BONSAI: Browser-Optimized Neural Gaussian Avatar Streaming

Sina Kitz
University of Bonn
Germany
kitz@cs.uni-bonn.de

Simon Katz
University of Bonn
Germany
katz@cs.uni-bonn.de

Jan Spindler
University of Bonn
Germany
jspindle@uni-bonn.de

Zhastay Yeltay
University of Bonn
Germany
jastayeltay@gmail.com

Reza Rastegari
University of Bonn
Germany
srastegari@uni-bonn.de

Julian Kaltheuner
University of Bonn
Germany
kaltheun@cs.uni-bonn.de

Patrick Stotko
University of Bonn
Germany
stotko@cs.uni-bonn.de

Reinhard Klein
University of Bonn
Germany
rk@cs.uni-bonn.de

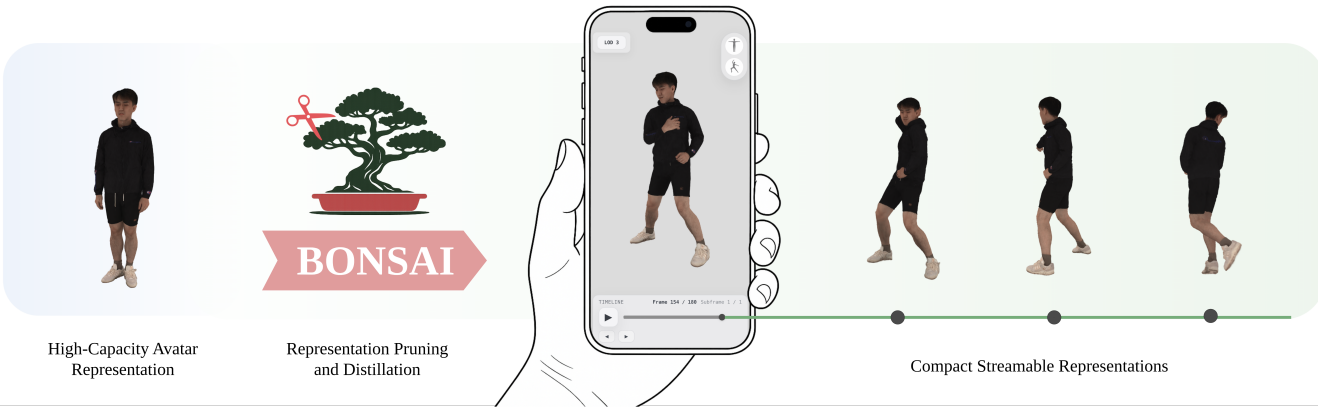

Figure 1: We present BONSAI, a deployment framework for browser-native neural Gaussian avatars. Starting from a trained high-capacity avatar representation, BONSAI derives a compact runtime decoder that preserves visual fidelity while enabling progressive streaming and interactive WebGPU playback. On desktop-class hardware, the compact decoder achieves real-time frame rates (340 FPS); on mobile devices, progressive LoD streaming enables browser-based neural avatar playback where it was previously not feasible.

Abstract

Neural Gaussian avatar representations can encode high-fidelity clothed human performances as compact canonical models, but their deployment in standard web browsers remains difficult because each animation frame requires neural decoding of tens to hundreds of thousands of Gaussian primitives. Existing efficient avatar representations achieve impressive mobile rendering performance by designing compact representations from scratch. Instead, we address the complementary deployment problem of converting existing high-capacity neural Gaussian avatars into compact browser-native runtime representations.

We present BONSAI, a deployment framework that converts high-capacity neural Gaussian avatars into compact, progressively streamable representations for browser-native WebGPU playback. At its core is a joint pruning and distillation pipeline that transforms high-capacity reference decoders into compact runtime decoders using activation-guided structured pruning and Gaussian-attribute distillation with lightweight image-space supervision. Critically, conversion requires only outputs from a trained reference decoder rather than the original multiview capture data. The compact runtime decoders execute within a fully fused WebGPU runtime supporting neural inference, SMPL-X attachment, sorting, and rendering. Combined with progressive hierarchical streaming for adaptive level-of-detail (LoD) playback, BONSAI enables browser-native deployment that adapts to available bandwidth and device capabilities—achieving interactive frame rates on desktop-class GPUs and enabling browser-based playback of neural Gaussian avatars on mobile devices. Across our experiments, BONSAI reduces decoder parameter counts by more than 98% while maintaining high visual fidelity and increasing browser MLP throughput from 5 FPS to 340 FPS on an RTX 4090.

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

Web3D '26, Doha, Qatar

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2810-5/2026/10
https://doi.org/10.1145/3822516.3834888

CCS Concepts

• **Computing methodologies** → **Graphics systems and interfaces**; *Shape representations*; • **Software and its engineering** → *Browsers.*

Keywords

Neural Avatars, Gaussian Splatting, WebGPU, Browser Rendering, Model Compression, Knowledge Distillation, Pruning, Progressive Streaming

ACM Reference Format:

Sina Kitz, Simon Katz, Jan Spindler, Zhastay Yeltay, Reza Rastegari, Julian Kaltheuner, Patrick Stotko, and Reinhard Klein. 2026. BONSAI: Browser-Optimized Neural Gaussian Avatar Streaming. In *The 31th International Conference on 3D Web Technology (Web3D '26), October 13–15, 2026, Doha, Qatar*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3822516.3834888

1 Introduction

Photorealistic human avatars are becoming a central building block for immersive communication, virtual production, telepresence, online education, and web-based extended reality. These application domains particularly benefit from browser-native deployment, where users can access interactive avatar experiences without dedicated software installation and across heterogeneous hardware platforms. Recent neural avatar representations based on 3D Gaussian splatting have made substantial progress in reconstructing clothed human performances with high visual fidelity while retaining the rasterization efficiency of Gaussian primitives [Kerbl et al. 2023]. In particular, anchor-based neural Gaussian avatars such as PiG-Avatar [Kaltheuner et al. 2026b] represent a dynamic human as a set of canonical Gaussian anchors whose pose-dependent attributes are generated by neural decoders. This design is attractive because it preserves a compact canonical structure, supports animation through body-model parameters, and avoids transmitting a per-frame explicit set of Gaussians.

However, these advantages come at a significant deployment cost. High-fidelity avatar representations often require large neural decoders to model complex dynamic regions such as clothing and hair. At runtime, these decoders must be evaluated for every anchor, leading to substantial per-frame inference overhead before rendering. While this is feasible in offline pipelines or on high-end desktop GPUs, it is poorly matched to the constraints of commodity browsers, where compute shaders, memory bandwidth, shader compilation times, GPU watchdogs, and data-transfer overheads impose tight latency budgets. This creates a gap between high-quality neural avatar reconstruction and practical web delivery: existing high-capacity neural decoders provide quality, but their direct evaluation is prohibitively expensive for responsive browser playback. Concretely, a single forward pass of a 512-wide, five-layer decoder over $\sim 10^5$ anchors per frame amounts to roughly 0.2–0.3 TFLOPs of arithmetic, well beyond the throughput typically achievable by general-purpose browser ML runtimes such as ONNX Runtime Web or TensorFlow.js at interactive frame rates, a point we revisit quantitatively in Sec. 3.1.4.

A direct response to the browser deployment problem would be to retrain a neural Gaussian avatar with a smaller decoder. While generally feasible, we find that this approach has two practical drawbacks. First, reducing decoder capacity does not always lead to a graceful degradation in visual quality. As shown in Fig. 2, compact decoders are particularly prone to artifacts in highly dynamic regions such as loose clothing. Second, every deployment target requires a new end-to-end retraining from the original multiview capture data, making this approach impractical for content pipelines that already possess high-quality trained avatars.



(a) Reference (512) **(b) Retrained (64)** **(c) BONSAI (64)**

Figure 2: In comparison to a 512-wide reference decoder (a), direct 64-wide retraining introduces artifacts in dynamic clothing (b), while BONSAI preserves much of the visual quality of the reference at the same compressed width (c).

We therefore propose BONSAI (Fig. 1), a deployment-oriented framework for browser-native neural Gaussian avatars. BONSAI converts an existing high-capacity reference decoder into a compact runtime decoder while preserving the original canonical anchor representation. The resulting decoder is integrated into a browser-native WebGPU runtime featuring fused neural inference, progressive streaming, level-of-detail playback, SMPL-X attachment, sorting, and rendering.

Our key observation is that high-capacity reference decoders contain substantial redundancy for interactive playback. BONSAI identifies informative neurons using activation statistics and a self-sensitivity-guided importance measure, followed by Gaussian-attribute distillation with lightweight image-space supervision that emphasizes visually important regions through opacity-, visibility-, and projected-area-aware sampling. The resulting runtime decoder is trained from cached Gaussian-attribute targets, preserving the visual fidelity of the reference avatar while client-side inference within the WebGPU runtime becomes feasible.

To accommodate a wide range of device capabilities and network conditions, we combine the compact runtime decoders with progressive hierarchical Gaussian streaming, allowing playback to adapt to available compute and bandwidth. Users can begin interacting with a coarse avatar before all anchors have arrived, while additional data progressively refines the visual quality. This is particularly important for mobile devices, where the reference decoder was previously entirely infeasible: BONSAI's combined compression and LoD streaming make browser-based neural avatar playback possible on smartphones, achieving interactive frame rates at lower LoD tiers (Sec. 4.2.1).

Terminology. Throughout this paper, a *neural Gaussian avatar* refers to a dynamic human representation that uses a learned neural network (*decoder*) to predict per-frame Gaussian primitive attributes from a compact canonical representation (*anchors*) attached to a parametric body model (SMPL-X). *Structured pruning* removes entire neurons from the decoder network to reduce its width, while *knowledge distillation* trains the resulting compact network to match the output of the original high-capacity reference. *Level-of-detail (LoD) streaming* partitions anchors into cumulative subsets that are delivered progressively to the browser.

In summary, our contributions are:

- We present BONSAI, a reference-to-runtime decoder conversion pipeline that transforms high-capacity neural Gaussian avatar reference decoders into compact runtime decoders while preserving the original avatar representation.
- We introduce an importance-guided decoder conversion method that combines activation-guided structured pruning with Gaussian-attribute distillation and lightweight image-space supervision. We further implement a self-sensitivity-guided importance variant that biases pruning toward visually salient regions.
- We develop a browser-native WebGPU runtime that integrates fused neural inference, SMPL-X attachment, GPU depth sorting, and Gaussian splat rendering for efficient cross-platform playback.
- We enable incremental client delivery through hierarchical Gaussian streaming and cumulative LoD playback, reducing startup latency while progressively refining visual quality, while also making browser-based playback feasible on mobile devices.

Code is available at: https://github.com/vc-bonn/bonsai.

2 Related Work

Gaussian splatting and neural human avatars. 3D Gaussian splatting is capable of high-quality real-time rendering by rasterizing anisotropic Gaussian primitives instead of performing expensive volumetric ray marching [Kerbl et al. 2023]. Related dynamic reconstruction methods have shown that spatially structured neural representations can improve the stability and efficiency of reconstructing temporally coherent 4D geometry [Ebbed and Lähner 2026; Kaltheuner et al. 2026a, 2025; Yao et al. 2024]. In the human-avatar setting, recent avatar methods attach Gaussian primitives to parametric human models or learn canonical Gaussian representations whose attributes vary with pose and time [Hu et al. 2024a; Li et al. 2024; Moreau et al. 2024; Qian et al. 2024]. We use *anchor-based neural Gaussian avatars* to denote the family of methods in which canonical anchors are attached to a parametric body model and one or more learned neural decoders predict pose-dependent Gaussian attributes for each anchor at every rendered frame. Within this family, some methods employ a single shared decoder for all anchors [Hu et al. 2024b; Kaltheuner et al. 2026b; Moreau et al. 2024; Qian et al. 2024], while others distribute multiple smaller decoders across spatial regions and interpolate their outputs [Li et al. 2024; Zhan et al. 2025]. BONSAI is evaluated on the shared-decoder architecture used by PiG-Avatar, whose hierarchical LoD organization naturally supports progressive streaming, although the underlying idea of decoder compression is not inherently restricted to a particular decoder organization.

Efficient and mobile Gaussian avatars. Recent work has begun to target efficient full-body Gaussian avatar rendering on mobile and resource-constrained devices. SqueezeMe and TaoAvatar distill or compress pose-dependent Gaussian attributes into lightweight representations [Chen et al. 2025; Iandola et al. 2025], while Mobile Avatars with Pruned Local Blendshapes replace expensive neural decoders by local linear blendshapes [Zhan et al. 2026]. These approaches achieve impressive runtime performance by designing compact representations or simplified deformation models from the outset. BONSAI addresses a complementary deployment problem. Rather than introducing a new avatar representation, we start from an already trained shared-decoder anchor- based neural Gaussian avatar and convert its high-capacity reference decoder into a compact browser-executable runtime decoder. The underlying canonical Gaussian representation, neural decoder formulation, and hierarchical LoD structure are preserved; only the decoder itself is compressed through structured pruning and Gaussian-attribute distillation. This enables deployment of existing high-fidelity avatars without requiring access to the original multiview capture data or retraining the avatar.

This distinction also affects the visual quality—performance trade-off: Blendshape-based methods obtain efficiency by restricting pose-dependent Gaussian attributes to linear combinations of learned bases. This is highly efficient, and local blendshape models reduce the detail loss observed in earlier global formulations, but they still constitute a different representation. Our compressed neural decoder retains the reference decoder's original functional form more directly and targets a different use case: fast deployment of high-fidelity neural-field avatars without replacing the underlying avatar model or progressive streaming architecture.

Web-based Gaussian and volumetric rendering. WebGL-based Gaussian viewers have demonstrated interactive rendering of static or precomputed splats [Kellogg 2023; Kwok 2023, 2024]. WebGPU extends browser graphics with compute shaders and GPU-resident buffers, enabling neural inference, sorting, and splatting directly on the client. BONSAI builds on these capabilities to execute compressed neural Gaussian avatar decoders within a fully client-side rendering pipeline.

Knowledge Distillation and Structured Pruning. Knowledge distillation transfers information from a large reference model to a smaller runtime model. Classical distillation uses softened output predictions as supervision [Hinton et al. 2015], while feature-based distillation matches intermediate or final representations of the teacher and student. Recent projector-based approaches revisit representation distillation and show that a simple projection head, normalization, and robust distance functions can improve teacher–student matching, especially under large capacity gaps [Miles and Mikolajczyk 2024]. These methods are valuable for training compact students, but they are primarily loss-side techniques: they do not by themselves determine which architecture should be executed at inference time. In our setting, this distinction is important because browser-based hardware acceleration requires a physically smaller neural decoder, not only a better distillation objective.

Network pruning removes redundant parameters or structures from a trained model to reduce memory and computation. Early compression pipelines combine pruning with quantization and entropy coding to reduce storage and improve deployment efficiency [Han et al. 2016]. Unstructured pruning can achieve high sparsity, but irregular sparse computation often does not translate into proportional speedups on commodity GPUs. Structured pruning instead removes entire channels, filters, neurons, or blocks and is therefore better aligned with hardware acceleration [He and Xiao 2024]. Network Slimming learns channel-level sparsity and removes insignificant channels to obtain thinner networks [Liu et al.

2017], while ThiNet formulates filter pruning using statistics from subsequent layers [Luo et al. 2017]. Other methods estimate parameter or neuron importance by incorporating gradient-information; for example, Taylor-based pruning approximates the effect of removing neurons or filters on the loss [Molchanov et al. 2019], and SNIP performs data-dependent connection pruning in a single shot before training [Lee et al. 2018].

Our approach is closest in spirit to structured neuron pruning, but differs in both target and importance criterion. We prune a pose-conditioned decoder MLP that is evaluated for large numbers of Gaussian anchors at every rendered frame, making runtime cost directly proportional to decoder width. Neuron importance is estimated from activation statistics and a self-sensitivity-guided measure that emphasizes regions with high visual impact. The resulting compact decoder is refined through Gaussian-attribute distillation, yielding a dense runtime network that remains well suited for fused WebGPU execution.

3 Method

The key challenge of browser-native neural avatar deployment is that the high-capacity reference decoders used during avatar authoring are too expensive for real-time execution in the browser (Sec. 3.1.4). BONSAI addresses this problem through a reference-to-runtime decoder conversion pipeline that preserves the original avatar representation, LoD hierarchy, and appearance while producing compact WebGPU-friendly runtime decoders.

3.1 Representation and Browser Pipeline

3.1.1 Neural Gaussian Avatar Representation. We assume a trained neural Gaussian avatar represented by a set of canonical anchors

$$\mathcal{A} = \{(\mathbf{x}_i, \mathbf{f}_i)\}_{i=1}^{N}, \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^3$ is the canonical anchor position and $\mathbf{f}_i$ is a learned latent code. Each anchor is attached to a parametric human model such as SMPL-X [Pavlakos et al. 2019], providing a local frame and a kinematic transform for a pose θ_t at time t. For an input condition

$$\mathbf{z}_{i,t} = [\mathbf{f}_i, \gamma(t), \phi(\theta_t), \ell], \tag{2}$$

where $\gamma(t)$ encodes time, $\phi(\theta_t)$ encodes pose, and ℓ denotes the LoD level, a neural decoder predicts Gaussian attributes

$$\Psi_{\text{Ref}}(\mathbf{z}_{i,t}) = \left(\Delta\boldsymbol{\mu}_{i,t}, \Delta\mathbf{q}_{i,t}, \mathbf{s}_{i,t}, o_{i,t}, \mathbf{c}_{i,t}\right). \tag{3}$$

Here $\Delta\boldsymbol{\mu}$ is a local position offset, $\Delta\mathbf{q}$ a residual rotation represented by a quaternion, $\mathbf{s}$ the Gaussian scale, o opacity, and $\mathbf{c}$ color expressed in spherical harmonics coefficients of a view-dependent color function. The final posed Gaussian is obtained by combining the predicted local position offset with SMPL-X-based transport.

The reference decoder Ψ_{Ref} in our implementation consists of two parallel high-capacity MLPs: a splat decoder that predicts $(\Delta\boldsymbol{\mu}, \Delta\mathbf{q}, \mathbf{s}, o)$ and a color decoder that predicts $\mathbf{c}$. Both networks use five layers with hidden width 512. While suitable for high-quality offline rendering, evaluating these networks for (N) anchors each frame is prohibitively expensive for real-time browser deployment.

The goal of BONSAI is therefore to replace the pair of reference networks with a pair of compact runtime networks. To achieve this, we jointly prune and distill the splat and color decoders, optimizing both runtime networks simultaneously using Gaussian-attribute supervision and soft image-space rendering losses. Our runtime analysis in Sec. 4.2.1 indicates that a hidden width of 64 provides the best trade-off between visual fidelity and browser performance.

3.1.2 Progressive LoD Streaming. In PiG-Avatar [Kaltheuner et al. 2026b] anchors are organized into cumulative LoD sets

$$\mathcal{A}_0 \subset \mathcal{A}_1 \subset \cdots \subset \mathcal{A}_{L-1} = \mathcal{A}. \tag{4}$$

The reference decoder is conditioned on anchor identity, pose, time, and active LoD level through the input $\mathbf{z}_{i,t,\ell}$, and predicts the corresponding Gaussian attributes extends Equation 3:

$$\Psi_{\text{Ref}}(\mathbf{z}_{i,t,\ell}) = \left(\Delta\boldsymbol{\mu}_{i,t,\ell}, \Delta\mathbf{q}_{i,t,\ell}, \mathbf{s}_{i,t,\ell}, o_{i,t,\ell}, \mathbf{c}_{i,t,\ell}\right). \tag{5}$$

This conditioning allows the decoder to produce attributes that are consistent with the currently available anchor subset. We build upon this hierarchical representation to partition avatar assets into per-LoD files, allowing the coarsest representation to be displayed immediately while higher-detail anchors stream in asynchronously. Consequently, the browser can begin playback using only $\mathcal{A}_0$ and progressively refine the displayed avatar as additional anchor data arrives. This yields a web delivery model in which start-up latency, bandwidth, and decoding cost can be traded against visual quality.

3.1.3 WebGPU Runtime. The runtime pipeline consists of the following four stages:

(1) Decode active anchor attributes using the compact runtime decoder MLPs in a fused WebGPU compute pass.
(2) Compute and apply SMPL-X attachment and local-to-world Gaussian transformations.
(3) Preprocess and depth-sort Gaussian splats into discrete bins using GPU compute kernels.
(4) Render the splats using alpha blending in a browser canvas.

The original high-capacity reference decoder is never evaluated in the browser. Only the compact runtime decoder, anchor data, LoD metadata, and motion sequence are streamed to the client.

Framework and portability. We implement the runtime on top of the `wgpu` crate instead of targeting the WebGPU JavaScript API directly. `wgpu` provides a safe Rust abstraction that compiles to WebAssembly for execution in the browser, while exposing a unified graphics and compute API across WebGPU, Vulkan, Metal, and D3D12 backends.

In this work, all runtime experiments are conducted in the browser. The choice of Rust and `wgpu` was motivated by software engineering and deployment considerations (rather than native execution): Rust provides strong type safety, memory safety, and a mature tooling ecosystem for large-scale systems development, while `wgpu` allows the same rendering and compute pipeline to remain portable across future deployment targets. Although we do not evaluate native execution in this paper, the architecture can be compiled to the previously mentioned native graphics backends without re-implementing the runtime.

Pipeline structure. Each frame is dispatched as a single WebGPU submission containing two command encoders. The first encoder executes all neural inference work: optional SMPL-X pose interpolation, SMPL-X linear blend skinning, attachment evaluation and the fused runtime decoder pass that emits all Gaussian attributes. When the animation state remains unchanged between consecutive

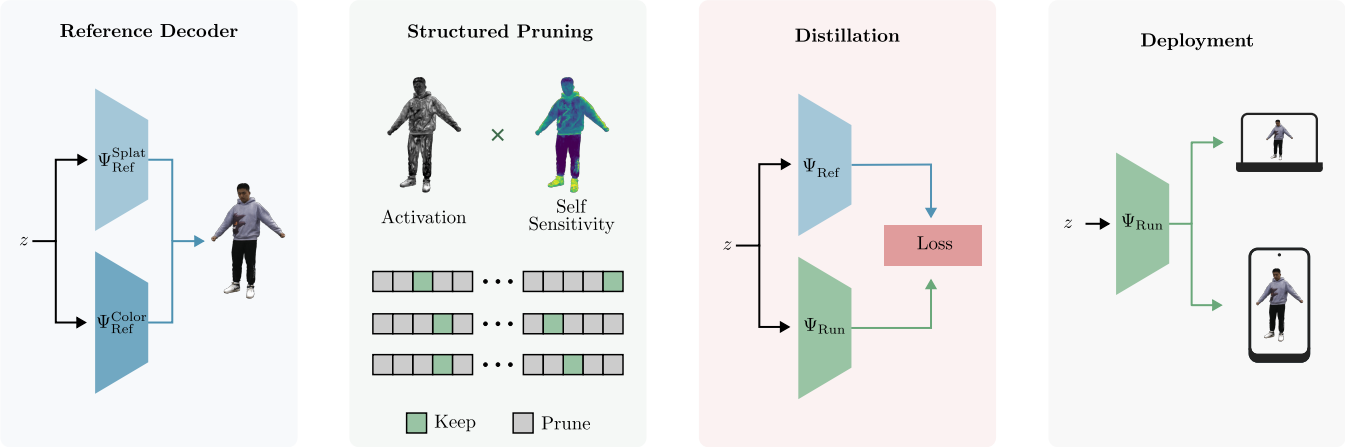

Figure 3: BONSAI pipeline. The reference decoders are pruned and jointly distilled into compact runtime decoders using Gaussian-attribute and optional image-space supervision. The resulting networks preserve the appearance of the reference avatar while enabling efficient browser-native inference.

frames, decoder evaluation is skipped and previously computed Gaussian attributes are reused. The second encoder consumes those attributes and performs splat preprocessing, depth sorting, and rendering. For depth ordering, we employ a wait-free radix sorting pipeline similar to WebSplatter [Han et al. 2026], which avoids synchronization assumptions that, while given by native execution on most desktop GPUs, are not guaranteed across WebGPU implementations and therefore improve portability across desktop and mobile GPUs.

Fused decoder kernel and memory traffic. The runtime decoder is implemented as a single compute pass that chains all hidden layers without writing intermediate activations to global memory. Standard layer-by-layer MLP evaluation incurs $2L-1$ global memory round-trips per anchor, with hidden width k this corresponds to roughly $8k(L-1)N$ bytes of intermediate write traffic per frame in FP32. For $N = 2{\times}10^5$ and $L = 5$, an unfused $k = 512$ reference decoder would write $\sim$3.3 GB of intermediates per frame, against $\sim$0.4 GB for a $k=64$ runtime decoder. Fusing the layers into a single pass keeps activations in registers and removes this traffic almost entirely, but the technique only applies up to the register budget of the compiled WGSL kernel: as k grows, the compiler eventually spills to shared or global memory and the per-anchor cost climbs sharply. This memory-traffic constraint, in addition to the raw FLOP count (Sec. 3.1.4), motivates the compact runtime decoder width targeted by our pruning procedure. In addition, geometry and color prediction are evaluated within the same compute dispatch. Because both branches share the same input features, this avoids reloading anchor conditioning data and removes the overhead of a second dispatch.

Shared Memory Caching. Each workgroup cooperatively loads the runtime decoder's current layer weight matrix into WebGPU workgroup (shared) memory before evaluating the layer. Subsequent dot products read weights from shared memory rather than global memory, reducing redundant memory traffic across threads. This optimization is possible only for decoder widths of 64 or smaller, where the layer weights fit within the available 16KB workgroup memory budget. In our experiments it reduced decoder inference time by up to 1.8×, where available.

Cross-platform watchdog handling. Modern graphics backends terminate long-running compute kernels via GPU watchdog mechanisms (e.g. D3D12 TDR or browser-imposed execution limits). To remain stable across mid-range and high-end hardware, we partition the per-frame decoder workload into chunks of up to 8,192 anchors on mid-range GPUs (e.g. NVIDIA RTX 3060); on high-end GPUs (e.g. RTX 5090) the full anchor set is dispatched in a single workgroup grid. Chunk size is selected once per device from a short warm-up calibration and held constant during playback.

3.1.4 Why the Reference Decoder Is Not a Browser Baseline. The most direct deployment baseline would be to execute the original reference decoder Ψ_{Ref} in the browser. However, as we show in Sec.4.2.1 the prohibitive computational cost of the reference decoder makes this approach impractical for interactive web deployment. A five-layer, 512-wide MLP is dominated by its three inner 512×512 weight matrices and requires $C_{\Psi_{\text{Ref}}} \approx 3{\cdot}2\,(512)^2 \approx 1.6$ MFLOPs per anchor. At the anchor budgets targeted by our WebGPU runtime ($N \approx 10^5$–$2{\times}10^5$ per frame), a single reference decoder evaluation per frame therefore amounts to $\sim$0.2–0.3 TFLOPs of decoder arithmetic, or $\sim$6–10 TFLOPs/s at a 30 Hz interaction target. Although this is in principle within the raw FP32 capability of a high-end discrete GPU, it is far above what general-purpose browser ML runtimes deliver in practice. In our preliminary experiments, neither ONNX Runtime Web nor TensorFlow.js approached interactive frame rates for Ψ_{Ref} on any tested browser/device pair. The dominant overheads are *not* arithmetic but per-layer dispatch, host/device buffer round-trips, the absence of register-resident activation chaining, and shader-compilation latency. None of these can be removed without the targeted fused-kernel design described in Sec. 3.1.3, which is itself enabled by the compact width of the runtime decoder.

We therefore treat the reference decoder as the *offline reference* that defines the visual-quality ceiling. All quality, fidelity, and

parameter-count measurements are reported relative to this reference. The effectiveness of the conversion process is evaluated through image-space comparisons between renderings produced by the reference decoder and the runtime decoder, using standard reconstruction metrics including PSNR, SSIM, and LPIPS. This evaluation directly measures the visual impact of compression independently of deployment concerns.

Because all evaluated runtime-decoder variants are exported as dense MLPs with the same target width, they share the same browser execution path and identical computational complexity. The purpose of the pruning, low-rank preprocessing, and initialization ablations is therefore not to compare browser performance, but to evaluate which conversion strategy best preserves the quality of the original reference decoder.

3.2 Reference-to-Runtime Decoder Conversion

Figure 3 provides an overview of the BONSAI decoder conversion pipeline. Directly evaluating the high-capacity reference decoder in the browser is impractical at interactive frame rates (Sec. 3.1.4). We thus convert it into a compact runtime decoder Ψ_{Run} while preserving the canonical anchors, LoD hierarchy, and motion-conditioning inputs. The conversion first caches reference Gaussian attributes for sampled anchor–pose–time–LoD inputs, then initializes a 64-wide runtime decoder through reference-aware neuron selection, and finally refines it with Gaussian-attribute and lightweight image-space distillation.

3.2.1 Reference Attribute Cache. To train the runtime decoder without repeatedly evaluating the expensive reference model, we first cache reference predictions. For sampled anchor–pose–time–LoD conditions $\mathbf{z}_{i,t,\ell}$, we evaluate the reference decoder (Eq. 5) once and store the resulting Gaussian attributes

$$\mathbf{y}^{\text{Ref}}_{i,t,\ell} = \Psi_{\text{Ref}}(\mathbf{z}_{i,t,\ell}). \tag{6}$$

The cached target contains the position offset, residual rotation, scale, opacity, and color predicted by the reference avatar. This turns decoder conversion into a lightweight reference–runtime training problem and removes the need to access the original multiview capture data during compression.

We sample uniformly across anchors and frames, but bias additional samples towards visually relevant regions by favoring high-opacity, large projected-area, and strongly deforming anchors. LoD levels are sampled explicitly so that the compact decoder receives balanced supervision for all progressive streaming stages.

3.2.2 Reference-Aware Neuron Selection. We initialize the compact runtime decoder by selecting neurons from the reference decoder that are likely to preserve its input–output behavior. Intuitively, useful neurons should be both frequently active on representative avatar samples and have a strong influence on the following layer. For each hidden neuron (j) in layer ℓ, we therefore compute an importance score

$$I^{(\ell)}_j = \left(\frac{1}{N} \sum_{n=1}^{N} \left| h^{(\ell)}_j(z_n) \right| \right) \left\| W^{\text{Ref}}_{\ell+1,j} \right\|_2 . \tag{7}$$

where N denotes number of sampled cache entries, $h^{(\ell)}_j$ is the activation of neuron j, and $W^{\text{Ref}}_{\ell+1,j}$ contains its outgoing weights. The first term measures how strongly the neuron is activated on average over sampled avatar states, while the second term estimates its downstream influence.

For each hidden layer, we keep the top k neurons according to $I^{(\ell)}$ and copy the corresponding weights and biases into the runtime decoder. The importance scores are computed once during offline conversion from the reference attribute cache and are not part of browser playback. In all experiments, we use $k = 64$, yielding a compact MLP that can be exported directly to the fused WebGPU inference kernel.

3.2.3 Self-Sensitivity-Guided Neuron Selection. As an additional reference-aware initialization strategy, we bias neuron selection by the visual sensitivity of the avatar anchors. Recall that each anchor denotes a canonical avatar element whose predicted pose-dependent Gaussian attributes can affect the final rendering to different degrees. The activation-based score from Sec. 3.2.2 treats all cached samples equally, although errors in some avatar regions are much more visible than others. For example, small deviations near silhouettes, faces, clothing folds, or high-frequency texture details can noticeably affect the rendered appearance, while comparable errors in flat or mostly occluded regions may be less important. We therefore estimate a visual sensitivity score for each anchor and use it to bias neuron selection toward visually important regions.

To compute this score, we perturb the Gaussian attributes predicted by the reference decoder for sampled poses and viewpoints, render the perturbed avatar, and measure the image-space change relative to the unperturbed reference rendering. For a spatial anchor group g, viewpoint v, time step t, and perturbation type a, we define

$$\rho^{(a)}_{g,v,t} = \frac{1}{|\Omega_{g,v,t}| + \epsilon} \sum_{p \in \Omega_{g,v,t}} \left\| I^{\text{pert},a}_{v,t}(p) - I^{\text{base}}_{v,t}(p) \right\|_1 , \tag{8}$$

where $\Omega_{g,v,t}$ contains the pixels influenced by group g. The area normalization prevents large projected regions from dominating the score only because they cover more pixels. We average these scores over poses and viewpoints and assign the resulting sensitivity value to all anchors in the group, yielding a per-anchor weight s_i.

We incorporate this weight $\mathbf{s}_i$ into the neuron-importance score from Sec. 3.2.2. For cached sample $\mathbf{z}_n$ associated with anchor $i(n)$, the sensitivity-weighted importance from Eq. 7 becomes:

$$I^{(\ell)}_j = \left(\frac{1}{M} \sum_{n=1}^{M} s_{i(n)} \left| h^{(\ell)}_j(\mathbf{z}_n) \right| \right) \left\| W^{\text{Ref}}_{\ell+1,:,j} \right\|_2 . \tag{9}$$

As before, we keep the top k neurons per layer and copy the corresponding weights into the compact runtime decoder. The sensitivity computation is performed once during offline conversion and adds no cost to browser playback.

3.2.4 Gaussian-Attribute Distillation. After neuron selection, we refine the compact runtime decoder using the cached reference predictions from Sec. 3.2.1. For each cached input $\mathbf{z}_n$, the runtime decoder prediction $\Psi_{\text{Run}}(\mathbf{z}_n)$ is matched to the corresponding reference output $\Psi_{\text{Ref}}(\mathbf{z}_n)$. We define the attribute-space distillation

loss as

$$\mathcal{L}_{\text{attr}}(\mathbf{z}_n) = \sum_{a \in \{\mu,q,s,o,c\}} \lambda_a, d_a \left(\Psi_{\text{Run},a}(\mathbf{z}_n), \Psi_{\text{Ref},a}(\mathbf{z}_n)\right), \quad (10)$$

where a indexes the decoded Gaussian attributes: position offset, residual rotation, scale, opacity, and color. We use a sign-aligned quaternion distance for d_q and mean-squared error for all other attributes. Since the loss is computed from cached decoder outputs, conversion does not require access to the original multiview training images.

To reduce visible artifacts that are not fully captured by independent attribute losses, we add lightweight image-space supervision by rendering the reference and runtime avatars from sampled views and matching their RGB appearance with reconstruction and SSIM losses. The full loss then becomes

$$\mathcal{L} = \mathcal{L}_{\text{attr}} + \mathcal{L}_1 + \mathcal{L}_{\text{SSIM}}. \quad (11)$$

The distilled decoder remains a compact MLP and can therefore be exported directly to the fused WebGPU runtime.

4 Evaluation

4.1 Experimental Setup

4.1.1 Datasets and Avatars. We evaluate BONSAI on six neural Gaussian avatars from two public datasets spanning different capture resolutions, motion characteristics, and appearance complexity.

THUman4.0. THUman4.0 [Zheng et al. 2022] provides relatively stable motions and moderate geometric complexity and is our baseline dataset.

DNA-Rendering. DNA-Rendering [Cheng et al. 2023] provides captures with greater appearance variation, more dynamic motion sequences, and higher resolution. We include it to evaluate our method under more demanding and diverse conditions.

For each dataset, we select three subjects with diverse clothing, body shapes, and skin tones. All reference avatars are trained using the same 512-wide decoder architecture on sequences of 180–300 frames before BONSAI compression.

4.1.2 Metrics. **Visual Quality.** We evaluate compressed runtime decoders using PSNR, SSIM, and LPIPS between images rendered by the reference decoder and the corresponding runtime decoder. Since BONSAI aims to reproduce an existing high-capacity avatar rather than reconstruct the original capture data, the reference decoder serves as the ground truth, isolating compression and distillation error from reconstruction error.

Deployment Performance. We report decoder throughput, model size, per-stage GPU timings, total GPU frame time, and Time-to-First-Frame (TTFF) measured in the browser. Runtime results are evaluated across multiple desktop and mobile devices and, where applicable, across different level-of-detail configurations.

4.2 Runtime Decoder Compression

We evaluate BONSAI primarily against its own unpruned reference decoder and a naively retrained 64-wide baseline. Direct quantitative comparison against alternative efficient avatar methods such as SqueezeMe [Iandola et al. 2025], TaoAvatar [Chen et al. 2025], and pruned local blendshapes [Zhan et al. 2026] is methodologically challenging for several reasons. First, these methods design compact representations from scratch using specific parametric models and training protocols that differ from the PiG-Avatar framework on which BONSAI operates; running them within our pipeline would require reimplementing their full avatar representations, which is beyond the scope of a deployment-focused contribution. Second, and more fundamentally, BONSAI addresses a complementary problem: converting *already trained* high-capacity avatars without access to the original multiview training data, a setting that existing efficient methods—which train compact models from raw capture data—do not target. We therefore evaluate against the reference decoder, which provides the most direct measure of compression fidelity, and against a retrained-from-scratch compact decoder, which represents the natural deployment alternative for a content pipeline that already possesses a trained high-capacity avatar. The related work section (Sec. 2) provides a qualitative comparison of design trade-offs between these approaches.

Table 1: Effect of runtime decoder width on browser performance (RTX 4090).

MLP Layer Width	512	256	128	64
Inference Speed [FPS]	5	26	186	340
Model Size [MB]	46	41	40	39

4.2.1 Effect of runtime decoder Width. The reference decoder is optimized for reconstruction quality rather than deployment efficiency. Because it must be evaluated for every Gaussian anchor at every frame, decoder width directly impacts runtime cost. To quantify this trade-off, we evaluate decoder widths from 512 to 64 while keeping the underlying Gaussian representation unchanged. Table 1 reports model size and decoder throughput measured in Chrome on an RTX 4090.

Reducing decoder width substantially lowers neural inference cost. Throughput increases from 5 FPS for the original 512-wide decoder to 340 FPS for the 64-wide variant, a nearly two-order-of-magnitude improvement. The observed speedup exceeds the reduction in parameter count alone, suggesting that reduced decoder width improves GPU execution efficiency in addition to lowering arithmetic workload.

In contrast, asset size decreases only modestly from 46MB to 39MB, indicating that storage is dominated by the Gaussian representation rather than the decoder. Decoder compression therefore primarily addresses the runtime inference bottleneck, while startup latency and network delivery are handled separately through the progressive streaming hierarchy.

Based on this trade-off, we use a hidden width of 64 for all subsequent experiments, as it provides the highest throughput while retaining sufficient capacity for distillation to recover the visual quality of the reference decoder.

4.2.2 Runtime Decoder Initialization Strategies. Unless stated otherwise, all compressed models use the same runtime-decoder architecture (64-wide MLP), distillation objective, and training budget. The only difference between variants is the initialization strategy used before distillation.



(a) Geometric Self-Sensitivity (b) Reference (c) Activation × Out. Norm (d) Self-Sensitivity Weighted

Figure 4: Self-Sensitivity Weighting (a) guides pruning toward visually important regions, enabling the self-sensitivity-weighted model (d) to better preserve fine details than activation-based pruning (c), while remaining close to the reference (b).

Table 2: Importance score ablation for structured pruning evaluated on THUman4.0 for LOD 3.

Importance Score	PSNR↑	SSIM↑	LPIPS↓
Random	34.14	0.9878	0.0128
Activation × out. norm	34.86	0.9890	0.0119
Self-Sensitivity (Ours)	35.06	0.9894	0.0117

Table 3: Quality of the final 64-wide BONSAI runtime decoder relative to the corresponding 512-wide reference decoder.

Method	Width	PSNR↑	SSIM↑	LPIPS↓
THUman4.0	64	35.06	0.9894	0.0117
DNA-Rendering	64	32.33	0.9918	0.0173

Table 2 compares different runtime-decoder initialization strategies for structured pruning:

Random Initialization. The runtime decoder is initialized with random weights and subsequently optimized using the BONSAI distillation pipeline.

Activation-Based Pruning. The runtime decoder is initialized through structured neuron selection based on activation statistics and outgoing-weight magnitude.

Self-Sensitivity Pruning (BONSAI). The runtime decoder is initialized through the proposed self-sensitivity-guided importance score combined with activation statistics and outgoing-weight magnitude. Self-sensitivity is an optional refinement to the core activation-based pruning strategy; its purpose is to better preserve visually salient regions such as silhouettes and clothing details that contribute disproportionately to perceived quality.

Random initialization produces the lowest reconstruction quality, indicating that distillation alone is insufficient to fully recover the behavior of the reference decoder under a fixed runtime-decoder capacity. Activation-based pruning substantially improves all metrics by selecting neurons that are frequently used and strongly connected to subsequent layers.

The proposed self-sensitivity-guided importance score further improves performance across all evaluated metrics. Compared to activation-based pruning, self-sensitivity increases PSNR from 34.86 dB to 35.06 dB, improves SSIM from 0.9890 to 0.9894, and reduces LPIPS from 0.0119 to 0.0117. Although the numerical differences are modest under global frame-averaged metrics, they are consistent across all quality measures. We note that these global metrics may understate the perceptual benefit of self-sensitivity, since the method specifically concentrates capacity on visually salient regions whose local improvement is diluted when averaged over the full frame.

Figure 4 provides qualitative insight into this behavior. The self-sensitivity score assigns higher importance to regions whose appearance changes strongly under perturbations of the decoder outputs, concentrating model capacity on visually significant structures such as clothing details, silhouette boundaries, and other high-frequency regions. This leads to improved preservation of fine appearance details after pruning and subsequent distillation.

4.2.3 Runtime Decoder Quality. Table 3 reports the quality of the final 64-wide BONSAI runtime decoder relative to the corresponding 512-wide reference decoder. Across both datasets, the compressed decoder preserves the appearance of the reference avatar despite an eightfold reduction in hidden-layer width.

On THuman4D, BONSAI achieves 35.06dB PSNR, 0.9894 SSIM, and 0.0117 LPIPS relative to the reference decoder. On the selected DNA Rendering subjects, which exhibit higher-resolution captures and more dynamic motion, BONSAI attains 32.33dB PSNR, 0.9918 SSIM, and 0.0173 LPIPS. These results indicate that the proposed

Table 4: Per-frame pipeline timing (ms) for $N = 100{,}000$ anchors and optimization ablations.

Device	Optimization	SMPL-X Attachment	Splat MLP	Splat Preprocessing	Radix Sort	Draw	Total GPU Time
GTX 1050M	unfused	1.99	52.54	0.27	0.87	1.25	56.93
GTX 1050M	fused	2.02	51.27	0.26	0.88	1.22	55.65
GTX 1050M	fused shared memory	2.07	28.30	0.26	0.89	1.22	32.75
RTX 3080	unfused	0.59	7.91	0.46	0.29	0.42	9.67
RTX 3080	fused	0.30	4.45	0.20	0.13	0.69	5.77
RTX 3080	fused shared memory	0.31	2.60	0.23	0.14	0.71	3.99
RTX 4090	unfused	0.45	2.53	1.30	0.47	0.34	5.02
RTX 4090	fused	0.44	2.29	0.20	0.11	0.38	3.42
RTX 4090	fused shared memory	0.50	1.62	0.25	0.13	0.44	2.94

compression pipeline remains effective across datasets with different capture characteristics while producing a compact runtime decoder suitable for browser deployment.

4.3 Browser Runtime Evaluation

4.3.1 Runtime Pipeline Breakdown. Table 4 reports per-frame GPU timings for the BONSAI runtime under three implementation variants: an unfused baseline, a fused pipeline, and our final fused shared-memory implementation. Across all tested GPUs, decoder inference remains the dominant component of the frame budget, motivating the compression strategy proposed in this work.

Kernel fusion and shared-memory caching substantially reduce memory traffic and decoder overhead. Compared with the unfused implementation, total frame time decreases from 5.02ms to 2.94ms on the RTX 4090, from 9.67ms to 3.99ms on the RTX 3080, and from 56.93ms to 32.75ms on the GTX 1050M, demonstrating that browser-native neural Gaussian avatars benefit from both decoder compression and GPU-aware runtime optimization.

Overall, the combined optimizations reduce end-to-end frame time by 1.7×, 2.4×, and 1.7× on the RTX 4090, RTX 3080, and GTX 1050M respectively. These results demonstrate that efficient browser-native neural Gaussian avatars require both decoder compression and GPU-aware runtime optimization.

4.3.2 Mobile LoD Evaluation. While decoder compression reduces neural inference cost, the progressive level-of-detail (LoD) representation can balance quality and performance when rendering large Gaussian point sets. Table 5 reports total GPU frame time on three consumer mobile devices.

Rendering cost increases with the number of Gaussian primitives. On the evaluated smartphones, LoD 0 achieves 28–39 ms (25–35 FPS), whereas LoD 3 requires 122–275 ms per frame (3.6–8.2 FPS). Decoder compression and LoD selection address complementary bottlenecks: the former reduces neural inference cost, while the latter controls Gaussian rendering cost. We emphasize that prior to BONSAI, the reference decoder could not execute at any frame rate on these mobile devices; the ability to achieve interactive playback at LoD 0–1, combined with progressive refinement to higher LoDs where frame-rate requirements are less stringent, represents a qualitative step forward for browser-based neural avatar deployment on mobile hardware.

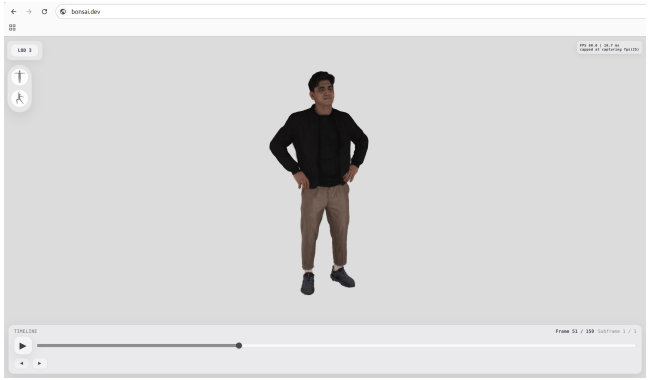

Figure 5: Avatar rendered in browser-native BONSAI runtime.

Importantly, LoD streaming addresses a different bottleneck than decoder compression: runtime-decoder pruning reduces neural inference cost, whereas LoD selection controls Gaussian rendering cost. Together, both mechanisms enable browser-native deployment across a wide range of devices.

4.3.3 Impact of Progressive LoD Streaming. Table 6 summarizes the progressive streaming hierarchy used by BONSAI. A single compressed runtime decoder is shared across all levels of detail; only the anchor set and its associated latent features are streamed incrementally. Consequently, increasing the LoD incurs additional storage and transfer costs only for newly introduced anchors, without duplicating decoder weights or requiring multiple model variants.

Figure 6 shows that even the coarser LoDs preserve the overall appearance of the avatar, while progressively increasing the number of anchors recovers finer geometric and appearance details. This design enables progressive refinement from 12.5k to 100k anchors while maintaining a single compact runtime decoder. Figure 5 shows the resulting browser-native runtime: the first avatar is displayed after approximately 2.2 s using the coarsest 4.9 MB representation, while the remaining LoDs are streamed asynchronously, reaching the full 39.2 MB avatar within an additional 54 ms.

5 Discussion

Role of Structured Pruning. Our experiments show that reference-aware initialization consistently outperforms random initialization

Table 5: Influence of LoD on mobile total GPU time.

Device	LoD 0 [ms]	LoD 1 [ms]	LoD 2 [ms]	LoD 3 [ms]
iPhone 15	31.85	61.90	123.06	274.54
Honor Magic 6 Pro	39.14	69.05	129.31	231.10
Samsung A73	28.25	41.73	68.03	122.35

Table 6: Influence of progressive LoD streaming on incremental asset size and delivery performance.

LoD	Anchors	Incremental Size [MB]	TTFF [ms]	ΔLOD0 [ms]
0	12.5k	4.90	2188.0	-
1	25k	4.90	2193.0	5.0
2	50k	9.80	2229.3	41.3
3	100k	19.60	2241.6	53.6



(a) LOD 0 **(b) LOD 1** **(c) LOD 2** **(d) LOD 3**

Figure 6: Visual comparison across progressive LoDs.

under the same distillation objective and training budget. Structured pruning transfers the most informative neurons into a compact runtime decoder, enabling distillation to recover lost capacity rather than learn the representation from scratch while remaining directly executable within our fused WebGPU pipeline. The core of this contribution lies in combining activation-based pruning with Gaussian-attribute distillation to achieve over 98% parameter reduction; the self-sensitivity weighting serves as an optional refinement that further improves preservation of visually salient details.

Limitations. BONSAI is a deployment framework rather than a reconstruction method and therefore inherits the limitations of the reference representation. Compression cannot recover details or dynamic behavior that are not represented by the original model.

We only evaluate shared-decoder neural Gaussian avatars built on PiG-Avatar. Although the proposed compression strategy is not inherently tied to this architecture, its effectiveness for distributed decoder networks, blendshape formulations, and other neural avatar representations remains an open question. Furthermore, while we have argued qualitatively that BONSAI addresses a complementary deployment problem to methods such as SqueezeMe, TaoAvatar, and local blendshape avatars [Chen et al. 2025; Iandola et al. 2025; Zhan et al. 2026], a systematic quantitative comparison under a common evaluation protocol would strengthen the analysis and is left for future work.

Our global reconstruction metrics (PSNR, SSIM, LPIPS) show only modest gains from self-sensitivity weighting over standard activation-based pruning. While the heatmap visualizations (Fig. 4) suggest that self-sensitivity improves detail preservation in perceptually important regions, localized metrics focused on areas such as the face, hands, and silhouette boundaries would provide a more targeted assessment of these benefits.

Finally, mobile browser performance at higher LoD tiers remains limited (3.6–8.2 FPS at LoD 3), reflecting the inherent rendering cost of large Gaussian point sets on tile-based mobile GPU architectures. Future work on mobile-specific rendering optimizations, such as tile-aware splat culling or half-precision compute, could further close the gap between mobile and desktop playback quality.

6 Conclusion

We presented BONSAI, a deployment framework for browser-native neural Gaussian avatars. Rather than retraining compact avatar models from scratch, BONSAI converts existing high-capacity neural avatar decoders into compact runtime decoders through structured pruning and Gaussian-attribute distillation while preserving the original canonical Gaussian representation. The resulting runtime decoder integrates directly with a WebGPU execution pipeline supporting neural inference, progressive streaming, level-of-detail playback, and real-time rendering in standard web browsers.

Our experiments demonstrate that substantial reductions in decoder complexity (over 98% parameter reduction) can be achieved while preserving the visual fidelity of the reference avatar. On desktop-class GPUs, BONSAI increases decoder throughput from 5 FPS to 340 FPS, enabling real-time interactive playback. On mobile devices, the combination of decoder compression and progressive LoD streaming makes browser-based neural Gaussian avatar playback feasible, with interactive frame rates at lower LoD tiers. These results establish BONSAI as a practical bridge between high-fidelity neural avatar authoring and cross-platform browser deployment.

Acknowledgments

This work was supported by the European Regional Development Fund (ERDF) and the State of North Rhine-Westphalia as part of the operational program EFRE/JTF-Programm NRW 2021-2027. The project, titled "Gen-AIvatar", was funded under the NEXT.IN.NRW competition with the grant agreement No. EFRE-20801085.

Additionally, it has been funded by the Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia as part of the Lamarr-Institute for Machine Learning and Artificial Intelligence and by the Federal Ministry of Education and Research under Grant No. 01IS22094A WEST-AI.

The work has also been funded by the Ministry of Culture and Science North Rhine-Westphalia under grant number PB22-063A (InVirtuo 4.0: Experimental Research in Virtual Environments), and by the state of North Rhine Westphalia as part of the Excellency Startup Center.NRW (U-BO-GROW) under grant number 03ESCNW18B.

References

Jianchuan Chen, Jingchuan Hu, Gaige Wang, Zhonghua Jiang, Tiansong Zhou, Zhiwen Chen, and Chengfei Lv. 2025. Taoavatar: Real-time lifelike full-body talking avatars

for augmented reality via 3d gaussian splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wei Cheng, Ruixiang Chen, Siming Fan, Wanqi Yin, Keyu Chen, Zhongang Cai, Jingbo Wang, Yang Gao, Zhengming Yu, Zhengyu Lin, et al. 2023. Dna-rendering: A diverse neural actor repository for high-fidelity human-centric rendering. In *IEEE International Conference on Computer Vision (ICCV)*.

Mohamed Ebbed and Zorah Lähner. 2026. MoAngelo: Motion-Aware Neural Surface Reconstruction for Dynamic Scenes. In *International Conference on 3D Vision (3DV)*.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*.

Yudong Han, Chao Xu, Xiaodan Ye, Weichen Bi, Zilong Dong, and Yun Ma. 2026. WebSplatter: Enabling Cross-Device Efficient Gaussian Splatting in Web Browsers via WebGPU. *arXiv preprint arXiv:2602.03207* (2026).

Yang He and Lingao Xiao. 2024. Structured Pruning for Deep Convolutional Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 46, 5 (2024).

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

Liangxiao Hu, Hongwen Zhang, Yuxiang Zhang, Boyao Zhou, Boning Liu, Shengping Zhang, and Liqiang Nie. 2024b. GaussianAvatar: Towards Realistic Human Avatar Modeling from a Single Video via Animatable 3D Gaussians. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Shoukang Hu, Tao Hu, and Ziwei Liu. 2024a. GauHuman: Articulated Gaussian Splatting from Monocular Human Videos. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Forrest Iandola, Stanislav Pidhorskyi, Igor Santesteban, Divam Gupta, Anuj Pahuja, Nemanja Bartolovic, Frank Yu, Emanuel Garbin, Tomas Simon, and Shunsuke Saito. 2025. SqueezeMe: Mobile-Ready Distillation of Gaussian Full-Body Avatars. In *ACM SIGGRAPH Conference Proceedings*.

Julian Kaltheuner, Hannah Dröge, Markus Plack, Patrick Stotko, and Reinhard Klein. 2026a. Neu-PiG: Neural Preconditioned Grids for Fast Dynamic Surface Reconstruction on Long Sequences. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Julian Kaltheuner, Alexander Oebel, Hannah Droege, Patrick Stotko, and Reinhard Klein. 2025. Preconditioned Deformation Grids. In *Computer Graphics Forum (CGF)*, Vol. 44. Wiley Online Library.

Julian Kaltheuner, Jan Spindler, Sina Kitz, Patrick Stotko, and Reinhard Klein. 2026b. PiG-Avatar: Hierarchical Neural-Field-Guided Gaussian Avatars. *arXiv preprint arXiv:2605.20185* (2026).

Mark Kellogg. 2023. GaussianSplats3D: Three.js-based implementation of 3D Gaussian splatting. https://github.com/mkkellogg/GaussianSplats3D. Accessed: 2026-04-01.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)* 42, 4 (2023).

Kevin Kwok. 2023. Splat: A WebGL 3D Gaussian Splatting Viewer. https://github.com/antimatter15/splat. Accessed: 2026-04-01.

Kevin Kwok. 2024. SplatV: A Video WebGL 3D Gaussian Splatting Viewer. https://github.com/antimatter15/splatv. Accessed: 2026-04-01.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. 2018. SNIP: Single-shot Network Pruning based on Connection Sensitivity. In *International Conference on Learning Representations (ICLR)*.

Zhe Li, Yipengjing Sun, Zerong Zheng, Lizhen Wang, Shengping Zhang, and Yebin Liu. 2024. Animatable and Relightable Gaussians for High-fidelity Human Avatar Modeling. *arXiv preprint arXiv:2311.16096v4* (2024).

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision (ICCV)*.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision (ICCV)*.

Roy Miles and Krystian Mikolajczyk. 2024. Understanding the role of the projector in knowledge distillation. In *AAAI Conference on Artificial Intelligence*, Vol. 38.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Arthur Moreau, Jifei Song, Helisa Dhamo, Richard Shaw, Yiren Zhou, and Eduardo Pérez-Pellitero. 2024. Human gaussian splatting: Real-time rendering of animatable avatars. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed AA Osman, Dimitrios Tzionas, and Michael J Black. 2019. Expressive body capture: 3d hands, face, and body from a single image. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhiyin Qian, Shaofei Wang, Marko Mihajlovic, Andreas Geiger, and Siyu Tang. 2024. 3DGS-Avatar: Animatable Avatars via Deformable 3D Gaussian Splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yuxin Yao, Siyu Ren, Junhui Hou, Zhi Deng, Juyong Zhang, and Wenping Wang. 2024. Dynosurf: Neural deformation-based temporally consistent dynamic surface reconstruction. In *European Conference on Computer Vision (ECCV)*.

Youyi Zhan, Tianjia Shao, Yin Yang, and Kun Zhou. 2025. Real-time High-fidelity Gaussian Human Avatars with Position-based Interpolation of Spatially Distributed MLPs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Youyi Zhan, He Wang, Tianjia Shao, and Kun Zhou. 2026. High-Fidelity Mobile Avatars with Pruned Local Blendshapes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zerong Zheng, Han Huang, Tao Yu, Hongwen Zhang, Yandong Guo, and Yebin Liu. 2022. Structured Local Radiance Fields for Human Avatar Modeling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.