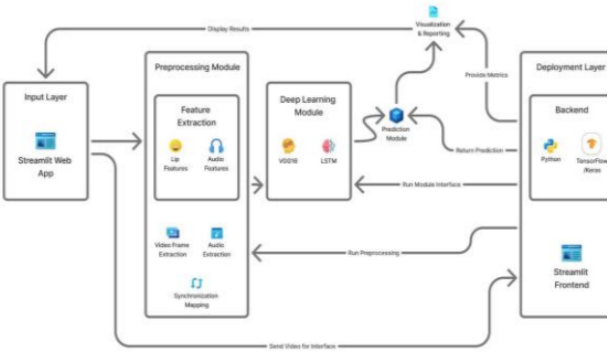

FIGURE 1. Architecture of Deep Fake Detection

FIGURE 2. Demonstrate the Formation and Testing Accuracy of the Restnet procedure model

tings, and assessment methodologies, which influence the development of our proposed hybrid CNN-LSTM model. This foundation supports the research direction and verifies the design choices used in the offered study. Ultimately, the literature analysis supports the paper's objective of constructing a high-accuracy, web-accessible detection system that answers the critical demand for trustworthy tools in media integrity verification.

METHODOLOGY

The structure of our AI Lip the Sync button identification system comprises of interconnected modules built for real-time, accurate identification of the edited movies. It starts with an Streamlit-based web interaction for user footage upload, providing real-time analysis. In the preliminary processing step, each video is divided into frames, and audio is individually extracted. Lip characteristics such as the height, breadth, and mouth element in the ratio—are extracted from preserves while auditory features like MFCCs are calculated. Data capture is carried out to the acquire isolated, frontal video data for dependable input [22].

Audio-visual features are synchronized using timestamps. Feature vectors are generated from lip sequences using a pre-trained ResNet-18 model [23]. These vectors are passed into a VGG19-LSTM hybrid, where CNN extracts spatial features and LSTM captures temporal dependencies. Results are visualized with metrics and confidence scores in the Streamlit interface. as shown in figure:1

Resnet

The Res2Net architecture aims at improving multi-scale representation by increasing the number of available receptive fields. This is achieved by connecting

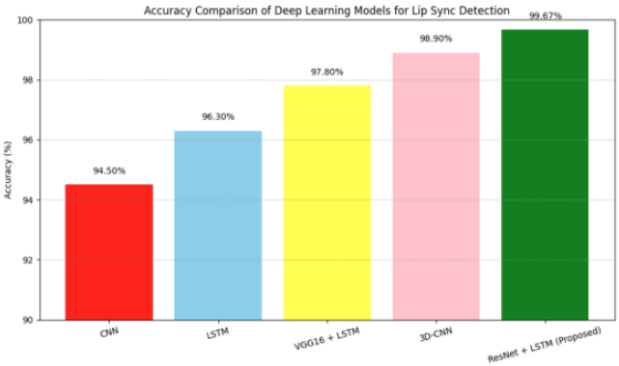

FIGURE 3. Provides a comparison effectiveness graph, highlighting several models is assessed

smaller filter groups within one block in a hierarchical residual-like style [24].

LSTM

With LSTM, neural networks can recall both current data and more distant past data. Since LSTM can discover without disregarding a long-term prerequisites using its memory indefinite model, it can be used to orderly and time series issues.

Video

The DeepFake videos are created in two major ways (i) Face swapping and (ii) Face morphing. Both of these techniques operate in different ways. The main difference lies in the way these techniques manipulate and transform facial features [25].

The suggested ResNet + LSTM hybrid building design greatly exceeds than others, obtaining an peak reliability of 99.67%. This emphasizes it is an greater

capabilities in collecting both spatial and temporal data for robust categorization.

The suggested technology may be used in a lot of different sectors. For communication sites, it can help verify popular videos and find political information that has been changed. In human resources, it may be used with video interview platforms like Zoom, Microsoft Teams, or specialist screening technologies like HackerRank and Codility to check the identification of candidates and make sure their interview answers are real. In journalism and media fabrication, it helps make sure that source footage is accurate so that false information doesn't get out. The Streamlit-based deployment makes it even easier for non-technical people to operate, allowing for smooth immediate operation throughout all sectors.

RESULT AND CONCLUSION

The recommended hybrid profoundly learning system shows outstanding an efficiency in identifying altered movies based on visual or audio coherence. By integrating an Long Short-Term Memory (LSTM) coalitions for temporal trends recognition with ResNet-based Convolutional Neural Networks (CNNs) for spatial features extraction, the simulation obtained an maximum test precision of 99.67% on a dataset of 20,000 labeled films. This indicates an major improvement greater than typical CNN or LSTM-only models.

Conclusion and Future Enhancement

The put forth hybrid LSTM-ResNet model efficiently identifies manipulated movies with an excessive accuracy of 99.67%, providing accurate lip-sync reliability analysis. Its combination with Streamlit provides immediate time user involvement, aiding media forensics and digital authentication. In future, the framework may be upgraded with real-time the flow identification, a bilingual datasets, and adversarial resilience. Handy and edge execution may boost accessibility. Utilizing explainable AI will boost transparency. Lastly, enabling automatic report production may help forensic record keeping.

One of the most captivating aspects about it is that it focuses on lip-sync designs, which is an area of deepfake detection that hasn't been studied enough. It also has an immediate, intuitive with an Streamlit interface. The model was evaluated on varied datasets (e.g., VoxCeleb2, FaceForensics++) encompassing Equal gender, different nations, various nations, and variable video/audio quality, displaying constant resilience. The system also gives understandable results through per-frame score confidence and audio-video alignment graphs. In the future, visual warmth maps will be added to make the system even better. The contribution is important for multimedia forensics and real-world use, but it will be more useful and have a bigger impact in production contexts if it is tested more thoroughly, made easier to understand, and given a wider range of data sets.

REFERENCES

1. T. Afouras, J. S. Chung, and A. Zisserman, "Deep lip reading: a comparison of models and an online application," *arXiv preprint arXiv:1806.06053*, 2018.
2. A. M. Sarhan, N. M. Elshennawy, and D. M. Ibrahim, "HLR-Net: a hybrid Lip-Reading model based on deep convolutional neural networks," *Computers, Materials & Continua*, vol. 68, no. 2, pp. 1531–1549, 2021.
3. X. Li et al., "A novel speech-driven lip-sync model with CNN and LSTM," *Proc. 14th Int. Congr. Image Signal Process., BioMedical Eng. Inform. (CISPB-MEI)*, 2021.
4. M. H. Alshahrani and M. S. Maashi, "A Systematic Literature Review: Facial expression and lip Movement Synchronization of an audio track," *IEEE Access*, vol. 12, pp. 75220–75237, 2024.
5. D. Pawar, P. Borde, and P. Yannawar, "Generating dynamic lip-syncing using target audio in a multimedia environment," *Natural Language Processing Journal*, vol. 8, p. 100084, 2024.
6. W. Zhong et al., "High-fidelity and Lip-synced Talking Face Synthesis via Landmark-based Diffusion Model," *arXiv preprint arXiv:2408.05416*, 2024.
7. Y. Gu, X. Zhao, C. Gong, and X. Yi, "Deepfake video detection using Audio-Visual Consistency," *Lecture Notes in Computer Science*, pp. 168–180, 2021.
8. M. Bohacek and H. Farid, "Lost in Translation: Lip-Sync Deepfake Detection from Audio-Video Mismatch," *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2024.
9. A. Kharel, M. Paranjape, and A. Bera, "DFTrans-Fusion: Multimodal deepfake detection via lip-audio cross-attention and facial self-attention," *arXiv preprint arXiv:2309.06511*, 2023.
10. S. Agarwal et al., "Detecting deep-fake videos from phoneme-viseme mismatches," *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops*, 2020.
11. S. Tipper, H. F. Atlam, and H. S. Lallie, "An Investigation into the Utilisation of CNN with LSTM for Video Deepfake Detection," *Applied Sciences*, vol. 14, no. 21, p. 9754, 2024.

12. L. He, B. Ding, H. Wang, and T. Zhang, "An optimal 3D convolutional neural network based lipreading method," *IET Image Processing*, vol. 16, no. 1, pp. 113–122, 2021.
13. S. A. Shahzad et al., "AV-Lip-Sync+: Leveraging AVHuBERT to exploit multimodal inconsistency for video deepfake detection," *arXiv preprint arXiv:2311.02733*, 2023.
14. T. Stafylakis and G. Tzimiropoulos, "Combining Residual Networks with LSTMs for Lipreading," *arXiv preprint arXiv:1703.04105*, 2017.
15. F. Abbas and A. Taeihagh, "Unmasking deepfakes: A systematic review of deepfake detection and generation techniques using artificial intelligence," *Expert Syst. Appl.*, vol. 252, p. 124260, 2024.
16. S. A. Shahzad et al., "Lip sync matters: A novel multimodal forgery detector," *Proc. APSIPA ASC*, 2022.
17. Y. Liu et al., "Lip-Audio Modality fusion for deep forgery video detection," *Computers, Materials & Continua*, vol. 0, no. 0, pp. 1–10, 2024.
18. W. Liu et al., "Lips are lying: Spotting the temporal inconsistency between audio and visual in lipsyncing deepfakes," *Advances in Neural Information Processing Systems*, vol. 37, pp. 91131–91155, 2024.
19. A. Malik, M. Kuribayashi, S. M. Abdullahi, and A. N. Khan, "DeepFake detection for human face images and Videos: a survey," *IEEE Access*, vol. 10, pp. 18757–18775, 2022.
20. S. Vekkot, T. S. Gupta, K. P. Karthik, and D. Kaushik, "Enhanced lip reading using deep model feature fusion: A study on the MIRACL-VC1 dataset," *Procedia Comput. Sci.*, vol. 258, pp. 1189–1198, 2025.
21. S. Petridis, Z. Li, and M. Pantic, "End-to-end visual speech recognition with LSTMs," *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2017.
22. N. Aripin and A. Setiawan, "Indonesian Lip-Reading Detection and Recognition based on lip shape using face mesh and Long-Term Recurrent Convolutional Network," *Appl. Comput. Intell. Soft Comput.*, vol. 2024, p. 6479124, 2024.
23. X. Li et al., "Replay and synthetic speech detection with res2net architecture," *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2021.
24. Ü. Atila and F. Sabaz, "Turkish lip-reading using BiLSTM and deep learning models," *Engineering Science and Technology, an International Journal*, vol. 35, p. 101206, 2022.
25. R. Khan et al., "Comparative study of deep learning techniques for DeepFake video detection," *ICT Express*, 2024.

Shubham Malhotra is a seasoned software engineer and technology innovator with extensive expertise in cloud computing, distributed systems, performance engineering and optimization, DevOps, and full-stack development. He has driven the design and deployment of scalable, secure systems at leading organizations like AmazonAWS, Microsoft Azure, where his work has streamlined operations and enhanced real-time data processing. Shubham's innovative approach has led him to develop distributed automation tools and API solutions that integrate cutting-edge technologies with robust cloud infrastructures. His passion for leveraging technology to solve complex challenges is further evident in his pioneering projects, such as an AI powered sales simulation platform and intelligent data analysis systems. In addition to his technical contributions, Shubham is a dedicated thought leader - actively participating in elite tech communities, reinforcing his commitment to shaping the future of modern software solutions. Contact him at shubham.malhotra28@gmail.com.

Dr. Meenu Gupta is with the Department of Computer Science & Engineering at Chandigarh University, Punjab, India. Her current research interests include artificial intelligence, deep learning, and computer vision. Contact her at gupta.meenu5@gmail.com.

Muhammad Saqib is a technology expert and researcher in the fields of artificial intelligence, cloud computing, and cybersecurity systems. He has applied intelligent technologies to real-world issues in areas like healthcare, financial services, and the manufacturing industries. In the area of cybersecurity, he has assisted in the creation of deep learning models that can identify threats in real-time through the use of pattern recognition and anomaly detection methods. These systems are designed to protect data, adapt to new threats, and reduce human intervention. He is also deeply involved in the formation of secure, scalable cloud-based platforms to support modern business applications. He is involved in the integration of AI with cloud infrastructure to develop systems that are not only technically powerful but also ethical, transparent, and user-friendly. Contact him at saqibraopk@hotmail.com.

Abdul Muqtadir Mohammed is a seasoned software engineer affiliated with the Department of Computer Science at the State University of New York. His work spans AI/ML, intelligent routing, and large-scale logistics systems. He holds professional honors as a Senior Member of IEEE and a Fellow of BCS. His focus lies in building scalable, real-world AI solutions. Contact him at am287@buffalo.edu.