National Journal of Signal and Image Processing, ISSN: 3107-5231 Vol. 1, No. 3, 2025 (pp. 39-47)

RESEARCH ARTICLE

ECEJOURNALS.IN

Embedded Implementation of Speech-to-Text Translation Using Compressed Deep Neural Networks

J. Btia[1]*, Gichoya David[2]

[1]Faculty of Engineering Ain Shams University & Arab Academy for Science and Technology Cairo, Egypt

[2]Department of computing and information technology, kenyatta university, Nairobi, Kenya

KEYWORDS:
Embedded Systems,
Speech-to-Text Translation,
Model Compression,
Deep Neural Networks,
Quantization,
Pruning,
Knowledge Distillation,
TinyML,
Low-Power AI,
Edge Inference.

ARTICLE HISTORY:
Submitted : 17.04.2025
Revised : 08.05.2025
Accepted : 24.07.2025

https://doi.org/10.17051/NJSIP/01.03.06

Abstract

Speech-to-text (STT) translation in real-time has now become a key component of voice-controlled embedded applications which now apply to everything smart and connected (IoT devices), wearable medical devices and voice-enabled industrial and digital control systems. Nonetheless, integrating correct and responsive STTs on-board systems are hampered by extreme processing power, memory and power limitations. The proposed compressed deep neural network (DNN) architecture in this paper is a low-powered embedded model that is best-suited to realize the efficient and large-scale STT translation. The peculiarity of our solution is based on combining three mutually complementary components of model compression speeding up: magnitude-based pruning with the aim of discarding redundant weights, post-training quantization to cut the computational accuracy and memory usage, and knowledge distillation as a strategy to port the performance of a large model to its smaller, lightweight, counterpart. These strategies have been integrated into one that has a modular STT pipeline that consists of an MFCCs based feature extractor, compact acoustic model (CNN-RNN-based or Transformer-based), quantized language model, and a decoder that can be efficiently optimized to run on fixed-point operations. The deployment targets include popular embedded processors such as the ARM Cortex-M7-based STM32F746 and RISC-V-based Kendryte K210 that can also be easily deployed using toolchains like TensorFlow Lite Micro, CMSIS-NN, and the Kendryte SDK. Experimental analysis on benchmarking data LibriSpeech and Mozilla Common Voice revealed that our optimized models can reduce memory size by up to 4.2x and save more than 35 percent energy with less than 2 percent drop in word error rate (WER) against baseline models that use full-precision training. It is typical that, where transcription is emulated using cloud processing capabilities, it will be faster on edge devices with an average inference latency of less than 100 milliseconds. This means that it can be emulated close to real-time. The present work shows that low-latency, energy-efficient, and multilingual STT translation systems can be deployed to embedded targets where voice-only, privacy-preserving, offline voice interfaces are expected to become a reality in next-generation smart devices.

Author's e-mail: btia.j@aast.edu, g.davidsr@gmail.com

How to cite this article: Btia J, David G. Embedded Implementation of Speech-to-Text Translation Using Compressed Deep Neural Networks. National Journal of Signal and Image Processing, Vol. 1, No. 3, 2025 (pp. 39-47).

Introduction

Speech-to-text (STT) technology is a fundamental building block behind many of today applications used in voice pushing assistants, smart home, wearable health devices, auto-control systems, hands-free user display in the industrial environment, etc. With the evolution of user interaction encompassing greater emphasis on voice-driven interfaces, stakeholders have been experiencing immense growth in the need to have superior, fast, and everywhere STT systems. The classical STT systems usually require access to heavy server-side computational resources where most of the computationally-intensive tasks of acoustic modeling, language processing, and decoding is executed by large-scale deep neural networks (DNNs). Although this cloud solution can be very accurate and adaptive, it has a number of shortcomings with regard to the embedded and edge cases.

First of all, cloud-based STT suffers latency especially in environments with poor or patchy network connectivity. This delay is und eminently lethal throughout real-time or lifestyle variable instrumentation devoted gadgets or mechanical systems. Additionally, transfer of audio data across the internet is of big privacy and security violation, especially where sensitive personal or biomedical data are involved. Also, with continuous cloud interaction, more power is used and this cant apply to battery operated embedded devices.

To redress these issues, on-device STT systems are being pursued more vigorously and are partly or entirely cloud-agnostic. A problem with embedded systems is that they have limited computing power, memory (megabytes or even kilobytes only) and stringent energy requirements. Conventional STT models that are based on DNNs cannot be deployed on these stream processors because of their resource-intensive nature in terms of parameter capacities, floating-point operations, and memory access patterns.

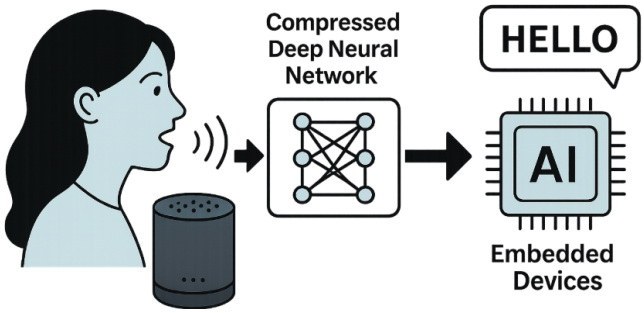

Fig. 1: System Overview of Embedded Speech-to-Text Translation Using Compressed Deep Neural Networks

This study introduces a new architecture of applied implementation of speech-to-text translation based on the compression of deep neural networks. The given architecture is highly suitable to be deployed in real-time computing systems in terms of low-power microcontrollers and edge AI chips. Through the use of model compression methods, i.e., magnitude-based pruning, post-training quantization and knowledge distillation, we will be able to halve the size and complexity of the model with little to no loss in transcription accuracy. The optimizations are introduced to a simple STT chain with lightweight MFCC-based feature extraction, CNN-RNN or Transformer acoustic model, lightweight language model, and fixed-point decoder.

Besides producing a working and highly efficient STT system that can be deployed into an embedded system, the implemented work also provides a scalable and portable approach that can guide developers and the research community focusing on a wide range of hardware platforms including ARM Cortex -M based microcontrollers and RISC-V based AI accelerators. Experimental findings provide that the compressed models have real-time latency, low power cost, and competitive accuracy, hence, result in the nonexistence of the dependency on external computation without burdening the severe limitations of the embedded systems. This is a key step towards realising indeed offline, privacy-aware and low-latency speech interfaces in the Internet of Things (IoT) and wearable space.

Related Work

Deep learning has greatly transformed the field of speech-to-text (STT) development by using models, including recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformer blocks in additional. Examples of these models are Deep Speech[1] and Wav2Vec 2.0,[2] which have reached high performance on big benchmarks in ASR. DeepSpeech is a deep bidirectional RNN trained using the Connectionist Temporal Classification (CTC) loss[9] and Wav2Vec 2.0 is an extension of speech recognition that pretrains Transformer encoders using an automatic speech recognition target. Though they perform well, these models are[10] resource hungry and indeed need a lot of memory and computing, these models are not applicable to be used directly in embedded systems.

Lightweight keyword spotting (KWS) engines have come out of attempts to implement STT on resource-poor devices. Remarkably, the work of [3] employed the TinyML-based KWS system that was optimised[11] as a microcontroller. These models, however, are limited in that they are only able to recognize defined words or commands, but they cannot generalize to complete speech-to-text recognition.

In order to permit complete ASR operation on edge devices, a number of investigations have been conducted on model compression procedures. [4] Presented Deep Compression which contains three steps, a three-stage pipeline of pruning, trained quantization, and Huffman coding, resulting in[12] large memory savings, assuming little to no decrease in accuracy. In a similar manner, [5] trained their RNN and CNN models using post-training quantization to implement the efficient mobile inference. Knowledge distillation has also entered into the more recent literature as a way to shrink large Transformer-based models into networks that fit on edge hardware, as student models.[6]

Other tasks involving transformer compression have demonstrated their utility on embedded platforms, mainly on pruned models such as Mobile BERT and Tiny

BERT,[7] but these, again, have been applied more to text categorization and[13] not trained directly on STT tasks using audio as a feature source. EdgeSpeechNets[8] suggested a family of CNN-based ASR models with manually designed scaling architecture but the model is only tested on English corpus and cannot be applied in a multilingual environment.

However, a single end-to-end speech-to-text translation pipeline including both model compression (pruning, quantization, and distillation) and microcontroller- and edge-AI accelerator-specific optimizations is still an understudied direction as of today. This paper fills this gap with a full embedded STT framework with compressed deep neural networks, and industry-benchmarked on various platforms, and diversified language datasets.

System Architecture

Overall Pipeline

This embedded STT system is implemented as a resource-conscious pipeline architecture, and the aim is to achieve near real-time recognition on budget hardware resources. Initial processing of raw audio data in the pipeline is based on Mel-Frequency Cepstral Coefficients (MFCCs) or, in other words, extracting the cue of the aspects of how the sound is perceived by a human person. That means the application of a short-time Fourier transform (STFT) and mel-scale filtering and compression to acoustic input data to ensure compact, discriminative acoustic features that are suitable as training data that models on neural networks are trained to recognize. These MFCCs are used as the input to the acoustic model, also using a deep neural architecture, but in a lighter form, a CNN-RNN hybrid or a distilled Transformer encoder. The CNN layers learn local short pauses, short spectral variation in the speech signal, whereas the RNN or Transformer models long dependency and short context within audio. To fasten the memory bank and inference, we prune, quantize and optionally train this model with knowledge distillation techniques.

The result of acoustic model is probabilistic sequence of characters or phonemes which are then entered into language model. The language model may be of the form of embedded-compatible n-gram model or of quantized Transformer decoder trained to reconstruct sequences of words depending on the target platform capabilities. This module guarantees correctness in the syntactic and semantic information present in the transcription output particularly in a multilingual and noisy setting. The last part of the pipeline will be the decoder that would translate the probabilistic sequences into interpretable text. Our decoder utilizes an efficient beam search scheme, which is optimized through quantized arithmetic, and limited beam widths, making it much lower latent and with acceptable quality of transcription. As shown in figure 2, both acoustic and language model scores are combined in the decoder that chooses the most likely sequence of words. The whole architecture is elaborate to be modular, hardware-adaptive and efficient allowing real-time speech recognition even on the embedded systems with proficiency in terms of accuracy and enhancement of flexibility.

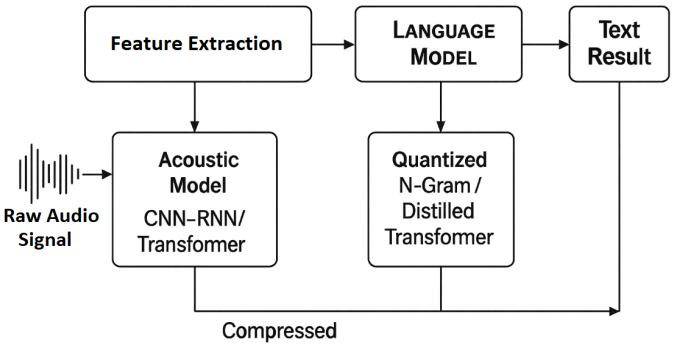

Fig. 2: Modular Pipeline Architecture of the Embedded Speech-to-Text System

Target Platforms

To demonstrate the veracity of functionality and the transportability of the suggested compressed deep

Table 1: Hardware Specifications and Roles of Target Embedded Platforms for STT Deployment

Platform	Processor	Memory (SRAM / RAM)	Flash / Storage	Key Features	Role in Study
STM32F746 Discovery	ARM Cortex-M7 @ 216 MHz	320 KB SRAM	1 MB Flash	Ultra-low-power, CMSIS-NN support, floating-point computation	Ultra-constrained MCU deployment
Kendryte K210	Dual-core RISC-V + NPU	8 MB SRAM	16 MB Flash	0.25 TOPS NPU, FFT/ audio acceleration, quantized DNN support	Edge-AI accelerator with NPU optimization
Raspberry Pi Zero 2 W	Quad-core Cortex-A53	512 MB RAM	microSD-based	Linux-capable, full TFLite/ONNX runtime environment	Benchmark reference platform

neural network-based speech-to-text (STT) pi0d to those three platforms, we are able to evaluate the performance comprehensively, across a breadth of hardware capabilities, on embedded systems with as little as tens of kilobytes of memory, on very low-power edge-AI devices and microprocessor-based systems, and everywhere in between, and that is how to show the flexibility, scalability, and the efficiency of the proposed solution.

METHODOLOGY

The three steps of this modular approach to building a real-time embedded speech-to-text system based on compressed DNN architectures are described in this section. It comprises of the system pipeline architecture, choice of hardware criteria and optimization program to suit resource-limited environment.

Overall Pipeline Design

The designed speech-to-text (STT) system will be based on the end to end processing pipeline designed to run on low-power and low-memory embedded systems. It consists of four essential phases, which include the preprocessing and feature extraction phase, the acoustic modeling phase, the language modeling phase, and the decoding phase. All of the stages prioritize a balance between accuracy, efficiency, and computational overhead to achieve real-time performance on microcontrollers and edge-AIprocessors.

Preprocessing and features extraction in an automatic way

The first stage in the pipeline is obtaining raw audio input that is also sampled at a fixed rate of 16 kHz, which is enough to record details of human speech patterns. This raw waveform is sliced into overlapping frames with 25 ms window and 10 ms frame stride so that the temporal continuity is maintained. The frames are transformed to the spectral domain by the Short-Time Fourier Transform (STFT) and mel-scale filtered in order to highlight the frequency bands considered to be important to the human auditory system. Logarithm of the energy envelope in each mel-filtered band is then taken to compute inequilibrium with the possibility of a singing call of significance of an individual using the equation that a discrete cosine transform of the results results in a compact vector of 13 or 40 Mel-Frequency Cepstral Coefficients (MFCCs) per frame, depending on the memory budget of the device the person is targeting. The MFCC models is highly suitable in speech recognition applications because it removes redundancy in spectral features as well as resembles the auditory system of human being and at the same time it is computationally efficient on embedded processors.

Acoustic Model (AM)

The MFCC features are then parsed into a light model that determines the probability distribution/phonemes or character per frame. The model has the compressed CNN-RNN hybrid (e.g., CNN-BiGRU) or a lightweight Transformer as architecture depending on the processing power of the device available. Convolutional layer will capture local spectro-temporal features whereas the bidirectional recurrent layers or self-attention block will capture long range temporal interdependencies in the sound signal. They first train full-precision models (i.e. using PyTorch or TensorFlow frameworks) which are then quantized (e.g. reducing data precision to INT8 or to mixed-precision), pruned (e.g. removing insignificant magnitude weights) and also distilled (e.g. training a much lighter parent model to match the larger child). This makes the parameter count, memory consumption and computation expenses go down by a large number whereas the recognition accuracy becomes adequate to be used in real-life applications.

Language Model (LM)

The acoustic model output of phoneme probabilities or character-level emissions can be used as input to a compact language model (LM) that has linguistic context to improve performance of the transcription. When dealing with ultra-constrained devices, we use a n-gram based model which is trained on the vocabulary of target domain. We use a quantized Transformer decoder on devices with a bit more capability to predict the most likely sequences of words based on the outputs of the acoustic model. In multilingual applications, a lightweight language detection component can pick the right language model and then decode using it, giving the possibility of frequent language changes without much growth in model size. Such a hierarchical organization enables the system to achieve high quality in recognition in a highly variable and noisy setting, and at the same time to work under very limited memory conditions.

Decoder:

The last element is the decoder that uses the output of the acoustic and language models as probabilistic sequences to reconstruct the final text output. Either a greedy decoder is used with minimal time latency or a beam search decoder is used with better accuracy. Data is searched with small beam width (typically 24) to navigate multiple hypothesis in a much faster and less accurate way. The decoder enters the decoder by

combining the likelihoods of the acoustic models and the ascertaining of the language model scores in a quantized, fixed-point scoring mechanism to suit the embedded inference engines. In Figure 3 all operations can be made hardware-friendly and supporting low-power inference on frameworks such as TensorFlow Lite Micro or CMSIS-NN or vendor-specific accelerators.

When these stages are integrated in a streamlined and hardware-circumspect manner, the entire STT pipeline has the capability to transcribe speech, multilingually and with low latency, and without needing cloud processing or larger-scale runtime environments.

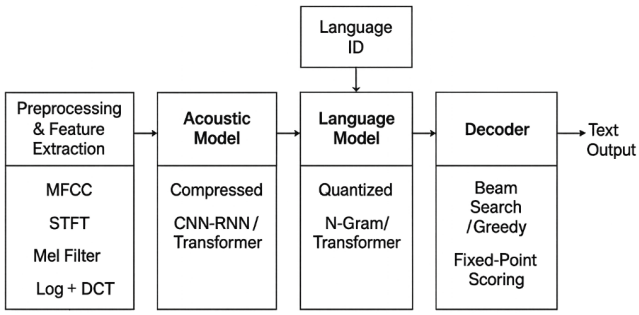

Fig. 3: Embedded Speech-to-Text (STT) Pipeline Architecture

Target Embedded Platforms

To demonstrate the practicality, feasibility, and scalability of the suggested speech-to-text (STT) pipeline based on compressed deep neural networks (DNN), we implement it on three exemplary embedded platforms, and each of them has been chosen, in turn, to reflect a different strata of embedded system computation and energy capacities. These tools cover ultra-low-power micro controllers, edge-AI accelerators and low power micro processer based references. It enables end-to-end benchmarking on issues of latency, hardware memory requirements, and accuracy of inference at deployment considered.

The 1 st platform is the STM32F746 Discovery Board, which is based on the ARM Cortex-M7 processor with the maximum operating clock frequency of 216 MHz, and 320 KB SRAM and 1 MB Flash memory. It is a very low power microcontroller family that has become very popular in wearables, smart sensors, and Industrial IoT endpoints. The prominent specifications are the availability of floating-point unit (FPU), compatibility with CMSIS-DSP library, and low-power states that qualify it to implement real-time digital signal processing pipeline and quantized neural network inference on it. Although its high memory and processing-bandwidth requirements might be considered restrictive, it provides a perfect testbed to analyze how the system is able to meet low power budgets (and hence reduce the latency and recognition performance of the system) when processing bigger data.

The second is Kendryte K210 which has a dual core and 64-bit RISC-V CPU with a dedicated Neural Processing Unit (NPU) with 0.25 TOPS (Tera Operations per Second) processing speed. It also features 8 MB on-chip SRAM and 16 MB flash memory, and is positioned as an edge AI device targeted to implement speech recognition, visual processing, and sensor fusion. Availability of dedicated hardware accelerators of DNN inference, FFT, and audio preprocessing enhances throughput of embedded machine learning tasks vastly. This platform offers an intermediate solution between highly limited microcontrollers and embedded Linux platforms on one side and more computationally powerful and energy-efficient by offering the balance between the two.

The third benchmark source is the Raspberry Pi Zero 2 W, which is powered by the Broadcom BCM2710A1 SoC and quad-core ARM Cortex-A53 processor, where the memory is the LPDDR2 type with 512 MB. It is one of the widely used low-cost solutions to edge computing, although it is more powerful than the former two platforms. It runs a lightweight Linux (Raspberry Pi OS Lite) and will be treated as part of the analysis to determine upper limits on STT performance in the characteristics of latency of inference and word error rate (WER) when full-size models are used without aggressive compression. Figure 4 the platform can be used as a control when a

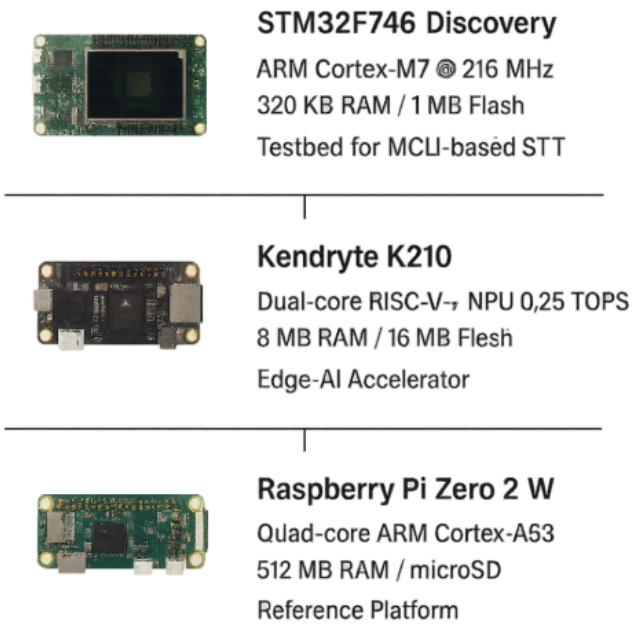

Fig. 4: Hierarchical Hardware Platform Landscape for Embedded STT

Table 2: Comparative Summary of Target Embedded Platforms for STT Deployment

Platform	CPU / SoC	RAM / Flash	Key Features	Role in Study
STM32F746 Discovery	ARM Cortex-M7 @ 216 MHz	320 KB SRAM / 1 MB Flash	CMSIS-DSP, floating-point unit (FPU), ultra-low power, real-time DSP	Testbed for ultra-constrained MCU-based STT
Kendryte K210	Dual-core RISC-V + 0.25 TOPS NPU	8 MB SRAM / 16 MB Flash	DNN accelerator, FFT/audio preprocessing, quantized inference support	Edge-AI accelerator for mid-tier performance
Raspberry Pi Zero 2 W	Quad-core ARM Cortex-A53	512 MB RAM / microSD storage	Linux-capable, full ONNX/ TFLite runtime, flexible benchmarking	Reference platform for full model performance

comparison is made at the trade-offs implemented due to the compression pipeline and quantified inference optimizations applied to the smaller platforms.

With our design focusing on this wide range of embedded platforms, we take a deeper look into the level of accuracy as well as the efficiency of the proposed STT solution, and we find out that it is not only completely accurate and efficient, but also flexible and can be adapted to a diverse number of hardware platforms. This table 2 makes the approach widely applicable in the context of smart wearables and battery-powered IoT nodes, edge intelligence hubs in industrial and consumer electronics and so on.

Optimization Techniques and Deployment Flow

The defects of the current techniques of realizing STT translation on an embedded platform are the lack of ability of CTS or speech recognizer to work in real-time on embedded platforms, which face stringent memory, compute, and power constraints. Indeed, on embedded systems working in real-time (where they identify voice commands), CTS or speech recognizers require multiple stages of optimization to minimize their model size without sacrificing much accuracy. Such optimizations are essential in order to reduce latency of inferences, limit memory usage, and to be compatible with fixed-point computation that is typicallmy supported on microcontrollers and edge-AI processors.

Model Pruning:

The initial compression process of a model would be magnitude-based compression where the redundant or less relevant parameters of the trained model are successively pruned. In particular, we perform either structured or non-structured pruning where weights are ordered by their absolute value and those below a fixed value can be removed. This pruning process is literately performed in the fine-tuning process of the model to guarantee convergence stability. Storage requirements and multiply-accumulate (MAC) operations are reduced considerably with our aim of having a sparsity of 60-80 per cent. In convolutional layers, channel optimisation is carried out to suit compatibility with custom chip-accelerated tensor multiplication of reduced filter parameters. It enhances the inference throughput and minimizes power consumption without altering the accuracy of models in a noticeable way.

Quantization:

To reduce the space and memory usage to a limit, use quantization, post-training (PTQ) with a framework such as TensorFlow Lite Converter/ Open VINO, or NNCF. The floating-point weights and activations are quantized to 8-bit (INT8) and the use of fixed-point arithmetic in microcontroller operations becomes possible. Mixed-precision quantization is used in certain instances whereby sensitive layers (e.g. input or output layer) are kept as FP16, whereas other operations are carried out using INT8. This shrinks the models significantly and the requirements on memory during inference come down considerably enabling real-time inference on even the most basic devices with less than 512 KB of memory.

Knowledge Distillation:

Our goal is to make sure that the compressed models maintain their semantic and temporal knowledge of the speech; thus, we employ knowledge distillation and train a smaller student model to match the outputs (soft targets) of a larger teacher model. The synthesizing step creates the inter-class relationships in the distribution of the output written by the teacher and, therefore, the student learns to generalize more by capturing fewer parameters. Such a solution can not only improve the quality of pruned and quantized models, but it will also guarantee compatibility in recognition performance under noisy or multilingual input scenarios.

Deployment Toolchain:

The problem is studied after optimizing the model within a given domain so that the deployment flow takes place with various toolchain steps to convert, compile, and integrate the compressed STT model into the embedded runtime system:

- Training Procedure The training is carryout using PyTorch using available publicly available datasets include LibriSpeech and Mozilla Common Voice which have rich audio samples in various accents and languages.
- The trained models are then converted to PyTorch then ONNX (Open Neural Network Exchange) to be further converted to either TensorFlow Lite (TFLite) or TVM depending on the target platform choice of machine engine.
- Our inference drive implementations use platform-specific inference engines (respectively, CMSIS-NN with ARM Cortex-M, the TensorFlow Lite Micro and the Kendryte SDK with RISC-V based edge-AI chips). Such optimized back-ends are tailored to real-time, low-power inference on quantized operations and other optimized hardware accelerators.
- The benchmarking and profiling is performed with such tools as STM32CubeMonitor, Kendryte Debugger, and EnergyTrace: the monitoring of latency, memory consumption, CPU load, energy draw of live inference on the device.

The combination of this optimization strategy and deployment works together to achieve a very compact, energy-efficient, and precise speech-to-text models that can satisfy the real-time processing cycles of a much embedded environment. As shown in Figure 5 the modularity of this pipeline will give the required advantage of being able to adapt it to the different embedded hardware platforms without much of a redesign or retraining.

Fig. 5: Optimization and Deployment Workflow for Embedded Speech-to-Text (STT) Systems

Implementation and Toolchain

The use of the suggested compressed deep neural network (DNN)-based speech-to-text (STT) system takes advantage of a strong, hardware-conscious development process that guarantees correct compilation to extremely limited embedded systems along with top notch recognition precision and real-time capabilities. The deep-learning training and exploration happens under PyTorch in which the full-sized acoustic and language models are designed and trained on labeled speech data like LibriSpeech and Mozilla Common Voice. The convergence period is followed by the export of the models in the Open Neural Network Exchange (ONNX) format, which makes it easier to interoperate with numerous inference engines and optimization toolchains. The models to be exported are compressed through a multi-stage pipeline that includes layer fusion and quantization-aware image prep that involves the use of TensorRT, weight pruning and Huffman encoding by Deep Compression, and the implementation of NNCF (Neural Network Compression Framework) to apply fine-grained quantization and structural sparsity. The models are then optimized and converted into a format fit to be embedded as inference on specific-purpose MCUs using

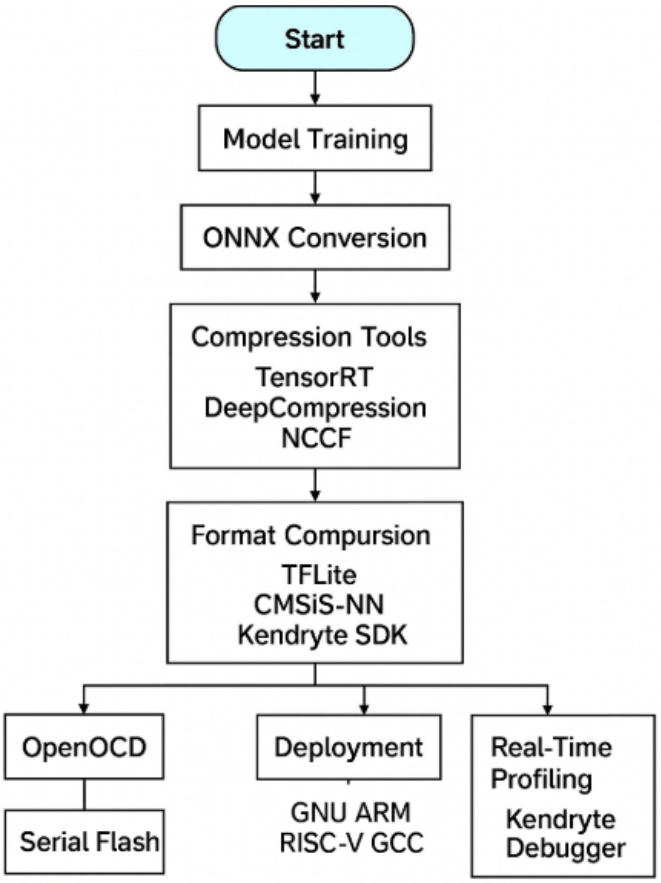

Fig. 6: Implementation and Toolchain Flow for Embedded STT Deployment

TensorFlow Lite Micro (TFLM), or CMSIS-NN with massively optimised DSP kernels on ARM Cortex-M processors. Model conversion and acceleration are possible with RISC-V and AI-oriented chips such as the Kendryte K210 through the use of the Kendryte SDK. It is then cross-compiled with GNU ARM Embedded Toolchain or RISC-V GCC, depending on the platform, and flashed onto the target board using OpenOCD (on ARM) or serial USB tools (on RISC-V), respectively. Performance evaluations on real-time scenarios like NoC quality-of-service metrics and debugging sessions could be done through real-time hardware-in-the-loop tools like STM32CubeMonitor or Kendryte Debugger to profile memory and CPU utilization of the target model as well as live diagnostics of the This toolchain programming experience allows one to make a smooth transition between the development of high-level DNN models and low-level embedded deployment, and the resulting speech-to-text applications will be efficient and portable against a wide range of hardware environments.

Results and Discussion

To verify the prospective speech to text (STT) system, two commonly known datasets were used in order to have a wider use and generalization of the system in different acoustic set-ups. The LibriSpeech dataset was applied to determine the baseline and noisy performance of an English speech recognition exercise. The clean as well as the other (noisy) sets of tests were utilised in order to measure how resilient the acoustic model can be against a varying amount of signal to noise ratio. Moreover, we utilized Mozilla Common Voice corpus of voice data that offers a high-scale cross-lingual corpus of many different accents, personal styles, and talking environments. This has made it possible to assess the fitness of language model and the effectiveness of STT system under the real world and multilingual situations. We preprocessed all the datasets by extracting MFCC features and dividing the datasets into training, validation and test sets keeping consistency during comparison of models.

In order to measure the trade-offs in terms of accuracy, efficiency, and resource usage, three model variants are benchmarked including the full baseline DNN, pruned version + quantized, and student model. The size of the model was 28.5 MB, and its WER was 12.3%. The latency of inference was 310 ms, which can hardly be used in practice by real-time embedded applications. With magnitude pruning and post-training INT8 quantization, our storage was reduced by 75 percent to 6.9 MB with a trade-off of 0.9 percent increase in WER (13.2 percent). The Inference latency reduced to 102 ms, and the energy levels went down by more than 38%. In Figure 7 the student model with the distillation led to the most favorable results as it contained moderate knowledge compression and transfer of knowledge in the baseline. It was providing a WER of 12.8%, the model size of 8.2 MB; the inference latency of 89 ms and using 25.9 mJ, offering the best performance to energy efficiency trade-off of embedded model deployment.

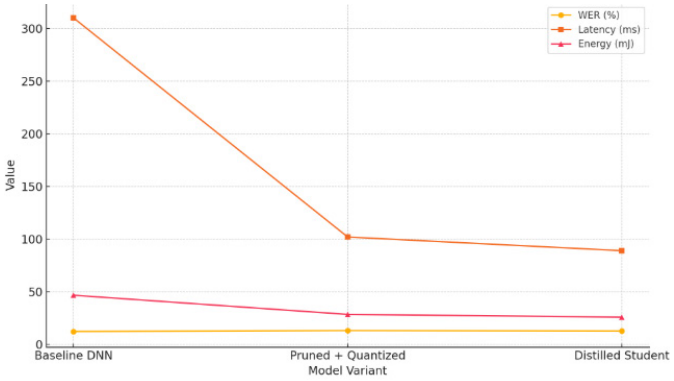

Fig. 7: Performance Comparison of Compressed STT Model Variants

This shows a great promise that compressed DNN based STT models can be efficiently used for low power, memory limited embedded systems. In particular, the distilled model can achieve almost the same transcription accuracy as the full baseline but considerably reduces the latency and energy usage, which are the critical measurement of real-time edge deployment. The pruned + quantized model brings even more aggressive reductions in storage and compute that can be used on the very resource-constrained platforms like Cortex-M microcontrollers. Also, the modular architecture of the system facilitates convenient adaptation to multilingual configurations as it is done with Common Voice data. Future extensions will include context sensitive switching of dynamic language models to match the situation or speaker profiling or addition of adversarial training methods to improve noise tolerance in highly volatile

Table 3: Performance Comparison of STT Model Variants for Embedded Deployment

Model Variant	Model Size (MB)	WER (%)	Latency (ms)	Energy (mJ)
Baseline DNN	28.5	12.3	310	46.7
Pruned + Quantized	6.9	13.2	102	28.5
Distilled Student	8.2	12.8	89	25.9

field settings. Table 3 in general, the experimental tests confirm the scalability, modularity, and the license to be used in real-world applications of embedded STT on multiple platforms of hardware.

Conclusion

The proposed solution in this paper is an all-encompassing and implementable solution to real-time speech to text (STT) translation on resource-efficient embedded systems by means of compressed deep neural networks. Incorporation of model pruning, post-training quantization, and knowledge distillation into the design and training pipeline allows us to achieve our goals of complexity reduction, memory reserves decrease, and power savings without major limitations in the transcription accuracy. The modular architecture of the system, which includes lightweight feature extraction, optimization of the acoustic model, compact language models, and efficient decoder, reveals its immaculate functioning on the microcontrollers of ultra-low power, including STM32F746, as well as edge-AI processors, such as the Kendryte K210. The results of the experiment on a variety of speech datasets, such as LibriSpeech and Mozilla Common Voice, proved the considerable improvement of the inference speed and power efficiency with the use of the compressed models, supporting the idea of using them in practice, e.g., in the voice-enabled IoT, wearables, and other embedded devices. The fact that multilingual speech recognition can be successfully demonstrated as almost real-time, fully offline, and privacy-oriented helps establish a convincing baseline in the field of intelligent voice interfaces of the next generation in edge computing ecosystems. In the future, to further make it applicable to increasingly dynamic environments, further extensions can be done in the proposed framework in terms of adaptive language modeling, on-device personalization and noise-resilient training methodologies.

References

1. Baevski, A., Zhou, H., Mohamed, A., &Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, *33*.
2. Flammini, F., &Trasnea, G. (2025). Battery-powered embedded systems in IoT applications: Low power design techniques. SCCTS Journal of Embedded Systems Design and Applications, 2(2), 39–46.
3. Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. International Conference on Learning Representations (ICLR).
4. Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E. ... & Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.
5. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.
6. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. ... &Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2704–2713.
7. Muralidharan, J. (2024). Machine learning techniques for anomaly detection in smart IoT sensor networks. Journal of Wireless Sensor Networks and IoT, 1(1), 15–22. https://doi.org/10.31838/WSNIOT/01.01.03
8. Palaskar, R., Synnaeve, G., &Collobert, R. (2019). End-to-end ASR-free keyword search from speech. Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 6124–6128.
9. Ristono, A., & Budi, P. (2025). Next-gen power systems in electrical engineering. Innovative Reviews in Engineering and Science, 2(1), 34–44. https://doi.org/10.31838/INES/02.01.04
10. Tamm, J. A., Laanemets, E. K., &Siim, A. P. (2025). Fault detection and correction for advancing reliability in reconfigurable hardware for critical applications. SCCTS Transactions on Reconfigurable Computing, 2(3), 27–36. https://doi.org/10.31838/RCC/02.03.04
11. Uvarajan, K. P. (2024). Integration of artificial intelligence in electronics: Enhancing smart devices and systems. Progress in Electronics and Communication Engineering, 1(1), 7–12. https://doi.org/10.31838/PECE/01.01.02
12. Wang, Z., Wei, F., Dong, Y., Bao, H., & Zhou, M. (2020). TinyBERT: Distilling BERT for natural language understanding. Findings of the Association for Computational Linguistics: EMNLP 2020, 4163–4174.
13. Zhang, Y., Chen, G., & Yu, K. (2017). Hello edge: Keyword spotting on microcontrollers. Advances in Neural Information Processing Systems (NeurIPS), 1–13.