I. J. Engineering and Manufacturing, 2026, 3, 30-41
Published Online on June 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijem.2026.03.03

Mask-Aware Localized Inpainting Method for CPU-Based Inference

Volodymyr Oliinyk*
Department of Information Systems and Technologies of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, 03056, Ukraine
E-mail: oliinyk.volodymyr@edu.kpi.ua
ORCID iD: https://orcid.org/0000-0002-4647-2658
*Corresponding Author

Serhii Hatsan
Department of Information Systems and Technologies of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, 03056, Ukraine
E-mail: serhiihatcan@gmail.com
ORCID iD: https://orcid.org/0009-0004-5011-0627

Received: 15 January, 2026; Revised: 08 March, 2026; Accepted: 18 May, 2026; Published: 08 June, 2026

Abstract: Image generation methods, including inpainting, are evolving rapidly; however, high memory requirements continue to limit their practical deployment. As a result, the efficient utilization of Latent Diffusion Models on edge devices has become increasingly important. This work explores techniques for reducing memory usage in Latent Diffusion Models while preserving their generative capabilities.
We propose a resource-efficient inpainting method optimized for CPU-based inference, based on a combination of VAE tiling, attention slicing, and dynamic region-of-interest slicing. Experimental results demonstrate that the model's memory footprint can be significantly reduced while maintaining output quality, without substantial increases in computation time, enabling execution on systems with as little as 4 GB of memory and only two processing cores. While the introduced optimizations, particularly those based on localized image processing, introduce an inherent trade-off between memory usage and computational cost, resulting in longer inference times compared to GPU-accelerated solutions, they demonstrate strong potential for deployment in memory-limited environments.
Additionally, we provide analysis of key deployment bottlenecks, including model compilation for cold-start overhead mitigation, proper runtime configuration and scheduler selection. These findings confirm the feasibility of effectively deploying Latent Diffusion Models for inpainting tasks on CPU-only, resource-constrained platforms, thereby broadening their applicability to edge computing scenarios.

Index Terms: Image Inpainting, Edge Computing, Model Compression, CPU Inference, Edge AI, Latent Diffusion Models, Resource-Constrained Systems

1. Introduction

The deployment of large-scale generative models, particularly Latent Diffusion Models (LDMs) such as Stable Diffusion, has traditionally been confined to environments equipped with high-performance Graphics Processing Units (GPUs). The inherent architectural overhead of orchestrating large-scale components such as the U-Net noise predictor and the Variational Autoencoder (VAE) decoder requires extensive matrix multiplication capabilities. As a result, unoptimized pipeline flow on CPU-based execution typically incurs prohibitive latency, making it impractical for real-time generation tasks.

However, the growing demand for widespread AI adoption necessitates efficient inference on edge devices and standard consumer systems that lack specialized accelerators. This study investigates the feasibility of structurally optimizing LDM inference workloads to CPU-based environments. The primary challenge lies in overcoming memory bandwidth limitations inherent to conventional x86 and ARM architectures.

The objective of this research is to identify an optimized pipeline execution strategy and configuration that minimize end-to-end latency – defined as the time required to generate a single image – while preserving 16-bit or 32-bit floating-point precision. This ensures that the visual fidelity of the generated outputs remains comparable to that of

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

GPU-accelerated implementations.

2. Existing Solutions

The evolution of generative image synthesis is increasingly shaped by the growing disparity between escalating model complexity and the demand for computational efficiency on local hardware. While the emergence of Multimodal Diffusion Transformers [1] has set new benchmarks in image fidelity, their practical deployment on edge devices remains dependent on aggressive optimization strategies.

At the architectural level, Transformer-based latent diffusion is exemplified by models such as Stable Diffusion 3.5, which achieve high precision across “Large” and “Medium” variants [2]. Similarly, Black Forest Labs’ FLUX.2 introduces the concept of “Unified Intelligence,” enabling advanced text-to-image and image-to-image editing within a single framework, supporting resolutions of up to 4 MP in the FLUX.2-dev configuration [3]. Other notable approaches include MAT [4], HD-Painter [5], SRInpaintor [6], and LatentPaint [7].

These models typically require two inputs: the original image and a corresponding mask that specifies the regions for modification. Mask generation techniques vary widely in complexity, ranging from manual annotation to automated generation using auxiliary neural networks such as YOLO [8] or SAM [9].

Despite these advancements, high VRAM requirements remain a significant barrier for consumer-grade hardware. Standard execution of models such as FLUX.2-klein-4B or Stable Diffusion 3.5 Medium typically requires 10–13 GB of VRAM. To address this, various quantization techniques have been proposed, including NF4 [10], GGUF [11], SEMQ [12], and QLLM [13]. While these approaches introduce minor precision loss, they substantially reduce memory consumption, enabling sub-second inference for models like FLUX.2-klein-4B with under 6 GB of VRAM. Additionally, knowledge distillation methods, such as Adversarial Diffusion Distillation (ADD) [14], employ a teacher–student paradigm to train smaller models that approximate the performance of larger counterparts with reduced resource requirements. However, training with real data in the loss function may bias the model toward perceptually plausible outputs rather than mathematically accurate reconstructions, potentially leading to contextual hallucinations or high-frequency artifacts.

Efficiency can be further improved through Autoregressive Distillation [15], which leverages trajectory history to better predict subsequent denoising steps, thereby reducing the number of sampling iterations required. This approach, however, introduces additional memory overhead due to key–value (KV) caching, which is necessary for efficient step prediction.

Finally, several approaches [16], [17] focus on improving the efficiency of bridging edge devices with cloud-based generative computation. While such solutions enable complex synthesis on low-specification devices, it introduces dependencies on network connectivity and subscription-based access.

Given that modern generative models often exceed the capabilities of consumer-grade hardware, and that existing optimization techniques involve trade-offs in accuracy or memory usage, targeted refinement of both model architecture and execution pipelines is essential. Such optimization serves as a critical bridge between increasing architectural complexity and the practical constraints of edge-based deployment, enabling high-fidelity, cloud-independent inference on resource-limited systems.

3. Proposed Solution

Traditional inpainting methods are typically based on Latent Diffusion Models (LDMs), which have demonstrated strong performance in both localized image manipulation and global synthesis. As a baseline, we employ the widely used Stable Diffusion 2 Inpainting model, a robust framework that provides a balance between generative quality and structural consistency, and serves as a benchmark for the optimizations proposed in this work.

To enable high-resolution generative synthesis within the constraints of edge-tier hardware, we propose the Mask-Aware Localized Inpainting (MALI) method. This approach bridges the gap between the high computational demands of modern neural architectures and the limited resource capabilities of consumer hardware and embedded systems.

By employing a three-tier optimization pipeline, MALI addresses both stochastic memory spikes and the inherent latency of CPU-bound inference, transforming high-fidelity inpainting into a computationally viable process for non-GPU-accelerated environments. A schematic overview of the optimization pipeline is presented in Figure 1.

High-resolution imagery presents a challenge for Variational Autoencoder (VAE) architectures, as peak memory usage during encoding phase often scales cubically with input dimensions. On hardware with a fixed RAM limits, monolithic tensor processing frequently leads to system instability or process termination.

MALI mitigates this risk in its first optimization tier through VAE tiling, decomposing high-resolution inputs into a set of discrete, overlapping spatial tiles that are processed sequentially. This strategy ensures that peak memory usage remains within a predefined, hardware-constrained limit. To further improve efficiency, attention slicing is employed, partitioning self-attention matrices within transformer blocks into smaller segments. By computing attention in segments rather than as a single large operation, the method avoids the quadratic memory scaling characteristic of standard attention mechanisms. This modular design enables high-fidelity synthesis on devices with limited memory, providing a scalable

solution for high-resolution tasks on low-resource architectures.

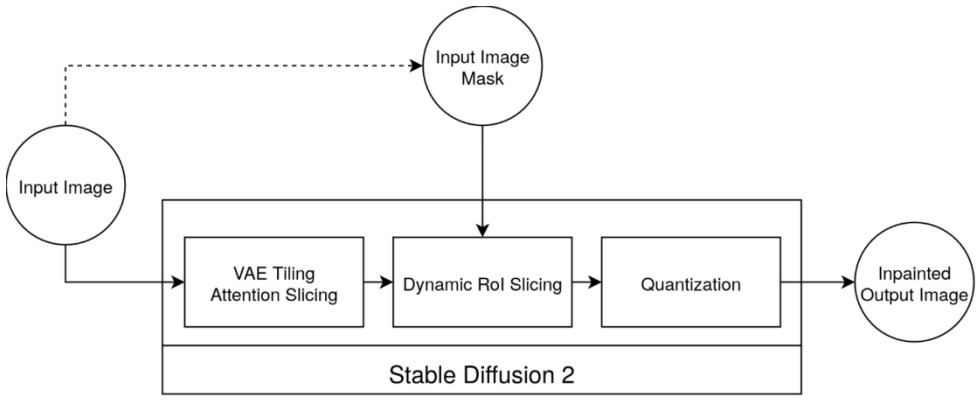


Fig. 1. Architectural diagram of the three-tier MALI optimization pipeline.

The second stage of the pipeline focuses on reducing redundant spatial computation through dynamic Region of Interest (RoI) slicing. Conventional inpainting approaches process the entire image as a uniform tensor, resulting in substantial unnecessary computation in regions unaffected by the target mask, whether defined manually or via external tools [8, 9]. MALI addresses this inefficiency by dynamically identifying the mask coordinates and extracting a localized subregion, augmented with a mathematically defined context buffer around the modification area. By restricting inference to this region, the method significantly reduces the number of floating-point operations per forward pass. This localized strategy ensures that the limited computational capacity of edge CPUs is concentrated on relevant synthesis rather than unchanged background regions.

The third optimization tier in MALI is the application of BF16 post-training quantization. In common deep learning workflows, models are typically used in 32-bit floating-point (FP32) precision to ensure numerical stability during training. However, on resource-constrained devices like Raspberry Pi 4 Model B, the memory requirements of FP32 weights often exceed available capacity.

The use of half-precision FP16 on CPU architectures is generally not feasible due to the lack of native hardware support in general-purpose processors, which necessitates inefficient software emulation and eliminates potential performance gains. In contrast, modern CPU architectures support the Bfloat16 (BF16) format, which retains the dynamic range of FP32 by preserving its 8-bit exponent while reducing mantissa precision to 7 bits. This enables efficient hardware utilization with minimal impact on image synthesis quality, while achieving up to a 50% reduction in memory requirements. Furthermore, BF16 aligns well with specialized instruction sets available in modern processors and can be effectively leveraged through frameworks such as Intel Extension for PyTorch (IPEX) to further improve execution performance.

4. Experimental Setup and Hardware Constraints

To evaluate performance under limited computational resources, a constrained hardware test environment was configured. The host system was equipped with an Intel Core Ultra 7 255HX processor; however, to approximate an edge computing scenario comparable to a Raspberry Pi 4, execution was restricted to two energy-efficient cores operating in power-saving mode, with a maximum clock frequency of 1900 MHz.

All experiments were conducted on a test dataset comprising 500 standardized image-mask pairs (see Fig. 2).

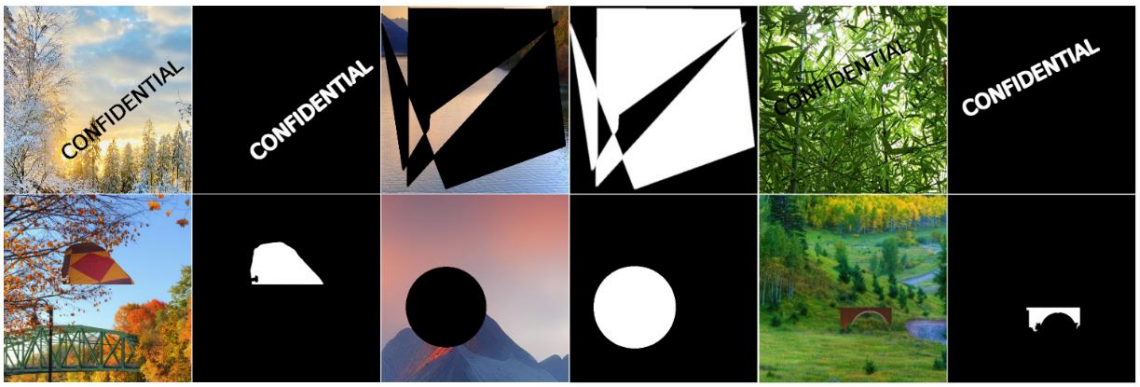


Fig. 2. Examples of image-mask pairs from a synthetically generated dataset used for inpainting task evaluation

The dataset was synthetically generated by compositing a single randomly selected object, together with its corresponding ground-truth mask, from a predefined set of 50 objects onto a randomly selected 512 × 512 background image drawn from a separate collection of 50 representative images. Object selection, background selection, and spatial placement were controlled via a pseudo-random sampling procedure with a fixed random seed to ensure full reproducibility of the dataset generation process. To promote diversity, object instances were placed at varying spatial locations and scales within the background images, subject to boundary constraints.

Nevertheless, the dataset remains subject to certain biases inherent to its synthetic construction, including a limited number of object categories and backgrounds, as well as simplified compositing conditions that may not fully capture real-world variability (e.g., lighting consistency, occlusions, and complex interactions between objects and scenes).

5. Experimental Results

A comparison of generation quality between the standard pipeline and the proposed approach is presented in Table 1. The standard pipeline corresponds to the unmodified Stable Diffusion implementation, instantiated without additional optimizations such as quantization or VAE tiling. For completeness, it is evaluated in both full precision (FP32) and reduced precision (BF16), enabling a direct comparison with the MALI configuration under corresponding precision and quantization settings. The Euler Ancestral Discrete scheduler is used across all configurations.

Table 1. Comparison of generative fidelity metrics (PSNR, SSIM, LPIPS, FID) between the MALI method and the standard pipeline

Model	Precision	PSNR↑	SSIM↑	LPIPS↓	FID↓
Standard Pipeline	FP32	23.88	0.6567	0.1346	31.41
Standard Pipeline	BF16	23.85	0.6545	0.1369	32.66
MALI	FP32	29.61	0.8904	0.0701	27.55
MALI	BF16	**29.58**	**0.8905**	**0.0709**	**27.16**

Quantitative and qualitative analyses of generated images demonstrate that the proposed MALI method preserves the global structural integrity of the input image by strictly constraining synthesis to the localized inpainting region. In contrast, the standard pipeline, which processes the full 512 × 512 image, often introduces stochastic artifacts or minor pixel shifts in non-masked areas (see Figure 3). The proposed method mitigates these unintended modifications by isolating the target region.



Fig. 3. Examples of artifacts, generated by LDM in a standard pipeline compared to MALI method

This architectural constraint is further reflected in the Peak Signal Noise Ratio (PSNR) metrics, where localized processing reduces global noise injection resulting in higher scores for MALI. Moreover, lower Learned Perceptual Image Patch Similarity (LPIPS) values indicate that proposed method produces more coherent and smoothly blended inpainted regions compared to the irregular structures often generated by unconstrained models. Higher Structural Similarity Index Measure (SSIM) scores are primarily due to the preservation of original pixel data outside the region of interest, thereby avoiding VAE reconstruction artifacts typically observed in global inference.

Finally, Fréchet Inception Distance results, computed over whole dataset with identical input data, indicate that the statistical quality of the generated latent space remains comparable to the FP32 baseline, suggesting that the optimization does not significantly degrade the model's generative capability.

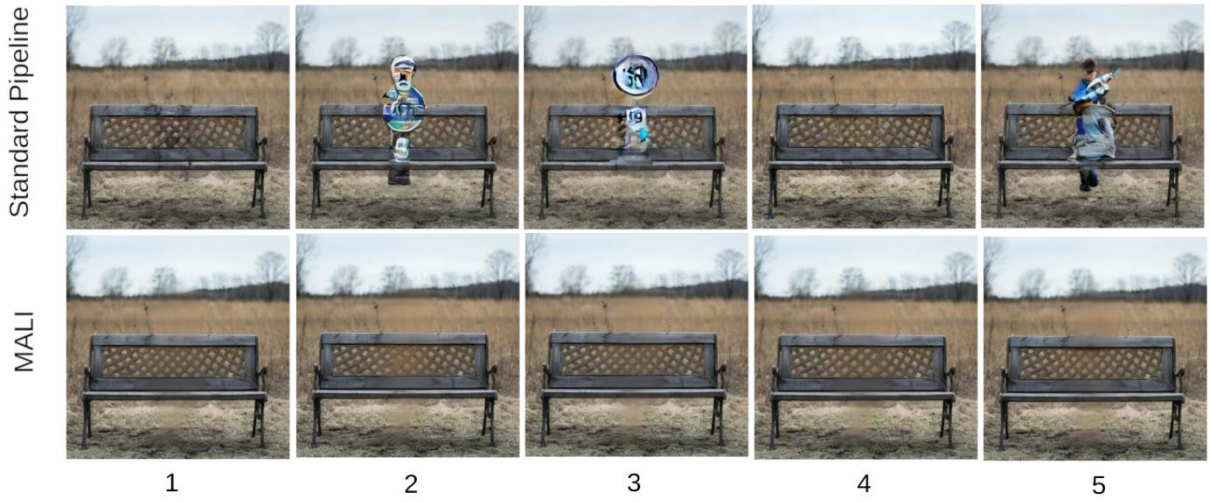


Fig. 4. Generation stability across inpainting steps achieved with the proposed pipeline modifications.

It should be noted that the reported results were obtained using a representative but synthetic dataset. Consequently, metric values in real-world scenarios may be slightly lower due to increased variability and potential biases inherent in synthetic data generation. These limitations should be taken into account when assessing the generalizability of the reported results. In addition, the MALI method was evaluated on real images, as illustrated in Figures 4, 5, and 6.

A side-by-side comparison of inpainting process steps, presented in Figure 4, shows that the MALI method produces properly inpainted images with substantially reduced boundary artifacts and more visually consistent results than the Standard Pipeline in most generations. This consistency is maintained even with Classifier-Free Guidance disabled, forcing the model to rely solely on its learned priors. This setup ensures that the observed stability improvements stem from structural changes rather than prompt-based guidance.

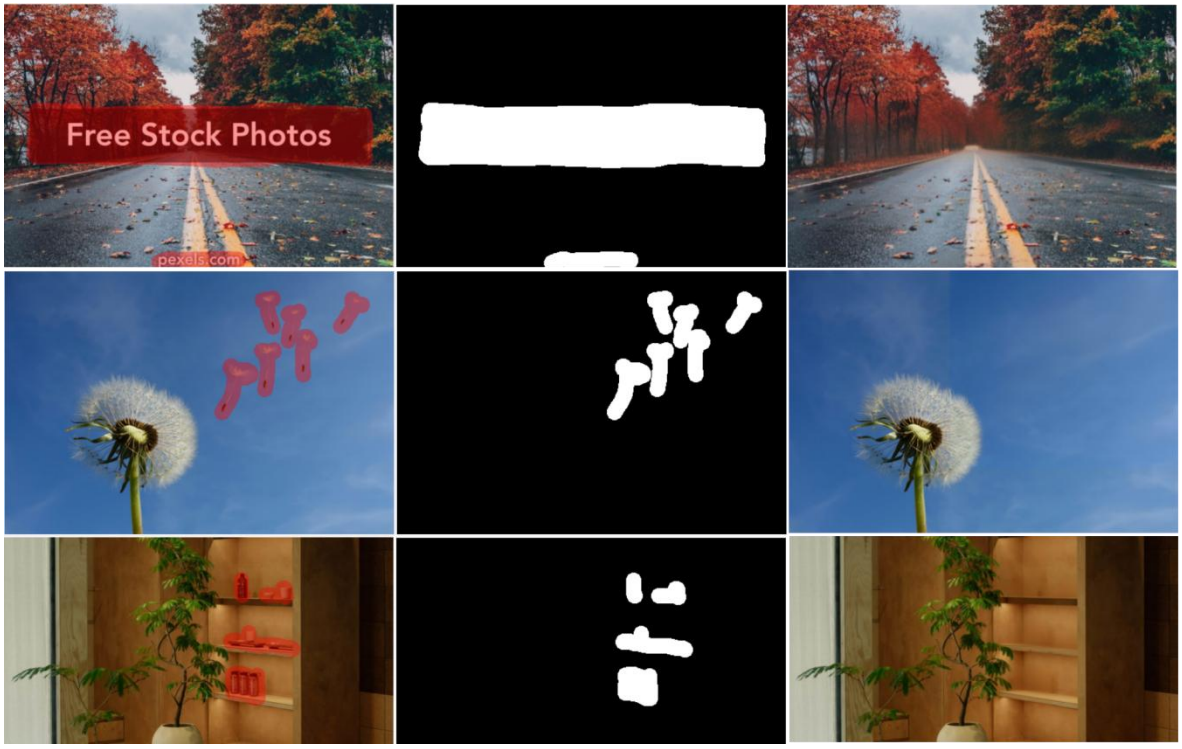


Fig. 5. Examples of object removal in high-texture environments using the proposed MALI method

Additional examples of object removal in high-texture environments are presented in Figure 5. As shown, the proposed localized diffusion method minimizes geometric artifacts common in global CPU-based inference while maintaining background semantic consistency at 512x512 resolution. The process of full object removal for complex images typically involves a greater number of steps. As on Figure 6, even after two sequential passes with 5+ steps there is still some artifacts in object removal.



Fig. 6. Demonstration of complex object removal requiring a greater number of processing steps

Table 2 summarizes key performance metrics, including memory consumption and inference time. The standard pipeline demonstrates the lowest memory usage; however, its five-step inpainting time is approximately twice as long as that of the optimized methods, limiting its practicality in real-world scenarios. Notably, the standard pipeline in BF16 format exhibits the longest total runtime, likely due to the absence of optimized hardware dispatch in a default PyTorch installation.

Table 2. Profiling of system resource utilization, including Peak memory footprint and CPU instruction throughput in resource-constrained environment

Method	Precision	Peak memory footprint	Average iteration time	5-step inpainting time (UNet)	Average total run time
Standard Pipeline	FP32	~5.5 GB	26.28 s/it	2:11	3 m 20 s
Standard Pipeline	BF16	**~3.7 GB**	25.79 s/it	2:08	8 m 18 s
OpenVINO	FP32	~12 GB	10.94 s/it	0:54	2 m 13 s
OpenVINO	BF16	~9.1 GB	**10.78 s/it**	**0:53**	**1 m 50 s**
MALI	FP32	~8.7 GB	13.01 s/it	1:05	2 m 14 s
MALI	BF16	~4 GB	68.53 s/it	5:42	12 m 07 s
MALI + IPEX	BF16	**~4 GB**	**11.42 s/it**	**0:57**	**2 m 13 s**

In contrast, OpenVINO-based approaches achieve the lowest latency, but their memory footprints (12 GB for FP32 and 9.1 GB for FP16) effectively preclude deployment on 4 GB or even 8 GB edge devices.

The proposed MALI approach provides a balanced trade-off between the standard pipeline and OpenVINO-optimized variants, reducing memory requirements to approximately 4 GB while maintaining competitive inpainting performance.

Furthermore, adopting BF16 quantization reduces peak RAM usage from 8.7 GB to approximately 4 GB (a 54% reduction) without increase of inpainting time when combined with hardware-specific optimizations such as Intel Extension for PyTorch (IPEX). Disabling IPEX results in a substantial performance degradation, increasing inpainting time by approximately fivefold due to reliance on software-based execution.

Overall, the experimental results demonstrate that the MALI optimization pipeline preserves high generative quality, mitigates key limitations of global diffusion-based approaches, and significantly reduces memory bandwidth demands while maintaining competitive execution time.

6. In-Depth Results Analysis

6.1. Torch compilation and Gradient Calculations

Figure 7 illustrates memory utilization and performance characteristics of the standard FP32 pipeline during a five-step inference process in an object removal scenario. A pronounced *cold-start* effect is observed, resulting in substantial initialization latency (approximately 60 seconds within a total runtime of around 3 minutes), followed by the main diffusion stage and a significant post-processing overhead (around 30 seconds)

Although the sources of this latency are well understood – including library initialization (notably PyTorch), model weight loading, and graph compilation – this study specifically evaluates the impact of TorchDynamo just-in-time (JIT) compilation on this phenomenon. Performance across primary Torch compilation modes (*reduce-overhead*, *default*, and *max-autotune*) was assessed and yielded largely similar results (see Table 3). Execution time was measured from program start to completion using the Linux time utility, with results averaged over ten runs.

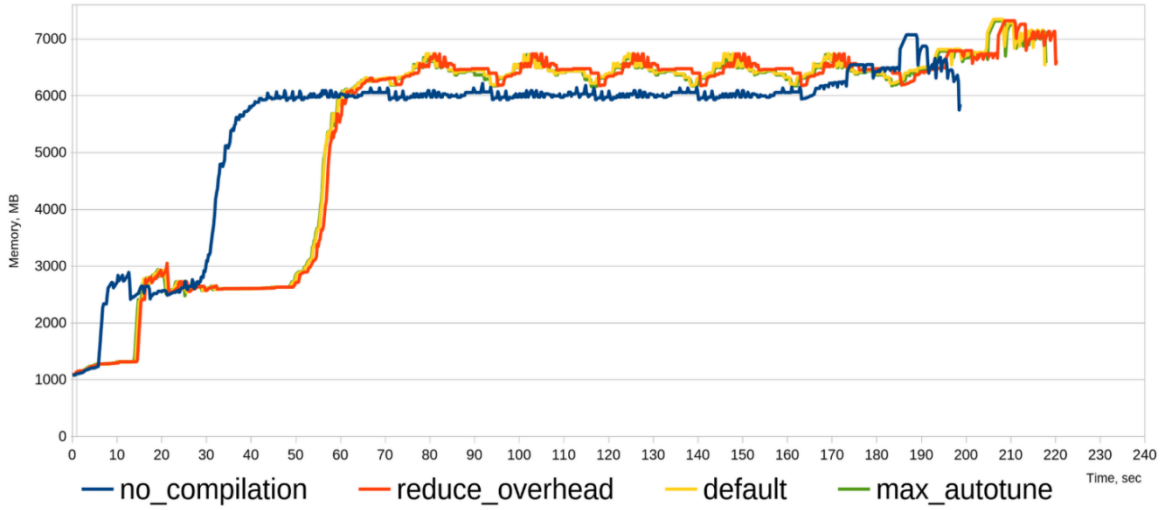


Fig. 7. Impact of different PyTorch compilation approaches on the processing time and memory usage

Notably, execution without compilation achieved the lowest average runtime (3 minutes 17 seconds), outperforming compiled variants. This result can be attributed to the non-amortized cost of the JIT warm-up phase. While compilation slightly reduced the duration of the main diffusion stage, the overall benefit was negligible. Consequently, enabling Torch compilation is not advantageous for single-image inference in resource-constrained environments and may only be beneficial in batch-processing scenarios or persistent runtime settings (see Section 6.4). Disabling compilation in our setup reduced warm-up time by approximately 25 seconds.

Additionally, a consistent post-processing overhead of approximately 30 seconds was observed at the end of execution, during which the framework finalizes computations and writes outputs to disk. This teardown latency remained constant and was unaffected by subsequent optimizations.

Memory usage remained relatively stable across configurations, with approximately 6.8 GB of RAM consumed under compiled modes and around 6.1 GB without compilation (see Figure 7). Although the non-compiled configuration initially exhibited lower memory usage, peak consumption during the final decoding stage (when images are reconstructed from the latent space) approached levels observed in compiled configurations.

From a memory-efficiency perspective, Torch compilation should be applied cautiously, or avoided, in such scenarios.

Table 3. Evaluation of end-to-end inference latency across standard and JIT-compiled execution backends

Gradient calculations	Reduce-overhead	Default	Max-autotune	Without compilation
Enabled	03:43.66	03:42.69	03:42.68	**03:17.09**
Disabled	03:29.18	03:39.46	03:43.29	**02:58.32**

Gradient computation is another important aspect in PyTorch that should be disabled during inference, as it is unnecessary and introduces avoidable computational overhead. Disabling gradient calculations resulted in a modest performance improvement in the non-compiled configuration, reducing the average execution time for a five-step inpainting task to 2 minutes 58 seconds – an 18-second improvement compared to runs with gradient computation enabled.

A comparative summary of execution times is provided in the Table 3.

6.2. Impact of Architectural Improvements on Performance

During testing, it was hypothesized that enabling VAE tiling and attention slicing would improve runtime performance on CPU-based architectures. To evaluate this assumption, a standard FP32 pipeline configuration was used as a baseline. The results in Figure 8 indicate that, when applied individually, both techniques degrade execution efficiency. Specifically, VAE tiling increases the average execution time to 209 s, while Attention Slicing further degrades performance to 225 s. However, the combined use of both techniques (without compilation) only increases the inference time up to 219 s, partially validating the proposed architectural optimizations.

In terms of memory utilization, both individual VAE Tiling, attention slicing, and the combined configurations exhibit the highest memory consumption during the U-Net processing phase at approximately 6 Gb, whereas VAE tiling results in comparatively lower memory usage during the final stage. All evaluated configurations demonstrate peak RAM utilization during the post-processing stage.

Given that the reduction in memory consumption against standard pipeline setup is marginal, roughly equivalent to the model size, the combined use of both techniques remains the more practical strategy for edge systems. While

individual optimizations may appear leaner in the current results, the combined approach prevents the localized memory bottlenecks from appearing. This configuration provides a balanced trade-off between memory efficiency and computational performance, enabling sufficient throughput for real-time applications while operating within the constraints of resource-limited environments.

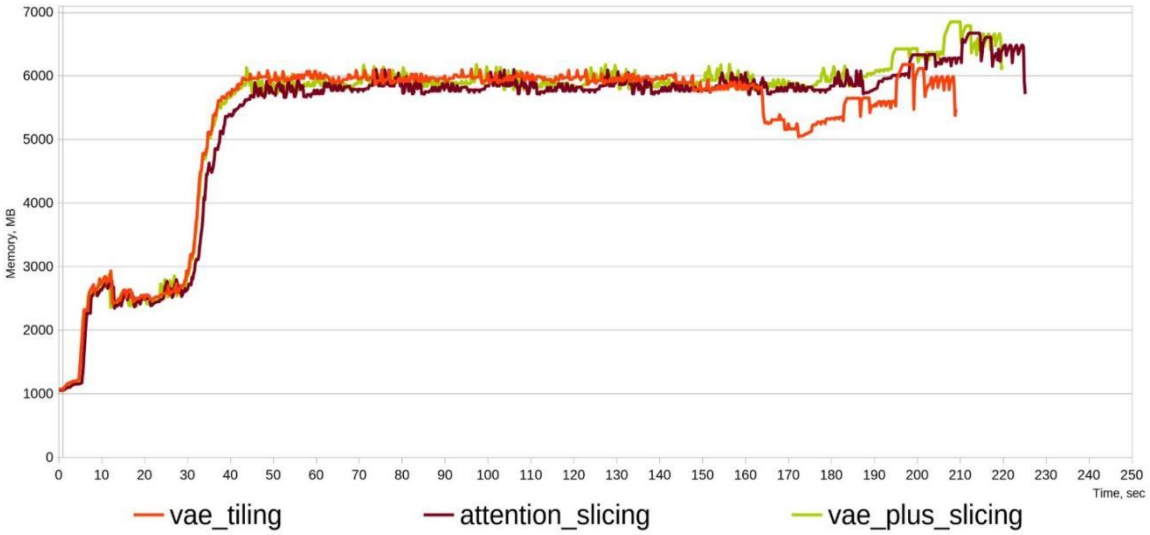


Fig. 8. Impact of VAE Tiling and Attention Slicing on peak memory consumption and performance

Additional experiments evaluating compilation modes for the VAE, U-Net, and text encoder modules (including *reduce-overhead*, *default* and *max-autotune*) show negligible impact on mitigating the cold-start effect.

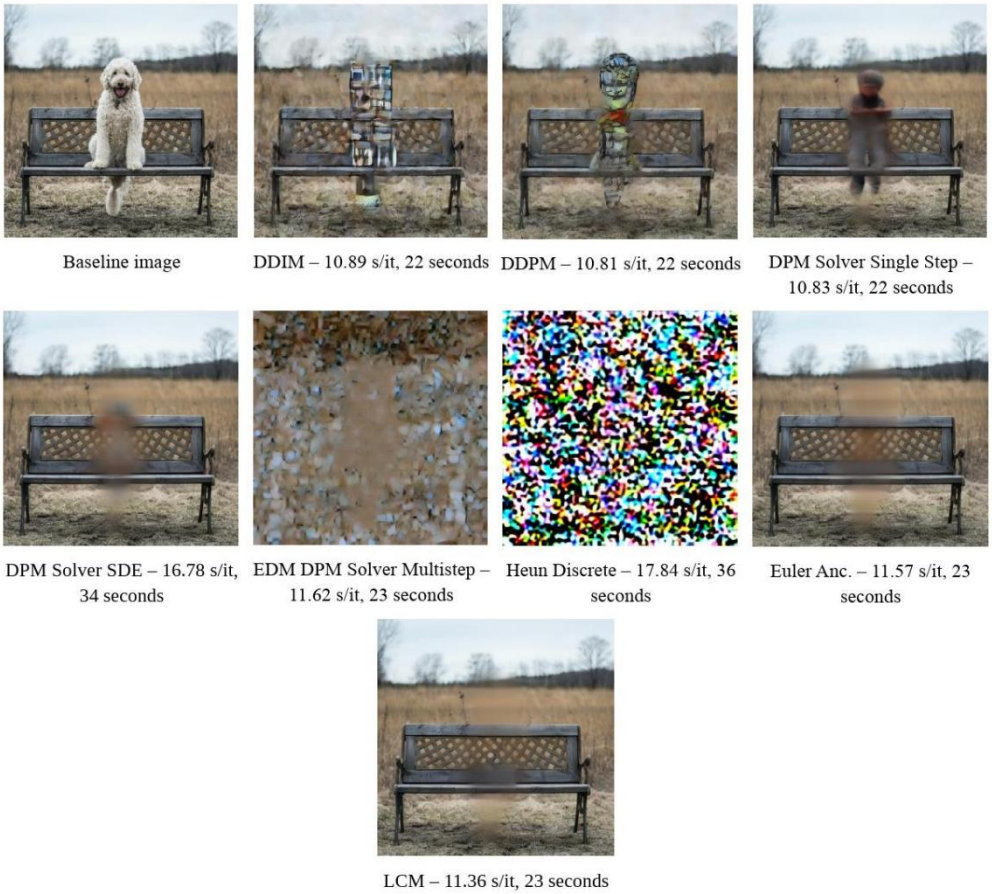


Fig. 9. Comparison of scheduler performance and resulting outputs after two-step inference

6.3. Scheduler Optimization

Schedulers are among the most critical, yet often underappreciated, components in LDMs [18]. To assess their impact on the inpainting pipeline, a set of widely used schedulers was evaluated. The results in Figure 9 present their performance for the MALI BF16 pipeline in a two-step object removal task. As shown, the choice of scheduler significantly affects image quality, convergence speed, and stability. While schedulers such as DDIM, DDPM, and single-step DPM achieve competitive inference times, they produce unacceptable output quality under low-step conditions.

The Euler Ancestral Discrete and Latent Consistency Model (LCM) schedulers demonstrate the best runtime performance in the evaluated scenario, completing a single iteration in 11.36-11.57 s. Both schedulers also achieve the highest visual quality among the tested options, with only minor differences.

The Euler Ancestral Discrete scheduler is used as the default configuration in all experiments (see Table 1), as it generally provides better results for complex, highly detailed images. Additionally, generation quality metrics are evaluated for the MALI BF16 configuration using the LCM scheduler. LCM demonstrates better PSNR (30.30) and SSIM (0.8940) scores, but worse LPIPS (0.0789) and FID (30.28) compared to the Euler Ancestral Discrete scheduler.

However, an important limitation of the Euler Ancestral Discrete scheduler is its inability to produce acceptable outputs in a single step, which may be critical for certain edge application scenarios; at least two inference steps are required (see Fig. 10.).



Fig. 10. Difference between 1-step and 2-step inference for Euler Ancestral Discrete scheduler

6.4. Persistent Runtime Environments and Classifier-Free Guidance

As discussed in Section 6.1, initialization latency in cold-start scenarios can be substantial. Transitioning from a conventional script-based execution model to a persistent runtime environment (e.g., Jupyter Notebook) significantly reduces the Time-to-First-Inference (TTFI) for subsequent runs. The use of a persistent kernel ensures that model components remain resident in memory, thereby eliminating repeated initialization overhead.

In this experiment, we assess the impact of environment persistence specifically on the main diffusion process. Experimental results show a TTFI reduction of approximately 2-3 s for a few-step inference in FP32 configurations (see Figure 11). The improvement is less pronounced for BF16 precision (below 1 s), but remains beneficial in scenarios involving multiple consecutive inferences.

The final optimization involves disabling Classifier-Free Guidance (CFG) by setting the guidance scale parameter to 1.0. In standard diffusion pipelines, CFG values greater than 1.0 require two forward passes per inference step: one conditioned on the input prompt and one unconditioned (null-prompt). Disabling CFG enables single-pass inference, eliminating redundant computations and achieving a more than twofold increase in generation speed (see Figure 11). Although higher CFG values generally improve prompt fidelity, experimental results indicate that, for object removal tasks with a given mask, contextual information from unmasked regions is sufficient to ensure convergence. Furthermore, the configuration with Classifier-Free Guidance (CFG) disabled achieves the lowest TTFI among all evaluated settings.

Under these conditions (BF16, CFG off), with IPEX enabled for hardware-aware optimization, each inference step completes in under 10 seconds in the evaluated environment. This configuration achieves the best performance among the tested setups and meets the requirements for practical real-world deployment.

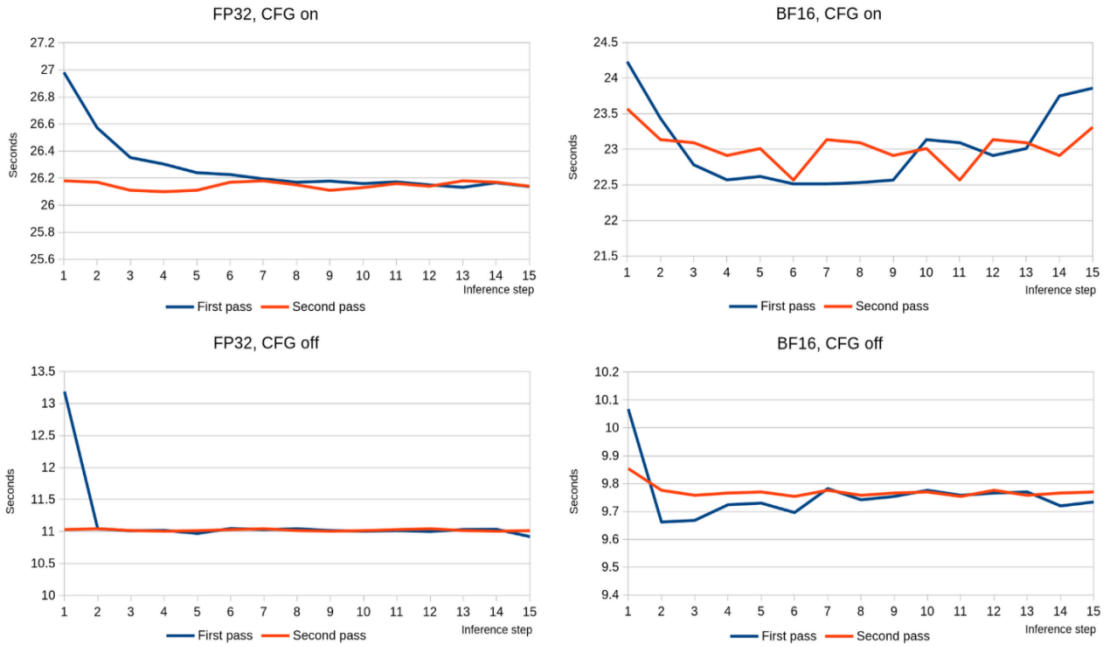


Fig. 11. Average inference time per step for the first and subsequent 15-step inpainting runs for MALI in FP32 and BF16 precision, with and without Classifier-Free Guidance

7. Summary

Overall, this work provides a practical roadmap for constructing an efficient inpainting pipeline that achieves a 4 GB memory footprint and an inference time per step of less than 10 seconds on a dual-core 1900 MHz CPU. These results demonstrate the feasibility of deploying high-fidelity Latent Diffusion Models on resource-constrained and edge CPU devices.

However, such deployment requires careful configuration and optimization of the execution environment, including judicious use of Torch compilation, disabling gradient computation, and deactivating classifier-free guidance, to minimize overhead. The results indicate that transitioning from generic software execution to hardware-aware dispatch, particularly through Intel Extension for PyTorch, can reduce inference time by a factor of 2-5x, especially for BF16 precision. Although the BF16 MALI configuration with IPEX and disabled CFG achieves the lowest Time-to-First-Inference among the evaluated settings, the use of a persistent runtime environment can further mitigate initialization overhead in subsequent runs and cold-start problem.

Beyond raw performance, the study highlights the importance of architectural strategies, including dynamic region-of-interest processing, VAE tiling, attention slicing, and proper Scheduler selection, as essential components for practical edge deployment. Empirical evaluation confirms that the proposed pipeline optimizations preserve high generative quality. Furthermore, constraining the denoising process to localized regions yields a 25.3% improvement in SSIM and reduces perceptual error, achieving a Learned Perceptual Image Patch Similarity score of 0.0709 compared to 0.1346 for baseline global approaches.

These findings suggest that localized processing is a promising optimization strategy and may, in certain scenarios, provide greater benefits than conventional global model compression techniques alone, thereby warranting further investigation.

All the Declarations and Statements

Author Contribution Statement

Serhii Hatsan – Methodology, data curation and software implementation: handled data acquisition, data set preprocessing and implementing research pipeline. Validated results using standard metrics and benchmarked performance against existing solutions. Drafted initial manuscript. Review and editing, project management: reviewed and edited the manuscript, ensured clarity and coherence.

Volodymyr Oliinyk – Conceptualization, Supervision: proposed research ideas and methodology, supervised project execution and results. Contributed to the literature survey, reviewed and edited the manuscript, ensured clarity and coherence.

All authors have read and agreed to the published version of manuscript.

Conflict of Interest Statement
The Authors declared no conflicts of interest.

Funding Declaration
This research received no external funding.

Data Availability Statement
The data that support the findings of this study are from the corresponding author upon reasonable request.

Ethical Declarations
This study did not involve human participants or animals.

Acknowledgments
We sincerely thank the experts for their professional evaluation and valuable recommendations, which have contributed to improving the quality of the experiment and the reliability of its results.
Special thanks to Igor Sikorsky Kyiv Polytechnic Institute for the continuous support without which we cannot done this work.

Declaration of Generative AI in Scholarly Writing
During the preparation of the manuscript, the authors carefully reviewed the content and take full responsibility for the accuracy and integrity of the manuscript.

Abbreviations
The following abbreviations are used in this manuscript:
GPU – Graphics Processing Unit
CPU – Central Processing Unit
LDM – Latent Diffusion Model
MALI – Mask Aware Localized Inpainting
VAE – Variational AutoEncoder
RAM – Random Access Memory
RoI – Region of Interest
FLOP – Floating point operation
PSNR – Peak Signal-to-Noise Ratio
SSIM – Structural Similarity Index Measure
LPIPS – Learned Perceptual Image Patch Similarity
FID – Fréchet Inception Distance
IPEX – Intel PyTorch Extensions
JIT – Just-in-Time
LCM – Latent Consistency Model
I/O – input-output
GB – GigaByte

APPENDIX
None

References

[1] Reuss, M., Yağmurlu, Ö. E., Wenzel, F., & Lioutikov, R. (2024). Multimodal diffusion transformer: Learning versatile behavior from multimodal goals. arXiv preprint arXiv:2407.05996.
[2] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10684-10695).
[3] Black Forest Labs. (2025). FLUX.2: Analyzing and enhancing the latent space of FLUX -- representation comparison. https://bfl.ai/research/representation-comparison
[4] Li, W., Lin, Z., Zhou, K., Qi, L., Wang, Y., & Jia, J. (2022). Mat: Mask-aware transformer for large hole image inpainting. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10758-10768).
[5] Manukyan, H., Sargsyan, A., Atanyan, B., Wang, Z., Navasardyan, S., & Shi, H. (2023). Hd-painter: high-resolution and prompt-faithful text-guided image inpainting with diffusion models. In The Thirteenth International Conference on Learning

Representations.
[6] Li, F., Li, A., Qin, J., Bai, H., Lin, W., Cong, R., & Zhao, Y. (2022). SRInpaintor: When super-resolution meets transformer for image inpainting. IEEE Transactions on Computational Imaging, 8, 743-758.
[7] Corneanu, C., Gadde, R., & Martinez, A. M. (2024). Latentpaint: Image inpainting in latent space with diffusion models. In Proceedings of the IEEE/CVF winter conference on applications of computer vision (pp. 4334-4343).
[8] Sapkota, R., Flores-Calero, M., Qureshi, R., Badgujar, C., Nepal, U., Poulose, A., ... & Karkee, M. (2025). YOLO advances to its genesis: A decadal and comprehensive review of the You Only Look Once (YOLO) series. Artificial Intelligence Review, 58(9), 274. DOI: https://doi.org/10.1007/s10462-025-11253-3
[9] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., ... & Girshick, R. (2023). Segment anything. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 4015-4026).
[10] Yoshida, D. (2023). NF4 Isn't Information Theoretically Optimal (and that's Good). arXiv preprint arXiv:2306.06965.
[11] Tripathi, O. M. (2024). "GGUF Models and Quantization". DOI: http://dx.doi.org/10.2139/ssrn.5044207
[12] Li, D., Xie, X., Zh11ang, D., Vasilakos, A. V., & Leung, M. F. (2025). SEMQ: Efficient non-uniform quantization with sensitivity-based error minimization for large language models. Future Generation Computer Systems, 108120.
[13] Liu, J., Gong, R., Wei, X., Dong, Z., Cai, J., & Zhuang, B. (2023). Qllm: Accurate and efficient low-bitwidth quantization for large language models. arXiv preprint arXiv:2310.08041.
[14] Sauer, A., Lorenz, D., Blattmann, A., & Rombach, R. (2024, September). Adversarial diffusion distillation. In European Conference on Computer Vision (pp. 87-103). Cham: Springer Nature Switzerland.
[15] Kim, Y., Anagnostidis, S., Du, Y., Schönfeld, E., Kohler, J., Georgopoulos, M., ... & Sanakoyeu, A. (2025). Autoregressive distillation of diffusion transformers. In Proceedings of the Computer Vision and Pattern Recognition Conference (pp. 15745-15756).
[16] Wei, Y., Tang, S., Zhao, L., & Yang, Q. (2025). DiffusionX: Efficient Edge-Cloud Collaborative Image Generation with Multi-Round Prompt Evolution. arXiv preprint arXiv:2510.16326.
[17] Yan, C., Liu, S., Liu, H., Peng, X., Wang, X., Chen, F., ... & Mei, X. (2024). Hybrid sd: Edge-cloud collaborative inference for stable diffusion models. arXiv preprint arXiv:2408.06646.
[18] Sabour, A., Fidler, S., & Kreis, K. (2024). Align your steps: Optimizing sampling schedules in diffusion models. arXiv preprint arXiv:2404.14507.

Authors' Profiles



Volodymyr Oliinyk received his PhD in 2011 from Igor Sikorsky Kyiv Polytechnic Institute, Ukraine. He is currently working as Associate Professor in the Department of Information Systems and Technologies at Igor Sikorsky Kyiv Polytechnic Institute, Ukraine.
His research interests include generative AI and deep learning, computer vision and natural language processing, agentic AI.



Serhii Hatsan is B.Sc. in Information Systems and Technologies and is currently pursuing the M.Sc. degree in Igor Sikorsky Kyiv Polytechnic Institute, Ukraine. Currently working as Network and Security Specialist at SPAN, Ukraine. His research interests include IOT, Embeded Systems, AI, Computer Vision

How to cite this paper

Volodymyr Oliinyk, Serhii Hatsan, "Mask-Aware Localized Inpainting Method for CPU-Based Inference", International Journal of Engineering and Manufacturing (IJEM), Vol.16, No.3, pp.30-41, 2026. DOI:10.5815/ijem.2026.03.03