# Implement, Test, and Economize an existing desnowing model on an embedded system

Fairuz Khan
*Mälardalens University*
Västerås, Sweden
fkn24006@student.mdu.se

Khalid Hasan Ador
*Mälardalens University*
Västerås, Sweden
khr24001@student.mdu.se

Rowshan Mannan Oni
*Mälardalens University*
Västerås, Sweden
roi24001@student.mdu.se

Samir Saffo
*Mälardalens University*
Västerås, Sweden
sso21004@student.mdu.se

***Abstract*—This report explores the importance of snow removal, or desnowing, in real-world embedded systems and IoT devices, where snowy weather often degrades the performance of computer vision applications. We address the challenges of implementing efficient snow removal in resource-constrained environments by utilizing the Laplace-prior-guided Mask Query Transformer (LMQFormer), as introduced by Lin et al. [1]. This architecture is both lightweight and high-performing, making it well-suited for our purposes.**

**To validate its capabilities on embedded systems, we deployed LMQFormer on Qualcomm AI platforms, fine-tuning it to achieve real-time performance on our target chip. Additionally, we conducted a comparative analysis of its performance on the CPU against the optimized version, evaluating results across various image sizes. Our findings underscore LMQFormer's transformative potential in enhancing snow removal for IoT and embedded applications, ultimately enabling reliable computer vision even under adverse weather conditions. The code for our implementation is available at: https://github.com/HIGH5ATURN/LMQFormer. The main code that our work is based on can be found here: https://github.com/JHLin42in/LMQFormer**

## I. Introduction

In real-world scenarios, snowy weather significantly impacts the performance of computer vision systems, particularly in IoT devices used in applications like autonomous driving, surveillance, and traffic monitoring. Snow obscures critical details in images, leading to poor visibility and degraded performance in high-level tasks such as object detection and scene understanding. Snow removal (desnowing) is, therefore, a crucial task for ensuring the reliability and accuracy of these systems, especially in environments where real-time processing and low computational resources are essential.

However, implementing effective snow removal in IoT devices and embedded systems poses significant challenges. These devices often have limited computational power, memory, and energy resources, making the deployment of complex deep learning models difficult. While effective, Traditional snow removal methods are computationally expensive and unsuitable for real-time applications. This creates a need for lightweight, efficient, and high-performing desnowing architectures. We explore the Laplace-prior-guided Mask Query Transformer (LMQFormer), a state-of-the-art lightweight snow removal network. The LMQFormer leverages two key components, where one generates a coarse mask to identify snow areas with minimal computational cost, and the other uses this mask to efficiently remove snow while preserving image details.

By deploying the LMQFormer on Qualcomm AI platforms, we further optimize the model for real-time performance, ensuring efficient execution on embedded devices. This deployment allows us to evaluate the model's effectiveness in real-world scenarios, demonstrating its ability to recover clean, high-quality images from snowy inputs.

In this report, we delve into the architecture of LMQFormer, its lightweight design, and its deployment on Qualcomm AI platforms, showcasing its potential to revolutionize snow removal in real-world embedded systems.

## II. Related Work

### A. Overview of Desnowing Architecture

Deep learning has revolutionized single-image snow removal, surpassing traditional methods reliant on prior knowledge. The first deep-learning-based approach, DesnowNet, was introduced by Liu et al. [2], employing a two-stage network to learn the mapping from snowy images to snow masks and subsequently recover clean images. Generative Adversarial Network (GAN) was later used to recover clean images in [3]. Chen et al. [4] proposed a JSTAR model to generate three different snow masks with a differentiable dark channel prior layer and guide image recovery with these masks. Chen et al. [5] also proposed HDCWNet, which removed snow with hierarchical dual-tree complex wavelet representation and contradict channel loss. Ye et al. [6] developed DAN-Net, a compact degradation-adaptive network comprising multiple expert sub-networks controlled by an adaptive gated neural unit. Ozdenizci et al. [7] present a novel patch-based image restoration algorithm based on denoising diffusion probabilistic models. The patch-based diffusion modeling approach enables size-agnostic image restoration by using a guided denoising process with smoothed noise estimates across overlapping patches during inference.

These methods achieve promising performances in snow removal, but at the cost of a large parameter size. To address this efficiency concern, Lin et al. [1] proposed LMQFormer, a lightweight yet high-performance snow removal network that integrates CNNs, Transformers, and VQVAE. This model is the focus of our study, and its detailed architecture is provided in subsection III-D.

## III. Approach and Method

### A. Initial Choice

When selecting a model for deployment on an embedded system, it is crucial to consider the resource constraints, like limited memory, limited processing power, and energy availability. The chosen model must strike a balance between performance and efficiency to ensure it can operate effectively within these limitations. Key metrics to evaluate include PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index Measure), which measure the quality of the reconstructed images, as well as the number of parameters (#Param) and computational complexity (#GMacs), which determine the model's resource requirements.

A model with high PSNR and SSIM values ensures that the image quality is maintained [7], and by improving image quality, desnowing enhances the performance, safety, and reliability of IoT systems across various applications, from autonomous vehicles and security systems to environmental monitoring. Models with fewer parameters and lower GMacs are more suitable for embedded deployment, as they require less memory and processing power, making them easier to run on hardware with limited resources. Figure 1 shows a quantitative comparison among different architectures [6]. From this, it is clear that HDCW-Net [5] is particularly well-suited for embedded systems due to its comparatively low number of parameters(699k) and efficient memory usage(9.78G).

| Method | CSD(2000) | | SRRS (2000) | | Snow 100K (2000) | | #Param | #GMacs |
|---|---|---|---|---|---|---|---|---|
| | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM | | |
| (TIP'18)Desnow-Net [26] | 20.13 | 0.81 | 20.38 | 0.84 | 30.50 | 0.94 | 15.6M | - |
| (ICCV'17)CycleGAN [10] | 20.98 | 0.80 | 20.21 | 0.74 | 26.81 | 0.89 | 7.84M | 42.38G |
| (CVPR'20)All-in-One [22] | 26.31 | 0.87 | 24.98 | 0.88 | 26.07 | 0.88 | 44 M | 12.26G |
| (ECCV'20)JSTASR [6] | 27.96 | 0.88 | 25.82 | 0.89 | 23.12 | 0.86 | 65M | - |
| (ICCV'21)HDCW-Net [7] | 29.06 | 0.91 | 27.78 | 0.92 | 31.54 | 0.95 | 699k | 9.78G |
| Desnowing Expert Net | 30.56 | **0.95** | 29.07 | **0.95** | 32.14 | **0.96** | 1.1M | 12.00G |
| Desnowing Expert Net-Tiny | 29.06 | 0.92 | 28.20 | 0.94 | 31.67 | 0.95 | 288K | 3.06G |
| DAN-Net | **30.82** | **0.95** | **29.34** | **0.95** | **32.48** | **0.96** | 2.73M | 31.36G |
| DAN-Net-Tiny | 29.12 | 0.92 | 28.32 | 0.94 | 31.93 | 0.95 | 1.02M | 13.47G |

Fig. 1. Quantitative comparisons among different architectures (Adapted from Ye et al. [6])

1) **Fewer Parameters (699k)**
   - **Reduced Memory Footprint**: Embedded systems often have limited memory (RAM and storage) [8]. Models with fewer parameters require less memory to store the model weights, making them easier to deploy on devices with constrained resources.
   - **Faster Loading Times**: Smaller models load faster, which is crucial for real-time applications where quick startup and response times are essential.
   - **Lower Power Consumption**: Fewer parameters mean fewer computations, which translates to lower power consumption. This is critical for battery-powered embedded devices.
2) **Lower Computational Complexity (9.78 GMacs)**
   - **Efficient Processing**: Embedded systems typically have limited processing power (micro-controllers or low-power CPUs) [8]. HDCW-Net's low computational complexity (measured in GMacs) ensures that it can run efficiently on these devices without requiring high-end hardware.
   - **Real-Time Performance**: Lower computational demands allow the model to process images in real-time, which is essential for applications like autonomous vehicles, drones, or IoT devices where delays are unacceptable.
   - **Heat Dissipation**: Less computation means less heat generation, which is crucial for energy-efficient and thermally constrained environments. Devices like mobile processors, edge AI systems, and embedded platforms often operate in compact enclosures with limited cooling solutions. By reducing computational load, HDCW-Net helps prevent thermal throttling, extends component lifespan, and improves overall system reliability.

Despite its small size, HDCW-Net achieves competitive PSNR (29.09 on CSD, 27.78 on SRRS and 31.54 on Snow 100k database) and SSIM (0.91 on CSD, 0.92 on SRRS and 0.95 on Snow 100k database) scores [6], meaning it maintains high-quality image reconstruction while being lightweight. It combines high performance with low resource requirements. This makes it a practical and cost-effective solution for real-world embedded applications.

### B. Compilation Issues in Qualcomm AI Hub for initial choice

Our first choice was HDCW-Net [5], which showed promising results in terms of performance and efficiency on embedded systems. However, we encountered compilation issues when attempting to deploy this model on Qualcomm AI Hub [9] because of its dependence on TensorFlow 1.x and usage of complex operations in some of the layers. Following this, we shifted our focus to JSTASR [4], another model that demonstrated strong performance in desnowing tasks. Similar to HDCW-Net, JSTASR also relies on TensorFlow 1.x, which presented compatibility issues with Qualcomm AI Hub. While both models were suitable in terms of their architectural design and performance metrics, the reliance on TensorFlow 1.x made them incompatible with the deployment environment we were targeting.

Despite these challenges, we were able to successfully run inference for both HDCW-Net and JSTASR on our local CPU.

This allowed us to validate their performance and confirm their effectiveness in desnowing tasks.

- **HDCW-Net**
  During our efforts to deploy HDCW-Net, we encountered a significant challenge related to a custom layer in the algorithm called the Inverse DWT Layer. This layer involves a complex operator that takes two real numbers as input and returns a complex number as output. While this operation is integral to the model's functionality, it posed a major obstacle during deployment on Qualcomm AI Hub.
  The key issues we faced in this approach were:
  – **ONNX Runtime Incompatibility**
    Complex numbers (a + bi) are not natively supported in ONNX as there is no built-in ONNX operator for handling complex arithmetic. Even if it is possible to export the model to ONNX ( by representing complex numbers as separate real and imaginary parts), ONNX Runtime do not have a corresponding implementation for the custom operator. As so the Inverse DWT Layer may not be properly serialized and will throw a runtime error.

  **Workaround Attempt**
  To address this issue, we tried modifying the approach by concatenating the two real numbers (which were originally inputs to the complex operator) to produce a real number as output instead of a complex number. While this modification allowed the model to run, it led to two critical problems:

  1) The output images suffered from significant degradation in quality, rendering the desnowing results unusable.
  2) The modification disrupted the integrity of the saved_model structure, making it unstable and unreliable for further use.

- **JSTASR**
  The JSTASR model was implemented using TensorFlow 1.6.0 and Keras 2.2.0. For compilation in Qualcomm AI Hub, it's necessary to have the ONNX format as well as TFLite version, and for that, the model needs to be saved in the saved_model format (a directory containing the model's architecture, weights, and metadata) [10]. But converting the model to saved_model format and TensorFlow Lite (TFLite) requires TensorFlow 2.x, leading to compatibility issues.

  1) **Deprecated Operations**: TensorFlow 1.x models may use operations that are deprecated or removed in TensorFlow 2.x.
  2) **Custom Layers**: The JSTASR model contains custom layers or operations that are not supported by TFLite.
  3) **API Changes**: TensorFlow 2.x introduced significant changes to the API, causing compatibility issues.

The failure to convert the model to TFLite prevents deployment on Qualcomm AI Hub. However, the inability to deploy these models on Qualcomm AI Hub due to TensorFlow 1.x limitations led us to reconsider our approach and explore alternative solutions that align better with the platform's requirements, such as any model written in PyTorch.

### C. Final Choice

QUANTITATIVE EVALUATIONS ON SYNTHETIC DATASETS.

| Dataset (testset: 1200) | IQA | DesnowNet (TIP'18) | JSTASR (ECCV'20) | HDCWNet (ICCV'21) | DDMSNet (TIP'21) | TKL (CVPR'22) | LMQFORMER |
|---|---|---|---|---|---|---|---|
| Snow100K (trainset: 10000) | PSNR(↑) / SSIM(↑) | 23.125/0.788 | 18.648/0.554 | 18.188/0.561 | 29.058/0.891 | 28.098/0.851 | 31.883/0.917 |
| | MAE(↓) / LPIPS(↓) | 100.387/0.257 | 129.854/0.437 | 164.356/0.406 | 168.364/0.103 | 98.232/0.140 | 93.999/0.085 |
| SRRS (trainset: 8000) | PSNR(↑) / SSIM(↑) | 21.307/0.835 | 21.593/0.770 | 25.148/0.893 | 25.967/0.932 | 25.718/0.909 | 31.040/0.964 |
| | MAE(↓) / LPIPS(↓) | 119.795/0.290 | 178.618/0.245 | 140.252/0.103 | 136.799/0.058 | 97.407/0.082 | 119.230/0.025 |
| CSD (trainset: 8000) | PSNR(↑) / SSIM(↑) | 20.632/0.777 | 20.628/0.705 | 28.669/0.892 | 24.976/0.905 | 30.122/0.933 | 32.643/0.963 |
| | MAE(↓) / LPIPS(↓) | 152.851/0.309 | 193.095/0.410 | 131.587/0.118 | 130.517/0.084 | 123.973/0.058 | 121.900/0.029 |
| SnowKITTI2012 (trainset: 4500) | PSNR(↑) / SSIM(↑) | 16.487/0.682 | 18.470/0.532 | 20.085/0.575 | 30.520/0.934 | 22.863/0.756 | 33.145/0.959 |
| | MAE(↓) / LPIPS(↓) | 96.785/0.353 | 143.152/0.407 | 166.774/0.329 | 112.250/0.053 | 110.321/0.221 | 81.847/0.051 |
| SnowCityScapes (trainset: 6000) | PSNR(↑) / SSIM(↑) | 21.020/0.705 | 19.047/0.550 | 20.385/0.651 | 33.371/0.956 | 24.412/0.804 | 39.172/0.984 |
| | MAE(↓) / LPIPS(↓) | 79.821/0.343 | 129.134/0.419 | 190.366/0.310 | 105.535/0.031 | 162.100/0.208 | 67.639/0.010 |
| Parameters(million) | | 26.151 | 83.347 | 18.958 | 229.454 | 28.712 | 2.181 |
| Runtime(s) | | 2.992 | 0.356 | 0.141 | 0.554 | 0.051 | 0.042 |

Fig. 2. Quantitative comparisons among different architectures (Adapted from Lin et al. [1])

Figure 2 shows a quantitative comparisons among different architectures [1]. LMQFORMER is a good choice because it combines high performance (best PSNR, SSIM, MAE, and LPIPS) with efficiency, as it has the fewest parameters and the fastest runtime. Its consistency across different datasets and suitability for real-time applications make it an ideal candidate for practical deployment, especially in resource-constrained environments like IoT devices.

### D. Architecture overview of LMQFORMER [1]

The LMQFormer is a hybrid snow removal network combining CNNs, Transformers, and VQVAE for efficient and high-quality snow removal. It consists of two main sub-networks:

- **Laplace-VQVAE:** Generates a coarse mask to locate snow regions.
- **MQFormer:** Removes snow using residual learning and attention mechanisms, and the coarse mask is used as guidance.

*1) **Laplace-VQVAE**:* It generates a coarse mask to identify snow regions. It takes Snowy image $I_{\text{snow}}$ and gives a coarse mask $I_{\text{prior}}$ (low-entropy representation of snow regions). This component uses spatial attention (SACM) to preserve snow edge details. It has two parts:

I. **Laplace Operator:** A 3×3 kernel ([[0, -1, 0], [-1, 4, -1], [0, -1, 0]]) applied to the input image to highlight high-frequency snow edges while suppressing background.
II. **VQVAE(Vector Quantized Variational Autoencoder):** It has got three units:
  - **Encoder:** Downscales the image into latent features.
  - **Codebook:** Discretizes latent features into a set of learned vectors.

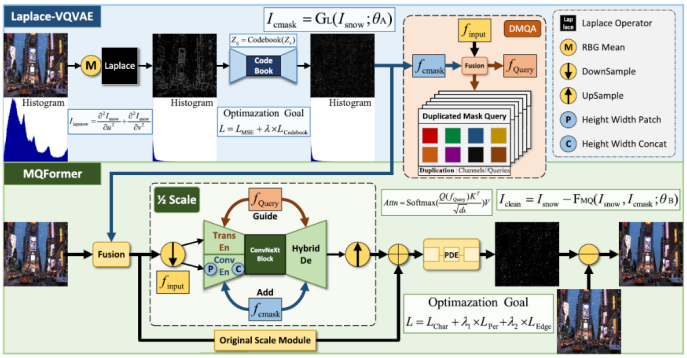


Fig. 3. The proposed LMQFormer with two sub-networks: Laplace-VQVAE and MQFormer. The Laplace-VQVAE extracts a prior of the coarse mask while the MQFormer recovers clean images with the Laplace prior. Due to the Laplace operation and the attention-based design, the network is lightweight but highly efficient. (Adapted from Lin et al. [1])

- **Decoder:** Reconstructs the coarse mask from quantized features.

So, the workflow will be: Snowy Image → Laplace Operator → VQVAE → Coarse Mask.

*2) **MQFormer(Mask Query Transformer)**:* Removes snow using the coarse mask and the difference between snowy and clean images. This component got five units:

(I) **Parallel Encoders:**
- **Transformer Encoder:** Captures global snow patterns using self-attention.
- **Convolutional Encoder:** Extracts local features (textures, edges) using Channel Attention Convolution Modules.

(II) **Hybrid Decoder:** Combines global and local features for reconstruction. It up-samples features to the original resolution.

(III) **Pixel Detail Enhancement (PDE):** Refines edges and textures using 3×3 depthwise convolutions and channel attention.

(IV) **Duplicated Mask Query Attention (DMQA):** It converts the coarse mask into multiple queries to focus on snow regions. It computes attention at 1/4 resolution for efficiency. The output from this unit is the residual snow map R (snow-only components).

(V) **Residual Learning and Clean Image Generation** From MQFormer, it takes the residual snow map as input. Then subtract this difference from the snowy image to obtain the clean image. It follows this equation: $I_{\text{clean}}$=$I_{\text{snow}}$-R .
This will make it easier for the network to learn the difference (snow) than the full clean image. Also, it preserves background details by avoiding direct pixel reconstruction.

*3) **Attention Mechanisms**:* This architecture uses three attention mechanisms (a technique in deep learning that allows models to focus on the most relevant parts of the input when making predictions [11]):

- **Spatial Attention (SACM):** It highlights snow edges in feature maps. It follows the following operation for computation: average/max pooling → concatenates → 7×7 conv → sigmoid.
- **Channel Attention (CACM):** It enhances snow-relevant feature channels. This attention mechanism works by Global average pooling → 1D conv → sigmoid.
- **Duplicated Mask Query Attention (DMQA):** It focuses computation on snow regions. At first, it duplicates the coarse mask into 8 queries. Then computes attention at 1/4 resolution (efficiency).
  - **Query:** Q=Conv($F_{\text{stem}}$+mask)
  - **Key/Value:** K,V=Conv($F_{\text{stem}}$)

  Here,$F_{\text{stem}}$ is the output feature map from the Stem Module, which serves as the foundational input for subsequent layers in MQFormer.

*4) **Loss Functions**:* These are the loss functions used in the LMQFormer architecture for training the model to achieve high-quality snow removal while maintaining computational efficiency.

(a) **MSE (VQVAE)**
- **Mathematical Formulation:**
  $\mathcal{L}_{\text{MSE}} = \|I_{\text{snow}} - I_{\text{reconstructed}}\|_2^2$
- **Purpose:** Ensures accurate reconstruction of the input image in Laplace-VQVAE.

(b) **Codebook Loss**
- **Mathematical Formulation:**
  $\mathcal{L}_{\text{Codebook}} = \sum_{k}^{2} \sum_{j}^{n_i} \|z_{i,j} - e_i\|$
- **Purpose:** Forces latent codes to align with discrete codebook vectors for compact representation.

(c) **Charbonnier Loss**
- **Mathematical Formulation:**
  $\mathcal{L}_{\text{char}} = \sqrt{\|I_{\text{clean}} - I_{\text{pred}}\|^2 + \epsilon^2},\ \epsilon = 10^{-5}$
- **Purpose:** Robust loss that handles outliers (heavy snow occlusions) better than MSE.

(d) **Perceptual Loss (VGG19**
- **Mathematical Formulation:**
  $\mathcal{L}_{\text{per}} = \|\phi(I_{\text{clean}}) - \phi(I_{\text{pred}})\|_2^2$
- **Purpose:** Maintains perceptual similarity between predicted and clean images using VGG19 features.

(e) **Edge Loss**
- **Mathematical Formulation:**
  $\mathcal{L}_{\text{edge}} = \frac{1}{N} \sum_{i=1}^{N} \sqrt{\|\Delta(I_{\text{pred}}) - \Delta(I_{\text{clean}})\|^2 + \varepsilon^2}$
- **Purpose:** Preserves sharp edges and fine details by comparing image gradients.

In general, LMQFormer demonstrates that lightweight architectures can achieve high-quality snow removal without excessive computational costs, making it a practical solution for

real-world deployment in autonomous vehicles, surveillance systems, and other edge devices operating in snowy environments. Future work could explore dynamic mask refinement or multi-scale attention to further improve performance under extreme conditions.

## IV. Implementation in Qualcomm AI hub

Qualcomm AI Hub [9] is a specialized platform designed to optimize and deploy machine learning models on Qualcomm's hardware, such as the QCS8550 chip. It provides tools for model compilation, inference, and profiling, ensuring that models are tailored to run efficiently on IoT devices. The platform supports various frameworks like TensorFlow, PyTorch, and ONNX, making it versatile for different use cases.

### *A. Converting to Onnx Format*

Converting a model to ONNX enables further optimizations, such as quantization and hardware-specific tuning, which are essential for improving performance on embedded systems. These models are compatible with Qualcomm AI Hub.

We converted our trained model (PyTorch) to ONNX format using the appropriate tools (torch.onnx.export) [10]. During conversion, it was ensured that the model's architecture and weights were preserved while making it compatible with the ONNX runtime.

### *B. Compilation and Profiling*

Compiling the model for a specific target device (QCS8550) ensures that it takes full advantage of the hardware's capabilities, such as accelerators and parallel processing units. Profiling helps identify bottlenecks in the model's performance, such as high memory usage or slow layers, allowing for targeted optimizations. These are necessary steps for deploying a model on Qualcomm AI Hub and making it optimized for that hardware.

After converting the model to ONNX, we used Qualcomm AI Hub to compile it for the QCS8550 chip. Then we profiled the compiled model to measure key metrics such as inference time, memory usage, and energy consumption. This step ensures that the model meets the performance requirements for deployment on embedded systems.

### *C. Quantization Process*

Quantization refers to the process of reducing the precision of the model's weights and activations from floating-point (e.g., 32-bit floating-point, or FP32) to lower precision (e.g., 8-bit integers, or INT8). For resource-constrained systems like embedded systems, to improve their inference speed and reduce memory usage, quantization is done.

For our desnowing architecture, we followed four approaches to do quantization:

- Using PyTorch's Static Quantization
- Using PyTorch's Dynamic Quantization
- Using ONNX Runtime's Static Quantization
- Using ONNX Runtime's Dynamic Quantization

We faced failure, and by doing a deep analysis, we found the reasons that were causing the issues, which are:

1) **State Dictionary Mismatch**
   The saved state dictionary (Snow100K.pth) did not match the architecture of the LMQFORMER model, leading to missing keys.
2) **Unsupported Custom Layers**
   The LMQFORMER model contains custom layers (**VQVAE.Laplace_op**) that are not fully supported by PyTorch's or ONNX Runtime's quantization tools.
3) **Unsupported Operations**
   The model contains operations (**ReduceMax**) that are not fully supported by ONNX Runtime's quantization tools. These operations lack the necessary quantization support.
4) **Missing Initializers**
   The model lacks the necessary initializers for quantization, causing the quantization process to fail.

## V. Results And Evaluation

Image inference and memory usage are crucial considerations for IoT and embedded systems due to their resource constraints, real-time processing requirements, and energy limitations.

### *A. Comparison between not optimized and optimized versions*

#### *1) Performance on CPU:*

- **Inference:**
  To evaluate how our architecture performs in a resource-constrained environment, we conducted inference tests on a CPU (Central Processing Unit) instead of a GPU (Graphics Processing Unit). CPUs are less computationally powerful compared to GPUs, making them a good benchmark for assessing the feasibility of deploying our model on IoT devices or embedded systems, which often rely on CPUs due to their lower cost and energy consumption.

  **Test Setup:**
  - Device: CPU (no GPU acceleration)
  - Scenario: Synthetic dataset (synth)
  - Weights Used: Pre-trained weights from Snow100K.pth [12]
  - Image Size: 256x256
  - Number of Images Processed: 2
  - 10 runs were performed, with the **first run discarded** (to account for initialization overhead), and the average of the remaining 9 runs was taken for accuracy.
  - Use Case: Baseline performance for resource-constrained environments.
- **Memory Allocation (RAM)**
  The Peak RSS (Resident Set Size) of **1463.71 MB** suggests that the model requires a significant amount of

```
!python test.py --device cpu

====> Scenario:  synth
====> Testing weights: ./checkpoints/Snow100K.pth
====> Save Dir:  ./results_new/LMQFormer/real1000/
Img Number:  2

Peak RSS: 1463.71 MB
Peak VMS: 3691.51 MB
Avg One Img Time: 5.4416s
```

Fig. 4. Performance on CPU

RAM for inference. While this is manageable on devices with sufficient memory, it may be too high for extremely resource-constrained IoT devices.

- **Inference Time**
  The average inference time of **5.4416 seconds** per image indicates that the model is computationally intensive when run on a CPU. This is expected, as CPUs lack the parallel processing capabilities of GPUs. For real-time applications, this latency may be a bottleneck, and optimizations such as model quantization or pruning could help reduce inference time.

*2) **Performance on QCS8550 (Proxy)**:* After optimizing the model for the chip QCS8550 (Proxy) on Qualcomm AI Hub, the performance of the model was as below:

- **Inference**
  The Qualcomm AI Hub provides optimization tailored to the target device, making it an ideal platform for assessing performance in real-world IoT and embedded scenarios.

  **Qualcomm AI Hub:**
  – Target Device: QCS8550 (Proxy)
  – Optimizations: Automatic quantization and hardware-specific optimizations.
  – Use Case: Real-world deployment on embedded systems.


Fig. 5. Performance on QCS8550 (Proxy)

- **Memory Allocation (RAM)**
  Memory usage is drastically reduced to 0-41 MB, making it feasible to deploy the model on low-memory devices without risking system crashes or slowdowns.

- **Inference Time**
  The inference time drops dramatically to **52.2** ms per image, which is **100X** faster than the CPU. This makes the model highly suitable for real-time applications on embedded systems.

TABLE I
PERFORMANCE COMPARISON

| Metric | CPU | Qualcomm AI Hub (QCS8550) |
|---|---|---|
| **Inference Time** | 5.4416 seconds | 52.2 ms |
| **Peak Memory Usage** | 1463.71 MB (RSS) | 0-41 MB |
| **Hardware** | General-purpose CPU | QCS8550 (Proxy) |
| **Optimizations** | None | Chip-specific |

Table I summarizes the results, which demonstrate the significant advantages of using the Qualcomm AI Hub for deploying our model on embedded systems compared to running it on a general-purpose CPU.

### *B. Comparison among different image sizes on Qualcomm AI Hub*

TABLE II
PERFORMANCE COMPARISON FOR DIFFERENT INPUT SIZES

| Input Image | Inference Time(ms) | Peak Memory Usage(MB) |
|---|---|---|
| **128x128** | 16.1 | 0-30 |
| **256x256** | 52.2 | 0-41 |
| **512x512** | 257.4 | 56-134 |

- **Inference Time Analysis**
  Inference time scales almost linearly with image size. Reducing the image size by a factor of 4 (from 512x512 to 256x256) reduces the inference time by approximately **5X**, and further reducing it to 128x128 results in a 16x improvement compared to 512x512.

- **Memory Usage Analysis**
  Memory usage also scales with image size, but the relationship is not strictly linear. Smaller images require significantly less memory, making them more suitable for low-memory devices.

- **Trade-offs:Accuracy vs. Speed**
  Larger images (e.g., 512x512) may provide higher accuracy due to more detailed information, but at the cost of slower inference times and higher memory usage. Smaller images (e.g., 128x128) sacrifice some detail but offer faster processing and lower resource consumption.

These trade-offs will help to select the appropriate image size to optimize performance for the target application and hardware.

## VI. Conclusion

In this report, we evaluated several models for snow removal, focusing on their suitability for deployment on embedded systems. Our initial choices, HDCW-Net and JSTASR, faced significant challenges. HDCW-Net's custom Inverse DWT Layer was incompatible with the ONNX runtime, and attempts to modify it led to poor image quality. JSTASR, while effective, relied on TensorFlow 1.x, making it incompatible with modern platforms like Qualcomm AI Hub. These limitations prompted us to explore LMQFormer, which emerged as a superior choice due to its lightweight design, state-of-the-art performance, and real-time capabilities.

LMQFormer stands out for its lightweight architecture (only 2.181 million parameters) and efficient use of resources, making it ideal for deployment on resource-constrained devices. It achieves top-tier performance on snow removal datasets, with fast inference times ( 52.2 ms for 256x256 images Table II) and minimal memory usage (0-41 MB Table I). Its compatibility with Qualcomm AI Hub further enhances its optimization for a specific chip, enabling it to be deployed in an IOT device. The performance can also be improved via quantization, representing a potential area for future development.

In conclusion, LMQFormer addresses the limitations of other models while offering superior performance, scalability, and efficiency. Its ability to operate in real-time with minimal resource usage makes it an ideal solution for real-world applications such as autonomous driving, surveillance, and beyond.

## References


[1] J. Lin, N. Jiang, Z. Zhang, W. Chen, and T. Zhao, "Lmqformer: A laplace-prior-guided mask query transformer for lightweight snow removal," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 11, pp. 6225–6235, 2023.

[2] Y.-F. Liu, D.-W. Jaw, S.-C. Huang, and J.-N. Hwang, "Desnownet: Context-aware deep network for snow removal," *IEEE Transactions on Image Processing*, vol. 27, no. 6, pp. 3064–3073, 2018.

[3] Z. Li, J. Zhang, Z. Fang, B. Huang, X. Jiang, Y. Gao, and J.-N. Hwang, "Single image snow removal via composition generative adversarial networks," *IEEE Access*, vol. 7, pp. 25 016–25 025, 2019.

[4] W.-T. Chen, H.-Y. Fang, J.-J. Ding, C.-C. Tsai, and S.-Y. Kuo, "Jstasr: Joint size and transparency-aware snow removal algorithm based on modified partial convolution and veiling effect removal," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*. Springer, 2020, pp. 754–770.

[5] W.-T. Chen, H.-Y. Fang, C.-L. Hsieh, C.-C. Tsai, I. Chen, J.-J. Ding, S.-Y. Kuo *et al.*, "All snow removed: Single image desnowing algorithm using hierarchical dual-tree complex wavelet representation and contradict channel loss," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 4196–4205.

[6] T. Ye, S. Chen, Y. Liu, E. Chen, and Y. Li, "Towards efficient single image dehazing and desnowing," *arXiv preprint arXiv:2204.08899*, 2022.

[7] O. Özdenizci and R. Legenstein, "Restoring vision in adverse weather conditions with patch-based denoising diffusion models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 8, pp. 10 346–10 357, 2023.

[8] "Cracking the complexity code in embedded systems development," https://www.mckinsey.com/industries/industrials-and-electronics/our-insights/cracking-the-complexity-code-in-embedded-systems-development, 2022, [Online].

[9] "Qualcomm ai hub." [Online]. Available: https://aihub.qualcomm.com/

[10] "Qualcomm ai hub documentation," https://app.aihub.qualcomm.com/docs/index.html, [Online].

[11] "Attention mechanisms in deep learning: Enhancing model performance," https://medium.com/@zhonghong9998/attention-mechanisms-in-deep-learning-enhancing-model-performance-32a91006092a, 2023, [Online].

[12] "Snow100k," https://paperswithcode.com/dataset/snow100k, [Online].