Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

SACHITRA: Scalable Application of CPU-based HTML Integrated Transformer for Regenerative Augmented Image Generation Model

SHASHWAT PANDEY

info.shashwatpandey@gmail.com

Indian Institute of Technology Madras https://orcid.org/0009-0005-3946-6611

Research Article

Keywords: Flow Matching, CPU Inference, WebAssembly, HTML5 AI Pipeline, Vision Transformer, Quantization, Edge AI, Browser-Native Generation, Multi-Armed Bandit, Attention Propagation, SACHITRA, Aatmanirbhar Bharat

Posted Date: May 14th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9702958/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

SACHITRA: Scalable Application of CPU-based HTML Integrated Transformer for Regenerative Augmented Image Generation Model

Shashwat Pandey
Digital India BHASHINI Division
Ministry of Electronics & Information Technology
Government of India
New Delhi, India
shashwat.dibd@gmail.com

***Abstract*—We present SACHITRA (Scalable Application of CPU-based HTML Integrated Transformer for Regenerative Augmented Image Generation Model), a generative image and video synthesis framework engineered for browser-native, CPU-bounded execution without dependence on dedicated GPU hardware. Existing state-of-the-art synthesis pipelines—including flow-matching models, attention-propagating Vision Transformers (ViTs), and large-scale diffusion architectures—achieve high perceptual fidelity but demand GPU clusters or NVIDIA A100-class accelerators, rendering them inaccessible to the vast majority of global computing endpoints, particularly in emerging economies.**

SACHITRA addresses this accessibility gap through three formally grounded contributions: (1) a Quantized HTML Pipeline (QHP), which orchestrates compressed generative sub-networks within an HTML5/WebAssembly execution context, exploiting browser SIMD intrinsics and SharedArrayBuffer for intra-session parallelism; (2) an Adaptive Finite-Difference Trajectory Approximation (AFDT), a training-free ODE acceleration method integrated with a UCB-1 multi-armed bandit (MAB) controller that dynamically skips redundant integration steps with formal error guarantees of order $O(|S|/T^3)$ under INT4 quantized arithmetic; and (3) a Cross-Layer Historical Attention Distillation (CHAD) mechanism that propagates compressed 4-bit integer attention matrices across encoder layers, reducing attention buffer memory by $16\times$ while sustaining accuracy through an analytically derived dynamic blending schedule. Experiments demonstrate a $2.1\times$–$3.4\times$ speedup over CPU-only diffusion baselines on commodity hardware (Intel Core i5, 8 GB RAM), with CLIP-IQA scores within 4.7% of GPU-resident reference models. SACHITRA constitutes the first formally verified, browser-deployable generative synthesis pipeline designed for the Aatmanirbhar Bharat digital infrastructure mandate.

***Index Terms*—Flow Matching; CPU Inference; WebAssembly; HTML5 AI Pipeline; Vision Transformer; Quantization; Edge AI; Browser-Native Generation; Multi-Armed Bandit; Attention Propagation; SACHITRA; Aatmanirbhar Bharat**

I. Introduction

The proliferation of generative artificial intelligence for image and video synthesis has been accompanied by an exponential growth in computational demand. Contemporary generative models—encompassing flow-matching architectures [4], [20], diffusion-based models [22], and transformer-based decoders [10]—routinely require 40–80 GB of GPU VRAM and multi-thousand-watt accelerator clusters for real-time inference. This creates a structural divide in which the benefits of generative AI are concentrated among well-resourced laboratories and commercial cloud providers, while individual developers and edge deployment contexts remain excluded.

India presents a paradigmatic case. With over 820 million internet users, a majority access the web through mid-range smartphones or sub-$300 computers [25]. The Government of India's MeitY mandate for indigenous digital infrastructure (Digital India, Aatmanirbhar Bharat) creates a compelling imperative for inference frameworks that operate within the constraints of consumer-grade processors, commodity RAM, and browser-based execution environments.

The technical challenge is substantial. Generative synthesis via flow-matching requires solving a time-continuous ordinary differential equation (ODE) governed by a neural velocity field $v : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$. Each integration step involves a full forward pass through a deep transformer network, typically comprising billions of floating-point multiply-accumulate operations. On CPU hardware constrained to 64-bit SIMD lanes (AVX-512) without tensor cores, this translates to latencies of 30–120 seconds per 512×512 image—far exceeding interactive thresholds.

This paper presents SACHITRA (Sanskrit-rooted acronym: *sa-chitra* also means "with image"), an end-to-end generative synthesis framework architected for browser-native, CPU-bounded execution. SACHITRA integrates three formally grounded innovations:

1) An adaptive ODE trajectory approximator governed by a multi-armed bandit, building on FastFlow [1];
2) A cross-layer compressed attention propagation system extending HAViT [2];
3) A quantized HTML pipeline that compiles the complete inference graph to a portable, single-file WebAssembly artifact.

Together these contributions enable practical generative synthesis on commodity hardware, without GPU dependency, cloud connectivity, or software installation.

A. Contributions

1) **Quantized HTML Pipeline (QHP):** A compilation and runtime framework encoding compressed generative model weights within structured HTML5 data attributes, executing inference via WebAssembly SIMD kernels, and orchestrating parallel sub-model inference through SharedArrayBuffer-backed Web Worker pools.
2) **Adaptive Finite-Difference Trajectory Approximation (AFDT):** A training-free ODE acceleration method using first-order Taylor expansion with finite-difference velocity estimation to skip redundant integration steps, governed by a UCB-1 multi-armed bandit with formally bounded trajectory deviation $e_T = O(|S|/T^3)$, adapted for INT4-quantized arithmetic.
3) **Cross-Layer Historical Attention Distillation (CHAD):** A 4-bit compressed attention propagation mechanism blending previous-layer attention logits with current-layer self-attention via a dynamically scheduled coefficient, maintaining an attention memory at 16× reduced memory footprint with analytically guaranteed distortion bounds.
4) **Formal Error Guarantees:** Two theorems establishing end-to-end error bounds for AFDT and CHAD under quantized arithmetic, stated in a form amenable to independent verification.
5) **SACHITRA Application Platform:** A complete, browser-native deployment validated on GenEval [16], CIFAR-100 [26], and TinyImageNet [27] benchmarks.

II. Related Work

A. Flow-Matching Generative Models

Flow Matching (FM; [4]) provides a deterministic framework for generative modelling by learning a continuous-time velocity field that transports samples from a source distribution π_0 to a target distribution π_1. Unlike score-based diffusion models, FM trains directly on conditional vector fields and achieves straighter sampling trajectories, permitting high-quality generation with fewer function evaluations [20], [21].

Acceleration of FM inference has proceeded along several directions. Static methods include knowledge distillation [8], [9], piecewise rectification [7], and higher-order ODE solvers [19]. Dynamic caching methods such as TeaCache [6] reuse intermediate representations across timesteps using handcrafted polynomial criteria.

FastFlow [1] most directly informs our work: it introduces a multi-armed bandit to adaptively skip ODE steps using finite-difference velocity approximation, demonstrating 2.6× speedup on GPU hardware. SACHITRA extends this framework to the fundamentally different regime of integer-quantized CPU inference within a browser WebAssembly environment, requiring new analysis of error accumulation under quantization noise (Section IV-B).

B. Vision Transformers and Attention Mechanisms

Vision Transformers (ViTs; [10]) apply self-attention to non-overlapping image patches, enabling long-range dependency modelling. Architectural variants have explored efficiency [11], [15], multi-scale attention [14], and cross-layer information reuse [13].

HAViT [2] introduces a cross-layer historical attention propagation mechanism that blends the self-attention matrix $A'_{self} = Q_l K_l^T / \sqrt{d_k}$ with the attention history H'_{prev} using a fixed coefficient α, demonstrating consistent accuracy improvements of +1.01% to +1.33% on CIFAR-100 and TinyImageNet. SACHITRA adapts this mechanism to 4-bit integer space, introduces dynamic α scheduling (Section IV-C), and deploys it within the broader QHP inference graph.

C. On-Device and Browser-Native AI Inference

Browser-native inference has been enabled by WebAssembly (WASM) [23], which provides near-native execution speed through SIMD intrinsics, and by WebGPU [24], which exposes GPU compute shaders to web applications. Transformers.js (Hugging Face) and ONNX Runtime Web (Microsoft) provide JavaScript-accessible inference runtimes. For flow-matching models, Low-Bit High-Fidelity quantization via Optimal Transport [17] demonstrates near-lossless 4–8 bit quantization.

No prior work integrates adaptive step-skipping, cross-layer attention propagation, and formal error bounding into a single browser-deployable generative synthesis pipeline. SACHITRA fills this gap.

D. AI for Mathematical and Scientific Knowledge

The broader AI4Math landscape [3] has demonstrated that foundation models can achieve graduate-level mathematical reasoning when equipped with formal verification environments and agentic workflows. This motivates SACHITRA's commitment to formally stated error bounds, consistent with the emerging standard of certified AI systems.

III. Problem Formulation and System Overview

A. Formal Problem Statement

Let $\mathbf{M}$ a trained flow-matching generative model parameterized by a deep transformer network with L encoder layers, d_{model}-dimensional embeddings, and K ODE integration timesteps. Standard inference requires K full forward passes through $\mathbf{M}$, each of complexity $O(n^2 d_{model})$, where n is the sequence length. On a CPU-only device with throughput B FLOPS, total latency is:

$$T_{latency} = \frac{K \cdot n^2 \cdot d_{model} \cdot L}{B}. \quad (1)$$

For a representative configuration (K=50, n=256, d_{model}=768, L=12, B=100 GFLOPS), this yields $T_{latency} \approx 118$ seconds—impractical for interactive use.

SACHITRA seeks a modified inference procedure $\mathbf{M}'$ satisfying:

$$\text{(i)} \quad T_{latency}(\mathbf{M}') \le \frac{T_{latency}(\mathbf{M})}{\gamma}, \quad \gamma \ge 2.0, \quad (2)$$

$$\text{(ii)} \quad \| x_T^{approx} - x_T^{true} \| \le \varepsilon, \quad \varepsilon \text{ user-specified}, \quad (3)$$

$$\text{(iii)} \quad \mathbf{M}' \text{ executes entirely in browser/WASM}, \quad (4)$$

without GPU access, cloud round-trips, or software installation.

TABLE I
SACHITRA SYSTEM COMPONENTS

Component	Function	Technology	Overhead
MCM	Quantization, WASM codegen	OT-Quant, emscripten	One-time offline
AFDT Ctrl	Adaptive ODE skip	UCB-1, FD approx.	<0.1 ms/step
CHAD Mod	Attention propagation	INT4 buffer, blend	$O(Bhn^2)$ INT4
BRE	Worker orchestration	Web Workers, SAB	Negligible

B. System Architecture

SACHITRA consists of four interlocking components summarized in Table I. The Model Compilation Module (MCM) accepts a pre-trained FM backbone and applies OT-Quantization (INT4/INT8), structural pruning, and WASM kernel generation. The resulting compressed model is encoded as base64 binary within HTML5 `<data>` attributes—constituting the QHP.

The AFDT Controller governs ODE integration at runtime via UCB-1 bandit-selected skip lengths. The CHAD Module manages the cross-layer INT4 attention memory buffer. The Browser Runtime Engine (BRE) orchestrates execution across Web Workers using SharedArrayBuffer for zero-copy inter-worker communication.

IV. METHODOLOGY

A. Flow Matching Background

Given source distribution π_0 (standard Gaussian) and target distribution π_1 (data), FM learns a time-dependent velocity field $v : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ satisfying the ODE:

$$\frac{dx_t}{dt} = v(x_t, t), \quad x_0 \sim \pi_0, \quad t \in [0,1]. \tag{5}$$

Forward Euler discretization on grid $\{t_0, t_1, \ldots, t_K\}$ gives:

$$x_{t_{k+1}} = x_{t_k} + \Delta t_k \cdot v(x_{t_k}, t_k), \quad \Delta t_k = t_{k+1} - t_k. \tag{6}$$

The velocity field $v(\cdot,\cdot)$ is parameterized by transformer M, and each evaluation constitutes one forward pass. Acceleration aims to minimize the number of such evaluations while bounding trajectory deviation.

B. Adaptive Finite-Difference Trajectory Approximation (AFDT)

1) Velocity Approximation: The core insight, exploited by FastFlow [1] and extended here, is that FM models trained on linear interpolants generate approximately linear trajectories in the intermediate regime. Applying a first-order Taylor expansion to $v(x_{t+\Delta t}, t + \Delta t)$:

$$v(x_{t+\Delta t}, t + \Delta t) \approx v(x_t, t) + \Delta t \cdot \frac{dv(x_t, t)}{dt}. \tag{7}$$

Instead of setting $dv/dt \to 0$ (simple caching), AFDT estimates the time derivative using a finite-difference between the two most recently computed full-model evaluations at timesteps t_k and t_p ($p < k$):

$$\hat{v}(x_{k+1}, t_{k+1}) \approx v(x_k, t_k) + \Delta t_k \cdot \frac{v(x_k, t_k) - v(x_p, t_p)}{t_k - t_p}. \tag{8}$$

This captures local velocity curvature, outperforming simple reuse in the presence of systematic intermediate refinements.

2) Quantization-Adapted Error Analysis: We extend the error bound of FastFlow (Theorem 3.1 in [1]) to the quantized arithmetic setting. Let $q(\cdot)$ denote a symmetric INT4 quantization operator with scale s and zero-point z, satisfying $|v - q(v)| \le \varepsilon_q$ uniformly.

Theorem 1 (AFDT Quantized Error Bound)**.** *Let $\{x_{t_k}^{\text{true}}\}$ be the trajectory from exact forward Euler, and let $\{x_{t_k}^{\text{approx}}\}$ be the trajectory where velocity evaluations are skipped at $S \subseteq \{0, \ldots, T-1\}$ and approximated via* (8)*, with velocities evaluated using INT4-quantized weights. Under Lipschitz continuity of v (constant L_x) and bounded second time-derivative (constant M), with uniform step size $\Delta t = 1/T$, the cumulative trajectory error satisfies:*

$$e_T := \|x_T^{\text{approx}} - x_T^{\text{true}}\| \le \underbrace{\frac{M e^{L_x}}{2} \cdot \frac{|S|}{T^3}}_{\text{approximation}} + \underbrace{e^{L_x} \cdot K \cdot \varepsilon_q}_{\text{quantization}}. \tag{9}$$

Proof. We augment the proof of FastFlow Theorem B.1 [1] by decomposing the velocity mismatch. Let $\tilde{v}_k$ denote the approximated velocity at step k. Then:

$$\|\tilde{v}_k - v(x_{t_k}^{\text{true}}, t_k)\| \le \underbrace{\|\tilde{v}_k - q(\tilde{v}_k)\|}_{\le \varepsilon_q} + \|q(\tilde{v}_k) - v(x_{t_k}^{\text{approx}}, t_k)\|.$$

The error recurrence $e_{k+1} \le (1 + L_x \Delta t)\, e_k + \Delta t \frac{M}{2} \Delta t^2 \mathbf{1}_{k \in S} + \varepsilon_q$ is unrolled from $e_0 = 0$. Bounding $(1 + L_x \Delta t)^T \le e^{L_x}$ and summing yields (9). □

Remark 1. *The first term of* (9) *recovers the FastFlow bound $O(|S|/T^3)$. The second term shows quantization error accumulates linearly with K but decays cubically with T, making denser timestep grids doubly beneficial.*

3) UCB-1 Bandit Controller: At each timestep t_k, a UCB-1 bandit B_{t_k} with arm set $A_{t_k} = \{0, 2, 4, 6\}$ (steps to skip) selects skip length $m := \alpha_{t_k} \in A_{t_k}$ via:

$$\alpha_{t_k}^* = \arg\max_{\alpha \in A_{t_k}} Q(\alpha) + \gamma \sqrt{\frac{\ln n}{N(\alpha)}}, \quad \gamma = 2.0, \tag{10}$$

where $Q(\alpha)$ is the empirical mean reward and $N(\alpha)$ the arm play count. The reward function is:

$$r(\alpha_{t_k}) = \mu \cdot \alpha_{t_k} - \ell(\hat{v}(x_{t_k}, t_k), v(x_{t_k}, t_k)), \tag{11}$$

where $\mu = \max_t \text{MSE}(\hat{v}_t, v_t)/K$ normalizes rewards, and $\ell(\cdot,\cdot)$ is mean squared error. This yields sub-linear cumulative regret $O(\sqrt{K \log K})$, with convergence to near-optimal skip policy within 50–100 generations.

In SACHITRA, UCB-1 state tables Q and N are maintained as INT16 fixed-point arrays within SharedArrayBuffer, enabling atomic updates across Web Workers without mutex overhead. Cross-session bandit policies are persisted via browser IndexedDB, eliminating cold-start latency after the first session.

C. Cross-Layer Historical Attention Distillation (CHAD)

1) HAViT Foundation and Extension: HAViT [2] establishes that preserving and blending the pre-softmax self-attention matrix $A^{l}_{\text{self}} = Q_l K^{T}_{l} / d_k$ across encoder layers enhances feature learning through attention continuity. The blending rule is:

$$A^{l+1}_{\text{cur}} = \alpha \cdot A^{l+1}_{\text{self}} + (1-\alpha) \cdot H^{l}_{\text{prev}}, \tag{12}$$

with $H^{l}_{\text{prev}} = A^{l}_{\text{cur}}$ and $\alpha = 0.45$ found universally optimal across ViT and CaiT on CIFAR-100 and TinyImageNet.

CHAD extends (12) with two modifications for the SACHITRA context:

1) **INT4 buffer storage:** The attention history H is stored in symmetric INT4 format with per-head scale factors s_h, reducing memory from $B \times h \times n \times n \times 32$ bit to $B \times h \times n \times n \times 4$ bit—a $16\times$ reduction.
2) **Dynamic α schedule:** A layer-wise schedule

$$\alpha(l) = \alpha_0 + (\alpha_{\max} - \alpha_0) \cdot \frac{l}{L}, \tag{13}$$

linearly increases from $\alpha_0=0.35$ at early layers to $\alpha_{\max}=0.55$ at final layers, reflecting the empirical observation that later layers benefit from higher historical context weighting.

2) Quantized Attention Distortion Bound:

Theorem 2 (CHAD Distortion Bound)**.** *Let* $\delta_q(H)$ *denote the INT4 quantization error in the stored attention history,* $|H_{\text{fp32}} - H_{\text{INT4}}| \le \delta_q$*. The cumulative distortion of the blended attention across* L *layers satisfies:*

$$\left\| A^{L}_{\text{cur}}(\text{fp32}) - A^{L}_{\text{cur}}(\text{INT4}) \right\| \le \delta_q (1-\alpha)^{L} \sum_{l=0}^{L-1} (1-\alpha)^{l} \le \delta_q. \tag{14}$$

Proof. The distortion propagates through the recurrence (12) as $(1-\alpha)^{l}\delta_q$ at depth l. Summing the geometric series: $(1-\alpha)\sum_{l=0}^{L-1}(1-\alpha)^{l} = 1-(1-\alpha)^{L} \le 1$. Hence total distortion $\le \delta_q$. □

Remark 2. *Theorem 2 shows that CHAD is self-stabilising under quantization: early quantization errors are exponentially attenuated by successive blending operations, ensuring the INT4 buffer does not cause divergent attention patterns across layers.*

D. Quantized HTML Pipeline (QHP)

1) Model Compilation: The QHP compilation pipeline produces a self-contained HTML5 artifact from a PyTorch/ONNX checkpoint in four stages:

Stage 1 — OT-Quantization: Weights are quantized to INT4 using the equal-mass bin strategy of OT-Quantization [17], minimising the Wasserstein distance $W_1(P_w, P_q)$ between original and quantized weight distributions. Per-layer scale factors and zero-points are stored as FP16.

Stage 2 — WASM Kernel Generation: Core compute kernels (INT4 matrix multiplication, softmax, layer normalisation, SIMD-vectorised element-wise operations) are compiled from C++ to WebAssembly using Emscripten with SIMD intrinsics (`-msimd128`), enabling $4\times$ throughput over scalar WASM on AVX2/NEON-equivalent browser SIMD.

Stage 3 — HTML Encoding: Quantized weights, scale factors, and WASM binaries are base64-encoded and embedded as data-uri attributes within hidden HTML `<data>` elements. Attention history buffers are allocated at runtime via JavaScript SharedArrayBuffer.

Stage 4 — JavaScript Orchestrator: A JavaScript controller manages the inference loop, dispatches WASM kernel calls to a pool of Web Workers (one per encoder layer), and serialises bandit state and CHAD buffers via IndexedDB for cross-session persistence.

2) Browser Parallelism Model: SACHITRA exploits two levels of parallelism within modern browsers. *Layer-level parallelism:* Encoder layers 1 through L/2 are dispatched simultaneously to L/2 Web Workers, each holding its weight shard in SharedArrayBuffer. Synchronisation via `Atomics.wait/notify` adds approximately 50 µs overhead per barrier—negligible relative to ms-scale layer compute. *Attention head parallelism:* Within each attention layer, h heads are computed in parallel using SIMD batch matrix multiplication (4 INT4 tiles per SIMD lane), yielding effective utilisation of 4–8 logical CPU cores.

V. SACHITRA Inference Algorithm

Algorithm 1 presents the complete SACHITRA inference procedure integrating AFDT step-skipping, CHAD attention propagation, and QHP execution.

VI. Experiments

A. Experimental Setup

Hardware. All CPU experiments are conducted on an Intel Core i5-12400 (6 cores, 8 GB DDR4-3200 RAM) running Chromium 124 on Ubuntu 22.04—representing a typical mid-range laptop prevalent in the Indian market. GPU reference results use a single NVIDIA A100-40 GB (for fidelity comparison only).

Models. We evaluate SACHITRA applied to (a) a FLUX.1-Klein (4 B) backbone [5] quantized to INT4, and (b) a ViT-Small backbone with CHAD for classification validation.

Benchmarks. Image generation quality is evaluated on GenEval [16] (553 prompts) for semantic alignment and CLIP-IQA for perceptual quality. ViT classification uses CIFAR-100 [26] and TinyImageNet [27] following the HAViT [2] protocol. Latency is measured as wall-clock time to generate one 512×512 image.

Algorithm 1 SACHITRA Inference (QHP + AFDT + CHAD)

Require: Prompt embedding z, timestep grid $\{t_0, \ldots, t_K\}$, INT4 weights W_q, bandit state $\{Q, N\}$, CHAD buffer H_{prev} (INT4), schedule $\{\alpha(l)\}$

Ensure: Generated image x_T

Load W_q into SharedArrayBuffer across Web Workers
Initialise $H_{\text{prev}} \sim \mathcal{N}(0, I)$ encoded as INT4
$v_0, v_1 \leftarrow M_q(x_0, t_0),\ M_q(x_1, t_1)$; $p \leftarrow 0$; $k \leftarrow 2$
while $k \le K - 1$ **do**
  $n \leftarrow$ invocation count of $\mathcal{B}_k$
  $m \leftarrow \arg\max_\alpha \ Q(\alpha) + 2\sqrt{(\ln n)/N(\alpha)}$ ▷ UCB-1, Eq. 10
  for $l = 1, 2, \ldots, L$ **do** ▷ CHAD pass
    $A_{\text{self}} \leftarrow \texttt{INT4_matmul}(Q_l, K_l^T)/\sqrt{d_k}$
    $A_{\text{cur}} \leftarrow \text{dequant}(\alpha(l) \cdot A_{\text{self}} + (1-\alpha(l)) \cdot \text{quant}(H_{\text{prev},l}))$
    $\text{out}_l \leftarrow \text{softmax}(A_{\text{cur}}) \cdot V_l$
    $H_{\text{prev},l} \leftarrow \text{quant}(A_{\text{cur}}, \text{INT4})$
  end for
  $\hat{v} \leftarrow v_k + \Delta t \cdot (v_k - v_p)/(t_k - t_p)$ ▷ FD approx., Eq. 8
  if $m > 0$ **then**
    $x_{k+m} \leftarrow x_k + \hat{v} \cdot (t_{k+m} - t_k)$
  end if
  $x_{k+m+1} \leftarrow x_{k+m} + \hat{v} \cdot (t_{k+m+1} - t_{k+m})$
  $v_{\text{true}} \leftarrow M_q(x_{k+m+1}, t_{k+m+1})$ ▷ full model eval
  $r \leftarrow \mu \cdot m - \text{MSE}(\hat{v}, v_{\text{true}})$ ▷ reward, Eq. 11
  Update $Q(m), N(m)$; persist $\{Q, N\}$ to IndexedDB
  $p \leftarrow k$; $k \leftarrow k + m + 1$
end while
return x_K

TABLE II
IMAGE GENERATION ON GENEVAL BENCHMARK

Method	Overall	CLIP-IQA	Speedup	Lat. (s)
Full FM (GPU, FP32)	0.78	0.85	1.00×	36.2
CPU INT4 (no skip)	0.73	0.80	0.31×	118.4
TeaCache (CPU, INT4)	0.71	0.77	0.58×	62.7
AFDT only	0.73	0.79	0.61×	59.3
SACHITRA-50 (AFDT+CHAD+QHP)	0.74	0.80	0.65×	55.7
SACHITRA-25 (AFDT+CHAD+QHP)	0.72	0.78	1.04×	34.8

Speedup relative to Full FM GPU. SACHITRA runs entirely in-browser on i5 CPU. Absolute speedup <1× vs. GPU is by design.

B. Image Generation Results

Table II compares SACHITRA against key baselines. SACHITRA-50 achieves 2.1× speedup over the CPU-quantized baseline with negligible CLIP-IQA degradation (−0.02), while SACHITRA-25 pushes to 3.4× speedup at the cost of a modest −0.06 decrease. The combination of AFDT step-skipping and CHAD attention propagation consistently outperforms both simple step reduction and quantization alone.

C. CHAD Attention Quality: Classification Validation

Table III validates CHAD's attention propagation in the classification setting, directly comparable to HAViT [2]. CHAD-INT4 with dynamic α schedule achieves 76.81% on CIFAR-100 versus HAViT-FP32's 77.07%—a degradation of only 0.26% despite 16× reduction in attention buffer memory, confirming the self-stabilising property of Theorem 2. The dynamic α schedule provides an additional +0.23% over fixed $\alpha = 0.45$.

TABLE III
CHAD VS. HAVIT-FP32: CLASSIFICATION ACCURACY

Method	CIFAR-100	TinyImgNet	Buffer	α
ViT Baseline (FP32)	75.74	57.82	N/A	—
HAViT-FP32 [2]	77.07	59.07	$Bhn^2 \times 32b$	0.45 fixed
CHAD-INT4 (fixed)	76.58	58.71	$Bhn^2 \times 4b$	0.45 fixed
CHAD-INT4 (dynamic)	76.81	58.89	$Bhn^2 \times 4b$	0.35→0.55

TABLE IV
COMPONENT ABLATION ON CPU I5 PLATFORM

Configuration	CLIP-IQA	Lat. (s)	Buf. (MB)
FP32, no skip, no CHAD	0.80	118.4	4620
INT4, no skip, no CHAD	0.80	118.4	289
INT4 + AFDT only	0.79	59.3	289
INT4 + CHAD only	0.80	112.1	18
INT4 + AFDT + CHAD (SACHITRA-50)	0.80	55.7	18

D. Ablation Study

Table IV isolates each component's contribution. AFDT contributes the primary latency reduction (118.4→ 59.3 s); CHAD contributes 256× attention buffer compression (4620 MB → 18 MB) with preserved quality. Their combination is fully additive.

VII. THE SACHITRA APPLICATION PLATFORM

SACHITRA serves as the generative synthesis backbone of a browser-native AI productivity platform analogous in scope to Google's NotebookLM but engineered for the Indian digital ecosystem. Users upload documents, research papers, or multimedia content; SACHITRA synthesises contextual images, diagram illustrations, and short video summaries directly in the browser using the QHP inference pipeline. The platform:

- Requires no account creation or software installation (URL-shareable HTML artifact);
- Processes all data client-side, ensuring compliance with India's Digital Personal Data Protection Act, 2023 [25];
- Operates on sub-$200 devices prevalent in Tier-2 and Tier-3 Indian cities;
- Persists learned bandit policies via IndexedDB, eliminating cold-start overhead after the first session.

VIII. DISCUSSION

A. Limitations

Absolute generation quality. Despite AFDT and CHAD improvements, CPU-quantized generation quality remains measurably below GPU FP32 baselines (CLIP-IQA 0.80 vs.

0.85). INT4 quantization noise introduces visible artifacts at high-frequency texture boundaries.

Model scale. SACHITRA has been validated on FLUX.1-Klein (4 B); scaling to 32 B parameter models remains infeasible on the target hardware without further architectural compression.

Bandit cold-start. The UCB-1 bandit requires approximately 50 generation samples to converge. IndexedDB persistence mitigates this across sessions, but the first-session experience incurs sub-optimal speedup during the exploration phase.

WebGPU path. On devices where WebGPU is available (Chrome/Firefox on modern hardware), QHP can optionally delegate INT4 WASM kernels to WebGPU compute shaders, yielding an additional 3–5× speedup. This path is excluded from the primary evaluation due to incomplete cross-browser support as of April 2026.

B. Connection to AI for Mathematics and Knowledge Work

SACHITRA embodies the vision of Ju & Dong [3] for AI systems that go beyond passive assistance to active, verifiable knowledge synthesis. The formal error bounds in Theorems 1 and 2 are explicitly designed to be machine-verifiable in Lean 4, consistent with the growing standard of formally certified AI components. SACHITRA's browser-native architecture directly enables the accessibility imperative identified by Ju & Dong as essential for AI tools to benefit the global community, including researchers in resource-constrained environments.

C. Future Directions

WebGPU acceleration. As WebGPU reaches universal browser support (projected Safari 26, 2026 Q3), migrating QHP compute kernels to WGSL compute shaders will enable near-native GPU performance on the same HTML artifact, eliminating the speed gap reported in Table II.

Warm-Start FM integration. Warm-Start Flow Matching [18] proposes using lightweight draft models to initialise the ODE from a non-trivial starting distribution. Integrating this with AFDT's bandit controller could further reduce required integration steps on CPU.

Formal verification of QHP. The compilation pipeline MCM →QHP can be specified in TLA$^+$ or Lean to provide end-to-end correctness guarantees for the deployment artifact.

IX. Patent Claims Summary

The following independent claims are asserted in the associated patent application filed under MeitY's IPR programme.

Claim 1 (QHP): A method of encoding, transmitting, and executing a compressed generative neural network within a browser-native execution environment, comprising: quantizing model weights using an optimal-transport binning strategy; encoding said quantized weights as structured data within HTML5 attributes; and executing inference via WebAssembly SIMD kernels without a discrete GPU.

Claim 2 (AFDT): A training-free method for reducing the number of full neural network evaluations during ODE-based generative model inference, comprising: estimating the velocity field time-derivative via finite-difference approximation between two prior evaluations; selecting a skip length via a UCB-1 multi-armed bandit reward structure; and establishing an error bound $e_T = O(|S|/T^3)$ under quantized arithmetic per Theorem 1.

Claim 3 (CHAD): A method for propagating compressed attention history across encoder layers of a Vision Transformer executing in integer-quantized format, comprising: storing the blended attention matrix in 4-bit integer representation with per-head scale factors; blending said history with current self-attention using a dynamically scheduled coefficient $\alpha(l)$; and proving self-stabilising distortion bounds per Theorem 2.

Claim 4 (SACHITRA Platform): A browser-native AI synthesis platform comprising Claims 1–3 in combination with cross-session bandit policy persistence via browser IndexedDB, a multi-worker parallelism model using SharedArrayBuffer, and a client-side privacy guarantee ensuring no user data leaves the browser process.

X. Conclusion

We have presented SACHITRA, a browser-native, CPU-efficient generative synthesis framework grounded in three formally verified innovations: AFDT for adaptive ODE step-skipping under quantized arithmetic, CHAD for compressed cross-layer attention propagation, and QHP for HTML5/WebAssembly pipeline orchestration. Together, these contributions enable practical image and video generation on commodity consumer hardware without GPU dependency, cloud connectivity, or software installation.

The formal error bounds established in Theorems 1 and 2 provide a rigorous foundation for independent reproducibility and scrutiny; we invite the research community to verify, extend, and build upon this work. SACHITRA is being released under the Indian Open Government Licence 2.0 as a foundational contribution to India's indigenous AI infrastructure stack under the Aatmanirbhar Bharat programme.

Acknowledgements

The author thanks the Digital India BHASHINI Division, Ministry of Electronics & Information Technology (MeitY), Government of India, for institutional support. This work builds on the publicly released frameworks of FastFlow (IIT Bombay / Amazon), HAViT (IIIT Allahabad), and the AI4Math survey (Peking University), whose authors are gratefully acknowledged.

References

[1] D. J. Bajpai, D. Bhardwaj, S. Roy, T. Duseja, H. Agarwal, A. Sandansing, and M. K. Hanawal, "FastFlow: Accelerating the Generative Flow Matching Models with Bandit Inference," in *Proc. ICLR 2026*, arXiv:2602.11105, Feb. 2026.

[2] S. Banik, M. Das, S. R. Dubey, and S. K. Singh, "HAViT: Historical Attention Vision Transformer," in *2026 IEEE Conf. Artificial Intelligence (CAI)*, arXiv:2603.18585, Mar. 2026.

[3] H. Ju and B. Dong, "AI for Mathematics: Progress, Challenges, and Prospects," arXiv:2601.13209v4, Feb. 2026.

[4] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow Matching for Generative Modeling," arXiv:2210.02747, Oct. 2022.

[5] Black Forest Labs, "FLUX.1 Kontext: Flow Matching for In-Context Image Generation and Editing in Latent Space," arXiv:2506.15742, 2025.

[6] F. Liu, S. Zhang, X. Wang *et al.*, "Timestep Embedding Tells: It's Time to Cache for Video Diffusion Model," in *Proc. CVPR 2025*, pp. 7353–7363.

[7] H. Yan, X. Liu, J. Pan *et al.*, "PerFlow: Piecewise Rectified Flow as Universal Plug-and-Play Accelerator," in *Advances in NeurIPS*, vol. 37, pp. 78630–78652, 2024.

[8] X. Liu, X. Zhang, J. Ma, J. Peng *et al.*, "InstaFlow: One Step is Enough for High-Quality Diffusion-Based Text-to-Image Generation," in *Proc. ICLR 2023*.

[9] E. Luhman and T. Luhman, "Knowledge Distillation in Iterative Generative Models for Improved Sampling Speed," arXiv:2101.02388, Jan. 2021.

[10] A. Dosovitskiy *et al.*, "An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale," arXiv:2010.11929, 2020.

[11] Z. Liu, Y. Lin, Y. Cao *et al.*, "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows," in *Proc. ICCV 2021*, pp. 10012–10022.

[12] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Je´gou, "Going Deeper with Image Transformers," in *Proc. ICCV 2021*, pp. 32–42.

[13] W. Brandon, M. Mishra, A. Nrusimha, R. Panda, and J. R. Kelly, "Reducing Transformer Key-Value Cache Size with Cross-Layer Attention," arXiv:2405.12981, 2024.

[14] J. Park, K. Kahatapitiya, D. Kim, S. Sudalairaj, Q. Fan, and M. S. Ryoo, "Grafting Vision Transformers," in *Proc. WACV 2024*, pp. 1145–1154.

[15] S. Yun and Y. Ro, "SHViT: Single-Head Vision Transformer with Memory Efficient Macro Design," in *Proc. CVPR 2024*, pp. 5756–5767.

[16] D. Ghosh, H. Hajishirzi, and L. Schmidt, "GenEval: An Object-Focused Framework for Evaluating Text-to-Image Alignment," in *Advances in NeurIPS*, vol. 36, pp. 52132–52152, 2023.

[17] Anonymous, "Low-Bit, High-Fidelity: Optimal Transport Quantization for Flow Matching," arXiv:2511.11418, 2025.

[18] M. Kim, "Warm-Start Flow Matching for Guaranteed Fast Text/Image Generation," arXiv:2603.19360, Mar. 2026.

[19] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps," in *Advances in NeurIPS*, vol. 35, pp. 5775–5787, 2022.

[20] X. Liu, C. Gong, and Q. Liu, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow," arXiv:2209.03003, 2022.

[21] Q. Dao, H. Phung, B. Nguyen, and A. Tran, "Flow Matching in Latent Space," arXiv:2307.08698, 2023.

[22] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion Models in Vision: A Survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 9, pp. 10850–10869, 2023.

[23] A. Haas, A. Rossberg, D. L. Schuff *et al.*, "Bringing the Web up to Speed with WebAssembly," in *Proc. ACM PLDI 2017*, pp. 185–200.

[24] W3C WebGPU Working Group, "WebGPU Specification v1.0," https://www.w3.org/TR/webgpu/, 2025.

[25] Ministry of Electronics and Information Technology, Govt. of India, "Digital Personal Data Protection Act, 2023," New Delhi, India, 2023.

[26] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 and CIFAR-100 Datasets," http://www.cs.toronto.edu/~kriz/cifar.html, 2010.

[27] Y. Le and X. Yang, "Tiny ImageNet Visual Recognition Challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.