GraphKWS: Exploring Graph Neural Networks for Keyword Spotting

Eleonora Mesaglio, Marlon Joshua Helbing

***Abstract*—Graph Neural Networks (GNNs) are neural architectures designed to operate on graph-structured data, offering structural flexibility, dynamic information flow and parameter sharing. These properties make GNNs particularly appealing for resource-constrained environments, where achieving high accuracy with minimal computational overhead is essential. In this work, we investigate the use of GNNs for the keyword spotting (KWS) task, using the Google Speech Commands Dataset V2 as our benchmark. Our key innovation lies in the graph representation of audio: spectrograms are modeled as graphs where temporal frames serve as nodes, connected through cosine similarity-based adjacency matrices with dilated connectivity patterns. This design efficiently captures both short- and long-range temporal dependencies. Together with our graph construction strategy, we propose two additional key contributions: (1) graph convolutional layers for balanced node influence and attention mechanisms for improved noise robustness, and (2) reduced node representations that optimize the accuracy-efficiency trade-off. Our models achieve state-of-the-art performance. A low-footprint variant (18.4k parameters) matches the accuracy of current leading models while reducing multiply operations by a factor of 3. A medium-footprint model (151k) reaches 94.68% accuracy while using 23% fewer parameters than previous attention-based approaches. To our knowledge, we are the first to explore GNN architectures for keyword spotting, and our results demonstrate their potential as an efficient alternative to traditional approaches for speech-based interfaces.**

***Index Terms*—Keyword Spotting, Graph Neural Networks, Attention Mechanisms, Dilation**

I. Introduction

Keyword Spotting (KWS) is a speech analysis technology that enables the detection of specific target words within continuous audio streams. Unlike full speech recognition, KWS is designed to operate efficiently in real-time and on resource-constrained devices, allowing for applications such as virtual assistants on mobile phones (e.g., Siri) or voice-controlled devices in smart home environments (e.g., Alexa).

Thus, achieving a good trade-off between accuracy and model footprint is one of the central challenges in keyword spotting. Convolutional Neural Networks (CNNs) have emerged as the state-of-the-art for KWS and general audio classification tasks, thanks to architectural elements like pooling layers, residual connections, and parameter sharing that help manage this trade-off effectively. Notable works by Sainath and Parada [1] and Tang et al. [2] exemplify this approach, achieving competitive accuracy while maintaining lightweight architectures suitable for deployment.

Nevertheless, these models have a static input, limiting information to a fixed spatial relationship. In our study, we utilize Graph Neural Networks (GNNs), which exploit the natural versatility of graph architectures to dynamically capture inter-frame relationships.

In this paper, we show how audio can be embedded into a graph representation, where nodes represent temporal frames containing a set of features and edges represent either static neighborhood connections or dynamic cosine-similarity based links. We further incorporate dilated connectivity patterns that create skip connections between non-adjacent time frames, to capture longer time dependencies. We analyze various node-to-node message passing mechanisms, comparing simple unweighted aggregation, weighted edge-based methods and Graph Convolutional Network (GCN) layers. We investigate multiple graph-level aggregation strategies for word classification, including basic pooling operations and attention-weighted aggregation mechanisms.

In particular, we demonstrate:

- how modeling audio as a graph using a sparse, cosine-similarity-based adjacency matrix with dilation effectively captures both short-term and long-term dependencies, achieving high accuracy with few parameters and multiplies;
- how GNNs can be enhanced with GCN layers and attention mechanisms. Our GCN-based low-footprint model reduces multiplies by a factor of 3 and our attention-based medium-footprint model reduced parameters by 23% w.r.t. comparable CNN models, while maintaining the same or achieving a higher accuracy;
- how a reduced node representation can be used to flexibly trade-off accuracy for less multiplications.

Recent studies have demonstrated the effectiveness of GNNs in audio-related tasks such as speech emotion recognition (Kim et al., 2022, [3]) and environmental audio tagging (Singh et al., 2024, [4]). Yet, to our knowledge, their potential in keyword spotting has not been explored. We investigate GNN architectures specifically tailored for the KWS domain.

This paper is structured as follows. In Section II we review related work in keyword spotting. The overall processing pipeline is presented in Section III, while Section IV details the proposed signal processing and graph construction techniques. The learning framework and GNN architectures are described in Section V, and performance evaluation is carried out in Section VI. Finally, Section VII concludes the paper with a summary and directions for future work.

II. Related Work

CNNs have dominated recent work in audio processing due to their ability to learn hierarchical acoustic patterns from

spectrograms, treating them as 2D time-frequency images. Their computational efficiency through parameter sharing and local connectivity has made them particularly attractive for KWS applications.

Early keyword spotting systems, based on Hidden Markov Models (HMMs), suffered from limited representational capacity and high computational overhead. Chen et al. [5] initiated the transition to neural methods, by demonstrating that deep fully-connected networks outperformed traditional HMM-based systems, achieving a 45% relative improvement while maintaining compact footprints suitable for mobile deployment. However, fully-connected networks ignore the spatial structure of spectrograms, limiting their effectiveness for acoustic pattern recognition. Thus, Sainath and Parada [1] addressed this issue by introducing compact CNNs tailored for embedded KWS tasks. Their models leveraged frequency-domain pooling and efficient filter design to achieve higher accuracy with fewer parameters. Tang and Lin [2] advanced CNN-based KWS by incorporating residual learning and dilated convolutions, achieving 90% accuracy on the 12-word Google Speech Command Dataset V1 with only 20k parameters and $\sim$6M multiplies, establishing new benchmarks for low-footprint models. Yet, the input to CNN models is constrained to a fixed spatial relationship, restricting their ability to model inter-frame dependencies. Andrade et al. [6] introduced attention mechanisms to KWS and combined CNNs with recurrent layers. Their attention-based model achieved 94.1% accuracy on the 35-word Google Speech Command Dataset V2 with 202k parameters, demonstrating that dynamic feature weighting could improve performance.

Our work addresses the limitations of existing KWS approaches by introducing learnable graph structures that enable flexible connectivity between spectral-temporal features without architectural constraints. We demonstrate that GNNs achieve superior accuracy-efficiency trade-offs compared to existing approaches, outperforming both Tang et al.'s [2] compact CNN (20k parameters) and Andrade et al.'s [6] attention-based model (202k parameters) on the 35-word Google Speech Command Dataset V2 while requiring fewer computational resources.

III. Processing Pipeline

Our system processes raw audio data to extract features suitable for graph-based keyword spotting using Graph Neural Networks. The overall processing pipeline includes the following stages: audio preprocessing, spectrogram generation, data augmentation, feature extraction, graph construction, and model inference.

Audio Preprocessing: To ensure consistent input size, audio clips of varying lengths are either padded with zeros or trimmed to a fixed 1-second window. We also optionally apply background noise injection, using six types of ambient noise to enhance model robustness during training. A random time-shift of [-100 ms, 100 ms] can also be applied to the audio clips, as in [2].

Spectrogram Generation: Audio signals are split into overlapping frames (25ms with 10ms shift) and windowed using a Hamming window. These are then transformed into spectrograms, representing the time-frequency energy distributions of the signals.

Data Augmentation: We incorporate an optional data augmentation step using SpecAugment [7], a technique that masks random regions of the spectrogram along both the frequency and the time dimension.

Feature Extraction: The spectrograms are passed through filterbanks to warp frequency bins based on human auditory perception. We experiment with both Mel filterbanks and Gammatone filterbanks. After applying log-scaling, we perform a Discrete Cosine Transform (DCT) to obtain cepstral coefficients: either MFCCs (mel-based) or GFCCs (gammatone-based). Each time frame is then described using a 39-dimensional feature vector consisting of static coefficients, their first- and second-order derivatives (deltas), and corresponding energy terms.

Graph Construction: From these frame-wise features, we construct graph representations. Frames are treated as nodes, and adjacency matrices are computed using one of two approaches: a basic sliding window (unweighted) or a cosine similarity sliding window (weighted).

Model Inference: These graphs are then fed into GNN-based classifiers. We experiment with different GNN architectures to evaluate their performance on the keyword spotting task.

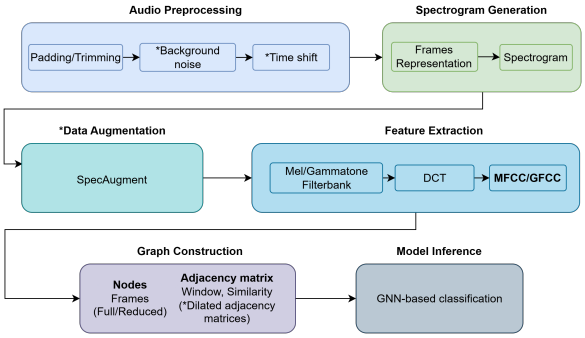


Fig. 1: *Processing Pipeline.*

Further implementation details, motivations, and the reasoning behind specific choices (e.g., filterbank types, feature dimensionality, or graph construction strategies) are provided in Sec. IV.

IV. Signals and Features

In the following, we discuss the technical details of the processing blocks introduced in Sec. III.

A. Dataset

The Google Speech Commands Dataset V2, [8], is a curated collection of approximately one-second audio recordings, specifically designed for training and evaluating keyword spotting systems. It comprises $105,829$ single-word utterances

across 35 unique commands, spoken by a diverse set of speakers, providing essential variability for building robust voice-controlled applications. Background noise recordings are also included to support data augmentation and enhance model performance under realistic acoustic conditions. We retain the official train−validation−test split (approximately $80\%-10\%-10\%$) to ensure consistency and comparability with prior work.

B. Signal Preprocessing

Padding/Trimming: Neural networks require fixed-size inputs. Since the majority of recordings are one second long at a 16kHz sampling rate, we standardize all audio clips to exactly 16000 samples. Shorter recordings are zero-padded at the end, while longer recordings are truncated to maintain consistent input dimensions across the dataset.

Background noise: The dataset includes six types of background noise (e.g., white noise, running tap water). While our implementation supports selecting a specific noise type, we default to randomly sampling one of the six types and adding it to each training sample with a probability of 0.8, in line with [2], to increase robustness and prevent the model from learning fixed noise patterns. As noise recordings are longer than one second, we randomly extract one-second segments. The noise is added at a randomly sampled signal-to-noise ratio (SNR) between $[-5\,\mathrm{dB}, +10\,\mathrm{dB}]$, following the approach in [1].

Data Augmentation: Data Augmentation techniques are essential for training robust neural networks, especially when working in limited or noisy conditions, as they artificially expand the training dataset with realistic variations. In this work, we use SpecAugment [7], a technique that masks parts of the spectrogram (definitions and properties in Sec. IV-C) along both the time and frequency dimensions. By hiding specific spectral and temporal details, the model is encouraged to focus on broader patterns, which improves generalization and performance. Specifically, we apply:

- frequency masking: randomly masking 3 to 15 consecutive frequency bins;
- time masking: randomly masking 10 to 30 consecutive frames.

The number of masked elements is sampled uniformly within these ranges for each augmentation instance. This augmentation is applied to each training sample with a probability of 0.8.

C. Feature Extraction

A spectrogram is a visual representation of how the frequency content of an audio signal evolves over time, capturing both temporal and spectral information needed for speech pattern recognition. It is computed by applying the Short-Time Fourier Transform (STFT) to overlapping frames of the signal. In our setup, the waveform is divided into frames of 25 ms with a 10 ms frame shift, resulting in a 98×201 spectrogram (time frames $\times$ frequency bins).

To better align the representation with human auditory perception and to reduce dimensionality, we apply filterbanks to the raw spectrogram. Specifically, we experiment with mel-scale and gammatone filterbanks to determine the most effective transformation.

1) Mel-Filterbanks: Mel-filterbanks consist of a set of triangular filters spaced according to the mel scale, which reflects the human ear's greater sensitivity to lower-frequency differences.

2) Gammatone Filterbanks: Inspired by Power-Normalized Cepstral Coefficients (PNCCs) [9], we also experiment with gammatone filterbanks. These filters, characterized by an impulse response that is the product of a gamma distribution and a sinusoidal tone, are spaced according to the Equivalent Rectangular Bandwidth (ERB) scale. Since gammatone filters more closely mimic the frequency selectivity of the human auditory system, we explore their potential for keyword spotting.

In our implementation, we apply a bank of 26 chosen filters to the spectrogram. Afterwards, we take the logarithm of the filter outputs and apply the Discrete Cosine Transform (DCT) to decorrelate the features and retain only the most significant terms. We then compute their delta and delta-delta features, along with the corresponding energy terms, yielding a final 39-dimensional vector. This representation is referred to as MFCC when using mel filterbanks, or GFCC with gammatone filterbanks.

D. Graph representation

Audio sequences are represented as undirected graphs where temporal frames serve as nodes, with each node containing 39-dimensional MFCC/GFCC feature vectors. We adopt an undirected graph formulation, which enables bidirectional information flow. This allows each frame to both influence and be influenced by its temporal neighbors, facilitating the propagation of temporal patterns in both directions. As a result, each frame can access contextual information from both past and future acoustic events within the sequence. Since graphs are inherently unordered structures, preserving the temporal information is non-trivial. To address this, we encode temporal relationships in the adjacency matrix.

1) Adjacency Matrix: We construct the adjacency matrix using one of 2 sliding window approaches, where each frame is connected to its temporal neighbors within a specified window size. Specifically, frame i is connected to frames $[i - Window\ Size, i + Window\ Size]$, resulting in up to $2\times$*Window Size* per frame. Boundary frames naturally have fewer connections due to sequence limits.

- **Mode simple window**: The adjacency matrix is unweighted, with binary connections (1 if frames are neighbors, 0 otherwise) based solely on temporal proximity.
- **Mode cosine window**: Creates a weighted adjacency matrix where edge weights are computed using cosine similarity between the MFCCs/GFCCs of neighboring frames. The adjacency matrix can be thresholded using a *Cosine Window Threshold* parameter, to filter out

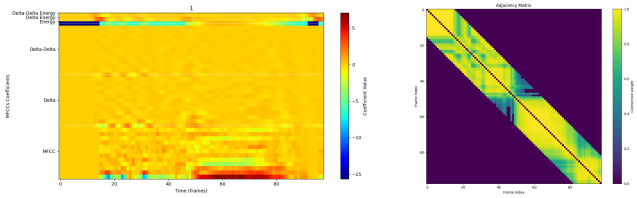


Fig. 2: *MFCCs and Cosine Window Adjacency Matrix*. Both plots consistently highlight the salient speech segment, approximately between frames 45 and 80, distinguishing it from silence and noise.

weak connections. As noted by Kim and Kim [3], cosine similarity-based connections help suppress irrelevant edges between voice and non-voice regions (Fig. 2).

2) Multi-Scale Connectivity through Dilation: Inspired by the use of dilated convolutions in [2], we extend this idea to the graph domain by introducing dilated adjacency matrices (see Fig. 3). These matrices allow each graph to encode multiple connectivity patterns, with each matrix capturing a distinct temporal scale (e.g., first degree: consecutive frames, second degree: skip one frame, and so on). A dilated adjacency matrix A_d with dilation rate $d > 1$ is obtained as:

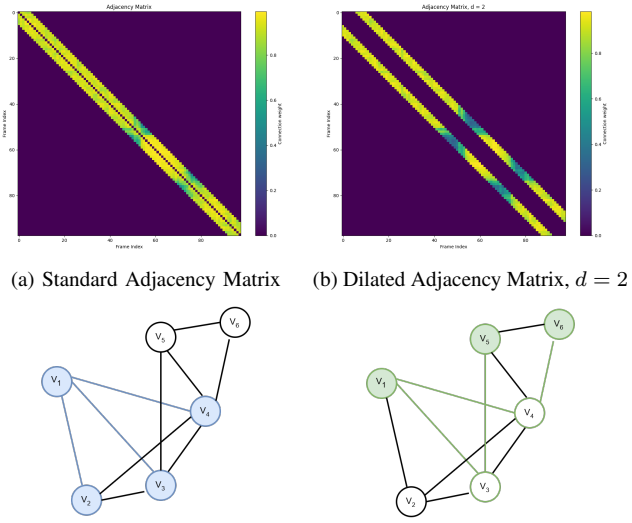
$$A_{d;i,j} = \begin{cases} w_{ij} & \text{if } \tilde{A}_{d;i,j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\tilde{A}_d = A^d - [A^{(d-1)} + \cdots + A + (d-1) \cdot I]$ isolates node pairs connected by paths of length exactly d, and w_{ij} is defined by the selected connection mode. In practice, we construct a sequence of dilated adjacency matrices using even dilation rates, skipping intermediate connections to promote faster access to long-range temporal dependencies.

During training, we assign these matrices to successive message passing layers, ensuring that each layer processes a distinct temporal scale. We refer to these layers as dilation layers. Overall, this approach brings several advantages:

1) **Efficient long-range dependency modeling**: By connecting frames that are multiple steps apart, dilated adjacency matrices enable small local neighborhoods to yield large effective receptive fields through layer composition.
2) **Temporal diversity through rotation**: Using different dilation patterns across different layers prevents repeated aggregation over the same local neighborhoods. This promotes the learning of diverse temporal features and reduces redundancy in both representations and message-passing paths.

Reduced Nodes Representation: We experiment with trading off some temporal detail for computational efficiency by introducing an optional node pooling step. This process groups k consecutive frames and computes the average value for each MFCC or GFCC feature dimension across the grouped frames. This reduces the number of nodes (and consequently the number of multiplies required), while preserving the original feature dimensionality per frame group.



(a) Standard Adjacency Matrix (b) Dilated Adjacency Matrix, $d = 2$

(c) Direct Connections (d) Dilated Connections, $d = 2$

Fig. 3: *Connectivity Patterns*. Standard adjacency matrix (a) shows local connectivity, while dilated adjacency matrix (b) with $d = 2$ creates longer-range connections. Bottom figures illustrate corresponding graph structures: with respect to v_1, we show (c) direct nearest-neighbor connections vs (d) dilated connections with $d = 2$ that skip intermediate nodes.

The obtained graph representations serve as input for our GNN models.

V. Learning Framework

Before presenting our specific model variants, we first describe the core architectural components and GNN operations shared across all our approaches.

A. Initial Node Encoding

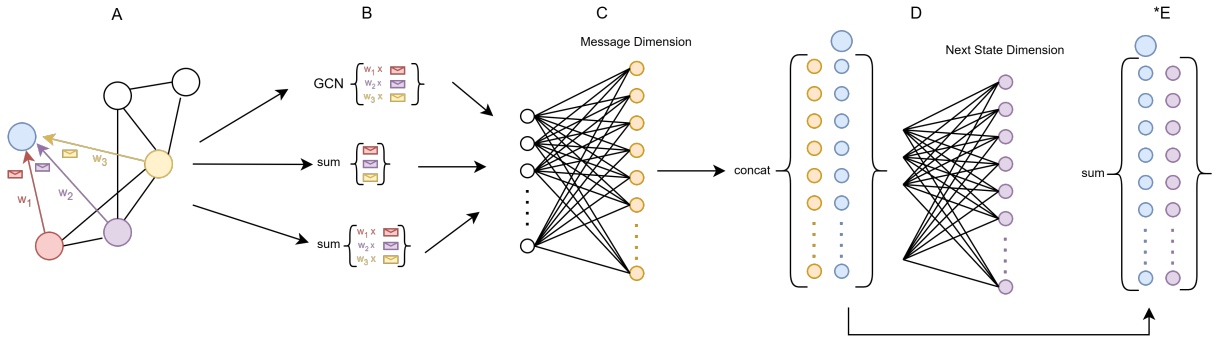
Rather than using raw MFCCs or GFCCs as static node features, we make these features learnable through one of the following encoding strategies:

- *Normal*: Input features passed through a single dense layer, resulting in a vector of dimension *Node Features*.
- *Multi-branch encoding*: The first 12 coefficients, first-order delta coefficients, second-order delta coefficients and the 3 energies are each passed through separate dense layers of sizes 24, 24, 24 and 8, respectively. The resulting embeddings are then concatenated and passed through an additional dense layer, resulting in a vector of dimension *Node Features*. This design allows each type of coefficient to be processed independently, encouraging the model to learn richer and more specialized representations for different temporal dynamics, at the cost of only a modest increase in parameters.

B. Message Passing

GNNs operate through message passing, an iterative mechanism that enables information exchange between the different components of a graph (nodes, edges, context node). In this

Fig. 4: *Message Passing Scheme.* **A** - Messages are collected from neighboring nodes to the receiver node (blue). **B** - We propose 3 different ways for Message Aggregation: simple summation, edge-weighted summation, and weighted GCN layers. **C** - Aggregated messages are passed through a dense layer to map them to size *Message Dimension*. **D** - Message is concatenated with current receiver node state and passed through a dense layer to map it to a vector of size *Next State Dimension*, representing the new state of the receiving node. ***E** - An optional residual connection can sum the original and updated state.



work, we focus on node-to-node communication (see Fig. 4). This process consists of three fundamental steps:

1) Message Gathering: Each node collects feature embeddings from all its neighboring nodes, according to the graph's connectivity structure.
2) Message Aggregation: The collected neighbor embeddings are combined through an aggregation function and transformed through a dense layer to a fixed vector with size *Message Dimension*. Note that this step excludes the receiver node's current state.
3) Node State Update: The node's representation is updated by combining its current state with the aggregated neighborhood information and transforming the result through another dense layer to a fixed vector with size *Next State Dimension*. Optionally, a residual connection can be applied by adding the original node representation to the dense layer output.

All dense layers share the same architecture: a fully connected layer with 20% dropout, layer normalization, and ReLU activation. Importantly, GNNs achieve parameter efficiency through weight sharing: the same transformation matrices are applied to all nodes regardless of their position in the graph, similar to how CNNs share convolutional kernels across spatial locations.

Through multiple message passing iterations, each node progressively aggregates information from increasingly distant neighbors: after k layers, a node incorporates features from all nodes within k hops in the graph. This allows GNNs to capture long-range dependencies, based on the depth of the message-passing process. For all tests, we set the number of message passing layers to 5.

In our model, we will extend this mechanism by incorporating on one hand edge weights, that modulate the influence of messages, and on the other hand dilation, as described in Sec. IV-D2. When using dilation layers, we match their number to the number of message passing layers (5), ensuring that each layer gathers information from a distinct set of neighboring nodes within a single forward pass. Furthermore, we propose the usage of a GCN layer for the Message Aggregation Step (see Sec. V-D2).

C. Context Node & Classification

Another feature that we incorporate is the context (or master) node, a special hyper-node that connects to all other nodes in the graph. Its role is to capture and aggregate global information from the entire graph. Specifically, information from all frame-level nodes is aggregated and passed to the context node, either by averaging across feature dimensions or via an attention mechanism using a Graph Attention Network (GATv2) layer (Fig. 5) (described in detail in Sec. V-D3). This allows the model to construct a global representation of the sequence. Additionally, the context node can be used as learnable feature that is updated by concatenating its current representation with the pooled messages and passing it through a dense layer, resulting in the new context node state.

Many prediction tasks can be addressed using graphs, but in this work we focus on graph-level prediction - specifically, predicting the utterance-class that the graph represents as a whole. To achieve this, we apply a final dense layer with linear activation at the end of the GNN, which maps the learned graph representation, captured in the context node, to the 35 target classes.

D. Models

Our main architecture can be described as presented in Fig. 6. The following sections describe the models employed in our experiments.

1) Base GNN model: We begin with a basic model that implements our GNN pipeline. This baseline serves to evaluate whether our theoretical motivation for applying GNNs to keyword spotting translates into effective practical results. We

Fig. 5: *Context Node Aggregation & Classification*. We propose three pooling strategies for classification: **Purple** - Direct averaging of node features followed by dense layer mapping to 35 classes. **Yellow** - Pooled messages concatenated with context node state, updated through a dense layer, then classified. **Blue** - GATv2 attention mechanism dynamically weights node importance across multiple heads, concatenates outputs with context state for updated representation before classification.

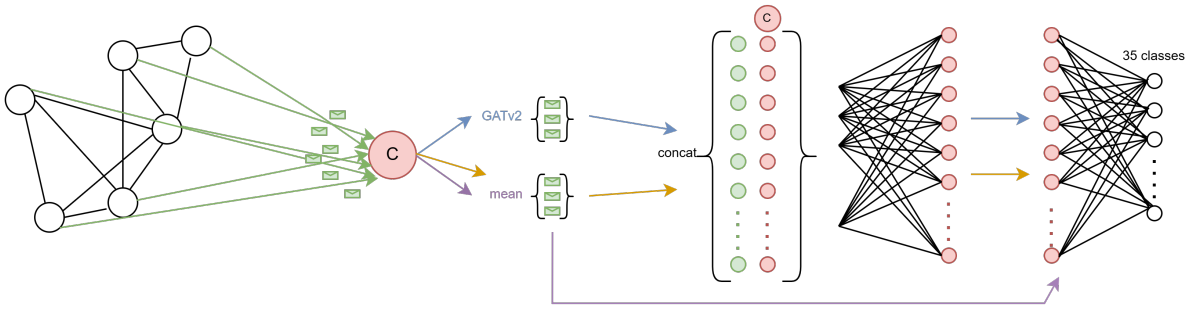


Fig. 6: *GNN Model Pipeline.*

conduct comprehensive ablation studies to analyze how various design choices (from input representations to architectural components) affect model performance. To start off, we use MFCCs and the *Normal* mode initial node encoding. We use the *Simple Window* adjacency matrix with a window size of 25 and no dilation layers. *SpecAugment*, *TimeShift*, and the residual connection during the Node State Update are not used. Furthermore, the context node is not learnt for final classification, i.e. we use simple mean pooling (color purple in Fig. 5). We set *Node Features*, *Message Dimension*, and *Next State Dimension* to 64. For brevity, we will refer to these three collectively as the model dimension when they share the same value.

2) The GCN model – Graph Convolutional Network: In this variant, we replace the basic message passing mechanism with the graph convolutional operator proposed by Kipf and Welling [10], which performs localized spectral convolutions combined with symmetric normalization. This normalization is particularly important in graphs with high variation in node degrees, as it prevents nodes with many neighbors from disproportionately influencing the feature aggregation process. The GCN layer operates through a dual normalization scheme: first, sender nodes apply degree-based normalization to their outgoing messages, which are then weighted by our edge weights and aggregated (summed) at the receiver node. Subsequently, the aggregated messages undergo receiver-side normalization based on the receiving node's degree. (Fig. 7). As a result, all nodes maintain feature magnitudes on a comparable scale, and connections with higher acoustic similarity remain emphasized. We adopt the optimal hyperparameter settings from our base GNN experiments while reducing the model dimension from 64 to 32 to reduce computational costs for a low-footprint model. The pipeline can be seen in Fig. 8.

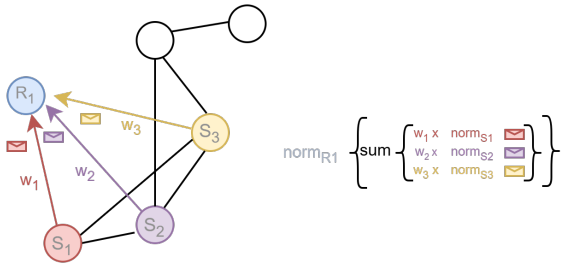


Fig. 7: *Graph Convolutional Operator*. We use R to indicate the receiver node and S to indicate sender nodes.

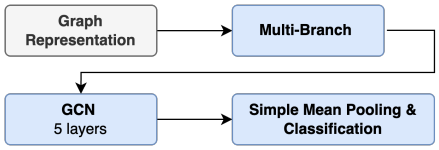


Fig. 8: *GCN Model Pipeline.* Graph Representation as found in Base GNN experiments.

3) The GAT-GCN model – Adding attention weighted Context Node Aggregation: While GCN normalization effectively addresses degree-based bias, a fundamental problem remains in our message passing framework: problematic signals such as noise clusters or silent regions may exhibit high internal cosine similarity, thus playing a disproportionate role in the final classification and transmitting noisy or silent information to the context node. To address this limitation, we incorporate an attention mechanism inspired by GATv2 [11]. After GCN message passing, each frame node sends information to the context node through an attention-weighted mechanism, where attention weights determine the contribution of each frame to

the final representation (Fig. 9). The outputs from all attention heads are concatenated and combined with the context node state. This combined representation is then processed through a dense layer to produce the final context node representation, which is subsequently passed through a classification layer for the final prediction (Fig. 5). In our implementation, we use 5 attention heads at 128 features per head as well as a model dimension of 64. Other hyperparameter settings are adopted from the best base GNN. The pipeline can be seen in Fig 10.

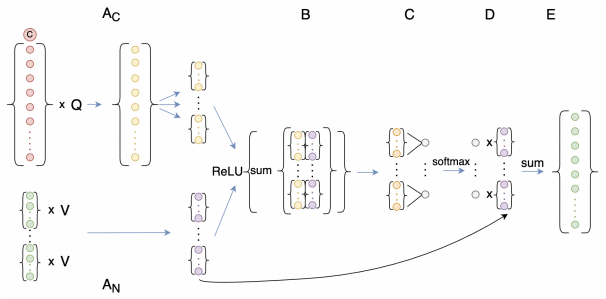


Fig. 9: *GATv2 Mechanism (single head).* $\mathbf{A}_C$ - The context node state is multiplied with the learnable query matrix Q and subsequently broadcasted to each node in the graph, resulting in a matrix of size $num\_nodes \times per\_head\_channels$. $\mathbf{A}_N$ - Each node state is multiplied with the learnable value matrix V. **B** - Query and value vectors of each node are summed along the feature dimension and passed through an activation function (ReLU). **C** - Using a dense layer, we then map the result to a single value for each node and pass it through softmax to create *attention weights* for each node w.r.t. the context node. **D** - Each of the node representations obtained in A_N is then multiplied with the attention weight, resulting in a matrix of size $num\_nodes \times per\_head\_channels$. **E** - Finally, we sum along their feature dimension to create a vector of size $1 \times per\_head\_channels$.

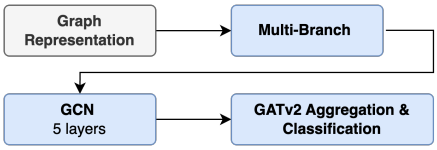


Fig. 10: *GAT-GCN Model Pipeline.* Graph Representation as found in Base GNN experiments.

E. Training

For model training, we employ categorical cross-entropy loss with the Adam optimizer (initial learning rate set to 0.001). We implement early stopping with a patience of 5 epochs based on validation loss, and apply learning rate decay (0.5 factor) when validation loss stagnates for 2 consecutive epochs. Training is conducted primarily under noisy conditions, as detailed in Sec. IV-B. However, to enable direct comparison with Andrade et al. [6], we also train models under clean conditions matching their experimental setup. To ensure reproducible results, we initialize models with fixed random seeds and train each configuration across three different seeds. All models are trained for a maximum of 30 epochs on a T4 GPU with a batch size of 64.

F. Metrics

We evaluate model performance using mean accuracy and standard deviation across the three random seeds. This choice enables direct comparison with existing keyword spotting literature. We also report computational efficiency metrics including parameter count, multiply operations, inference time (averaged per sample, in milliseconds), and model memory requirement.

VI. Results

A. Ablation studies on Base GNN model

We present here our basic GNN model, with starting architecture and parameters as detailed in Sec. V-D1. We achieve an accuracy of $85.57\% \pm 0.95$ at 68k parameters, 6.474M multiplies, an inference time of approximately 1.41 ms and memory cost of 264 kb. The evolution of models can be seen in Fig. 11.

1) Mitigating redundant local connectivity with dilation layers: We replace our fixed 25 window size architecture with dilated graph convolution using 5 progressively dilated adjacency matrices (A, A_2, A_4, A_6, A_8) and reduced neighborhood size of 5. This change maintains the receptive field while introducing connections to distant nodes, addressing the limitation of static neighborhood structures to force nodes to receive information repeatedly through identical communication pathways. The new structure improves accuracy from $85.57\% \pm 0.95$ to $90.17\% \pm 0.21$, though it increases the inference time to circa 1.89 ms due to processing multiple adjacency matrices. We adopt this dilated architecture for all subsequent experiments.

2) Using weighted message sending based on cosine similarity: Switching to cosine-weighted adjacency at a threshold of 0.3 yields modest accuracy gains ($90.21\% \pm 0.11$) at the cost of some computational overhead (6.600M multiplies, 2.12 ms inference time). We attribute this modest gain to the narrow window: most nodes exhibit high similarity within their neighborhood, with minimal variation. In scenarios with greater neighborhood diversity, the benefits of cosine weighting become more evident. In fact, testing with a larger window size of 15, 3 dilation layers and 3 message passing layers results in a more substantial improvement ($87.25\% \pm 0.21$ vs. $88.81\% \pm 0.38$), confirming the effectiveness of cosine weighting. We adopt this scheme, with a window size of 5, for subsequent experiments.

3) Residual connections for more efficient long-term dependency learning: Adding residual connections to our message passing framework improves accuracy to $90.48\% \pm 0.54$, while decreasing inference time to 2.01 ms. We attribute this to the model's ability to adaptively balance local and distant context, which is beneficial for speech data where optimal receptive fields vary with phonetic content, speaking rate, and

word length. Therefore, we include residual connections in all subsequent experiments.

4) Better acoustic representation via separate MFCC component processing: Despite a modest increase in parameters (72k), computation (6.968M multiplies), inference time (2.06 ms), and memory (279 kb), we gain over 0.5% in accuracy ($91.08\% \pm 0.44$) using our *Multi-Branch Node Encoding*, which justifies its use in all subsequent experiments.

5) Improved generalization via Time Shift and SpecAugment: Time Shift achieves $91.13\% \pm 0.22$ accuracy, further adding SpecAugment reaches $91.87\% \pm 0.23$. However, SpecAugment also increases training time by generating additional samples, whereas Time Shift is a lightweight, in-place operation. For all subsequent experiments, we employ Time Shift and designate the model with our current hyperparameter configuration as Best GNN, with a model dimension of 64. We will additionally report the effect of SpecAugment.

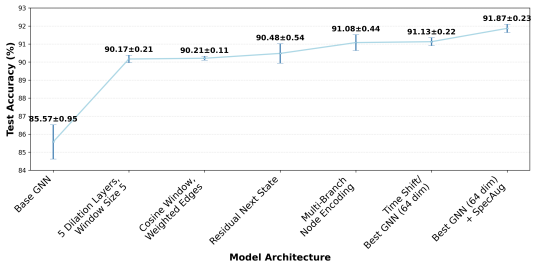


Fig. 11: *Evolution from Base GNN to Best GNN.*

6) Other parameter variations lead to no benefits: GFCCs yielded comparable or slightly worse performance than MFCCs despite substantial noise training, suggesting MFCCs provide sufficient noise robustness while preserving discriminative spectral details. A learnable context node increased parameter count and computational cost without performance gains. Window size experiments for cosine similarity consistently showed optimal performance at size 5 (Fig. 12). Finally, higher cosine window thresholds decreased accuracy, indicating that weak connections between dissimilar regions (e.g., voice and non-voice) still contribute valuable information and should be preserved.

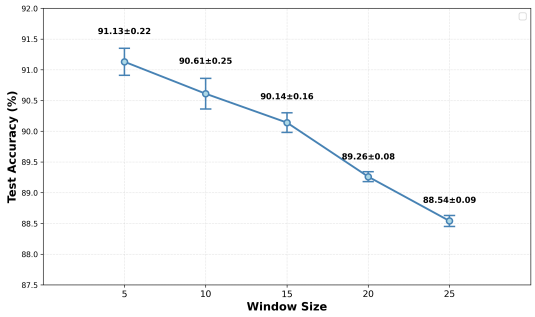


Fig. 12: *Effects of Window Size on Accuracy.* Higher window sizes lead to lower accuracy. The results in the figure are based on Best GNN (dim 64) hyperparameter settings.

B. Low-Footprint Models

We implemented and trained Tang's res8-narrow on the 35-word task, comparing its performance against two GNN variants: our developed architecture and our GCN-based version. By reducing the model dimension from 64 to 32, our models have around 21k parameters (slightly higher than Tang's 20.8k), while requiring only roughly 2M multiplies – approximately one-third of res8-narrow's computational cost. For fair comparison, we trained all models without SpecAugment, though we also report results with SpecAugment.

1) Low-footprint GNN: Our compact 21.2k parameter GNN model achieves $86.87\% \pm 0.22$ accuracy, outperforming res8-narrow and demonstrating the scalability of our GNN architecture (Table 1). Even with reduced features (32), the model retains strong performance.

2) Low-footprint GCN: Using our GCN-variant, we achieve a slight performance improvement while both parameters and multiplies decrease slightly (Table 1). As discussed in Sec. V-D2, we attribute this gain to degree normalization.

Reducing parameters: While our models outperform res8-narrow in both multiplies and accuracy, they have slightly more parameters (21.2k/20.9k vs 20.8k). To demonstrate parameter efficiency, we use the GCN-based model and replace the multi-branch initial node encoding with our normal encoding, reducing the parameter count to 18.4k. The model achieves $85.83\% \pm 0.70$, remaining competitive while using fewer parameters (Table 1). Utilizing SpecAugment, we outperform in accuracy ($87.81\% \pm 0.25$).

C. GAT-GCN Medium-Footprint Model

An attention mechanism increases both parameter count and computational cost, therefore this variant serves as our 'medium-footprint model', which we compare against the CRNN-attention architecture from Andrade [6]. Our model outperforms Andrade's by roughly 0.7% accuracy while using 47k fewer parameters. Adding SpecAugment further improves accuracy by approximately 0.1% (see Table 2).

D. Flexibility of Multiplies using a Reduced Node Representation

To flexibly reduce computational multiplies, we introduced the possibility to reduce the amount of frames using a mean pooling operation. Halving the frames led to an accuracy drop of about 0.5-1%, while quartering resulted in a 3-4% decrease (see Fig. 13), with inference time reduced by 0.2-0.3 and 0.4-0.5 ms, respectively. Since only the number of nodes in the graph was modified, parameter counts and model memory requirements remained identical to those reported in our previous results.

VII. Concluding Remarks

In this work, we introduced GNNs to the keyword spotting domain for the first time, demonstrating their effectiveness on the Google Speech Command Dataset V2. We proposed a novel graph-based representation of audio spectrograms using sparse, cosine similarity-based adjacency matrices with dilated

TABLE 1: *Best GNN vs GCN vs res8-narrow*

Model	**Test Acc**	**#P**	**#Mu**	**Time (ms)**	**#Me**
Best GNN (dim 32)	86.87 ± 0.22	21.2k	2.034M	2.03 ± 0.04	83kb
Best GNN (dim 32) + SpecAug	87.56 ± 0.38	21.2k	2.034M	2.02 ± 0.06	83kb
GCN	87.12 ± 0.27	20.9k	1.989M	2.18 ± 0.05	82kb
GCN + SpecAug	87.92 ± 0.20	20.9k	1.989M	2.20 ± 0.04	82kb
GCN + Normal Node Encoding	85.83 ± 0.70	18.4k	1.750M	2.09 ± 0.01	72kb
GCN + SpecAug + Normal Node Encoding	**87.81 ± 0.25**	**18.4k**	**1.750M**	**2.09 ± 0.02**	**72kb**
res8-narrow	85.93 ± 0.85	20.8k	5.65M	1.34 ± 0.42	82kb

TABLE 2: *GAT-GCN vs. CRNN w/ attention*

Model	**Test Acc**	**#P**	**#Mu**	**Time (ms)**	**#Me**
GAT-GCN	94.58 ± 0.11	155k	11.102M	2.30 ± 0.03	607kb
GAT-GCN + SpecAug	**94.68 ± 0.10**	**155k**	**11.102M**	**2.30 ± 0.04**	**607kb**
CRNN w/ attention	93.9	202k	/	/	/

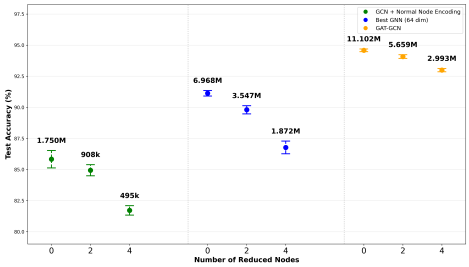


Fig. 13: *Effects of Reduced Node Representations on Accuracy.* In the figure we can see the performances of 3 of our models, trained without SpecAugment, with respect to different reduced node representations.

connectivity, and incorporated weighted message passing, graph convolution with degree normalization and attention mechanisms for improved context aggregation. Our results show that GNNs provide a compelling alternative to traditional CNNs for resource-constrained applications, offering scalability across model sizes – from low-footprint models with fewer than 20k parameters to medium-scale architectures under 200k – while outperforming CNN-based baselines. Sparse connectivity and dilated layers keep the computational cost low, further reduced by compact node representations.

Throughout the project, we gained hands-on experience with graph-based neural networks and developed a deeper understanding of model evaluation beyond accuracy, recognizing the importance of computational cost in real-world deployment. For the first time, we also benchmarked our models against state-of-the-art baselines, which presented challenges such as re-implementing a CNN originally trained on a smaller dataset. One of the main difficulties we faced was the limited documentation available for modeling GNNs in TensorFlow. Additionally, designing an appropriate graph structure required careful consideration: we needed a sparse representation to maintain a low footprint while still preserving relevant information. We believe our proposed solution provides a solid foundation, though future refinements could further improve its performance and efficiency.

Looking ahead, several promising directions can enhance our approach. First, our best models can be fine-tuned using hyperparameter optimization techniques. Secondly, future work may explore alternative graph construction strategies, such as asymmetric adjacency matrices, that prioritize future over past frames, or directed graph structures, that only pass the message in the declared direction. Furthermore, it could be interesting to investigate whether the cosine similarity patterns evident in Figure 2 could be exploited for voice activity detection.

REFERENCES

[1] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Interspeech*, (Dresden, Germany), Sept. 2015.

[2] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Calgary, Canada), pp. 5484–5488, Apr. 2018.

[3] J. Kim and J. Kim, "Representation learning with graph neural networks for speech emotion recognition." arXiv preprint arXiv:2208.09830, Aug. 2022.

[4] S. Singh, C. J. Steinmetz, E. Benetos, H. Phan, and D. Stowell, "Atgnn: Audio tagging graph neural network," *IEEE Signal Processing Letters*, vol. 31, pp. 825–829, Jan. 2024.

[5] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Florence, Italy), pp. 4087–4091, May 2014.

[6] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition." arXiv preprint arXiv:1808.08929, Aug. 2018.

[7] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," in *Interspeech*, (Graz, Austria), Sept. 2019.

[8] P. Warden, "Speech commands: A dataset for limited vocabulary speech recognition." arXiv preprint arXiv:1804.03209, Apr. 2018.

[9] C. Kim and R. M. Stern, "Power-normalized cepstral coefficients (pncc) for robust speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, pp. 1315–1329, Mar. 2016.

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907, Sept. 2017.

[11] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?." arXiv preprint arXiv:2105.14491, May 2022.