Optimizing ImageBind: Enhancing Multimodal Embedding Models

***Abstract*—ImageBind has proven to be a powerful model that generates unified embeddings across different modalities, providing a great approach to cross-modal understanding. In this work, we experiment with architectural and post-training optimizations for improving the resource usage and ease of use of ImageBind. We purpose a modular approach that separates the model into independent, modality-specific submodels, allowing for on-demand loading and inference based on the modalities relevant to a given task. In addition, we explore model quantization, experimenting with both dynamic and static quantization techniques. These optimizations lead to a notable decrease in memory usage and inference time with minimal impact to the integrity of multimodal embeddings. We also investigate compatibility with edge devices and multi-threaded inference setups, highlighting how our optimizations maintain performance while expanding deployment options, enabling real-time applications such as AR, robotics, and assistive tech. Thus, making ImageBind more scalable and applicable in real-world scenarios where resources are constrained.**

***Index Terms*—ImageBind, Embeddings, Optimizations, Quantization, Decomposition**

I. Introduction

ImageBind [1] represents a novel contribution in multimodal representation learning by learning a unified embedding space across six modalities: images, text, audio, depth, thermal, and inertial measurement unit (IMU) data. Its innovation lies in bringing all these modalities together based on just image-paired data—successfully removing the requirement of large co-occurring multimodal datasets. By leveraging the innate connections that images have to other sensory inputs, ImageBind is able to learn unified representations capable of facilitating powerful zero-shot performance across a wide range of modalities. Thus enabling functionalities like cross-modal retrieval tasks, compositional reasoning via embedding arithmetic, in addition to detection and generation tasks across diverse modalities.

The work presented herein purposes a series of enhancements for improving the efficiency, modularity, and overall usability of ImageBind without compromising its robust capabilities for cross-modal representation. As an initial step, we restructured the architecture into discrete, modality-specific components, enabling selective inference based on the modalities relevant to the task. Additionally, we investigated quantization, both dynamic and static techniques, where we significantly reduced the model's memory footprint with minimal degradation in performance. Collectively, these enhancements make ImageBind more flexible and efficient for a wide range of real-world applications where resources are constrained.

II. Related Works

The swift advancement of multimodal embedding models has been driven by the success of contrastive learning models, including CLIP [Radford et al., 2021] [2], ALIGN [Jia et al., 2021] [3], and Florence [Yuan et al., 2021] [4]. These models have shown that big image-text alignment can be attained effectively using paired datasets and scalable architectures. AudioCLIP [Guzhov et al., 2021] [5] expanded this to the audio modality, and ImageBind [Girdhar et al., 2023] [1] expanded upon this by jointly embedding six distinct modalities within one representational space at once. In spite of their state-of-the-art performance, increasing modality coverage tends to result in much larger model sizes.

The inherently modular structure of models such as CLIP and ImageBind opens up promising avenues for optimization. These models rely on separate modality-specific encoders, allowing for targeted improvements such as weight pruning, quantization, and selective activation, without disrupting the entire architecture.

In model compression, Bondarenko et al. [2021] tackled the challenges of efficient transformer quantization [6], a key problem because transformer-based backbones are the foundation for numerous such multimodal models. Their work is to highlight trade-offs between latency and precision, with an emphasis on the difficulty of maintaining performance when compressing the model.

Likewise, Zhu and Gupta [2017] examined pruning methods in their work ”To Prune or Not to Prune” [7], demonstrating that big sparse models can achieve better results than the same-sized small dense models in most tasks. Their incremental pruning method provides an effective means of compressing deep neural networks to be implemented on resource-limited environments, and their results strongly speak in favor of the importance of hardware accelerators specialized in sparse computation.

Taking advantage of pruning, Kwon et al. [2022] proposed a post-training pruning framework for transformers [8] that achieved up to 50% FLOPs reduction and latency savings with negligible loss in accuracy—all in a matter of seconds of computation. This demonstrates the feasibility of quick, training-free model optimization.

A complementary approach is knowledge distillation. Gou et al. [2021] provided a thorough review of this approach [9], in which a smaller ”student” model learns to imitate a larger pretrained ”teacher.” This strategy enables substantial size and inference cost reductions with much of the original performance preserved—a central concern for real-time or edge deployment of multimodal models.

Our work builds upon these prior efforts, providing a strong foundation for exploring optimizations for multimodal embedding models like ImageBind in resource-constrained environments.

III. Methodology and Experimental Setup

We propose a modular architecture for the ImageBind model to reduce its memory footprint during inference, especially when working with a subset of modalities. Our approach decomposes the monolithic architecture into independent submodules per modality, allowing selective loading and inference. In addition to modularization, we explore model quantization techniques, both dynamic and static, to further compress the model and accelerate inference while maintaining reasonable performance across modalities.

A. Modular Decomposition

The original ImageBind model, released as a single monolithic architecture, includes multiple modality pipelines bundled together which is memory inefficient when only a subset of modalities is required. To overcome this limitaion, we split the model into individual submodules, one for each of the supported modalities: vision, text, audio, depth, IMU, and thermal. Each submodule includes its own preprocessor, trunk, head, and postprocessor. The proposed architectural restructuring enables loading only the necessary components at inference, significantly reducing the GPU memory footprint. The resulting sizes of the modular weights are summarized in Table I.

TABLE I
Per-Modality Checkpoint Sizes After Splitting

Modality	Checkpoint Size
Audio	329.018 MB
Depth	83.490 MB
IMU	74.798 MB
Text	1.318 GB
Thermal	328.925 MB
Vision	2.357 GB
Total (Combined)	**4.473 GB**

To facilitate modular execution, we extended the model loading mechanism to accept a list of desired modalities. Thus only the relevant submodules are instantiated and loaded into memory. This effectively enables the model to operate in a reduced configuration without impacting other modality pipelines or the embedding performance.

We validated the memory optimization using several tools and techniques:

- **Python's `memory_profiler`** was employed to monitor peak memory usage during inference with different modality subsets.
- **Manual calculation** of model memory requirements was performed using the parameter and buffer counts per submodule.
- **Runtime diagnostics** were conducted to ensure that unnecessary submodules were not inadvertently loaded.

B. Quantization of ImageBind

Quantization is an effective technique for reducing the memory footprint and computational load of neural networks by transforming floating-point weights and activations into lower-bit representations. This section presents our experimental results on model quantization applied to ImageBind. We explored both dynamic and static quantization approaches using PyTorch's Eager Mode Quantization framework.

As of PyTorch 2.6, there are three quantization methods available: Eager Mode Quantization (beta), FX Graph Mode Quantization (prototype), and PyTorch 2 Export Quantization (prototype). Due to its more mature implementation, we selected Eager Mode for our experiments, focusing specifically on dynamic and static quantization techniques.

The ImageBind architecture incorporates diverse layer types including `Conv2d`, `Conv3d`, `Linear`, `Embedding`, and `MultiheadAttention` layers. Different quantization modes offer varying levels of support for these layers. Dynamic quantization operates at inference time, converting weights to lower precision while keeping activations in full precision. Static quantization quantizes both weights and activations using a calibration process on representative data.

1) Dynamic Quantization: offers an accessible approach that does not require architectural modifications. We applied PyTorch's `quantize_dynamic` method with the following configuration:

- `qconfig_spec`: Targeted `nn.Linear`, `nn.LayerNorm`, `nn.Embedding`, `nn.Dropout`, and `nn.GELU` layers
- `dtype`: `int8` precision for weights

For `Embedding` layers, we set the quantization configuration to `float_qparams_weight_only_qconfig`. We evaluated the quantized model using a combination of representative examples and random inputs across all modalities. Our results demonstrate significant improvements in model efficiency summarized in Table II and Table III shows the cosine similarity between outputs from the original and quantized models across different modalities.

TABLE II
Dynamic Quantization Performance (All Supported Layers)

Metric	Value
Original model size	4581.14 MB
Quantized model size	2246.47 MB
Size reduction	50.96%
Speed improvement	33.20%

TABLE III
Cosine Similarity Between Original and Dynamically Quantized Model Outputs

Modality	Average Cosine Similarity
Vision	0.943
Text	0.906
Audio	0.989
Depth	0.993
Thermal	0.999
IMU	0.999

Since `Linear` layers account for the majority of the model's parameters, we conducted an additional experiment quantizing only these layers. The results were comparable to quantizing all supported layers, as shown in Tables IV and V.

TABLE IV
DYNAMIC QUANTIZATION PERFORMANCE (LINEAR LAYERS ONLY)

Metric	Value
Original model size	4581.14 MB
Quantized model size	2246.47 MB
Size reduction	50.96%
Speed improvement	32.93%

TABLE V
COSINE SIMILARITY FOR LINEAR-ONLY QUANTIZATION

Modality	Average Cosine Similarity
Vision	0.948
Text	0.898
Audio	0.990
Depth	0.993
Thermal	0.999
IMU	0.999

These results demonstrate that dynamic quantization achieves substantial efficiency gains while maintaining output similarity above 90% across all modalities.

2) Static Quantization: offers a more comprehensive optimization by quantizing both weights and activations, resulting in greater size reduction and performance improvements. However, it requires architectural modifications to support the quantization process.

ImageBind's architecture consists of four module types for each modality: preprocessor, trunk, head, and postprocessor. We excluded preprocessors from quantization due to their relatively small parameter count and critical impact on downstream modules. The following modifications were necessary to support static quantization:

1) Addition of `QuantStub` modules at input points to capture and quantize input tokens from each modality
2) Custom handling of the `Normalize` module, which uses `torch.nn.functional.normalize` (not directly quantizable)
3) Integration of `QuantStub` and `DeQuantStub` wrappers around the `DropPath` module from the TIMM library
4) Custom mapping for `MultiheadAttention` modules using `custom_module_config` to their quantizable counterparts
5) Addition of a final `DeQuantStub` to convert output embeddings back to float32

These modifications maintain the original architecture's functionality while enabling quantization support. Our static quantization process followed three key steps:

1) **Preparation**: Attaching observer or fake quantization modules and propagating quantization configurations
2) **Calibration**: Passing representative data through the model for observers to analyze tensor value distributions
3) **Conversion**: Converting modules to their quantized versions and removing observer modules

We configured quantization parameters using PyTorch's `QConfig` with `HistogramObserver` for activations (`quant_max=255`, `quant_min=0`) and `default_per_channel_weight_observer` for weights. Results are presented in Tables VI and VII.

TABLE VI
STATIC QUANTIZATION PERFORMANCE

Metric	Value
Original model size	4581.14 MB
Quantized model size	1316.30 MB
Size reduction	71.27%
Speed improvement	68.96%

TABLE VII
COSINE SIMILARITY BETWEEN ORIGINAL AND STATICALLY QUANTIZED MODEL OUTPUTS

Modality	Average Cosine Similarity
Vision	0.779
Text	0.407
Audio	0.929
Depth	0.979
Thermal	0.985
IMU	0.998

Our full implementation, including training scripts and model configurations, is available at [10] and [11]

IV. ANALYSIS AND DISCUSSION

The modularization experiment demonstrated that decomposing the ImageBind model into modality-specific submodules enabled selective loading which reduced memory usage significantly during inference. The observed memory usage during inference was in line with the expected theoretical values calculated from the sizes of the loaded checkpoints. For example, using only the vision and text modalities resulted in approximately 3764 MB of memory usage, which matched the combined sizes of the corresponding submodules.

Our quantization experiments revealed significant differences between dynamic and static quantization approaches. Dynamic quantization achieved a 50.96% size reduction with 33.20% speed improvement while maintaining high output similarity (>90%) across all modalities. Static quantization delivered superior efficiency gains with 71.27% size reduction and 68.96% speed improvement but exhibits substantial accuracy degradation in text (40.7%) and vision (77.9%) modalities.

Several factors may contribute to the accuracy degradation in static quantization:

- **Calibration data quality**: Our proof-of-concept used random inputs rather than truly representative data, potentially leading to suboptimal quantization parameters.
- **Quantization granularity**: Text and vision modalities may require more nuanced per-layer quantization strategies or selective quantization of critical layers.
- **Architecture sensitivity**: These modalities might incorporate operations particularly sensitive to activation quantization.

Future work should explore improved calibration techniques with domain-specific data, per-layer quantization configurations, and quantization-aware training to mitigate accuracy loss.

V. Conclusion

We proposed and evaluated two optimization techniques for the ImageBind model: modularization and quantization. Our modularization approach successfully reduced memory overhead by allowing selective loading of sub-modules based on required modalities. Memory usage during inference was consistent with theoretical expectations,thereby confirming the feasibility of fine-grained modular deployment.

Quantization experiments also reduced resource requirements. Dynamic quantization provided a good balance between model size, speed, and accuracy, making it a practical solution for real-world applications. Although static quantization offered more substantial efficiency improvements, it introduced unacceptable performance degradation in certain modalities. Addressing these issues through better calibration and quantization-aware training remains a promising avenue for future research.

Together, these results highlight the potential of modular, quantized multi-modal embedding models for efficient, inference in resource-constrained settings.

References

[1] R. Girdhar, A. El-Nouby, Z. Liu, M. Singh, K. V. Alwala, A. Joulin, and I. Misra, "Imagebind: One embedding space to bind them all," 2023. [Online]. Available: https://arxiv.org/abs/2305.05665

[2] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[3] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y. Sung, Z. Li, and T. Duerig, "Scaling up visual and vision-language representation learning with noisy text supervision," 2021. [Online]. Available: https://arxiv.org/abs/2102.05918

[4] L. Yuan, D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li, C. Liu, M. Liu, Z. Liu, Y. Lu, Y. Shi, L. Wang, J. Wang, B. Xiao, Z. Xiao, J. Yang, M. Zeng, L. Zhou, and P. Zhang, "Florence: A new foundation model for computer vision," 2021. [Online]. Available: https://arxiv.org/abs/2111.11432

[5] A. Guzhov, F. Raue, J. Hees, and A. Dengel, "Audioclip: Extending clip to image, text and audio," 2021. [Online]. Available: https://arxiv.org/abs/2106.13043

[6] Y. Bondarenko, M. Nagel, and T. Blankevoort, "Understanding and overcoming the challenges of efficient transformer quantization," 2021. [Online]. Available: https://arxiv.org/abs/2109.12948

[7] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," 2017. [Online]. Available: https://arxiv.org/abs/1710.01878

[8] W. Kwon, S. Kim, M. W. Mahoney, J. Hassoun, K. Keutzer, and A. Gholami, "A fast post-training pruning framework for transformers," 2022. [Online]. Available: https://arxiv.org/abs/2204.09656

[9] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, p. 1789–1819, Mar. 2021. [Online]. Available: http://dx.doi.org/10.1007/s11263-021-01453-z

[10] A. Saed, "Modular imagebind - load modules based on selected modailites," https://github.com/facebookresearch/ImageBind/pull/138, 2025, contribution to the open-source project *ImageBind* on GitHub.

[11] ——, "Quantized imagebind - applying dynamic and static quantization of imagebind," https://github.com/ahmedsaed/ImageBind/tree/quantization, 2025, contribution to the open-source project *ImageBind* on GitHub.