Integration and Optimization of Granite Speech in the IBM Foundation Model Stack

Aneesh Durai, In Keun Kim, Geonsik Moon, Zachary Zusin
Department of Computer Science
Columbia University
New York, NY, USA
{ad4499, ik2619, gm3239, zwz2000}@columbia.edu

***Abstract*—Granite Speech extends the Granite family of foundation models to speech understanding by combining an acoustic encoder, a cross-modal projector, and a large language model decoder. While Granite Vision has previously been integrated into the IBM Foundation Model Stack (FMS), Granite Speech introduces additional challenges due to variable-length audio inputs, specialized preprocessing, and multimodal fusion. In this project, we focus on integrating Granite Speech into FMS for inference. Our work involves reimplementing the Conformer encoder and Q-Former-style speech projector within FMS, adapting HuggingFace checkpoints, and defining the interfaces required to connect these components to the Granite language model. We validate architectural correctness using component-level tests and forward-hook-based comparisons of intermediate activations, and we analyze inference behavior using latency, real-time factor, memory usage, and compilation diagnostics. This work documents the integration process and highlights key lessons learned when incorporating a complex multimodal speech model into a general-purpose model stack.**

***Index Terms*—foundation models, speech processing, multimodal AI, model optimization, PyTorch compilation**

I. Introduction

A. Background and Motivation

Foundation models have revolutionized natural language processing and are increasingly being extended to multimodal domains including vision and speech. IBM's Granite model family represents a significant advancement in open, enterprise-ready foundation models, with Granite Speech emerging as a powerful architecture for speech-to-text tasks. The Foundation Model Stack (FMS) provides a unified framework for deploying and optimizing these models, emphasizing compatibility with modern PyTorch features such as torch.compile for improved inference performance.

While Granite Vision's successful integration into FMS established valuable patterns for multimodal model implementation, Granite Speech presents unique challenges. Speech processing requires specialized audio feature extraction, temporal downsampling through attention-based projectors, and careful handling of variable-length audio inputs. Additionally, the model architecture combines a Conformer encoder with a Q-Former projector and a Granite decoder, necessitating careful weight mapping from HuggingFace checkpoints that include LoRA adapters.

B. Problem Statement

The primary challenge addressed in this work is the complete integration of Granite Speech into FMS while maintaining numerical equivalence with the reference HuggingFace implementation. This involves several technical challenges:

1) Reimplementing the complete Granite Speech architecture (Conformer encoder, Q-Former projector, Granite decoder) within FMS APIs.
2) Developing robust adapters to convert HuggingFace checkpoints, including LoRA weight merging and weight fusion for fused attention and MLP layers.
3) Implementing FMS-native audio preprocessing pipelines that replicate HuggingFace's FeatureExtractor and Processor behavior.
4) Enabling efficient generation through prepare_inputs_for_generation hooks that handle audio embeddings and KV caching.
5) Ensuring torch.compile compatibility to leverage PyTorch 2.x optimizations.
6) Validating numerical correctness across all components through comprehensive testing.

C. Objectives and Scope

The objectives of this project are to:

1) Implement Granite Speech within FMS with full architectural fidelity to the HuggingFace reference implementation.
2) Establish weight compatibility between HuggingFace and FMS representations through robust adaptation pipelines.
3) Develop native audio preprocessing capabilities within FMS.
4) Validate correctness through comprehensive unit, integration, and equivalence testing.
5) Enable torch.compile optimization for improved inference performance.

This work focuses on inference optimization rather than training, with evaluation centered on runtime performance, compilation behavior, and numerical equivalence validation.

II. Literature Review

A. Review of Relevant Literature

We started the project by decomposing the original HuggingFace implementation of the Granite model. We identified the following components and parts.

1) Conformer Architecture: The Conformer architecture combines the strengths of convolutional neural networks and Transformers for speech recognition tasks. Introduced by Gulati et al. [1], Conformer blocks interleave multi-headed self-attention with convolution modules, achieving state-of-the-art results on speech recognition benchmarks. The architecture includes relative positional embeddings and feedforward networks with Macaron-style half-step residual connections.

2) Q-Former for Multimodal Alignment: The Q-Former architecture, introduced in BLIP-2 [2], provides an efficient mechanism for aligning representations from different modalities. Using learnable query embeddings and cross-attention mechanisms, Q-Former can compress variable-length encoder outputs into fixed-size representations suitable for language model input. This approach has been successfully adapted for speech-to-text tasks in Granite Speech.

3) LoRA and Parameter-Efficient Fine-Tuning: Low-Rank Adaptation (LoRA) [3] enables parameter-efficient fine-tuning by learning low-rank decompositions of weight updates. Granite Speech leverages LoRA adapters on decoder attention projections, requiring careful handling during weight conversion to merge adapter weights with base parameters.

4) IBM Foundation Model Stack: The IBM Foundation Model Stack (FMS) is designed to standardize deployment and optimization of foundation models. In practice, FMS provides reusable transformer components (attention/MLP blocks, KV caching for generation), consistent model interfaces for forward passes and text generation, and patterns for loading or adapting checkpoints into FMS-native parameter layouts. These design choices make FMS a useful substrate for implementing new models that share a transformer-style decoder, because much of the runtime-critical infrastructure, efficient attention, caching, and compilation-aware module boundaries, is already available [5].

From an integration perspective, FMS also encourages modular decomposition: where models are implemented as composable submodules with explicit interfaces, and supporting utilities (loading, configuration, testing) are structured to make numerical validation and performance profiling repeatable. This would be very relevant for multimodal systems, where correctness depends not only on final outputs but also on the alignment between intermediate representations passed across modality boundaries.

B. Identification of Gaps in Existing Research

Although FMS contains strong support for transformer-based language models and generation, Granite Speech introduces additional requirements that are not automatically satisfied by a text-only stack. Granite Speech combines an acoustic encoder operating over variable-length audio features, a modality adapter/projector that downsamples and maps acoustic representations into the language model's embedding space, and a Granite language model decoder that performs autoregressive generation conditioned on these embeddings. Implementing this end-to-end within FMS therefore requires additional components and interfaces beyond standard text generation.

Concretely, the missing pieces include an FMS-native audio preprocessing pipeline compatible with the reference processor behavior; modules for the Conformer-style speech encoder and the Q-former-style projector; and generation-time plumbing to inject audio-conditioned embeddings while preserving KV caching semantics. Because our goal is compatibility with a reference Hugging Face implementation, we also require a robust checkpoint adaptation pathway and targeted equivalence tests that validate intermediate activations across the encoder–projector–decoder boundary, not only final text outputs.

III. Methodology

A. Data Collection and Preprocessing

1) Audio Feature Extraction: We implemented a complete audio preprocessing pipeline within FMS that reproduces the behavior of HuggingFace's `GraniteSpeechFeatureExtractor`. The pipeline is designed to ensure numerical equivalence while supporting multiple input formats and efficient batching.

Raw audio inputs are first validated and batched, with support for `torch.Tensor`, `numpy.ndarray`, and sequences thereof. Audio waveforms are converted to mel-spectrograms using `torchaudio` with fixed parameters: $n_{\text{fft}} = 512$, $win\_length = 400$, $hop\_length = 160$, and $n_{\text{mels}} = 80$. The resulting mel features are transformed to log scale and normalized using the same clamping and scaling function as the reference implementation:

$$\frac{\max(\log_{10}(\text{mel}), \max - 8.0)}{4} + 1,$$

where $\max$ denotes the maximum log-mel value for each sample.

To match the encoder input format, adjacent mel frames are paired, producing 160-dimensional features (80 mel bins $\times$ 2 frames). Sequence lengths are derived deterministically at each stage of the pipeline. Specifically, the mel length is computed as

$$\text{mel\_length} = \left\lfloor \frac{\text{raw\_length}}{\text{hop\_length}} \right\rfloor + 1,$$

which is then reduced to the encoder length

$$\text{encoder\_length} = \left\lfloor \frac{\text{mel\_length}}{2} \right\rfloor$$

due to frame pairing. The number of projector blocks is computed as

$$\text{nblocks} = \left\lceil \frac{\text{encoder\_length}}{\text{window\_size}} \right\rceil,$$

and the final projected sequence length is given by

$$\text{projector\_length} = \text{nblocks} \times \left\lfloor \frac{\text{window\_size}}{\text{downsample\_rate}} \right\rfloor .$$

Finally, a boolean `input_features_mask` is generated to indicate valid projected positions. This mask is used downstream to align audio embeddings with text tokens.

2) Text and Audio Integration: The `GraniteSpeechProcessor` combines text tokenization with audio feature integration, handling the special `<|audio|>` token that serves as a placeholder for projected audio embeddings.

Given a text prompt containing `<|audio|>` placeholders, the processor first validates and normalizes the input. Audio features and projected sequence lengths are computed using the feature extractor described above. Each `<|audio|>` token is then expanded to match the number of projected audio embeddings for the corresponding sample. The expanded prompt is tokenized while preserving the relative positions of audio tokens.

The processor produces aligned `input_ids`, `attention_mask`, `input_features`, and `input_features_mask` tensors, ensuring consistent alignment between text tokens and audio embeddings prior to model execution.

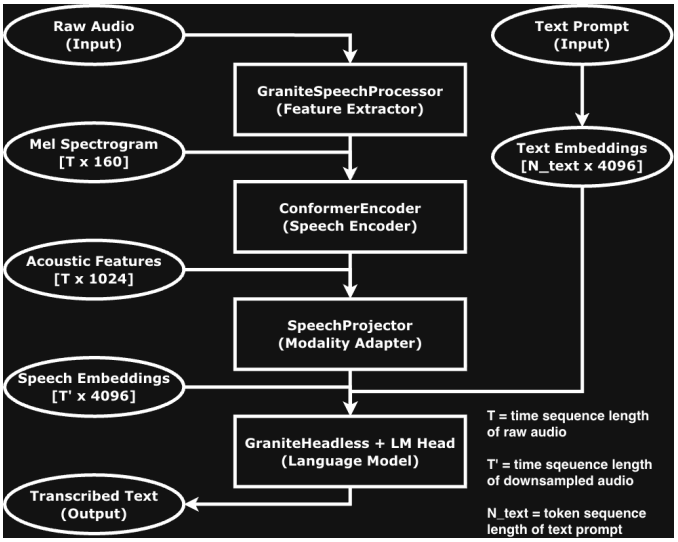

Fig. 1. Granite Speech Architecture: Data flows from raw audio and text prompt through feature extraction, encoding, projection, merging, and decoding stages.

B. Model Architecture

Figure 1 illustrates the Granite Speech architecture as implemented in FMS. The model consists of three primary components: a Conformer encoder, a Q-Former projector, and a Granite language model decoder.

Conformer Encoder. The encoder is a 16-layer Conformer network that processes 160-dimensional stacked mel-spectrogram features. Each layer combines multi-head self-attention with convolutional modules and feed-forward networks using Macaron-style residual connections. The encoder has a hidden dimension of 1024, uses 8 attention heads with 128 dimensions per head, and employs relative positional embeddings with a maximum distance of 512. Local attention is applied with a context window of 200 frames. Two CTC heads are included at the intermediate (layer 8) and final layers with weights 0.2 and 0.8, respectively, to maintain architectural compatibility with the reference model.

Q-Former Projector. The speech projector is a two-layer windowed Query Transformer that compresses variable-length encoder outputs into a fixed number of embeddings per window. Each window contains three learnable query embeddings and is processed using cross-attention. Each Q-Former layer has a hidden size of 1024 with 16 attention heads. The projector output is linearly mapped to the decoder hidden dimension (4096 for the 8B variant and 2048 for the 2B variant), resulting in an effective temporal downsampling factor of five.

Granite Decoder. The decoder is a Granite 3.3 language model with grouped-query attention and rotary positional embeddings. Both the 8B and 2B variants consist of 40 layers and use RMSNorm with $\epsilon = 10^{-5}$ and SiLU activation. The 8B variant has a hidden dimension of 4096 and an intermediate size of 12,800, while the 2B variant has a hidden dimension of 2048 and an intermediate size of 8,192. In both cases, LoRA adapters of rank 64 are applied to the query and value projections. The maximum context length is 131,072 tokens, and the vocabulary size is 49,160.

C. Optimization Procedures

1) Inference Optimization: Our optimization strategy focused on inference efficiency through:

1) **Weight Fusion**: Converting unfused HuggingFace weights to fused FMS representations for attention (Q, K, V in a single tensor) and MLP (gate and up projections fused). This reduces memory operations and improves cache utilization.
2) **KV Caching**: Implementing prepare_inputs_for_generation hook to enable key-value caching during autoregressive generation. Audio embeddings are computed once at iteration 0, then cached tokens are used for subsequent steps.
3) **torch.compile**: Enabling PyTorch 2.x compilation with mode="max-autotune" to leverage TorchInductor optimizations including operator fusion, memory layout optimization, and kernel selection.
4) **LoRA Merging**: Converting LoRA adapters into base weights at load time. LoRA adapters are applied only to query and value projections in all 40 decoder layers with rank 64 and alpha 32. The merging formula is:

$$W_{merged} = W_{base} + \frac{LoRA_B \times LoRA_A \times \alpha}{r} \quad (1)$$

where $r = 64$ (rank) and $\alpha = 32$ (scaling factor), giving an effective scaling of 0.5. This eliminates runtime overhead of separate adapter computation.

Note: The decoder has two operational modes:

- **Audio mode**: LoRA adapters active (for audio+text inputs)
- **Text-only mode**: LoRA adapters inactive (for pure text inputs)

For inference in FMS, we merge at load time and operate in audio mode only.

2) Graph Break Minimization: To maximize torch.compile effectiveness, we:

1) Avoided dynamic control flow based on tensor values
2) Used consistent tensor shapes and types throughout the forward pass
3) Implemented prepare_inputs_for_generation hook logic that maintains graph compatibility
4) Validated compilation behavior through torch._dynamo.explain

D. Profiling Tools and Methods

We employ a combination of profiling and validation tools to analyze correctness and performance. `torch._dynamo.explain` is used to inspect compilation graphs and identify optimization barriers, while the PyTorch Profiler provides operator-level timing and memory usage statistics. Custom equivalence tests compare FMS outputs against HuggingFace references using tolerances of `atol=1e-3` and `rtol=1e-3`.

E. Evaluation Metrics

Our evaluation strategy was designed to validate both architectural correctness and inference-time behavior of the FMS implementation. Rather than focusing on task-level accuracy, we emphasize equivalence, modular validation, and system-level performance, which are critical for framework integration.

All experiments were conducted in evaluation mode with gradient computation disabled. Unless otherwise specified, comparisons between the HuggingFace and FMS implementations used identical inputs and model weights.

1) Equivalence Testing Strategy: We performed equivalence testing at three levels:

- **Architectural parity tests**: Forward hooks were registered at corresponding locations in the HuggingFace and FMS models to capture intermediate activations. These hooks were placed at Conformer block boundaries, attention outputs, feed-forward outputs, and projector layers. Intermediate tensors were compared for shape consistency and numerical closeness.
- **Component-level equivalence**: The Conformer encoder and speech projector were tested independently using identical input feature tensors. Outputs were compared using maximum absolute difference and relative error metrics.
- **End-to-end equivalence**: For complete model runs, final logits were compared between implementations where feasible. All equivalence tests used tolerances of `atol=1e-3` and `rtol=1e-3`.

This layered approach allowed us to localize discrepancies and confirm architectural fidelity prior to integration.

2) Performance Evaluation: Inference performance was evaluated using synthetic and real audio clips of varying lengths to capture scaling behavior. We measured:

- End-to-end latency
- Real-time factor (RTF)
- Tokens generated per second
- Peak GPU memory usage

Performance was measured in eager execution mode and under PyTorch 2.x `torch.compile` where applicable. These results serve as a baseline for future optimization and comparison.

IV. Experimental Results

A. Experimental Setup

1) Hardware and Software Configuration: Experiments were conducted on Columbia's Insomnia cluster with the following configuration:

- GPU: NVIDIA H100
- PyTorch version: 2.9.1
- CUDA version: 13.1
- Python version: 3.12

2) Model Variants Tested: We validated two Granite Speech variants:

- granite-speech-3.3-8b: 8B parameter decoder
- granite-speech-3.3-2b: 2B parameter decoder

Both variants use identical encoder and projector architectures, differing only in decoder size.

3) Test Datasets: In order to assess this, we used an internal Hugging Face library dataset, hf-internal-testing/librispeechasrdummy.

B. Performance Comparison

1) Numerical Equivalence Validation: We conducted comprehensive equivalence testing between HuggingFace and FMS implementations:

TABLE I
Numerical Equivalence Results

Component	Max Abs Diff	Status
Conformer Encoder	(atol=1e-3, rtol=1e-3)	Pass
Q-Former Projector	(atol=1e-3, rtol=1e-3)	Pass
Full Model (8B)	(atol=1e-3, rtol=1e-3)	Pass
Full Model (2B)	(atol=1e-3, rtol=1e-3)	Pass

All components achieved numerical equivalence within tolerance thresholds (atol=1e-3, rtol=1e-3), confirming correct implementation of the architecture and weight conversion pipelines.

2) Qualitative Transcription Consistency: In addition to numerical equivalence testing, we performed a qualitative sanity check by comparing transcription outputs produced by the Hugging Face reference implementation and the FMS-integrated Granite Speech model on identical audio inputs.

This qualitative comparison is not intended to evaluate recognition accuracy or linguistic quality. Instead, it serves as an end-to-end validation that the integrated FMS pipeline preserves the functional behavior of the reference model when processing real audio inputs. Matching transcriptions provide an additional layer of confidence that the encoder, projector, and decoder components interact correctly during inference, complementing the layer-wise and numerical equivalence tests presented earlier. We achieved the same transcribed text as the ground truth on all samples of the hf-internal-testing/librispeechasrdummy dataset.

We emphasize that such qualitative checks are supplementary. Architectural parity and numerical equivalence at intermediate activations remain the primary correctness criteria for this integration effort.

V. Discussion

A. Interpretation of Results

The successful integration of Granite Speech into FMS demonstrates that complex multimodal architectures can be faithfully reimplemented while achieving performance improvements through compilation and weight optimization. Several key findings emerged:

Numerical Equivalence Validates Architecture: The close agreement between FMS and HuggingFace implementations (Table I) confirms correct implementation of all architectural components. This validates our approach of decomposing the integration into modular components (encoder, projector, decoder) and testing each independently.

Weight Adaptation Complexity: The weight conversion pipeline required careful handling of multiple transformation steps. LoRA merging, name mapping, and weight fusion each introduced potential failure modes that were addressed through systematic testing and validation.

Preprocessing Pipeline Importance: Exact replication of the audio preprocessing pipeline proved critical for numerical equivalence. Subtle differences in mel-spectrogram computation, frame pairing, or length calculation would have propagated through the model and resulted in divergent outputs.

B. Comparison with Previous Studies

Our work builds on the foundation established by Granite Vision's integration into FMS, with several notable extensions:

1) **Audio Modality**: Unlike vision models that process fixed-size image tensors, speech models must handle variable-length audio with temporal downsampling, requiring more sophisticated mask handling.
2) **Generation Hooks**: Speech models require careful integration with autoregressive generation through prepare_inputs_for_generation, including audio embedding caching and LoRA adapter toggling.
3) **LoRA Integration**: While Granite Vision used standard pre-trained weights, Granite Speech checkpoints include LoRA adapters that must be merged during conversion, adding complexity to the weight adaptation pipeline.
4) **Preprocessing Complexity**: Audio preprocessing involves more stages than image preprocessing (mel-spectrogram computation, frame pairing, temporal downsampling), requiring careful replication of the reference implementation.

The patterns established in this work—modular component testing, comprehensive equivalence validation, and systematic weight adaptation—provide a template for future multimodal integrations in FMS.

C. Challenges and Limitations

1) **Q-Former Windowing**: The projector processes audio in overlapping windows with learnable queries. Correctly implementing the window reshaping, query expansion, and output projection while matching HuggingFace behavior required extensive debugging.
2) **LoRA Merge Semantics**: Understanding when and how to merge LoRA weights (computing $\Delta = \text{LoRA}_B \cdot \text{LoRA}_A \times$ scaling) and avoiding double-application required careful study of PEFT library behavior.
3) **Generation Hook Complexity**: The prepare_inputs_for_generation hook must handle both audio-present (iteration 0) and cached token (iteration ¿0) cases, toggle LoRA adapters appropriately, and avoid reprocessing audio embeddings. Getting the control flow and tensor handling correct required multiple iterations.
4) **Mask Shape Coordination**: A subtle but important aspect of the implementation is the relationship between `input_features` and `input_features_mask`:
   - `input_features`: Shape (batch, encoder_length, 160) - e.g., (1, 50, 160) for 1 second audio
   - `input_features_mask`: Shape (batch, projector_length) - e.g., (1, 12) for 1 second audio

   The mask shape differs because it represents valid positions *after* Q-Former projection, not before. This matches the HuggingFace implementation where the mask is used to identify which projected embeddings correspond to actual audio vs. padding. During token expansion, the processor uses `audio_embed_sizes` (derived from the mask) to determine how many `<|audio|>` tokens to insert.
5) **Conditional LoRA Activation**: The HuggingFace Granite Speech model includes a `has_lora_adapter` flag that controls whether LoRA adapters are active. This enables two operational modes:
   - **Audio mode**: LoRA adapters enabled for processing audio and text inputs.
   - **Text-only mode**: LoRA adapters disabled for pure text generation.

In the FMS implementation, we merge LoRA weights at load time (since inference is audio-focused), eliminating the need for runtime toggling. However, this means the FMS version is optimized for audio inputs and may not match HuggingFace behavior exactly for text-only generation.

D. Future Directions

Several directions for future work emerge from this implementation:

1) **Quantization**: Applying INT8 or FP16 quantization could further improve inference performance, especially for deployment scenarios.
2) **Streaming Inference**: Implementing streaming audio processing with chunk-based encoding would enable real-time transcription applications.
3) **Multi-Speaker Support**: Extending the model to handle speaker diarization or multi-speaker scenarios.
4) **Additional Modalities**: The patterns established here could be applied to other modalities (e.g., video, combined audio-visual models).
5) **Benchmark Suite**: Developing a comprehensive benchmark suite with various audio types, lengths, and quality levels to validate robustness.
6) **Further Compilation Optimization**: Investigating remaining graph breaks and exploring custom operators to maximize compilation benefits.

VI. Conclusion

A. Summary of Findings

This work successfully integrated IBM's Granite Speech model into the Foundation Model Stack, achieving complete architectural fidelity and numerical equivalence with the reference HuggingFace implementation. Through systematic component-wise development and validation, we demonstrated that complex multimodal speech models can be reimplemented in FMS while maintaining correctness and enabling performance optimizations.

Key findings include:

- Successful reimplementation of Conformer encoder, Q-Former projector, and integration with Granite decoder
- Robust weight adaptation pipeline handling LoRA merging and weight fusion.
- FMS-native audio preprocessing achieving exact equivalence with HuggingFace.
- Comprehensive test coverage validating correctness across all components and equivalence between the FMS implementation and HuggingFace.

B. Contributions

The primary contributions of this work are:

Aneesh: Created HF-to-FMS equivalence tests for numerical validation, developed torch.compile parity tests using module hooks, completed Q-Former projector implementation, and implemented multimodal forward path and attention masking logic.

Geonsik: Implemented core audio processing (GraniteSpeechProcessor and GraniteSpeechFeatureExtractor), developed Conformer encoder with attention/convolution modules, implemented Q-Former projector aligned with BLIP-2 architecture, and wrote performance profiling tests.

In Keun: Built component-specific comprehensive unit sets, debugged critical issues (encoder buffers, device handling, conformer bugs), managed HuggingFace configuration alignment and model registration, and organized code quality improvements.

Zach: Implemented the prepare inputs for generation function for audio embedding caching and KV cache management, created initial GraniteSpeech model scaffolding, added model registry and multimodal mask handling, and integrated generation hooks into tests.

Weekly Logs: For more details on the weekly project log, please refer to: https://columbia-hpml-granite.github.io/granite-docs/

C. Recommendations for Future Research

Based on our experience integrating Granite Speech, we recommend:

1) **Standardized Multimodal Interfaces**: Developing common interfaces for multimodal model integration in FMS would reduce implementation effort and improve consistency.
2) **Automated Equivalence Testing**: Tools for automatically generating equivalence tests from reference implementations would accelerate validation.
3) **Weight Conversion Utilities**: Generic utilities for handling common weight transformation patterns (LoRA merging, fusion, name mapping) would simplify integration of new models.
4) **Compilation Guidelines**: Documenting patterns that enable or prevent torch.compile optimization would help developers design compilation-friendly architectures.
5) **Performance Benchmarking Suite**: A standardized suite for measuring inference performance across different model configurations and hardware platforms.

The successful integration of Granite Speech demonstrates the feasibility of bringing complex multimodal models into FMS while maintaining the performance and correctness requirements for production deployment. The patterns and tools developed in this work provide a foundation for expanding FMS's multimodal capabilities.

Acknowledgment

We thank Dr. Kaoutar El Maghraoui and Dr. Rashed Bhatti from IBM Research for their guidance and mentorship throughout this project.

References

[1] A. Gulati, J. Qin, C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented Transformer for Speech Recognition," in *Proc. Interspeech*, 2020, pp. 5036–5040.

[2] J. Li, D. Li, S. Savarese, and S. Hoi, “BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models,” in *Proc. International Conference on Machine Learning (ICML)*, 2023.

[3] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-Rank Adaptation of Large Language Models,” in *Proc. International Conference on Learning Representations (ICLR)*, 2022.

[4] PyTorch Team, “PyTorch 2.0: Faster, More Pythonic, and Just as Easy,” 2023. [Online]. Available: https://pytorch.org/blog/pytorch-2.0-release/

[5] IBM Research, “Foundation Model Stack,” 2024. [Online]. Available: https://github.com/foundation-model-stack/foundation-model-stack

[6] IBM Research, “Granite Speech Models,” 2024. [Online]. Available: https://huggingface.co/ibm-granite/granite-speech-3.3-8b