Practical Visual Question Answering at the Edge

Gloria Giorgetti[1], Simone Bianco[2], Pietro Firpo[3], Riccardo Berta[4], and Danilo Pau[5]

[1] STMicroelectronics, Agrate Brianza 20864, Italy gloria.giorgetti@gmail.com
[2] DISCO, Università di Milano-Bicocca, Italy simone.bianco@unimib.it
[3] DITEN, University of Genoa, Genova, Italy pietro.firpo@edu.unige.it
[4] DITEN, University of Genoa, Genova, Italy riccardo.berta@unige.it
[5] STMicroelectronics, Agrate Brianza 20864, Italy danilo.pau@st.com

Abstract. Edge devices require AI models that are carefully tailored to their technical specifications and resource constraints. Deploying multimodal or generative models at the edge is particularly challenging due to their high demands in terms of memory and processing power. Nevertheless, the advantages of on-device inference continue to drive interest in Edge AI at large and, specifically, in Generative EdgeAI (Gen EdgeAI) solutions. In particular, the latter is emerging as a key enabler of natural and conversational human–machine interactions in future scenarios, such as humanoid robotics. This paper presents a practical methodology for developing lightweight multi-modal Visual Question Answering (VQA) models for edge deployment on the STM32MP2 low power platform. The full workflow is outlined, with a focus on architectural design choices and training strategies. In addition, a proof-of-concept end-to-end generative system is presented. This system integrates VQA models with foundational models for language generation, speech-to-text, and text-to-speech, enabling users to interact through visual content, natural language and spoken dialogue. This work aims to provide a practical and accessible guide for students and young practitioners interested in Gen EdgeAI. By focusing on intermediate models such as VQA workloads, it demonstrates feasible lightweight AI deployment at the edge and the potential for extension to generative applications. The developed code, trained models, and step by step instructions are available on GitHub, offering a practical reference for easy experimentation and adoption.

Keywords: Tutorial · Edge Devices · Generative AI models · Quantization · Knowledge Distillation · Micro-processors · Low power processing.

1 Introduction

Latest advancements of Artificial Intelligence (AI) are proving superior performance. This often involves increasing the size and parameters of models. However, this trend line encounters significant limitations when the models shall be deployed on edge devices, which are known to be severely constrained by finite

amount of resources such as computational power, memory, and energy. Consequently, designing AI workloads for edge devices requires to carefully consider the device's specifications and trading resource utilization with model performance. Despite the inherent constraints of edge devices, their usage into AI applications presents numerous compelling advantages, including enhanced privacy, independence from internet connectivity, scalability, reduced latency, and real-time data processing capabilities. Moreover, certain applications, such as generative AI systems, are proposed to enable more natural, intuitive and conversational human–machine interactions. EdgeAI is known to be focused on the development of AI models to be deployed on edge devices, such as smartphones, IoT devices, near-user computers (NUCs), multi processors (MPUs), neural processing units (NPUs) and microcontrollers (MCUs). In recent years, there has also been a surge of interest in a particular niche of EdgeAI called Generative EdgeAI (Gen EdgeAI), which focuses on deploying Generative AI models on such devices. However, deploying generative or multi-modal models at the edge is a significant challenge due to the very high amount of computational and memory resources these models require [1]. Furthermore, there is still a very little established practice, within the EdgeAI community, about designing and integrating GenAI models into end-to-end systems for application-specific cases. This may be due to the long-standing focus on more traditional fixed AI, TinyML solutions, such as those based on convolutional or recurrent neural networks, for which ample literature and zoo's models are widely available as resources on internet. All of that and the scarcity of practical guidance on Gen EdgeAI motivate the contribution subject of this paper. Here, these challenges are addressed through a practical tutorial for building lightweight multi-modal Visual Question Answering (VQA) models specifically optimized for the STM32MP2 MPU platform [6], ensuring efficiency and compatibility with the on device's NPU acceleration. In addition, the feasibility of integrating these models with foundational generative and speech-processing models is explored, resulting in a proof-of-concept end-to-end system that allows users to pose spoken questions and receive natural-language conversational answers on resource-constrained devices. The remainder of the paper is organized as follows. Section 2 presents the main contributions in more detail. Section 3 reviews related work on VQA, including the evolution of architectures and representative examples of deployment, and discusses the foundational generative models integrated into the proposed pipeline. Section 4 presents the proposed end-to-end generative system and its main components. Section 5 outlines the design and training of the lightweight VQA models for the STM32MP2 MPU, and Section 6 reports the performances. Section 7 analyzes the deployability of the complete pipeline on the STM32MP2 MPU. Finally, Section 8 summarizes the contributions and provides concluding remarks.

[6] https://www.st.com/en/microcontrollers-microprocessors/stm32mp2-series.html

2 Contributions of this paper

The main contributions of this work include: (i) a tutorial for developing and training lightweight VQA models optimized for STM32MP2 edge devices, and (ii) a proof-of-concept generative pipeline that combines the proposed VQA models with language, speech-to-text (STT), and text-to-speech (TTS) edge oriented models to enable spoken question–answer interactions on embedded hardware. This manuscript is intended as a practical tutorial for newcomers to Edge AI interested in Gen EdgeAI —including students, early-stage researchers, young professionals and developers—who are looking to a practical approach to begin with. The focus is on building lightweight multi-modal VQA models optimized for the STM32MP2 MPU, leveraging its integrated NPU for efficient execution. One objective is not targeting (above) state-of-the-art performance. While the tutorial offers a practical educational example that could be trained on commercially available hardware such as NUC and GPUs. Readers are guided through model design, training, and deployment onto edge devices, to the architecture, multi-modal fusion, and knowledge distillation (KD). The code, trained models, and usage instructions are publicly released on GitHub, providing a practical entry toward advanced Edge AI concepts and a bridge toward Gen EdgeAI. Moreover, as an exploratory attempt, the paper also presents a proof-of-concept end-to-end generative system that combines the lightweight VQA models with foundation models: a language model for text generation, STT, and TTS. This whole pipeline allows users to pose spoken questions helped by visual content and receive natural-language conversational answers: This will showcase multi-modal understanding, aided by reasoning skills, with natural human interaction to edge devices. Although not fine-tuned or evaluated for accuracy, the whole system is built with edge-optimized lightweight models and evaluated on the STM32MP2 MPU. Metrics on latency, energy, and memory usage on STM32MP2 demonstrate the feasibility of deploying such pipelines in resource-constrained environments, pointing to a path for extending lightweight VQA toward interactive generative applications on embedded hardware. And it also inspires a development path toward future performance enhancements.

3 Related Work

This section reviews prior work that framed and informed about the development of the lightweight VQA models presented in this paper, as well as the foundational workloads that were incorporated into the proposed generative pipeline. The review is about three main areas: the evolution of VQA architectures, examples of VQA models deployed on resource-constrained devices, and foundational models integrated into the proof-of-concept system.

3.1 Visual Question Answering

VQA consists of answering questions posed in natural language by analyzing visual content, typically images. This task has applications in several domains, in-

cluding assistive technologies for visually impaired users and automated surveillance systems. Early VQA models combined CNNs for image feature extraction with RNNs such as long short-term memory (LSTM) units for question encoding, resulting in fixed-length representations for both modalities. These features were then fused, usually through concatenation or element-wise operations. The fused representation was passed to a classifier that selected the most likely answer from a predefined set, typically composed of the most common answers in the dataset [2]. However, these early architectures were limited in their ability to model complex interactions between visual and textual inputs. To overcome this, researchers introduced bilinear pooling techniques—more expressive fusion strategies which better captured the interactions between images and questions. In parallel, attention mechanisms were adopted to enable models to focus on salient image regions based on the given question, rather than treating the entire image uniformly [3]. Yu et al. [5] addressed these limitations by introducing the Multi-modal Factorized Bilinear (MFB) pooling module to fuse features more effectively. Based on this module, they proposed several architectures—MFB Baseline, MFB with Attention, and MFB with Co-Attention—which explore different strategies for integrating attention with bilinear pooling. The models proposed by this tutorial are based on these architectures and further developed. More recent methods leverage transformer architectures with self-attention mechanisms, and multi-modal pre-training [4]. These models achieved excellent performance but are unsuitable for most edge devices due to their large model size, high computational demands, and resource-intensive nature. For this reason, despite achieving state-of-the-art accuracy, they were not considered by this tutorial.

3.2 VQA on Edge Devices

Deploying VQA models on resource-constrained devices requires to achieve carefully a trade-offs between accuracy, computational cost, and memory usage. Several prior works provided valuable guidance on the different strategies that can be employed to address these challenges. Cao et al. [7] proposed MobiVQA, a set of optimizations that preserve accuracy while significantly reducing latency on devices such as Nvidia Jetson TX2 and Pixel 3XL. Yu et al. [8] developed the Bilaterally Slimmable Transformer (BST) framework, that enables a single Transformer-based model to be pruned into efficient sub-models tailored for diverse edge devices. Mishra et al. [10] proposed a novel Transformer architecture with advanced post-training quantization and deployed it on Samsung Galaxy S23. Rashid et al. [6] designed TinyVQA, a compact network optimized through knowledge distillation and quantization, achieving low latency and power use on a GAP8 processor. These works not only focus on device-specific optimization but also illustrate techniques such as pruning, quantization, and KD, which guided the design of the VQA models for the STM32MP2 MPU in this paper. Unlike these prior studies, this work provides a practical tutorial approach with code, enabling readers to implement and train lightweight multi-modal VQA models and explore their integration into a proof-of-concept generative pipeline.

3.3 Foundational Models for conversational Generative Systems

Foundational models—such as LMs, STT, and TTS—can be building blocks for generative pipelines, providing reusable representations pre-trained on large-scale datasets that support a wide range of downstream tasks. In this work, leveraging the capabilities of foundational models, a generative pipeline is composed by integrating the lightweight VQA model from the tutorial. The pipeline processes multi-modal input—specifically, spoken questions paired with images—and generates coherent natural-language responses using the VQA model, a STT model for query transcription, an LM for answer generation, and a TTS model for audio synthesis. The full system architecture is detailed in Section 4. For the STT component, Moonshine [15] was adopted. It is a family of transformer-based encoder-decoder architectures optimized for real-time operation on resource-constrained devices. By using Rotary Position Embeddings and processing variable-length audio segments without zero-padding, it minimizes computational overhead and delivers low-latency transcription, making it well suited for embedded and edge environments. Among many foundational models, Gemma 3 270M [17] was preferred, even if others can be possibly used. This compact decoder-only transformer model was pre-trained using knowledge distillation, similar to other models in the Gemma 3 series. With only 270 million parameters, it provides a balance between computational efficiency, power consumption and performance, making it a convenient resource-friendly option in the series and suitable for edge deployment. Finally, Piper [16] serves as the TTS. Piper is an open-source neural speech synthesis system designed for efficient inference on constrained hardware. It generates real-time audio while maintaining a low computational and memory footprint, converting textual input into speech with minimal overhead. Its efficiency makes it well suited for integration into edge-oriented pipelines that require natural audio output. Overall, all these components provide the building blocks for the end to end generative workload, enabling multi-modal reasoning, natural language generation, and speech synthesis deployable on STM32MP2 MPU.

4 System Overview

The proposed system is designed as an end-to-end pipeline that integrates speech, vision, and language models to enable generative capabilities on a resource-constrained edge device. Its architecture consists of four sequential components: STT, VQA, language modeling from foundational zoos, and TTS. Integrated, these components provide an end to end multi-modal interaction loop, from spoken input to generated verbal response. The STT component is implemented using Moonshine [15], which transcribes the audio query into text. The transcribed question is then paired with the corresponding image and processed by the lightweight VQA model devised by this work, responsible for multi-modal reasoning. The resulting intermediate representation fed Gemma 3 (270M) [17], a compact generative LM that produces coherent textual responses. The Piper

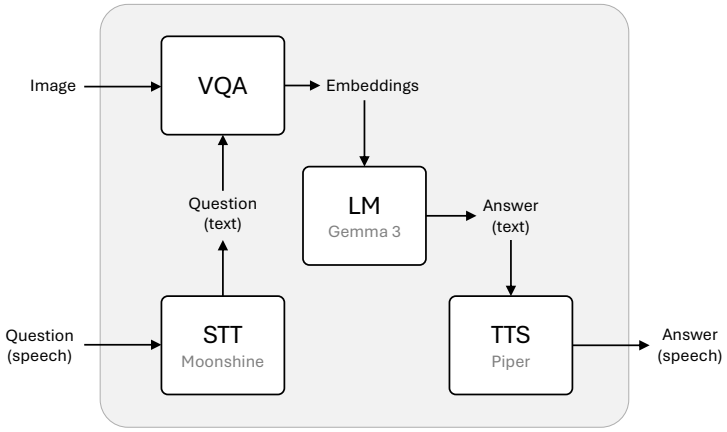

Fig. 1: Block diagram of the proof-of-concept generative system, illustrating the data flow across four components: speech-to-text (STT, Moonshine), visual question answering (VQA), language modeling (LM, Gemma 3), and text-to-speech (TTS, Piper). diagram illustrates a four-stage pipeline. The inputs are a spoken question and an image. The spoken question is transcribed into text by the speech-to-text (Moonshine) module. This text, together with the image, is processed by the visual question answering (VQA) module to generate an answer representation. The output is passed to the language model (Gemma 3), which produces a natural-language textual response. Finally, the text-to-speech (Piper) module converts this response into speech, providing the spoken answer as the system's output.

[16] module synthesizes natural-sounding speech from the generated text, completing the pipeline.

Figure 1 illustrates the block diagram of the system, showing the four components and their connections. In Section 7, the deployability of this complete pipeline on the STM32MP2 MPU is analyzed in detail.

5 VQA Model Development

This section presents the methodology, the design choices, the architectures, and the training strategies adopted for devising a VQA workload on edge devices. it clarifies what has been implemented and why. Practical resources—including python code, training and evaluation scripts, and configuration files—are provided in the associated GitHub repository, so that paper and repository together compose a comprehensive tutorial.

5.1 Models Architecture Design

The VQA models developed in this work are based on the neural network architectures proposed by Yu et al [5]: MFB Baseline, MFB with Attention, and MFB with CoAttention. These architectures were chosen because they offer good performance while keeping fusion and attention mechanisms relatively simple and suitable for deployment on STM32MP2 MPU. While effective in their original application context, they are not inherently optimized for efficient execution or hardware acceleration such as the STM32MP2 MPU. Therefore this work devised some focused modifications that preserved the overall architecture while replacing specific neural layers with more efficient alternatives, enabling effective inference on the edge device. In the original MFB Baseline network, image features are extracted using a ResNet-152 pre-trained on ImageNet, while textual features are obtained via an embedding layer followed by a two-layer LSTM recurrent network. These features are fused using the MFB module (shown Figure 2), and the resulting representation feeds a dense layer that predicts the answer. The MFB with Co-Attention variant extends this design by incorporating attention mechanisms. Question features are extracted in the same way as in the MFB Baseline, followed by the application of an attention mechanism to obtain question attention features. Image features are extracted using ResNet-152 pre-trained, fused with the question attention features via the MFB module, and then processed by a second attention mechanism to generate image attention features. These image and question attention features are then fused using another MFB module. Finally, the output is passed to a dense layer to predict the answer. The MFB with Attention variant is similar while omits the question attention block. To deploy efficiently these architectures on the STM32MP2 MPU and leverage its integrated NPU (designed to accelerate CNNs), several key modifications were introduced into the original design. ResNet-152 was replaced with MobileNet V3 Large, pre-trained on ImageNet, which is more suitable for resource-constrained devices. The LSTM layers for textual feature extraction were removed and blocks of temporal convolutional layers (TCLs) introduced, as shown in Figure 4. This to enable faster and hardware-friendly sequence processing. Additionally, the MFB module was simplified by removing power normalization and replacing the sum pooling with average pooling layer, while retaining the remaining structure. Word embeddings learned during training were concatenated with pretrained GloVe embeddings to enhance semantic representation. The MFB module hyperparameters were set to $k = 5$ and $o = 1024$, as they control the internal factorization and output dimensions. Figure 3 illustrates the optimized MFB with Co-Attention network. The MFB with Attention and Baseline variants can be derived by removing the question attention block, or both the question and image attention blocks along with the first MFB module, respectively. Figure 3 shows the full MFB with Co-Attention network. The image is first processed by MobileNet to extract a feature representation. The question is encoded using a learned embedding concatenated with GloVe embeddings, which is then processed by two Temporal Convolutional (TC) blocks. These image and question features are combined through MFB fusion modules

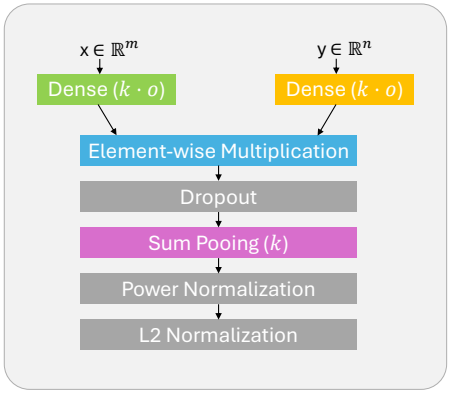

Fig. 2: Architecture of the Multi-modal Factorized Bilinear (MFB) module. k and o are hyperparameters of the module. It shows two input feature vectors which are first projected with dense layers, combined through element-wise multiplication and pooling, and then normalized to produce a joint representation. The hyperparameters k and o control the internal factorization and output dimensions.

and attention mechanisms, before being passed to a classifier that predicts the answer. The Attention and Baseline variants can be derived by removing the question attention block or both attention blocks and the first MFB module. Figure 4 shows the architecture of the TC block, which replaces LSTM layers and consists of stacked temporal convolutions.

5.2 Dataset and Data Processing

The dataset adopted by this tutorial is named VQAv2 [11][13], which consists of images, questions (approximately five per image), ten answers provided by different human annotators, and a single ground truth answer. Each question is also annotated with its question and answer type. The dataset is sub divided into training, validation, and test sets. As ground truth answers are unavailable for the test set, the training was performed on the training set and evaluation on the validation set. Table 1 summarizes the number of images and questions in each sub-division. All answers were preprocessed following the official guidelines—lowercased, number words converted to digits, articles removed, and other standard normalization steps. From over 20,000 processed answers in the training set, the 1,000 most frequent were retained along with their corresponding questions, reducing the training set to 87.5% of its original size. Models were trained on this filtered set to predict one of these 1,000 answers. Limiting the model to a predefined subset of possible answers is a standard approach in the VQA task. Regarding data processing, images were resized to 224×224 pixels and rescaled to values between –1 and 1, as required by MobileNet's input preprocessing. Questions were tokenized and either padded or truncated to ensure all

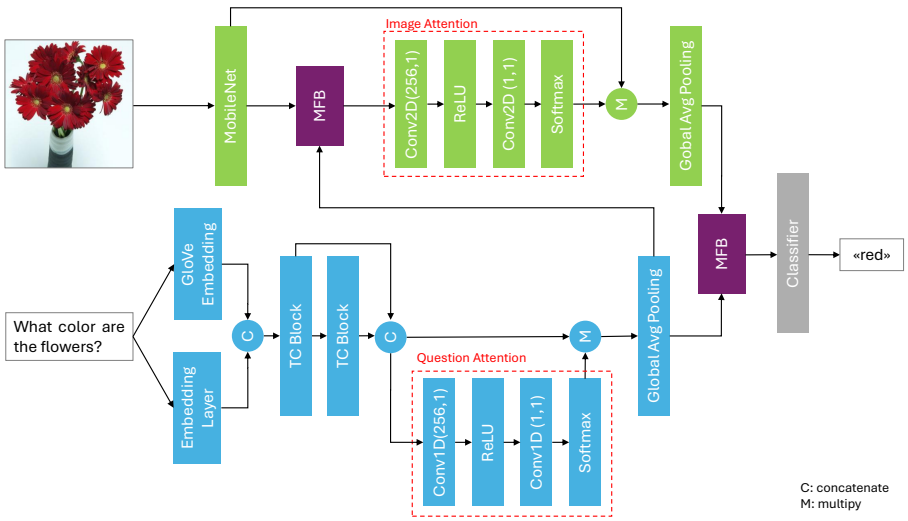

Fig. 3: MFB with CoAttention Network. Architecture modified for STM32MP2 MPU deployment. *(a)* The MFB with Co-Attention network used by this tutorial. The MFB with Attention and Baseline variants are derived by removing attention blocks and MFB modules.

Table 1: Distribution of images and questions across the VQAv2 dataset subdivision.

Set	**Questions**	**Images**
Training	443,757	82,783
Validation	214,354	40,504
Test	447,793	81,434

sequences had a fixed length of 15 tokens. Note that less than 0.2% of sequences were originally longer than 15 tokens. For the tokenizer vocabulary, only words that appeared at least 5 times in the complete training set were included, resulting in a vocabulary of 6,415 words. Words not in the vocabulary were replaced with the special "<unk>" token.

5.3 Training Procedure

Following early experiments and to achieve higher performance, KD was employed instead of training the models from scratch. This technique involves using a large, pre-trained teacher model with established high performance to transfer knowledge to a smaller student model. Among the various KD techniques, the method proposed by Hinton et al. [12] was the one adopted in this tutorial. This method relies exclusively on the teacher model's output logits, without considering its intermediate representations. When training a VQA model from scratch,

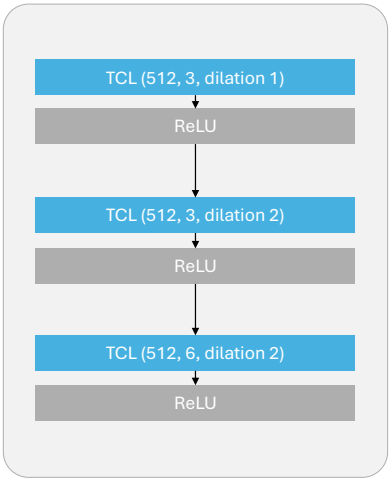

Fig. 4: Temporal convolutional (TC) block, composed of stacked temporal convolution layers (TCLs), replacing LSTM layers for improved efficiency on STM32 MPU.

the cross-entropy loss is typically employed, as the model functions as a classifier over the set of possible answers. For training with KD, the loss $\mathcal{L}$ function consists of two components:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{student}} + (1 - \alpha) \cdot \mathcal{L}_{\text{distillation}}, \tag{1}$$

$$\mathcal{L}_{\text{student}} = \text{CE}\big(y, \text{Softmax}(s)\big), \tag{2}$$

$$\mathcal{L}_{\text{distillation}} = T^2 \cdot \text{KL}\Big(\text{Softmax}\big(\frac{t}{T}\big) \parallel \text{Softmax}\big(\frac{s}{T}\big)\Big), \tag{3}$$

where s and t represent the student and teacher logits, respectively, and y denotes the ground truth answer. Here, KL is the Kullback-Leibler divergence, while CE is the cross-entropy loss; thus $\mathcal{L}_{\text{student}}$ corresponds to the cross-entropy loss used in training from scratch. The temperature parameter T produces a softened probability distribution over classes, while α balances the contributions of the two losses in the final loss function. Typically, α is chosen so that the contribution of $\mathcal{L}_{\text{student}}$ is smaller. For all experiments reported in the tutorial, the temperature parameter T was set to 3, while the balancing coefficient α was set to 0.1. The teacher model used is BEiT-3 fine-tuned on VQAv2 [4][14], which achieves an accuracy of 82.53% over the test-dev split. It features 683 M parameters and predicts among 3,129 answers, though for this tutorial logits outside the reduced answer set were discarded. Due to the dataset imbalance—where yes/no represent $\approx$ 40% of answers—loss weights were set inversely proportional to answer

Table 2: Performance comparison on VQAv2 validation set. Accuracy is reported for all, yes/no (Y/N), number (Num), and other answer types. The table also shows the number of parameters (#Par.) for each model.

Model	**#Par.**	**Accuracy (%)**			
(lr)3-6		All	Y/N	Num	Other
MFB (original) [18]	68.0M	65.4	83.2	45.3	57.1
MFB base. (this work)	24.4M	56.0	76.7	36.5	45.4
MFB att. (this work)	40.4M	**57.0**	**77.7**	**37.2**	**46.5**
MFB co-att. (this work)	51.2M	56.4	76.9	36.8	46.1

Table 3: Average accuracy of MFB models, computed by ranking the 1000 possible answers by their per-answer accuracy. The columns report average accuracy for the top 10 highest-ranked answers, answers ranked 11-50, and answers ranked 51-1000.

Model	**Top-10 (%)**	**11-50 (%)**	**51-1000 (%)**
MFB base.	91.1	86.2	62.0
MFB att.	92.1	85.1	63.1
MFB co-att.	90.8	84.7	62.5

frequency, preventing overfitting to dominant classes. Training was performed on an NVIDIA GTX 1060 (6 GB) using Adam with learning rate 10^{-4}. Each model was trained for 10 epochs ($\approx$ 1 hour per epoch), totaling approximately 10 hours per model.

6 Results

The VQA model evaluation method was proposed in the original VQA paper [2]. Given the model's predicted answer a_i for question i, and the ten human-provided reference answers, the accuracy of a model is computed as follows:

$$accuracy = \frac{1}{N} \sum_i min\left(\frac{count(a_i)}{3}, 1\right) \tag{4}$$

where N is the number of questions and $count(a_i)$ denotes the number of annotators who provided the answer a_i to question i.

Table 2 summarizes the results on the validation set for the three models developed in this tutorial, together with the original model from which the modified architectures were derived. The reported metrics include overall accuracy as well as accuracy by answer type (yes/no, number, other), in addition to the number of parameters. Among the models developed in this tutorial, the MFB with attention (MFB att.) achieves the best overall accuracy (57.0%) as well as the highest accuracy across all answer types, outperforming both the baseline and co-attention variants. Consequently, this was the model selected for integration

Table 4: Performance comparison of the models composing the generative pipeline, evaluated on the STM32MP2 device. The table reports runtime measures (RTF).

Model	Framework	Execution Device	Runtime Measure
Moonshine [15]	onnx runtime	CPU	RTF: 1.50
MFB att.	this work	GPU/NPU	Inference time (ms): 56
Gemma 3 270M [17]	ollama	CPU	Throughput (tok./s): 4.07
Piper [16]	onnx runtime	CPU	RTF: 1.84

Table 5: Efficiency comparison of the models composing the generative pipeline, evaluated on the STM32MP2 device. The table reports power consumption, and memory footprint. CPU inference time for MFB att.: 434 ms.

Model	Power (W)	Flash (MB)	RAM (MB)
Moonshine [15]	0.89	245	1101
MFB att.	0.75	48	51
Gemma 3 270M [17]	1.00	292	789
Piper [16]	0.81	61	356

into the proof-of-concept generative pipeline described in Section 4. Table 3 provides a detailed breakdown of model average accuracy by answer rank ranges, showing performance for the top 10 highest-ranked answers, answers ranked 11-50, and the remaining answers up to rank 1000. All the developed models have got fewer parameters than the original MFB model (68 M), which reduces memory footprint and computational requirements and also contribute to lower accuracy (original MFB: 65.4%). The teacher model used for KD, by contrast, achieves 87.5% accuracy on the test-dev set and features 683 M parameters — exceeding by an order of magnitude both the original model and the student models. The results shown in Table 2 reflect this work design choices. As expected, the models do not achieve performances as high as the teacher, which is also consistent with the simplified training procedure and basic data preprocessing employed. Furthermore, the model was selected for deployability and ease of use, and further adapted to run efficiently on the STM32MP2 device. These results also illustrate the trade-offs inherent in Edge AI, where model performance shall be traded against the computational and memory constraints of the target hardware. Overall, they serve as a practical example for readers which aims to learn to develop models suitable for resource-constrained devices.

7 Edge Deployments: Analysis

As shown in figure 1, the proposed end-to-end generative system combines the Moonshine STT model [15], the lightweight VQA model developed by this work, the Gemma 3 270M LM for text generation [17], and the Piper TTS model [16]. Among the VQA variants developed, the MFB with attention model was

privileged, as it achieved the highest performance. Only the VQA models described in 5 were devised to leverage the STM32MP2's (embedded) NPU for the workload acceleration, thereby reducing inference time, CPU load, and power consumption. On the other hand, the remaining models (STT, LM, TTS) were executed on the CPU due to lack of acceleration at the time of writing. The VQA model required 434 ms per inference if executed by the STM32MP2 CPU, compared to 56 ms by its NPU, corresponding to a 7.8× speed-up. The pipeline was deployed on the STM32MP2 MPU, and a deployability analysis was performed to assess the feasibility of running each model under hardware constraints. Three key metrics were recorded for each model: runtime, average power consumption in watts, and memory footprint (FLASH and RAM) in megabytes. Runtime was measured differently depending on the type of model: for streaming models, it was recorded as real-time factor (RTF), defined as the ratio of processing time to audio duration, where RTF < 1 indicates faster-than-real-time processing; for single-shot tasks, it was measured as inference time in milliseconds; and for the LM, it was measured as throughput in tokens per second. While the Gemma 3 270M LM achieves up to 4.07 tokens per second, alternative small to medium-scale LMs—such as Phi 3 or 4, Qwen 0.5B or 0.6B, and Llama 3.2 or Granite 3.3—can also be employed, with throughput ranging approximately from 0.4 to 2.4 tokens per second on STM32MPU.

Table 5 reports the results for the individual models in the pipeline. These measurements provide insight into the trade-offs between runtime, energy, and memory usage. While the models' performance is inherently limited by the hardware, the results demonstrate that lightweight and edge-optimized models can be integrated to form a functional multi-modal system within the available resource envelope. This proof-of-concept is intended to highlight the practical feasibility of deploying generative and multi-modal AI models on embedded platforms. It provides a foundation for further exploration of Gen EdgeAI, where future work may focus on enhancing model quality and end-to-end system capabilities while trying to increase level of acceleration by using the NPU of the CPU's workloads.

8 Conclusions

This paper presented a practical methodology for developing lightweight multi-modal VQA models tailored for edge devices. By guiding readers through the design and training process, the work provides an easy entry point into Gen EdgeAI useful for everyone. The proposed approach prioritizes feasibility and reproducibility over state-of-the-art performance, offering models that can be trained on premises hardware and deployed on constrained platforms. Practical resources, including code, scripts, and configuration files, are provided in the accompanying GitHub repository to support hands-on implementation. In addition to the tutorial, an exploratory end-to-end pipeline has been introduced, combining lightweight VQA models with foundational generative and speech-processing models. This pipeline demonstrates the potential of enabling multi-modal and conversational interactions directly on edge devices. The evaluation

on the STM32MP2 demonstrates the deployability of this approach in terms of latency and energy efficiency. Future directions include further optimization of accuracy and efficiency, fine-tuning generative components for edge scenarios, increasing speedup by using NPU for language models, and exploring consumer and automotive applications.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. G. Giorgetti and D. P. Pau, "Transitioning from TinyML to edge GenAI: a review," *Big Data and Cognitive Computing*, vol. 9, no. 3, p. 61, 2025.
2. S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "Vqa: Visual question answering," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2425–2433.
3. A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach, "Multimodal compact bilinear pooling for visual question answering and visual grounding," *arXiv preprint arXiv:1606.01847*, 2016.
4. W. Wang, H. Bao, L. Dong, J. Bjorck, Z. Peng, Q. Liu, K. Aggarwal, O. K. Mohammed, S. Singhal, S. Som, and F. Wei, "Image as a foreign language: BEiT pretraining for vision and vision-language tasks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.
5. Z. Yu, J. Yu, J. Fan, and D. Tao, "Multi-modal factorized bilinear pooling with co-attention learning for visual question answering," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1821–1830.
6. H.-A. Rashid, A. Sarkar, A. Gangopadhyay, M. Rahnemoonfar, and T. Mohsenin, "TinyVQA: Compact Multimodal Deep Neural Network for Visual Question Answering on Resource-Constrained Devices," *arXiv preprint arXiv:2404.03574*, 2024.
7. Q. Cao, P. Khanna, N. D. Lane, and A. Balasubramanian, "Mobivqa: Efficient on-device visual question answering," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1–23, 2022.
8. Z. Yu, Z. Jin, J. Yu, M. Xu, H. Wang, and J. Fan, "Bilaterally Slimmable Transformer for Elastic and Efficient Visual Question Answering," *IEEE Transactions on Multimedia*, vol. 25, pp. 9543–9556, 2023.
9. Z. Yu, J. Yu, Y. Cui, D. Tao, and Q. Tian, "Deep modular co-attention networks for visual question answering," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 6281–6290.
10. A. Mishra, A. Agarwala, U. Tiwari, V. N. Rajendiran, and S. S. Miriyala, "Efficient Visual Question Answering on Embedded Devices: Cross-Modality Attention With Evolutionary Quantization," in *2024 IEEE International Conference on Image Processing (ICIP)*, 2024, pp. 2142–2148.
11. Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, "Making the v in vqa matter: Elevating the role of image understanding in visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6904–6913.
12. G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

13. "VQA: Visual Question Answering," https://visualqa.org/, Accessed on 23 July 2025.
14. "(BEiT-3) Image as a Foreign Language: BEiT Pretraining for Vision and Vision-Language Tasks," https://github.com/microsoft/unilm/tree/master/beit3, Accessed on 23 July 2025.
15. N. Jeffries, E. King, M. Kudlur, G. Nicholson, J. Wang, and P. Warden, "Moonshine: Speech recognition for live transcription and voice commands," *arXiv preprint arXiv:2410.15608*, 2024.
16. OHF-Voice contributors, "piper1-gpl: Fast and Local Neural Text-to-Speech Engine," https://github.com/OHF-Voice/piper1-gpl, Accessed on 26 August 2025.
17. Gemma Team, "Gemma 3," *arXiv preprint arXiv:2503.19786*, 2025.
18. Z. Yu and Y. Cui, "OpenVQA Benchmark and Model Zoo," https://openvqa.readthedocs.io/en/latest/basic/model_zoo.html, Accessed on 26 September 2025.