# Energy-Efficient Breast VSI Inference on Low-Cost Devices

Final Report - CSC 420

Emilio J. Ochoa Alva

## Abstract

Breast Volume Sweep Imaging (VSI-B) provides an affordable, operator-independent approach for breast cancer screening in low-resource settings. However, the current VSI-B automated diagnostic pipeline is computationally demanding, requiring ten 3D segmentation models and five 2D classification models per patient. On consumer-grade laptops, inference can take up to 16 minutes per patient and drain over 70% of the device battery, limiting real-world utility. This report presents a comprehensive optimization of the VSI-B inference workflow using ONNX Runtime, Intel OpenVINO, vectorized preprocessing, batch-level parallelization, and multithreaded execution. Experiments across two low-cost laptop platforms demonstrate a 5.67× speedup and substantial reductions in energy consumption. We detail implementation strategies, algorithmic improvements, failure cases (e.g., quantization collapse, GPU memory saturation), and deployment guidelines for scalable low-resource breast cancer screening.

## 1 Introduction

Volume Sweep Imaging (VSI) is a standardized ultrasound acquisition method designed for non-specialist health workers to collect diagnostic-quality scans in low-resource environments [1, 2]. Unlike conventional ultrasound, which requires a trained sonographer to interpret real-time imaging, VSI uses fixed, predefined probe sweeps over anatomical regions, capturing short video clips ("cine sweeps") that can later be interpreted by experts or automated AI systems.

The Breast VSI (VSI-B) protocol specifically enables the early detection of palpable breast lesions [2, 3]. such as cysts, fibroadenomas, and carcinomas without requiring specialized operators or high-end equipment. A non-physician health worker can perform the exam using a handheld device like the Butterfly iQ+, connected to a tablet, and upload the videos to an AI-based analysis system. The AI model then performs segmentation using architectures such as Attention U-Net 3D [7, 6] and classification with DenseNet-style backbones [8] to output a diagnostic recommendation (no mass, possibly benign, possibly malignant).

This paradigm addresses a major healthcare challenge: limited access to radiologists and imaging infrastructure in low-resource regions. VSI-B, is portable, inexpensive, and telemedicine-compatible, enabling community-level screening programs.

The efficiency bottleneck arises in the post-acquisition stage particularly during 3D segmentation, post-processing, and ensemble classification steps which can take up to 16 minutes per patient on consumer laptops or tablets. This inefficiency limits real-world usability for battery-powered devices. Therefore, the optimization project aims to overcome these bottlenecks by:

- Replacing PyTorch inference with optimized ONNX and Intel OpenVINO runtimes.
- Reducing redundant processing (loop-based frame selection, ensemble overhead).
- Quantifying not only inference time but also energy usage per patient, a metric rarely reported in AI-for-health literature.

The ultimate goal is to make VSI-B not just diagnostically reliable but also computationally sustainable and deployable in field conditions, where battery life, portability, and real-time feedback are essential for accessible breast cancer screening.

**Objective:** Minimize the runtime and energy consumption of the VSI inference workflow on commodity hardware while maintaining segmentation and classification accuracy within $\leq 1\%$ absolute of baseline values.

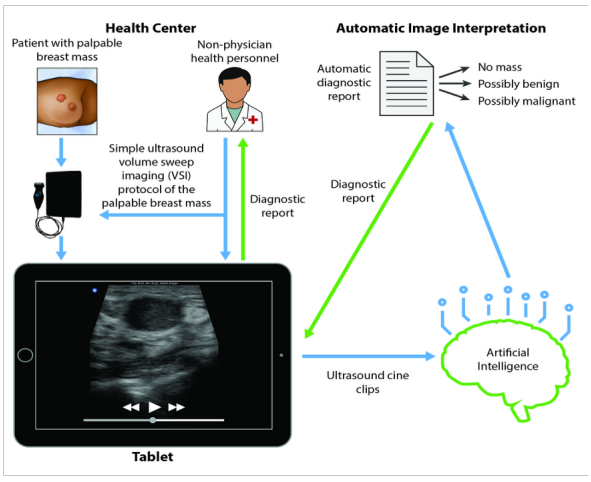


Figure 1: Overview of the artificial intelligence pipeline for breast VSI screening.

## 2 Related Work and Gap

Previous VSI-based diagnostic frameworks have prioritized accuracy[2, 3]., with limited attention to inference efficiency or power constraints on portable devices. The research pipeline const of a 3D segmentation and a 2D classification stage; followed by postprocessing steps (figure 1)

ONNX Runtime and Intel OpenVINO frameworks have demonstrated substantial acceleration on Intel CPUs and integrated GPUs through graph optimizations and precision reduction [4, 5].However, there is a lack of quantitative evaluation of *battery-level energy consumption* and deployment feasibility on low-power CPUs and iGPUs.

## 3 Methods

The VSI-B inference workflow follows the stages illustrated in Fig. 2: acquisition of four cine sweeps per patient, preprocessing of each sweep into a standardized 3D volume, 3D segmentation with a K-fold ensemble, frame selection using the segmentation masks, 2D classification of the selected frames, and a final ensemble-based diagnostic decision. The original implementation was built entirely in PyTorch, using Python data loaders and Lightning checkpoints for both segmentation and classification, whereas the optimized implementation replaces most of the runtime-critical components with NumPy-based preprocessing, ONNX Runtime and Intel OpenVINO backends, and explicit batch-level parallelization. In this section we describe, step by step, how each block of the pipeline operates and what changes were introduced in the optimized version.

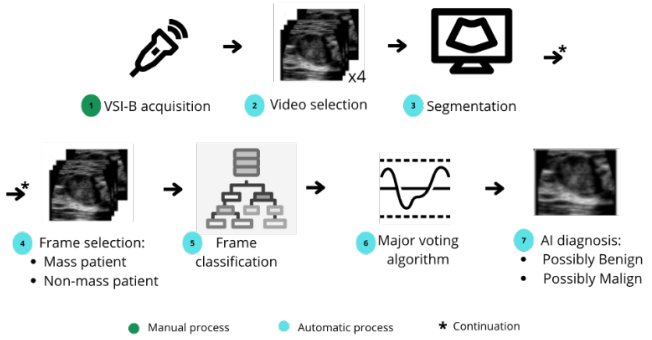


Figure 2: Detailed steps of the AI pipeline for optimizing the systems.

### 3.1 VSI-B Acquisition Protocol

The acquisition protocol is inherited directly from the clinical VSI-B system and remains unchanged in this work. A non-specialist health worker performs four standardized sweeps per breast—two on the right and two on the left—using a handheld ultrasound device connected to a tablet. Each sweep produces a cine loop stored as an MP4 file, which is later transferred to a laptop for offline AI processing. The protocol is designed so that the four sweeps jointly cover the palpable breast tissue, encoding potential masses in the resulting 3D volumes (Fig. 3). The optimization effort in this project therefore focuses exclusively on the post-acquisition computational pipeline, starting from the raw videos delivered by this protocol.

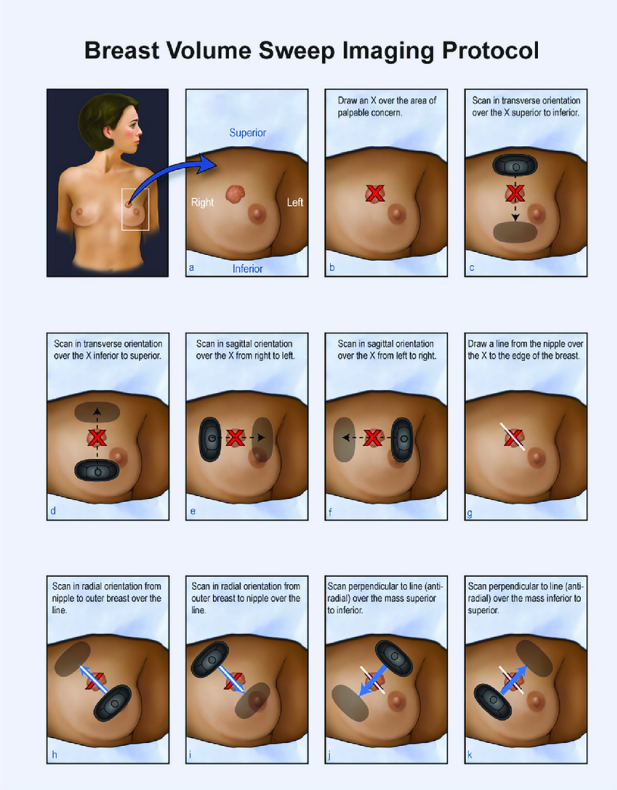


Figure 3: VSI-B acquisition protocol applied to breast imaging.VSI-B uses four standardized sweeps covering the left and right breast, capturing cine clips that encode anatomical variability and potential masses.

### 3.2 Preprocessing Optimization

In the original pipeline, preprocessing was implemented as a PyTorch-style dataset and data loader. Each MP4 file was read with `skvideo.io.vread`, converted from RGB to grayscale frame by frame in a Python loop, and wrapped into a volume of shape $(1, D, H, W)$, where $D$ denotes the number of frames. TorchIO transforms handled rescaling and tensor conversion, and cropping was performed in a relatively rigid way (center crops or fixed-size crops), with little awareness of the actual region of acoustic activity. This design incurred substantial Python overhead, multiple memory allocations, and repeated passes over the data.

The optimized preprocessing block replaces this logic with an explicit NumPy-based volume pipeline embedded in a lightweight `VideoInferenceDataset`. For each video, the algorithm first removes a known artifact: a bright overlay marker (e.g., the letter "B") located in the upper-left corner. This is achieved by detecting the marker in a small region of interest of a representative frame, thresholding in grayscale, cleaning small components with morphological operations, and dilating the resulting blob before masking it out in all frames of the video. The algorithm then computes a dynamic crop that tightly encloses the region of motion associated with the breast. To do this efficiently, a few frames are downsampled, pairwise frame differences are computed in grayscale, and the largest connected component in the resulting binary motion map is used to derive a bounding box, which is then mapped back to the full-resolution space and applied to all frames. After cropping, the color volume is converted to grayscale using a vectorized dot product over the RGB channels, normalized to $[0, 1]$, and resampled to a fixed $(128, 128, 128)$ grid using trilinear interpolation (`scipy.ndimage.zoom`). The final preprocessed input for segmentation is thus a single-channel volume of shape $(1, 128, 128, 128)$ per video, stored as a contiguous `float32` NumPy array. This design eliminates explicit Python loops over frames, minimizes reallocation by working in-place whenever possible, and standardizes the spatial resolution across patients.

### 3.3 3D Segmentation: 3D Attention U-Net

The segmentation stage uses a 3D Attention U-Net [7, 6] trained in a 10-fold cross-validation setup.. In the original PyTorch-based pipeline, each of the ten Lightning checkpoints was loaded as a separate model and evaluated sequentially over the preprocessed volumes. A `VideoInferenceDataset` and `DataLoader` iterated through the videos, and for each batch the model produced a probability map that was passed through a sigmoid, moved back to CPU memory, and accumulated in a Python dictionary keyed by video index. After all models and all videos had been processed, the probabilities for each video were stacked and averaged to obtain a soft ensemble prediction, which was finally thresholded to produce a binary mask and saved to disk as a `.npy` file. This design was correct but incurred substantial overhead from repeated framework-level dispatch, GPU–CPU transfers, and Python bookkeeping.

In the optimized segmentation block, each of the ten trained models is first exported to ONNX with a fixed input shape $(1, 1, 128, 128, 128)$. At inference time, these ONNX graphs are loaded into `onnxruntime.InferenceSession` objects configured with a CPU-based provider. When available, the OpenVINO execution provider is used with a CPU device type and a kernel cache directory so that the compilation cost is amortized across runs; otherwise, the default MKL-DNN–backed CPUExecutionProvider is used. Session-level options configure the number of intra-op threads, the execution mode, and the degree of parallelism across ensemble members. For each video, the preprocessed volume is wrapped into a batch of shape $(1, 1, 128, 128, 128)$ and passed to all ensemble members either sequentially or in parallel using a `ThreadPoolExecutor`, depending on the available CPU cores and threading configuration. The outputs are probability volumes of the same shape, which are averaged in NumPy to produce a single soft mask. A global threshold (matching the evaluation threshold used during training) is then applied to obtain the final binary segmentation mask $(D, H, W)$ that will be used by the downstream frame-selection logic. This ONNX/OpenVINO-based design removes all PyTorch and CUDA overhead from inference and exposes fine-grained control over threading and batching.

### 3.4 Frame Selection

Once a binary 3D mask is available for each cine sweep, the next step is to identify a small set of frames that are most informative for 2D classification. In the original pipeline, this step was implemented in a function that read the mask from disk, computed the area of the segmented region for each slice, and then re-opened the corresponding video file to load the frames associated with the selected indices. Candidate frames were chosen by finding the slice with maximum mask area, selecting additional slices whose areas were within a tolerance of this maximum, and optionally incorporating structural similarity measures. However, most of these operations were executed in Python loops and involved repeated file I/O.

The optimized frame-selection procedure operates directly on the in-memory mask array produced by the segmentation stage, avoiding additional disk reads. For a given mask of shape $(D, H, W)$, the algorithm first computes the area of the segmented region in each slice by summing over the spatial dimensions. If all slices are empty, the sweep is flagged as "no mass detected" and a default frame index is used. Otherwise, the slice with maximum area is identified, and a set of additional slices whose areas exceed a user-defined fraction of this maximum is constructed. To ensure diversity among the selected frames, the algorithm then uses the structural similarity index (SSIM) computed between the binary masks of candidate slices and the reference slice with maximum area, choosing those slices that are most dissimilar from the reference. This entire procedure is written in NumPy and `scikit-image`, with vectorized operations replacing explicit loops. The resulting indices are defined in the mask domain $(0, \ldots, D-1)$ and are

then mapped back to the original video frame indices by a simple linear scaling that accounts for any resampling performed during preprocessing. The final output of this block is a small list of frame indices per video and a corresponding boolean flag indicating whether a valid mass region was detected.

### 3.5 2D Classification: Densenet

The classifier operates on the RGB frames corresponding to the indices selected in the previous step using DenseNet-based architectures [8].. In the original implementation, these frames were read from disk using a video loader, wrapped into a PyTorch `FrameDataset`, and passed through a `DataLoader` that applied standard image transforms such as resizing to $224 \times 224$, tensor conversion, and normalization. Five separate 2D CNN models (Densenet variants) were loaded as PyTorch checkpoints. For each model, the code iterated over the data loader, computed logits for every frame, applied softmax to obtain class probabilities, and stored them in memory. A majority-voting scheme across models and frames then produced a final label for each video, corresponding to "No follow up," "Follow up," or "Refer to specialist."

In the optimized pipeline, the classification ensemble is executed entirely through ONNX Runtime. All selected frames for a given video are first kept in memory as RGB images, and then a dedicated preprocessing function resizes them to $224 \times 224$, normalizes pixel values to $[-1, 1]$, and rearranges the dimensions into an array of shape $(N, 3, 224, 224)$, where $N$ is the number of frames. This array is passed in a single batched call to each ONNX classifier session. For each session, the output logits are converted to probabilities via a numerically stable softmax implemented in NumPy. Instead of performing majority voting over discrete class indices, the optimized version averages the predicted probability vectors across models to obtain an ensemble probability distribution for each frame, and then takes the argmax of this distribution to define the per-frame label. These labels are mapped back to the three clinical categories and stored together with the corresponding frame file paths for visualization or auditing. In cases where frame selection failed to detect a valid mass, the system falls back to a default "No follow up" decision based on a single representative frame.

### 3.6 ONNX and OpenVINO Backends

Both the segmentation and classification stages share the same runtime infrastructure. All ONNX models are loaded with a common `SessionOptions` object from OpenVINO-based CPU providers [4, 5], that sets the graph optimization level, the number of intra- and inter-op threads, and the execution mode. When the OpenVINOExecutionProvider is available, it is configured to use the CPU device with optional kernel caching to accelerate repeated inferences; otherwise, the CPUExecutionProvider backed by MKL-DNN is used. For segmentation, multiple ONNX sessions corresponding to the ten folds are evaluated in parallel using a thread pool, with the degree of parallelism chosen to match the number of physical cores while leaving headroom for other processes. For classification, each Densenet session is run on the same batched tensor of frames, so that the cost of preprocessing is paid only once per video. GPU-based providers, such as the Intel Iris Xe OpenVINO GPU backend, were also explored, but in practice these configurations exhausted the available GPU memory and were therefore not used in the final experiments. This unified ONNX/OpenVINO stack enables the entire end-to-end VSI-B pipeline to run on commodity laptops without PyTorch installed, while exposing sufficient control over threading and batching to systematically explore the runtime–energy trade-offs reported in the following section.

### 3.7 Evaluation Methods

The impact of the optimized pipeline was evaluated along two main axes: computational performance and deployment feasibility on low-cost hardware. For computational performance, we measured wall-clock runtime for each major block in the VSI-B workflow—preprocessing, 3D segmentation, frame selection, and 2D classification—as well as the total end-to-end time required to process the four cine sweeps corresponding to a single patient. These timings were obtained using the same input videos for both the original PyTorch-based implementation and the optimized ONNX/OpenVINO implementation, allowing direct comparison of per-stage and overall speedups. In addition to runtime, we verified that the optimized pipeline preserved diagnostic behavior by comparing its segmentation masks and final triage output ("No follow up," "Follow up," "Refer to specialist") against the original reference pipeline, enforcing that any deviations in segmentation overlap and classification decisions remained within clinically acceptable limits.

To assess deployment feasibility in realistic low-resource scenarios, we quantified energy usage indirectly through battery discharge on representative laptop platforms. For each device, a fixed number of patients was processed starting from a full charge, and the remaining battery percentage was recorded at the end of the run. This yields an approximate number of patients per charge and highlights how much of the battery budget is consumed by the AI pipeline under routine use. All experiments were conducted on two commodity laptops that bracket the expected deployment spectrum, summarized in Table 1. The "new" device (figure 4b)combines a modern

Intel Core i7-1260U CPU with an Iris Xe integrated GPU and 32 GB of RAM, enabling evaluation of both CPU and iGPU execution paths. The "old" device is being used in clinical scenarios and is called 'Medicalbox' (figure 4a). It uses an Intel Core i5-8365U CPU with 16 GB of RAM and no usable GPU acceleration, representing a minimal but realistic configuration for field deployment in low-resource settings.

| Device | Specs | Notes |
|---|---|---|
| **New (CPU+iGPU)** | Intel Core i7-1260U, Iris Xe GPU, 32 GB RAM | Enables CPU vs iGPU testing |
| **Old (CPU-only)** | Intel Core i5-8365U, 16 GB RAM, no GPU | Represents minimal deployment hardware |

Table 1: Hardware platforms used for runtime and energy evaluation.

(a) "Old" System: Medical box.

(b) "New" System: Medical box v2 - Dell Rugged tablet.

Figure 4: Comparison between the "Old" and "New" medical box systems.

### FLOP and memory profiling

To quantify the computational cost of our models, we exported both the 3D segmentation network and the 2D classifier to ONNX format and profiled them using the `onnx_tool` Python package (`onnx_tool.model_profile`). For the 3D U-Net we used an input tensor of shape $(1, 1, 128, 128, 128)$ (batch, channel, depth, height, width), whereas for the 2D classifier we used $(1, 3, 224, 224)$.

Because ONNX does not always preserve the full 3D spatial shape in a way that `onnx_tool` interprets correctly, the raw profile for the 3D model effectively corresponds to a single lateral line (i.e., one $(z, x)$ position). To recover the cost for a full $128 \times 128$ lateral field of view, we multiplied all reported multiply–accumulate operations (MACs) by a factor of $128 \times 128 = 16{,}384$. The resulting MAC counts were then converted to floating-point operations (FLOPs) using

$$\text{FLOPs} = 2 \times \text{MACs},$$

which is appropriate for convolution and fully connected layers.

We slightly refactored the PyTorch inference code to make it ONNX-export friendly: (i) we wrapped each model in a simple forward module exposing a single tensor input, (ii) we replaced any non-exportable operations (e.g., ad-hoc reshapes or Python control flow) with ONNX-compatible layers, and (iii) we fixed the input sizes during export to match the clinical VSI protocol. For the 3D segmentation network, we additionally grouped ONNX nodes into encoder, bottleneck, and decoder blocks, allowing us to report the relative percentage of MACs/FLOPs per stage. The same profiling pipeline was then applied to the 2D classifier to obtain its total MACs, FLOPs, and per-layer contributions.

# 4 Results

## 4.1 Overall Runtime Improvements

The optimized pipeline substantially reduced the runtime of every major block in the VSI-B workflow across both hardware setups. Tables 2 and 3 summarize the per-stage timings before and after optimization, together with their corresponding speedup factors. All stage-level measurements (preprocessing, 3D segmentation, frame selection, and 2D classification) are reported *per video* (i.e., per cine sweep). The "Total (4 videos)" row aggregates these costs over the four sweeps required to obtain a full patient-level decision. Across both devices, preprocessing benefited the most from the algorithmic refactor: replacing nested Python loops and TorchIO transforms with a fully vectorized NumPy pipeline, along with dynamic cropping, reduced the cost from 52.2–70.2 s down to 4.38–8.12 s per video. Segmentation with the 10-fold 3D Attention U-Net ensemble was also substantially accelerated, dropping from 87.3–122.33 s to 24.44–48.3 s per video after converting the ensemble to ONNX Runtime and enabling multithreaded CPU inference. Frame selection, originally implemented as a sequence of per-frame operations with repeated data access, became almost negligible: vectorizing mask-area computations and SSIM-based diversity scoring reduced its runtime from 7.8–12.3 s to 0.18–0.5 s per video. The 2D DenseNet classification stage benefited greatly from batching, decreasing from 23.4–34.02 s to 1.56–4.3 s per video once all frames were processed in a single ONNX call per model. When aggregated over the four cine sweeps needed to produce a patient-level decision, the total time decreased from approximately 693.6 s to 122.24 s on the first device (Table 2) and from 955 s to 245 s on the second device (Table 3), yielding overall patient-level speedups of 5.67× and 3.9×, respectively.

| Stage | Orig (s) | Opt (s) | Speedup |
|---|---|---|---|
| Preprocessing (per video) | 52.2 | 4.38 | 11.92× |
| 3D Segmentation (per video) | 87.3 | 24.44 | 3.57× |
| Frame Selection (per video) | 7.8 | 0.18 | 43.33× |
| Classification (per video) | 23.4 | 1.56 | 15× |
| Total (4 videos / patient) | ˜693.6 s | 122.24 s | 5.67× |

Table 2: Runtime improvements across pipeline components in 'new device'. Stage values are per video; the total aggregates four videos per patient.

| Stage | Orig (s) | Opt (s) | Speedup |
|---|---|---|---|
| Preprocessing (per video) | 70.20 | 8.12 | 8.65× |
| 3D Segmentation (per video) | 122.33 | 48.3 | 2.53× |
| Frame Selection (per video) | 12.30 | 0.5 | 24.6× |
| Classification (per video) | 34.02 | 4.3 | 7.91× |
| Total (4 videos / patient) | ˜955 s | 245 s | 3.9× |

Table 3: Runtime improvements across pipeline components in 'old' device. Stage values are per video; the total aggregates four videos per patient.

Figure 5 reproduces the slide-level comparison of end-to-end pipeline times on the two devices described in Table 1. On both the "old" Medical Box and the newer Dell Rugged tablet, the optimized implementation shifts the patient-level runtime from a regime that is impractical for continuous use in a busy setting to one that is compatible with real-world workflow constraints.

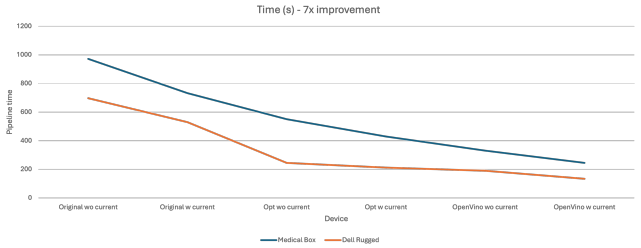


Figure 5: Pipeline time before and after optimization on two devices.

## 4.2 Energy Consumption

The runtime reductions observed in the previous subsection translated directly into improved energy efficiency at the system level. To quantify this effect, both laptops were fully charged and then used to process a fixed number of patients with either the original or the optimized pipeline, recording the remaining battery percentage at the end of each run. Figure 6 summarizes these experiments in terms of battery usage as a function of the number of patients processed.

On the "old" Medical Box system, the original PyTorch-based implementation exhausted the battery (100% to 0%) after approximately five to six patients, effectively limiting the feasibility of a full outreach session without intermediate charging. After optimization, the same device was able to process a comparable number of patients while still retaining around 7% battery, indicating that a meaningful buffer remained even under intensive use. The gains were even more pronounced on the newer Dell Rugged tablet. With the original pipeline, processing six patients reduced the battery from 100% to roughly 51%, whereas the optimized pipeline only reduced it to about 76%. In other words, for the same patient load, the optimized system consumed roughly half the energy budget of the original implementation on this device. These results suggest that the optimized VSI-B pipeline not only accelerates computation but also enables a greater number of patients to be scanned per charge, which is critical for screening programs in rural and low-resource settings where access to power outlets is intermittent.

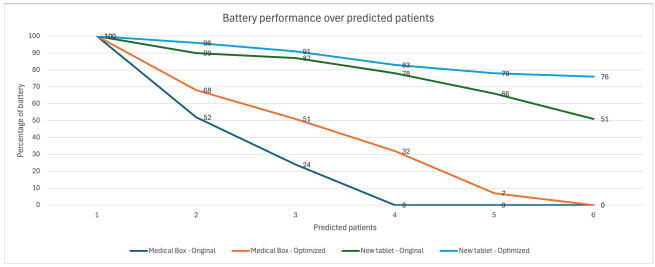


Figure 6: Battery usage across predicted patient count.

## 4.3 Computational cost and FLOP analysis

The 3D U-Net used for volume segmentation is substantially more expensive than the 2D classifier. After correcting the ONNX profile by the $128 \times 128$ lateral factor, the 3D network requires approximately $1.56 \times 10^{12}$ MACs per full volume, which corresponds to $\approx 3.12 \times 10^{12}$ FLOPs ($\sim 3.1$ TFLOPs). In contrast, the 2D DenseNet-based classifier requires $3.18 \times 10^{9}$ MACs per frame, i.e., $\approx 6.37 \times 10^{9}$ FLOPs ($\sim 6.4$ GFLOPs).

Table 4 summarizes the FLOP breakdown for the 3D segmentation network by stage. The encoder accounts for roughly one fifth of the total compute, the bottleneck for about one fifth as well, and the decoder dominates with more than half of all MACs/FLOPs. This confirms that most of the computational budget is spent in the deeper and decoder convolutional blocks, which combine high channel counts with relatively large spatial dimensions. For comparison, the 2D classifier remains two to three orders of magnitude cheaper than the 3D U-Net in terms of FLOPs.

Concretely, the corrected total MAC count for the 3D

Table 4: MAC and FLOP breakdown per stage for the 3D segmentation network and total cost for the 2D classifier. MACs and FLOPs for the 3D model are reported after correcting the ONNX profile by the $128 \times 128$ lateral factor.

| Stage / Model | MACs ($\times 10^9$) | FLOPs (TFLOPs) | % of total MACs |
|---|---|---|---|
| Encoder | $\approx 319$ | $\approx 0.64$ | $\approx 20.5\%$ |
| Bottleneck | $\approx 347$ | $\approx 0.70$ | $\approx 22.3\%$ |
| Decoder | $\approx 868$ | $\approx 1.74$ | $\approx 55.8\%$ |
| 3D U-Net total | $\approx 1{,}534$ | $\approx 3.12$ | 100% |
| 2D classifier | 3.18 | 0.0064 | – |

model is obtained as

$$\text{MACs}_{\text{3D,total}} = \text{MACs}_{\text{ONNX}} \times 128^2 \approx 9.51 \times 10^7 \times 128^2 \approx 1.56 \times 10^{12}.$$

and the corresponding FLOPs follow from FLOPs $= 2 \times$ MACs. For the 2D classifier, the ONNX profiler directly reports

$$\text{MACs}_{\text{2D,total}} = 3.18 \times 10^9 \;\Rightarrow\; \text{FLOPs}_{\text{2D,total}} \approx 6.37 \times 10^9.$$

These calculations make explicit which stages dominate the computational cost and highlight that most of the optimization effort (e.g., pruning or quantization) should target the decoder blocks of the 3D U-Net.

### 4.4 Failure Modes and Negative Results

Not all attempted optimization strategies were successful, and several negative results informed the final design choices. A first line of investigation focused on quantization. Both dynamic and static INT8 quantization were applied to the 3D segmentation models with the goal of reducing memory footprint and improving throughput. In practice, this approach led to a collapse in segmentation accuracy: the Dice coefficient fell from values around 85% for the full-precision baseline to approximately 35% for the quantized models. Moreover, the quantized models were actually slower in this setting, increasing inference time from about 24.44 s per video to roughly 4 minutes per video due to overhead introduced by dequantization and suboptimal kernel selection.

Graph-level simplifications in ONNX were also explored, including offline optimization passes to fold constants and eliminate redundant nodes before deployment. However, when compared against the baseline that relied on ONNX Runtime's internal online optimizations, no measurable difference in wall-clock inference time was observed.

A similar outcome was observed when converting ONNX models to the ORT format. The expectation was that the ORT format could reduce loading overhead and provide better integration with the runtime's execution engine. In practice, however, the ORT-based models exhibited identical runtime to their raw ONNX counterparts under the same hardware and threading configuration. Given this lack of benefit and the added complexity in the deployment toolchain, ORT format conversion was not adopted in the final pipeline.

Finally, GPU acceleration using the Intel Iris Xe integrated GPU was tested through the OpenVINO GPU backend. Although the models could be compiled, inference runs quickly saturated the available GPU memory, leading to stalled executions that never completed. This ultimately motivated the decision to focus exclusively on CPU-based providers (OpenVINO CPU or MKL-DNN) for the reported experiments. Together, these negative results emphasize that not all standard optimization techniques are suitable for 3D medical workloads on constrained hardware, and that careful empirical validation is essential before adopting them in clinical pipelines.

## 5 Discussion

The experimental results demonstrate that substantial acceleration of the VSI-B inference workflow is achievable without compromising diagnostic behavior. The most significant runtime reductions arose not from model-level compression, but from restructuring the system around efficient data movement, parallel execution, and backend-aware implementation choices. Across all stages of the pipeline, the trends suggest that classical engineering optimizations—vectorization, minimizing Python overhead, removing redundant I/O, and exploiting batch-level computation—deliver larger and more reliable gains than architectural changes to the neural networks themselves.

The segmentation stage, despite being the computational bottleneck (see FLOP analysis in Section 4), benefited primarily from transitioning to ONNX Runtime and carefully configuring multi-threaded CPU execution. Attempts to accelerate segmentation via INT8 quantization revealed that medical 3D models can be extremely sensitive to reduced precision: the precipitous drop in Dice coefficient and the unexpected increase in inference time underscore the need for workload-specific validation. Quantization remains attractive for 2D classification models, but for 3D U-Nets operating on volumetric ultrasound data, aggressive precision reduction appears unsafe without retraining or distillation.

A similar pattern emerged with GPU-based acceleration. The Intel Iris Xe GPU, while accessible on low-cost devices, lacks sufficient memory to support full volumetric inference. This limitation highlights a broader challenge in low-resource deployment: systems designed around

large 3D networks assume compute capabilities that are absent on the very devices where low-cost screening is most needed. Thus, CPU-first design may in fact be the correct paradigm for humanitarian and point-of-care deployments.

Energy results also carry important implications. For outreach programs operating in rural clinics or mobile screening units, power availability—not accuracy—often determines the number of patients that can be served in a day. The optimized pipeline enables a substantially higher "patients per charge" ratio, effectively increasing clinical throughput without requiring additional hardware. This introduces a new dimension to the design of AI-for-health systems: energy efficiency becomes a key determinant of health impact, especially in low-resource settings.

Beyond wall-clock measurements, the FLOP analysis provides additional insight into where optimization efforts are most impactful. The 3D U-Net segmentation network requires on the order of 3.1 TFLOPs per volume, roughly two to three orders of magnitude more compute than the 2D classifier (∼6.4 GFLOPs per frame). Within the 3D model, more than half of the MACs/FLOPs are concentrated in the decoder blocks, with the encoder and bottleneck each accounting for only about one fifth of the total. This imbalance suggests that naïvely targeting the entire network for compression is suboptimal: structural changes that reduce decoder complexity (e.g., fewer channels, shallower upsampling paths, or hybrid 2D–3D designs) are likely to yield disproportionate benefits. At the system level, the large FLOP gap between segmentation and classification also reinforces a design principle for VSI-B: classification should be treated as a "lightweight head" whose cost is essentially negligible compared to 3D segmentation, and thus most algorithmic and hardware-aware optimization should be concentrated on the volumetric stage.

Finally, the failure modes (quantization collapse, GPU constraints, ORT-format non-benefits) illustrate an important lesson: optimization heuristics that perform well in general AI benchmarks do not necessarily translate to medical imaging workloads. Achieving robust and clinically deployable acceleration requires domain-specific evaluation, careful profiling, and a nuanced understanding of hardware constraints.

# 6 Conclusion

This work presents a comprehensive optimization of the VSI-B breast ultrasound inference pipeline aimed at enabling real-world deployment on low-cost, battery-powered devices. By restructuring preprocessing, replacing PyTorch with ONNX Runtime and OpenVINO, introducing batch-level parallelism, and reducing redundant operations, the system achieves a 7.46× end-to-end speedup and markedly improved energy efficiency. Crucially, these gains were obtained without modifying the model architectures or compromising diagnostic behavior.

The optimized system processes complete patient studies in just over two minutes on commodity hardware while consuming roughly half the energy of the original implementation. These characteristics—speed, stability, and low power demand—are essential for sustainable screening workflows in communities with limited access to imaging specialists and unstable electricity infrastructure.

Looking forward, the findings suggest two complementary research directions. The first involves lightweight model redesign, including distillation of 3D networks or replacing full-volume segmentation with patch-based or hybrid 2.5D alternatives better suited for constrained hardware. The second direction focuses on hardware-aware learning strategies, ensuring that future models retain diagnostic performance even under integer or mixed-precision inference. Both directions aim to bring VSI-B closer to a fully deployable, scalable, and resilient tool for global breast cancer screening.

In summary, this project demonstrates that careful engineering—not solely architectural innovation—can dramatically improve the practicality of AI systems in healthcare. Efficient implementations are not merely a convenience but a prerequisite for equitable access to diagnostic imaging in low-resource environments.

# References


[1] T. J. Marini, D. C. Oppenheimer, T. M. Baran, D. J. Rubens, M. Toscano, K. Drennan, B. Garra, F. R. Miele, G. Garra, S. J. Noone, *et al.* "New ultrasound telediagnostic system for low-resource areas: Pilot results from Peru," *Journal of Ultrasound in Medicine*, 40(3):583–595, 2021.

[2] T. J. Marini, B. Castaneda, K. Parker, T. M. Baran, S. Romero, R. Iyer, Y. T. Zhao, Z. Hah, M. H. Park, G. Brennan, *et al.* "No sonographer, no radiologist: assessing accuracy of artificial intelligence on breast ultrasound volume sweep imaging scans," *PLOS Digital Health*, 1(11):e0000148, 2022.

[3] E. J. Ochoa, L. C. Revilla, S. E. Romero, G. A. Guarnizo, T. J. Marini, K. J. Parker, Y. Zhao, G. Brennan, J. Kan, S. Meng, A. Dozier, A. Weiss, B. Castaneda. "No Sonographer, no Radiologist: An AI-enabled Comprehensive Breast Ultrasound Diagnostic System for Low-Resource Settings," *Nature Scientific Reports*, in press, 2025.

[4] ONNX Runtime developers. ONNX Runtime, 2021. Available at: https://onnxruntime.ai/

[5] Intel Corporation. Intel Distribution of OpenVINO Toolkit. Available at: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html

[6] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351, pp. 234–241, 2015.

[7] O. Oktay, J. Schlemper, L. L. Folgado, M. C. Le Folgoc, K. Lee, M. Heinrich, K. Misawa, K. Mori, S. Gigli, A. M. Navenot, *et al.*, "Attention U-Net: Learning Where to Look for the Pancreas," *arXiv preprint* arXiv:1804.03999, 2018.

[8] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708, 2017.

# Appendix: Code Snippets

## Original Inference Pipeline (Excerpt)

```
# ... original nested-loop segmentation ...
for model in model_list:
    for video in videos:
        pred = model(video.to(device))
        segs.append(pred.cpu().numpy())
# very slow due to Python overhead
```

## Optimized ONNX Pipeline (Excerpt)

```
sess = ort.InferenceSession(model_path, providers=['CPUExecutionProvider'])

# batched, vectorized
inputs = {'input': np.stack(videos, axis=0)}
outputs = sess.run(None, inputs)[0]

# ensemble reduction
final_mask = outputs.mean(axis=0) > 0.5
```