Factorized Gradients for Scalable Highly-expressive Parametric Diffeomorphisms

Amit Aflalo[1] **Eran Treister**[1] **Chaim Baskin**[2] **Oren Freifeld**[1]

[1]The Stein Faculty of Computer and Information Science, Ben-Gurion University of the Negev, Israel
[2]School of Electrical and Computer Engineering, Ben-Gurion University of the Negev, Israel

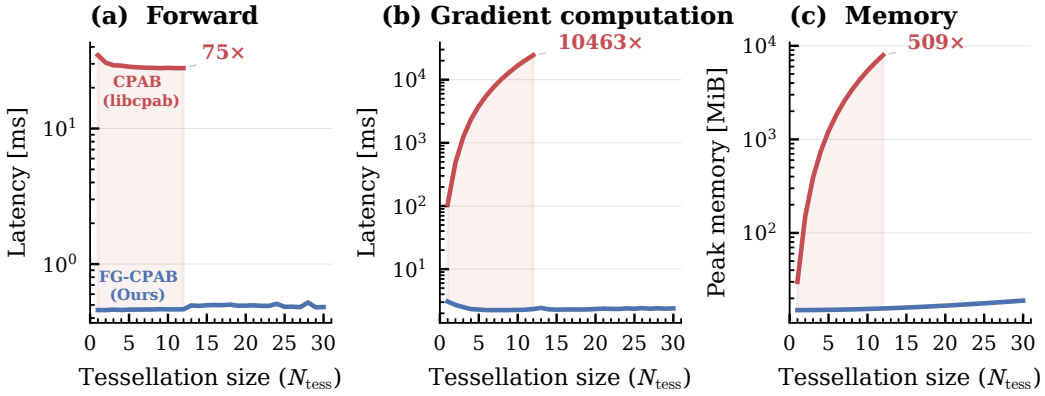


Figure 1: **Quantitative comparison, in 2D, of performance scaling with tessellation size** (N_{tess}), between the proposed method, called **FG-CPAB**, and `libcpab` [10], the standard implementation of CPAB diffeomorphisms [13]. The evaluation was performed on a single RTX A6000 GPU with 125,000 2D points and a batch size of 16. **(a)** Forward pass latency. **(b)** Gradient computation latency. **(c)** Peak GPU memory allocation. In 2D, the domain is partitioned into $C = 4N_{\text{tess}}^2$ triangular cells. Notably, the standard implementation does not support $N_{\text{tess}} > 12$, whereas ours scales efficiently.

Abstract

Diffeomorphisms provide a flexible, topology-preserving mathematical tool for modeling complex spatial or temporal deformations, and are used in various scientific fields. However, making these deformations simultaneously highly expressive and computationally efficient remains challenging. Continuous Piecewise-Affine Based (CPAB) transformations [13, 14] offer an attractive solution by parameterizing a family of diffeomorphisms via continuous velocity fields that are piecewise affine *w.r.t.* a chosen tessellation of the domain. Importantly, $d = \dim(\theta)$, the dimension of the parameter vector θ, depends on the fineness of the tessellation rather than on the data resolution. Despite this compact parameterization, CPAB optimization has long been hindered by the tight coupling between trajectory integration and gradient computation *w.r.t.* θ. We introduce **FG-CPAB**, a factorized-gradient formulation for scalable CPAB transformations. By separating integration from parameter-space projection, our method reduces gradient computation complexity from $\mathcal{O}(dTN)$ to $\mathcal{O}(TN + dC)$, where T is the number of integration steps, N is the number of transformed points, and C is the number of tessellation cells. This reformulation yields speedups of up to four orders of magnitude (e.g., $\approx 10^4\times$ in 2D), substantial memory savings (e.g., $\approx 500\times$ reduction in peak GPU memory usage in 2D), and dramatically higher expressiveness. This makes, for the first time, fine-tessellation CPAB practical in 2D and 3D, enabling highly expressive parametric diffeomorphisms. For code and resources, see our project page.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

A diffeomorphism is a differentiable invertible function whose inverse is also differentiable. Diffeomorphisms are a natural choice in spatial modeling tasks that require smooth, invertible, and topology-preserving deformations. They are widely used in applications such as registration, spatial alignment, and geometric warping, where folding, tearing, or self-intersection would result in geometrically invalid deformations. The standard way to construct such transformations is via integration of either a stationary velocity field (SVF) [2] or a time-dependent velocity field as in Large Deformation Diffeomorphic Metric Mapping (LDDMM) [4]. However, in practice, such methods often rely on a dense-grid discretization of the spatial domain. As a result, the number of parameters controlling the diffeomorphisms grows directly with the resolution of the domain (e.g., with the number of pixels in 2D). In modern 2D and especially 3D settings, this leads to high memory use, slow optimization, and difficult optimization landscapes.

Continuous Piecewise-Affine Based (CPAB) transformations, introduced by Freifeld et al. [13, 14], offer a compelling alternative. Instead of a dense-grid discretization, this approach tessellates the domain into cells (e.g., triangles in the 2D case) and constrains the velocity field to be a continuous piecewise-affine (CPA) function. A CPAB transformation is, by definition [13], obtained by integrating a CPA velocity field. Under a mild, easily enforced regularity condition, a CPAB transformation is a diffeomorphism[1]. A key point is that the number of parameters controlling a CPAB transformation depends on the tessellation, not the data resolution, thereby decoupling model complexity from data resolution. These transformations provide a compact low-dimensional parameterization while still capturing rich nonlinear deformations and have been used in diverse deep learning settings, including 1D applications such as time-series alignment and trainable activation functions [22, 28, 34, 38, 7, 26, 29, 23, 33], parameter-efficient multi-task fine-tuning [27], and 2D vision and generative modeling tasks [19, 11, 31, 32, 35, 37, 18].

Despite this compact parameterization, CPAB transformations remained hard to scale in practice. The bottleneck lies in computing the loss gradient, $\nabla_\theta \mathcal{L}$, which conventionally requires propagating parameter sensitivities through the transformation $x \mapsto \Phi^\theta(x)$. In the standard formulation, sensitivities *w.r.t.* all $d = \dim(\theta)$ parameters are propagated through all T integration steps for each of the N spatial points, yielding a backward complexity of $\mathcal{O}(dTN)$. This tight coupling between parameter dimension and trajectory integration makes optimization prohibitively expensive once tessellations become fine or spatial inputs become large. Consequently, existing CPAB implementations are often limited to coarse tessellations, which restricts practical expressiveness. Moreover, even with coarse tessellations in 2D, CPAB optimization is too slow for most practical applications. In 3D, the problem worsens, and scalable CPAB optimization has remained largely out of reach, despite the wide use of diffeomorphisms in this setting [5, 39, 42, 16, 8, 36, 17, 25, 6].

In this paper, we address this bottleneck with **FG-CPAB**, a factorized-gradient formulation that separates trajectory integration from parameter-space projection. Instead of propagating per-parameter sensitivities during integration, we perform a single adjoint sweep that accumulates low-dimensional cell-wise geometric statistics and recovers all parameter gradients through one projection onto the CPA basis. This reduces the backward complexity from $\mathcal{O}(dTN)$ to $\mathcal{O}(TN + dC)$, where C is the number of tessellation cells. We further introduce a sparse basis formulation that reduces memory use and allows the final projection step to fully exploit basis sparsity, thereby realizing the full computational benefit of the factorized gradients in practice.

Combined with optimized GPU kernels, the resulting method makes CPAB practical at scales that were previously inaccessible. For example, in 2D, our implementation achieves up to $75\times$ faster forward passes, $10^4\times$ faster backward passes, and over $500\times$ lower peak memory compared with the standard implementation [10]. Such gains enable efficient optimization of fine-tessellation CPAB diffeomorphisms in both 2D and 3D, and make CPAB competitive as a modern primitive for large-scale nonrigid spatial modeling. To summarize, our main contributions are:

- **FG-CPAB.** We introduce a gradient-factorized backward pass based on discrete-adjoint moment accumulation. Unlike standard adjoint methods, which still carry parameter dependence through

[1] CPAB transformations are homeomorphisms and even bi-Lipschitz functions (hence also almost-everywhere differentiable with an almost-everywhere differentiable inverse). Typically, but technically not always, they are also diffeomorphisms. For the sufficient regularity condition ensuring that a CPAB transformation is a diffeomorphism, see [1].

the backward integration, our method exploits the CPA basis structure to accumulate parameter-independent cell-wise moments and recover gradients only in a final sparse projection step.

- **An efficient implementation.** We combine factorized gradients, optimized GPU kernels, and a sparse basis construction to achieve large runtime and memory gains in practice.
- **Scalable diffeomorphic optimization in 2D and 3D.** We demonstrate that the resulting formulation makes fine-tessellation CPAB practical for spatial modeling and competitive with other diffeomorphic primitives in both 2D and 3D.

2 Background and Related Work

Let $\Omega \subseteq \mathbb{R}^K$ denote the spatial domain where $K \in \{1, 2, 3\}$. A diffeomorphism is a differentiable bijection $\Phi : \Omega \to \Phi(\Omega) \subseteq \mathbb{R}^K$ with a differentiable inverse. Such transformations preserve topology, avoiding tearing, folding, and self-intersection. A standard way to construct such a diffeomorphism Φ is to integrate a well-behaved velocity field $\boldsymbol{v}$ over time. Given an initial point $\boldsymbol{p}^{(0)} = \boldsymbol{p}_0 \in \Omega$, its trajectory follows the ordinary differential equation (ODE)

$$\dot{\boldsymbol{p}}^{(t)} = \boldsymbol{v}(\boldsymbol{p}^{(t)}, t), \quad t \in [0, 1], \tag{1}$$

with the resulting mapping defined as $\Phi(\boldsymbol{p}_0) = \boldsymbol{p}^{(1)}$. Some frameworks, such as Large Deformation Diffeomorphic Metric Mapping (LDDMM) [4], use time-varying velocity fields. A widely used alternative is the Stationary Velocity Field (SVF) formulation [2], which assumes the velocity field is constant over time (i.e., $\boldsymbol{v}(\boldsymbol{p}, t) = \boldsymbol{v}(\boldsymbol{p})$). This simplification significantly reduces computational complexity while still producing a rich class of diffeomorphisms.

Dense vs. Compact Parameterizations. Traditional methods for constructing diffeomorphisms [2, 4] parameterize $\boldsymbol{v}$ densely on the spatial grid. While this enables highly expressive deformations, it couples the number of parameters directly to the resolution of the domain. Even in 2D, this can already involve hundreds of thousands of parameters, and in 3D, the parameter count can reach millions, leading to high memory requirements and a large parameter space that can be difficult to optimize without explicit regularization. While alternative formulations restrict dense velocity fields to finite-dimensional Lie algebras [40, 41], they do not directly address the scaling bottlenecks.

Continuous Piecewise-Affine Based (CPAB) transformations [13, 14] address this limitation by imposing structure on the velocity field. Following [13], we consider tessellations of Ω into C convex polyhedral cells, possibly unbounded, and refer to these as *convex polyhedral tessellations*. A continuous piecewise-affine (CPA) function, *w.r.t.* a given tessellation, is a continuous function whose restriction to each cell is affine. A *simplicial tessellation* is the special case in which all cells are simplices: intervals in 1D, triangles in 2D, and tetrahedra in 3D. Our implementation uses simplicial tessellations when zero boundary conditions are imposed. Otherwise, we extend Ω to $\mathbb{R}^K$ and augment the simplicial tessellation with unbounded convex polyhedral cells, following [14]. For simplicity, the presentation below focuses on simplicial cells, but the formulation also applies to the additional unbounded cells.

CPA functions are Lipschitz continuous, and a CPAB transformation is defined by integrating a CPA velocity field. These fields form a finite-dimensional linear space, constructed as follows. Within each cell c, the stationary velocity field $\boldsymbol{v}$ takes the form

$$\boldsymbol{v}_c(\boldsymbol{p}) = \mathbf{A}_c \widetilde{\boldsymbol{p}}, \qquad \boldsymbol{p} \in \text{cell } c, \tag{2}$$

where $\mathbf{A}_c \in \mathbb{R}^{K \times (K+1)}$ contains the affine velocity coefficients for cell c, and $\widetilde{\boldsymbol{p}} = (\boldsymbol{p}^\top, 1)^\top$ denotes homogeneous coordinates. Defining the augmented matrix $\widetilde{\mathbf{A}}_c = \begin{bmatrix} \mathbf{A}_c^\top & \mathbf{0} \end{bmatrix}^\top \in \mathbb{R}^{(K+1)\times(K+1)}$, we can specialize Eq. 1 to the stationary affine velocity field within cell c, obtaining $\dot{\widetilde{\boldsymbol{p}}}^{(t)} = \widetilde{\mathbf{A}}_c \widetilde{\boldsymbol{p}}^{(t)}$. Its closed-form solution is $\widetilde{\boldsymbol{p}}^{(t)} = \exp(t\widetilde{\mathbf{A}}_c)\widetilde{\boldsymbol{p}}^{(0)}$, where exp denotes the matrix exponential. This solution is valid as long as the trajectory remains in c throughout $[0, t]$.

To ensure that the global velocity field $\boldsymbol{v}$ is continuous, we impose linear constraints requiring the affine velocity fields of neighboring cells to agree on their shared boundaries. Concatenating the cell-wise affine coefficients of a CPA velocity field yields a vector of length $D = CK(K+1)$, which must lie in the null space of the continuity constraint system. Let $d \leq D$ denote the dimension of this

null space and let $\mathbf{B} \in \mathbb{R}^{D \times d}$ have columns forming a basis for it. The stacked coefficients of any CPA velocity field can therefore be written as

$$\mathrm{vec}\big(\{\mathbf{A}_c\}_{c=1}^{C}\big) = \mathbf{B}\boldsymbol{\theta} \tag{3}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is the vector of parameters controlling the velocity field. Crucially, this guarantees that any choice of $\boldsymbol{\theta}$ produces a CPA velocity field, shifting the dimensionality of the optimization problem from the spatial resolution of the data to the complexity of the tessellation. The dimension d, which controls the expressiveness of the CPA velocity fields, increases with tessellation refinement.

We remark that, as shown in [13], additional optional linear constraints such as zero boundary conditions or ones that imply volume preservation can be simply added to the constraint system above. For a more detailed explanation of CPAB transformations, see [13, 14, 1].

Previous CPAB implementations. The original CPAB framework [13] and its main successor `libcpab` [10] compute gradients by propagating sensitivities through the integration trajectory (see [12] for details). This couples the spatial integration to the parameter dimension d, yielding a prohibitive $\mathcal{O}(dTN)$ backward complexity that restricts prior methods to coarse tessellations. Although early CPAB work [13, 14] also considered MCMC-based inference, this approach scales poorly in the fine-tessellation regime and cannot provide the gradients required for end-to-end learning; see Appendix J.

Adjoint methods and factorization. Adjoint sensitivity methods are standard for differentiating through ODE solvers with reduced memory cost [9]. However, for CPAB transformations, a generic adjoint formulation does not address the main computational bottleneck: gradients still involve the dependence of the velocity field on all d basis parameters. Moreover, as noted by ANODE [15], continuous adjoint formulations can yield inconsistent gradients under forward/backward discretization mismatch. Our approach is therefore closer in spirit to a discretize-then-optimize adjoint, but specialized to the CPAB structure: the linear CPA basis allows trajectory-dependent terms to be accumulated as cell-local quantities before a final projection onto the parameter basis.

3 Method

We introduce **FG-CPAB**, a gradient-factorization technique based on discrete-adjoint moment accumulation that decouples spatial trajectory integration from the parameter dimension. Existing CPAB implementations compute gradients *w.r.t.* the transformation parameters $\boldsymbol{\theta}$ by propagating forward sensitivities through the trajectory, incurring complexity $\mathcal{O}(dTN)$, where T is the number of integration steps, N is the number of spatial points, and $d = \dim(\boldsymbol{\theta})$. Since d grows with tessellation resolution and spatial dimension, this becomes prohibitive for fine tessellations and higher-dimensional inputs.

Our method's novelty is not merely in the use of a discrete adjoint, which alone does not remove the coupling to the parameter dimension. Rather, we also exploit the linear CPA basis to reorganize adjoint contributions into cell-local geometric moments that are independent of d. The full parameter gradient is then recovered by a single projection onto the basis, shifting all dependence on $\boldsymbol{\theta}$ to this final step and reducing the backward complexity from $\mathcal{O}(dTN)$ to $\mathcal{O}(TN + dC)$. **This enables a much higher d (i.e., finer tessellations), hence also much more expressive diffeomorphisms.**

Furthermore, to maximize these efficiency gains, we also introduce a sparse formulation of the CPA basis matrix $\mathbf{B}$ (detailed in Appendix F). This sparse construction improves the practical efficiency and memory footprint of both the forward and backward passes.

3.1 Trajectory integration and gradient computation

In the forward pass, the affine ODE within cell c admits the update

$$\widetilde{\boldsymbol{p}}^{(t+1)} = \exp(h\widetilde{\mathbf{A}}_c)\widetilde{\boldsymbol{p}}^{(t)} \tag{4}$$

where t indexes integration steps of size h. The update is exact provided that the trajectory remains within the same cell. Accordingly, we precompute $\mathbf{T}_c = \exp(h\widetilde{\mathbf{A}}_c)$, so that each forward step reduces to a matrix-vector multiplication. When a step crosses a cell boundary, in both the forward and gradient computation steps, we assume that the chosen integration step h is sufficiently small,

so the update is well approximated by averaging the velocity over the step. This is consistent with the standard CPAB formulation [10]. For the gradient computation, however, we use a different discretization. Specifically, we adopt an explicit midpoint rule (second-order Runge–Kutta),

$$\boldsymbol{p}^{(t+1)} = \boldsymbol{p}^{(t)} + h\,\mathbf{A}_c\,\widetilde{\boldsymbol{p}}_{\text{mid}}, \qquad \boldsymbol{p}_{\text{mid}} = \boldsymbol{p}^{(t)} + \tfrac{h}{2}\,\mathbf{A}_c\,\widetilde{\boldsymbol{p}}^{(t)}. \tag{5}$$

Using different forward and backward schemes is also the approach adopted in the original CPAB formulation [13] and is also natural from the Optimize-then-Discretize viewpoint [9]. However, since our velocity field is usually not differentiable at inter-cell boundaries, an Optimize-then-Discretize adjoint may yield inconsistent gradients (see ANODE [15]), because the backward adjoint trajectory can pass through a different sequence of cells than the forward trajectory. We therefore adopt a Discretize-then-Optimize treatment for the backward pass, and combine it with checkpointing (see § 3.3) for memory efficiency. See Appendix I for discussion and a numerical gradient check.

3.2 Adjoint factorization via moment accumulation

Let $\mathbf{A}_{c,\text{lin}} \in \mathbb{R}^{K\times K}$ denote the linear block of $\mathbf{A}_c$, and let $\mathbf{B}_{i,c}$ denote the i-th basis field's affine matrix in cell c. Differentiating Eq. 5 *w.r.t.* θ_i (the i-th component of $\boldsymbol{\theta}$), using $\partial\mathbf{A}_c/\partial\theta_i = \mathbf{B}_{i,c}$, yields the sensitivity recursion

$$\frac{\partial \boldsymbol{p}^{(t+1)}}{\partial \theta_i} = h\,\mathbf{B}_{i,c}\,\widetilde{\boldsymbol{p}}^{(t)}_{\text{mid}} + \tfrac{h^2}{2}\,\mathbf{A}_{c,\text{lin}}\,\mathbf{B}_{i,c}\,\widetilde{\boldsymbol{p}}^{(t)} + \mathbf{J}\,\frac{\partial \boldsymbol{p}^{(t)}}{\partial \theta_i}, \tag{6}$$

$$\text{where} \quad \mathbf{J} \coloneqq \frac{\partial \boldsymbol{p}^{(t+1)}}{\partial \boldsymbol{p}^{(t)}} = \mathbf{I} + h\,\mathbf{A}_{c,\text{lin}} + \tfrac{h^2}{2}\mathbf{A}^2_{c,\text{lin}} \tag{7}$$

is the Jacobian of one midpoint step with respect to the current position.Note that $\mathbf{J}$ depends on $\mathbf{A}_{c,\text{lin}}$ but not on the translation component of $\mathbf{A}_c$ (see Appendix H). The translation still influences the forward trajectory through the positions themselves.

A direct implementation of Eq. 6 would still require storing or propagating the full $N \times d$ sensitivities. Instead, we introduce the adjoint vector $\boldsymbol{\lambda}^{(t)}$, defined as the gradient of the loss *w.r.t.* the position at time t, and propagate it backward in time. The adjoint satisfies the discrete recurrence

$$\boldsymbol{\lambda}^{(t)} = \left(\mathbf{I} + h\,\mathbf{A}^\top_{c,\text{lin}} + \tfrac{h^2}{2}(\mathbf{A}^\top_{c,\text{lin}})^2\right)\boldsymbol{\lambda}^{(t+1)} = \mathbf{J}^\top\boldsymbol{\lambda}^{(t+1)}. \tag{8}$$

Using this adjoint recurrence and left-multiplying Eq. 6 by $\boldsymbol{\lambda}^{(t+1)\top}$ yields

$$\boldsymbol{\lambda}^{(t+1)\top}\frac{\partial \boldsymbol{p}^{(t+1)}}{\partial \theta_i} = \boldsymbol{\lambda}^{(t+1)\top}\left(h\,\mathbf{B}_{i,c}\,\widetilde{\boldsymbol{p}}^{(t)}_{\text{mid}} + \tfrac{h^2}{2}\,\mathbf{A}_{c,\text{lin}}\,\mathbf{B}_{i,c}\,\widetilde{\boldsymbol{p}}^{(t)}\right) + \boldsymbol{\lambda}^{(t)\top}\frac{\partial \boldsymbol{p}^{(t)}}{\partial \theta_i}. \tag{9}$$

The sensitivity recurrence propagates basis contributions through subsequent timesteps, making it difficult to group them by cell and factor out the basis. We note, however, that summing over all timesteps produces a telescoping series, so that all propagated terms cancel exactly (see Appendix G), leaving local contributions that can be grouped by cell to factor out the basis. Concretely, let $\mathcal{L}$ denote a loss and let $c_n^{(t)}$ denote the cell containing point n at step t. Then

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{n=1}^{N}\sum_{t=0}^{T-1}\boldsymbol{\lambda}_n^{(t+1)\top}\left(h\,\mathbf{B}_{i,c_n^{(t)}}\,\widetilde{\boldsymbol{p}}^{(t)}_{n,\text{mid}} + \tfrac{h^2}{2}\,\mathbf{A}_{c_n^{(t)},\text{lin}}\,\mathbf{B}_{i,c_n^{(t)}}\,\widetilde{\boldsymbol{p}}^{(t)}_n\right). \tag{10}$$

Moment accumulation. The remaining direct terms are bilinear in the adjoint $\boldsymbol{\lambda}$ and the homogeneous position $\widetilde{\boldsymbol{p}}$. Since the basis slice $\mathbf{B}_{i,c}$ is fixed for each cell c, we use the Frobenius identity $\boldsymbol{\lambda}^\top\mathbf{A}\mathbf{B}_{i,c}\widetilde{\boldsymbol{p}} = \langle\mathbf{B}_{i,c}, (\mathbf{A}^\top\boldsymbol{\lambda})\widetilde{\boldsymbol{p}}^\top\rangle_F$ to separate the basis from the trajectory-dependent factors. We then accumulate these factors over all visits to each cell. Defining $\mathcal{I}_c \coloneqq \{(n,t) \mid c_n^{(t)} = c\}$, the moment matrix for cell c is

$$\mathbf{M}_c = \sum_{(n,t)\in\mathcal{I}_c}\left[h\,\boldsymbol{\lambda}_n^{(t+1)}\,\widetilde{\boldsymbol{p}}^{(t)\top}_{n,\text{mid}} + \tfrac{h^2}{2}\,(\mathbf{A}^\top_{c,\text{lin}}\boldsymbol{\lambda}_n^{(t+1)})\,\widetilde{\boldsymbol{p}}^{(t)\top}_n\right]. \tag{11}$$

Crucially, $\mathbf{M}_c \in \mathbb{R}^{K\times(K+1)}$ is decoupled from $\mathbf{B}_{i,c}$, hence independent of the dimension d.

Gradient recovery. Once the moment matrices have been accumulated, all parameter gradients are recovered by a single sparse basis projection

$$\boxed{\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{c=1}^{C} \langle \mathbf{B}_{i,c},\, \mathbf{M}_c \rangle_F, \qquad \text{equivalently} \qquad \nabla_{\boldsymbol{\theta}} \mathcal{L} = \mathbf{B}^\top \operatorname{vec}\big(\{\mathbf{M}_c\}_{c=1}^{C}\big)} \,. \tag{12}$$

The core factorization result is Eq. 12. The trajectory-dependent computation is independent of d, with all explicit d-dependence confined to the final projection step. In practice, this final projection further benefits from a sparse basis construction of $\mathbf{B}$, which we derive in Appendix F. This sparse formulation reduces the memory footprint of the basis representation and improves the efficiency of the projection step.

3.3 Memory-efficient adjoint via segmented checkpointing

While factorization removes dependence on d, the reverse sweep still requires access to intermediate forward states. In our GPU implementation, each thread processes a single spatial point, so storing the full trajectory would require $\mathcal{O}(TN)$ storage across all threads (equivalently, $\mathcal{O}(T)$ per thread), which increases register pressure and can spill to local memory. Thus, we use segmented checkpointing with segment length S (we use $S = 4$). Instead of storing all $T + 1$ states along the trajectory, each thread stores only the states at segment boundaries, giving $\lceil T/S \rceil$ checkpoints per point. During the reverse sweep, each segment is replayed from its checkpoint and immediately used for adjoint propagation and moment accumulation. This reduces storage from $\mathcal{O}(TN)$ to $\mathcal{O}(N(T/S + S))$ overall, or from $\mathcal{O}(T)$ to $\mathcal{O}(T/S + S)$ per thread, at the cost of only constant-factor recomputation. See Appendix D.

3.4 Gradient computation algorithm

Incorporating segmented checkpointing, the factorization yields three sequential phases:

1. **Forward checkpointing** ($\mathcal{O}(TN)$). Integrate the trajectory forward, caching only sparse checkpoints at boundaries of length S.
2. **Adjoint sweep + moment accumulation** ($\mathcal{O}(TN)$). Iterate backward through the segments. For each segment, recompute the local forward trajectory from the checkpoint, propagate the adjoint via the transposed midpoint rule (Eq. 8), and atomically accumulate the products into $\mathbf{M}_c$ (Eq. 11).
3. **Sparse basis projection** ($\mathcal{O}(d\,C)$). Recover all d gradients via Eq. 12.

The total gradient computation cost is $\mathcal{O}(TN + d\,C)$: integration and adjoint propagation are independent of d, and projection is independent of N and T.

4 Results

In this section, we evaluate **FG-CPAB** along several axes: computational efficiency vs. the standard CPAB implementation, integration runtime vs. optimized diffeomorphic baselines, expressiveness and optimization behavior vs. dense SVFs, performance on 3D deformable registration, and CPAB as a low-dimensional geometry code for operator learning. We also show that composing CPAB transformations (which are not algebraically closed under composition) increases expressiveness further, and conclude with ablations on tessellation resolution, basis sparsity, and solver step count.

4.1 Acceleration and memory savings over the standard CPAB implementation

FG-CPAB substantially improves CPAB scaling over the standard `libcpab` [10] implementation. As Figure 1 shows, the gains are largest in the gradient computation phase, where standard CPAB is bottlenecked by parameter-dependent sensitivity propagation.

In 2D, FG-CPAB achieves up to $75\times$ faster forward passes, $10^4\times$ faster gradient computation, and over $500\times$ lower peak memory. Under our benchmark setup, the standard implementation does not run beyond $N_{\text{tess}} = 12$ because its backward pass couples the parameter dimension directly to kernel-launch geometry (see Appendix A.1), whereas FG-CPAB continues to scale to finer resolutions. This pattern is similar in other dimensions (see Appendix A): in 3D, up to $60\times$ forward

acceleration, $4{,}477\times$ backward acceleration, and a $180\times$ reduction in peak memory; in 1D, up to a $301\times$ backward speedup. These results match the proposed complexity reduction: by decoupling trajectory integration from parameter-space projection, FG-CPAB removes the main bottleneck that has historically restricted CPAB to coarse tessellations.

4.2 Forward and gradient computation runtime

We next evaluate FG-CPAB against optimized VoxelMorph SVF [3] and `torch-lddmm` [24] baselines on standard 2D and 3D grids. All methods are run with a batch size of 32. FG-CPAB uses a 15×15 tessellation in 2D (4×15^2 triangles) and a $15 \times 15 \times 15$ tessellation in 3D (5×15^3 tetrahedra). For each method, we measure the latency of the forward spatial transformation and the latency of computing the gradients *w.r.t.* the transformation parameters $\boldsymbol{\theta}$.

FG-CPAB is faster than both baselines, as reported in Table 1. In 2D, it reduces total runtime by $2.4\times$ relative to VoxelMorph and by $21.4\times$ relative to `torch-lddmm`. The same trend holds in 3D, with speedups of $2.3\times$ and $21.2\times$, respectively. These improvements are due to the combination of CPAB's low-dimensional parameterization and the proposed factorized gradient computation, which avoids propagating parameter-dependent sensitivities during trajectory integration.

	2D Integration (512×512)			**3D Integration** ($128 \times 128 \times 128$)		
Method	**Forward (ms)**	**Gradient (ms)**	**Peak Mem (MiB)**	**Forward (ms)**	**Gradient (ms)**	**Peak Mem (MiB)**
FG-CPAB	$\mathbf{1.69 \pm 0.06}$	$\mathbf{8.64 \pm 0.02}$	**197.97**	$\mathbf{31.10 \pm 0.18}$	$\mathbf{117.50 \pm 0.34}$	**2351.61**
VoxelMorph SVF	11.41 ± 0.02	13.42 ± 0.08	1330.13	119.14 ± 0.03	220.21 ± 0.24	14432.13
torch-lddmm	51.23 ± 0.41	169.75 ± 0.06	3162.13	732.81 ± 0.40	2414.12 ± 0.53	43288.13

Table 1: Forward transformation and parameter gradient ($\nabla_{\boldsymbol{\theta}}$) latency, alongside peak memory footprint, on 2D and 3D data.

4.3 Expressiveness and optimization stability compared with dense SVFs

We compare FG-CPAB against a dense stationary velocity field (SVF) baseline on a synthetic registration benchmark designed to isolate optimization behavior under a matched budget (1000 fit steps). In both 2D (512×512) and 3D ($128 \times 128 \times 128$), we generate blob-based source images, warp them with sampled smooth deformations to form targets, and fit either a CPAB transformation (using FG-CPAB) or a dense SVF integrated with 7 scaling-and-squaring steps. FG-CPAB uses a 15×15 tessellation in 2D (4×15^2 triangles) and a $15 \times 15 \times 15$ tessellation in 3D (5×15^3 tetrahedra); full implementation details are provided in Appendix C.

Table 2 reports reconstruction error, fitting time, and the fraction of non-positive Jacobian determinants, while Figure 2 shows qualitative results. Under the same optimization budget, FG-CPAB consistently reaches lower residual error, converges faster, and exhibits fewer topology violations. These gains arise from the compact CPAB parameterization: optimizing within the low-dimensional space spanned by $\mathbf{B}$ acts as an implicit regularizer, substantially improving optimization behavior, potentially due to a smoother optimization landscape than that of a dense SVF parameterization.

Importantly, our claim is not that dense SVFs lack representational power. Rather, the comparison highlights a practical trade-off between asymptotic flexibility and optimization efficiency. Dense SVFs can represent highly detailed deformations, but their large parameter space makes them much harder to optimize reliably under realistic compute budgets. In contrast, CPAB offers a more compact parameterization that appears to facilitate efficient optimization while retaining sufficient expressiveness for the evaluated deformations.

To further probe this trade-off, Appendix C.1 studies extended-budget optimization and regularized dense SVFs. Those results show that heavily regularized dense SVFs can eventually approach, and in some cases surpass, CPAB in final reconstruction error, but only after orders of magnitude more optimization steps. Thus, the main advantage of CPAB is practical: it achieves strong alignment quality much more reliably and with substantially less compute.

Dim	Params (C,S)	MSE C ↓	MSE S ↓	Fit C (s) ↓	Fit S (s) ↓	$\lvert J\rvert \leq 0$ (%) (C,S) ↓
2D	842 / 524288	**8.853e-05 ± 3.9e-05**	2.651e-03 ± 1.2e-03	**0.525 ± 0.044**	1.356 ± 0.007	**0.028%** / 0.178%
3D	8232 / 6291456	**7.255e-05 ± 2.5e-05**	1.676e-04 ± 4.1e-05	**6.140 ± 0.021**	12.906 ± 0.021	**0.012%** / 0.021%

Table 2: Quantitative synthetic registration results comparing FG-CPAB (C) and dense SVF (S) under the same optimization budget in 2D and 3D. Metrics report final reconstruction MSE, wall-clock fitting time, and the percentage of spatial locations with non-positive Jacobian determinant.

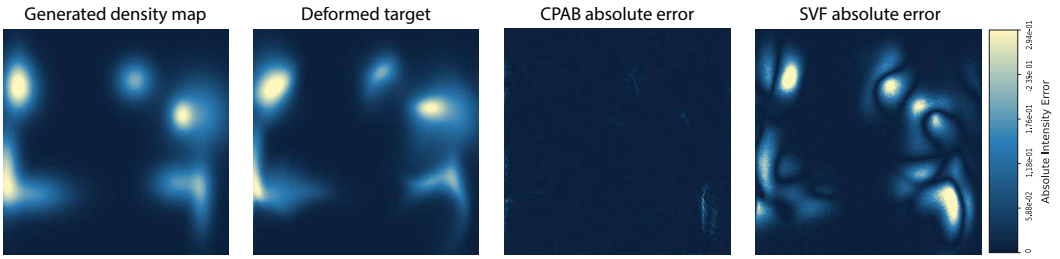


Figure 2: **Qualitative comparison of CPAB and dense SVF on synthetic deformation fitting.** Left to right: Source image, deformed target, absolute alignment error after optimization with CPAB, and absolute alignment error after optimization with dense SVF under the same optimization budget. CPAB achieves near-zero residual error despite its compact parameterization, whereas dense SVF more often exhibits structured residual artifacts under the matched budget.

4.4 CPAB for 3D deformable registration

We evaluate FG-CPAB in the classical *instance-specific optimization* setting for deformable registration. For each test volume pair, we directly optimize the compact CPAB parameter vector $\boldsymbol{\theta}$ (using the factorized gradients from § 3). No offline pre-training on any dataset is performed. This setup highlights the practical ease of optimization unlocked by our method: the compact parameterization and $\mathcal{O}(TN + dC)$ backward pass enable fast, stable, topology-preserving convergence per volume pair. Full implementation details, including the optimizer and loss function, are provided in Appendix E.1.

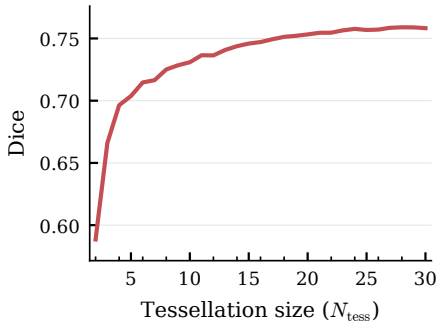
Table 3 reports results on the IXI brain MRI benchmark. A single CPAB transformation reaches 0.760 ± 0.023 Dice with near-perfect topology preservation ($< 0.0001\%$ non-positive Jacobians). This performance is competitive with strong learned baselines, despite requiring no offline pre-training. Direct per-pair optimization converges in ≈ 5 seconds per volume pair on an RTX A6000 GPU, compared with ≈ 135 seconds for LDDMM. FG-CPAB therefore offers a substantially better accuracy-runtime trade-off than classical optimization-based diffeomorphic methods.

We further observe that sequential composition of CPAB transformations increases expressiveness while preserving topology. Composing i transformations raises the Dice score from 0.760 ($i = 1$) to 0.765 ($i = 4$), with the fraction of non-positive Jacobians remaining below 0.0001 throughout. These results demonstrate that the proposed factorized formulation makes fine-tessellation CPAB a practical, optimization-friendly primitive for 3D deformable registration.

Table 3: Registration performance comparison on the IXI dataset. For FG-CPAB, i indicates the number of composed transformations.

Model	**Dice**	$\lvert J\rvert \leq 0$ **(%)**
Affine	0.386 ± 0.195	-
SyN	0.645 ± 0.152	< 0.0001
NiftyReg	0.645 ± 0.167	0.020 ± 0.046
LDDMM	0.680 ± 0.135	< 0.0001
VoxelMorph	0.732 ± 0.123	1.522 ± 0.336
CycleMorph	0.737 ± 0.123	1.719 ± 0.382
MIDIR	0.742 ± 0.128	< 0.0001
ViT-V-Net	0.734 ± 0.124	1.609 ± 0.319
CoTr	0.735 ± 0.135	1.298 ± 0.343
PVT	0.727 ± 0.128	1.858 ± 0.314
nnFormer	0.747 ± 0.135	1.595 ± 0.358
TransMorph	0.753 ± 0.123	1.579 ± 0.328
ScaMorph	0.755 ± 0.132	1.400 ± 0.329
FG-CPAB	0.760 ± 0.023	< 0.0001
FG-CPAB ($i = 2$)	0.762 ± 0.024	< 0.0001
FG-CPAB ($i = 3$)	0.763 ± 0.024	< 0.0001
FG-CPAB ($i = 4$)	$\mathbf{0.765 \pm 0.024}$	< 0.0001

Figure 3: Dice score vs. tessellation size on IXI. At tessellation 30 we have $5 \times 30^3 = 135,000$ unique cells.



4.5 CPAB as a low-dimensional geometry code for neural operator learning

DIMON [39] learns geometry-dependent PDE operators by transporting inputs and solutions from varying domains $\{\Omega^\theta\}$ to a reference domain Ω^0, and conditioning the operator on a low-dimensional geometry code. In the original DIMON pipeline, this code is obtained by compressing LDDMM high-dimensional deformation representation with principal component analysis (PCA).

We consider the same reaction–diffusion setting, but replace the geometry encoding with CPAB. For each target domain Ω^θ, we fit a CPAB diffeomorphism $\Phi_\theta : \Omega^0 \to \Omega^\theta$ and use the corresponding parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$ directly as the geometry code. Unlike PCA-based LDDMM encodings, CPAB provides a native low-dimensional diffeomorphic parameterization and therefore removes the need for a separate compression step. We match the CPAB parameter count to that of the PCA encoding.

This has two advantages: the code is directly tied to the transport map, and its dimension is controlled by the tessellation, yielding an interpretable expressiveness–complexity trade-off. Table 4 shows that FG-CPAB attains the lowest test relative L^2 error under the same operator backbone, suggesting that compact diffeomorphic parameterizations can serve as effective geometry descriptors for neural operator learning. We defer implementation details to Appendix E.2.

Experiment	Geometry model	Test rel. L^2 mean	Test rel. L^2 std
FG-CPAB	FG-CPAB parameterization	**0.052811**	**0.022436**
LDDMM deformation PCA	PCA deformation code	0.082821	0.033168

Table 4: Reaction–diffusion operator-learning results using different geometry encodings. Lower is better for both metrics.

4.6 CPAB composition for non-stationary velocity fields

Table 3 shows that FG-CPAB already performs competitively with, and in our setting better than, LDDMM. However, the two models differ in expressiveness: a single CPAB transformation is generated by a stationary CPA velocity field, whereas LDDMM can represent more general deformations through time-varying velocity fields. Therefore, there exist orientation-preserving diffeomorphisms that cannot be represented (or even well approximated) by a single CPAB diffeomorphism. Fortunately, the efficiency of our implementation also makes it practical to compose m CPAB transformations,

$$\Phi = \Phi_m \circ \cdots \circ \Phi_1. \tag{13}$$

Although such a composition is not necessarily itself a CPAB transformation (CPAB diffeomorphisms form a subset, but not a subgroup, of the diffeomorphism group and are not closed under composition [13]), it remains a diffeomorphism, since diffeomorphisms are closed under composition. This provides a practical way to increase expressiveness while retaining CPAB's efficiency at each stage.

Figure 4 illustrates this point on a synthetic example with two blobs following crossing trajectories. A single CPAB transformation cannot model this motion accurately and leaves a visible residual error. In contrast, a composition of CPAB transformations successfully captures the deformation by distributing the motion across multiple stages. Thus, CPAB composition provides a simple and efficient way to approximate deformations that would otherwise require a non-stationary velocity field, bringing CPAB closer in spirit to the flexibility of LDDMM, but at a much lower cost.

4.7 Ablations and hyper-parameter analysis

Basis sparsity. We first ablate the sparse basis construction by comparing sparse and dense representations of the CPA basis. The sparse formulation yields substantial memory savings, with additional runtime benefits that become more pronounced at larger tessellation sizes. This shows that sparsity is an important complement to gradient factorization for making high-resolution CPAB practical. Full results are provided in Appendix B.2.

Tessellation resolution. We next study tessellation resolution, the main expressiveness knob of CPAB. Performance improves as the tessellation becomes finer, both on IXI (Figure 3) and in the synthetic setting, showing that the additional local flexibility is useful in practice. This result also highlights the importance of our efficient backward pass, which makes these larger tessellations practical to optimize. Full results are provided in Appendix B.1.

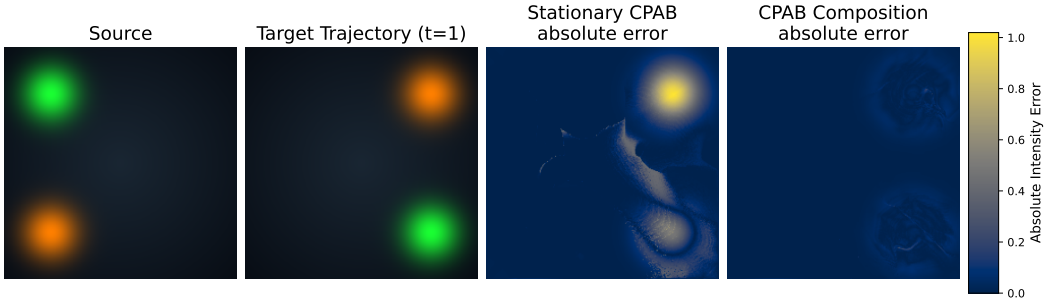


Figure 4: **A composition of CPAB transformations can model deformations that a single stationary CPAB cannot.** From left to right: source image, target image, absolute error for a single CPAB transformation, and absolute error for a composition of CPAB transformations. The target requires the two blobs to follow crossing trajectories, which cannot be captured well by a single stationary velocity field. While a single CPAB leaves substantial residual error, the composed model greatly reduces the error by realizing the deformation through multiple diffeomorphic stages.

Solver step count. We also study the number of ODE solver steps used for CPAB integration. All results in the paper use 50 CPAB solver steps unless noted otherwise. Accuracy improves rapidly at low step counts and then largely plateaus around 50 steps, while runtime continues to increase. This supports using 50 steps as a robust default, consistent with previous CPAB implementations. Full results are provided in Appendix B.3.

5 Discussion

Our results show that **FG-CPAB** makes CPAB substantially more practical as a diffeomorphic primitive for modern spatial or temporal modeling. By decoupling trajectory integration from parameter-space projection, the proposed formulation removes the main computational bottleneck of standard CPAB and enables efficient optimization with much finer tessellations in both 2D and 3D. In turn, this improved scalability translates into better practical expressiveness, competitive registration accuracy, and strong topology preservation.

While adjoint methods are standard for general ODEs [9], the linear CPA basis permits an exact factorization into cell-local moments that prior general-purpose adjoint solvers cannot exploit. The key insight is the reorganization of sensitivities into parameter-independent geometric statistics that are later projected through the sparse CPA basis. This delivers the $\mathcal{O}(TN + dC)$ complexity and the dramatic practical speedups reported here.

These efficiency gains, however, do not eliminate the dimensionality limitations of tessellation-based models. Although the factorization extends directly to any spatial dimension K, with $K \times (K+1)$ moments and parameter dependence confined to the final projection, the number of cells grows exponentially with K. We therefore restrict our empirical claims to spatial dimensions $K \leq 3$. Extending the approach to higher-dimensional domains would likely require additional structure, such as separable, low-rank, or adaptively refined velocity fields.

Overall, this work shifts CPAB from a promising but difficult-to-scale formulation toward a useful, optimization-friendly tool for large-scale diffeomorphic learning in both classical optimization and modern deep-learning pipelines.

References

[1] Amit Aflalo and Oren Freifeld. A regularity condition under which a cpab transformation is a diffeomorphism, 2026. URL https://arxiv.org/abs/2610.32369. 2, 4

[2] John Ashburner. A fast diffeomorphic image registration algorithm. *Neuroimage*, 38(1):95–113, 2007. 2, 3

[3] Guha Balakrishnan, Amy Zhao, Mert R Sabuncu, John Guttag, and Adrian V Dalca. Voxelmorph: a learning framework for deformable medical image registration. *IEEE transactions on medical imaging*, 38(8):1788–1800, 2019. 7, 30

[4] M Faisal Beg, Michael I Miller, Alain Trouvé, and Laurent Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *International journal of computer vision*, 61(2):139–157, 2005. 2, 3, 30

[5] Jian Bi, Qianliang Wu, Jianjun Qian, Lei Luo, and Jian Yang. Shaping without tearing: Controllable diffeomorphic deformations for topology-preserving 3d point cloud augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 19711–19719, 2026. 2

[6] Mathieu Carrière, Marc Theveneau, and Théo Lacombe. Diffeomorphic interpolation for efficient persistence-based topological optimization. *Advances in Neural Information Processing Systems*, 37:27274–27294, 2024. 2

[7] Irit Chelly, Shahaf E Finder, Shira Ifergane, and Oren Freifeld. Trainable highly-expressive activation functions. In *European Conference on Computer Vision*, 2024. 2

[8] Junyu Chen, Eric C Frey, Yufan He, William P Segars, Ye Li, and Yong Du. Transmorph: Transformer for unsupervised medical image registration. *Medical image analysis*, 82:102615, 2022. 2, 23

[9] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 4, 5, 10

[10] Nicki S. Detlefsen. libcpab. https://github.com/SkafteNicki/libcpab, 2018. 1, 2, 4, 5, 6, 14, 28, 30

[11] Nicki Skafte Detlefsen, Oren Freifeld, and Søren Hauberg. Deep diffeomorphic transformer networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4403–4412, 2018. 2

[12] Oren Freifeld. Deriving the cpab derivative. Technical report, 2018. 4

[13] Oren Freifeld, Soren Hauberg, Kayhan Batmanghelich, and John W Fisher. Highly-expressive spaces of well-behaved transformations: Keeping it simple. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2911–2919, 2015. 1, 2, 3, 4, 5, 9, 29

[14] Oren Freifeld, Søren Hauberg, Kayhan Batmanghelich, and John W. Fisher. Transformations based on continuous piecewise-affine velocity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2496–2509, 2017. doi: 10.1109/TPAMI.2016.2646685. 1, 2, 3, 4, 28, 29

[15] Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2019. 4, 5

[16] Kunal Gupta and Manmohan Chandraker. Neural mesh flow: 3d manifold mesh generation via diffeomorphic flows. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1747–1758. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1349b36b01e0e804a6c2909a6d0ec72a-Paper.pdf. 2

[17] Kun Han, Shanlin Sun, Thanh-Tung Le, Xiangyi Yan, Haoyu Ma, Chenyu You, and Xiaohui Xie. Hybrid neural diffeomorphic flow for shape representation and generation via triplane. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 7707–7717, 2024. 2

[18] Adam Hartshorne, Allen Paul, Tony Shardlow, and Neill DF Campbell. Arc-flow: Articulated, resolution-agnostic, correspondence-free matching and interpolation of 3d shapes under flow fields. In *2025 International Conference on 3D Vision (3DV)*, pages 11–33. IEEE, 2025. 2

[19] Søren Hauberg, Oren Freifeld, Anders Boesen Lindbo Larsen, John Fisher, and Lars Hansen. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In *Artificial intelligence and statistics*, pages 342–350. PMLR, 2016. 2

[20] Malte Hoffmann, Benjamin Billot, Douglas N Greve, Juan Eugenio Iglesias, Bruce Fischl, and Adrian V Dalca. Synthmorph: learning contrast-invariant registration without acquired images. *IEEE transactions on medical imaging*, 41(3):543–558, 2021. 19

[21] IXI Project. IXI Dataset. https://brain-development.org/ixi-dataset/. Information eXtraction from Images, EPSRC Grant GR/S21533/02. 30

[22] Ilya Kaufman, Ron Shapira Weber, and Oren Freifeld. Cyclic diffeomorphic transformer nets for contour alignment. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 349–353, 2021. doi: 10.1109/ICIP42928.2021.9506570. 2

[23] Alaa Kryeem, Shmuel Raz, Dana Eluz, Dorit Itah, Hagit Hel-Or, and Ilan Shimshoni. Personalized monitoring in home healthcare: An assistive system for post hip replacement rehabilitation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 1868–1877, October 2023. 2

[24] Brian Lee. torch-lddmm: Gpu and cpu implementation of lddmm image registration. https://github.com/brianlee324/torch-lddmm. GitHub repository, accessed 2026-04-26. 7

[25] Yuchen Liu, Ling Wang, Xiaolin Ning, Yang Gao, and Defeng Wang. Enhancing unsupervised learning in medical image registration through scale-aware context aggregation. *iScience*, 28(2), 2025. 2

[26] Krishna S Mantri, Xinzhi Wang, Carola-Bibiane Schönlieb, Bruno Ribeiro, Beatrice Bevilacqua, and Moshe Eliasof. Digraf: Diffeomorphic graph-adaptive activation function. *Advances in Neural Information Processing Systems*, 37:3649–3681, 2024. 2

[27] Krishna Sri Ipsit Mantri, Carola-Bibiane Schönlieb, Bruno Ribeiro, Chaim Baskin, and Moshe Eliasof. Ditask: Multi-task fine-tuning with diffeomorphic transformations. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 25218–25229, 2025. 2

[28] Iñigo Martinez, Elisabeth Viles, and Igor G Olaizola. Closed-form diffeomorphic transformations for time series alignment. In *International Conference on Machine Learning*, pages 15122–15158. PMLR, 2022. 2, 14, 30

[29] Nour Neifar, Achraf Ben-Hamadou, Afef Mdhaffar, Mohamed Jmaiel, and Bernd Freisleben. Leveraging statistical shape priors in gan-based ecg synthesis. *IEEE Access*, 12:36002–36015, 2024. 2

[30] Daniel Rueckert, Luke I Sonoda, Carmel Hayes, Derek LG Hill, Martin O Leach, and David J Hawkes. Nonrigid registration using free-form deformations: application to breast mr images. *IEEE transactions on medical imaging*, 18(8):712–721, 2002. 19

[31] Pola Schwöbel, Frederik Rahbæk Warburg, Martin Jørgensen, Kristoffer Hougaard Madsen, and Søren Hauberg. Probabilistic spatial transformer networks. In *Uncertainty in artificial intelligence*, pages 1749–1759. PMLR, 2022. 2

[32] Guy Shacht, Dov Danon, Sharon Fogel, and Daniel Cohen-Or. Single pair cross-modality super resolution. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 6378–6387, 2021. 2

[33] Ron Shapira Weber, Shahar Ben Ishay, Andrey Lavrinenko, Shahaf E Finder, and Oren Freifeld. Timepoint: Accelerated time series alignment via self-supervised keypoint and descriptor learning. In *International Conference on Machine Learning*, 2025. 2

[34] Ron A Shapira Weber, Matan Eyal, Nicki Skafte, Oren Shriki, and Oren Freifeld. Diffeomorphic temporal alignment nets. *Advances in neural information processing systems*, 32, 2019. 2

[35] Nicki Skafte and Søren Hauberg. Explicit disentanglement of appearance and perspective in generative models. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[36] Shanlin Sun, Kun Han, Deying Kong, Hao Tang, Xiangyi Yan, and Xiaohui Xie. Topology-preserving shape reconstruction and registration via neural diffeomorphic flow. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20845–20855, 2022. 2

[37] Hexiang Wang, Fengqi Liu, Qianyu Zhou, Ran Yi, Xin Tan, and Lizhuang Ma. Continuous piecewise-affine based motion model for image animation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 5427–5435, 2024. 2

[38] Ron Shapira Weber and Oren Freifeld. Regularization-free diffeomorphic temporal alignment nets. In *International Conference on Machine Learning*, pages 30794–30826. PMLR, 2023. 2

[39] Minglang Yin, Nicolas Charon, Ryan Brody, Lu Lu, Natalia Trayanova, and Mauro Maggioni. A scalable framework for learning the geometry-dependent solution operators of partial differential equations. *Nature Computational Science*, 4:928–940, 2024. doi: 10.1038/s43588-024-00732-2. 2, 9, 23, 30

[40] Miaomiao Zhang and P Thomas Fletcher. Finite-dimensional lie algebras for fast diffeomorphic image registration. In *International conference on information processing in medical imaging*, pages 249–260. Springer, 2015. 3

[41] Miaomiao Zhang and P Thomas Fletcher. Fast diffeomorphic image registration via fourier-approximated lie algebras. *International Journal of Computer Vision*, 127(1):61–73, 2019. 3

[42] Zhiwei Zhao, Changqing Liu, Yingguang Li, Zhibin Chen, and Xu Liu. Diffeomorphism neural operator for various domains and parameters of partial differential equations. *Communications Physics*, 8:15, 2025. doi: 10.1038/s42005-024-01911-3. 2

A Performance

In this section, we evaluate the computational efficiency of our optimized CPAB implementation against the strongest available baselines. In 2D and 3D, we compare against the standard `libcpab` [10] implementation. In 1D, we additionally compare against `DIFW` [28], which to our knowledge is the fastest available implementation in the 1D setting, although it only supports 1D transformations. To ensure reliable measurements, all reported results represent the mean of 50 execution iterations, preceded by a 10-iteration warmup phase. All experiments were performed on a standardized workload of 125,000 points with a batch size of 16.

A.1 Why the standard backward stops scaling

The failure of the standard implementation at fine tessellation is not an intrinsic limitation of CPAB, but a consequence of how the reference backward pass is organized on the GPU. In previous implementations, sensitivities are propagated separately for every parameter throughout the full integration trajectory. As tessellation becomes finer, the parameter dimension d grows, and the backward pass therefore scales directly with d.

The key hardware bottleneck is that the parameter axis is encoded directly in the backward-kernel launch geometry. As a result, increasing tessellation enlarges the kernel grid itself and eventually reaches a device-dependent launch ceiling; this is the *launch-limit problem* referred to in the main text. In practice, once tessellations become moderately fine, the reference implementation is governed by this parameter-dependent kernel structure rather than by the geometric integration alone.

FG-CPAB avoids this bottleneck by removing parameter dependence from the integration itself. The fused adjoint kernel is launched only over points and batch elements, accumulates parameter-independent cell-wise moments, and defers the dependence on d to a separate final projection step. This is the practical reason the optimized implementation continues to run at tessellation levels for which the standard backward becomes unusable.

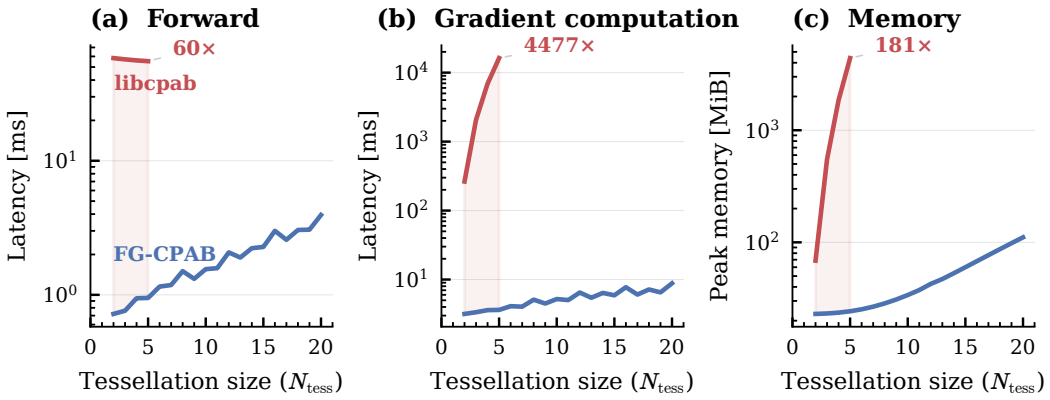


Figure 5: **Performance scaling with tessellation size (N_{tess}).** Comparison of our optimized 3D CPAB vs. standard implementation on a single RTX A6000 GPU. **(a)** Forward pass latency. **(b)** Gradient computation latency. **(c)** Peak GPU memory allocation. Our approach yields up to a $60\times$ speedup in the forward pass and a $4477\times$ speedup in the gradient computation, while reducing peak memory by over $180\times$. Notably, the standard implementation does not support $N_{\text{tess}} > 5$, whereas ours scales efficiently.

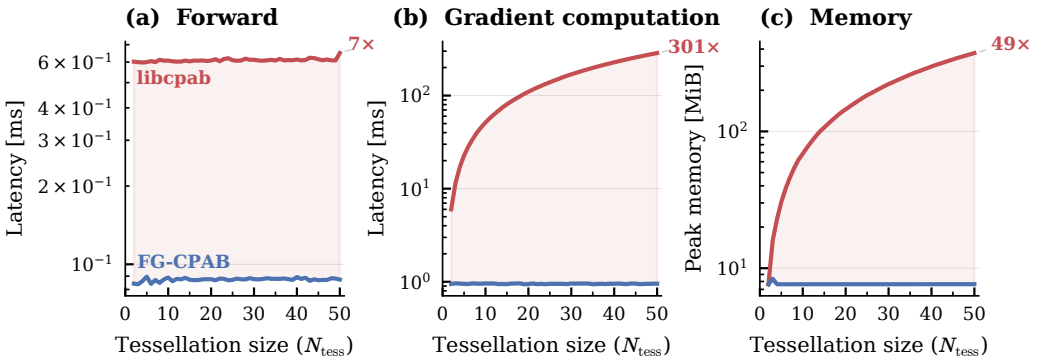


Figure 6: **Performance scaling with tessellation size (N_{tess}).** Comparison of our optimized 1D CPAB vs. `libcpab` implementation on a single RTX A6000 GPU. **(a)** Forward pass latency. **(b)** Gradient computation latency. **(c)** Peak GPU memory allocation. Our approach yields up to a $7\times$ speedup in the forward pass and a $301\times$ speedup in the gradient computation, while reducing peak memory by $49\times$.

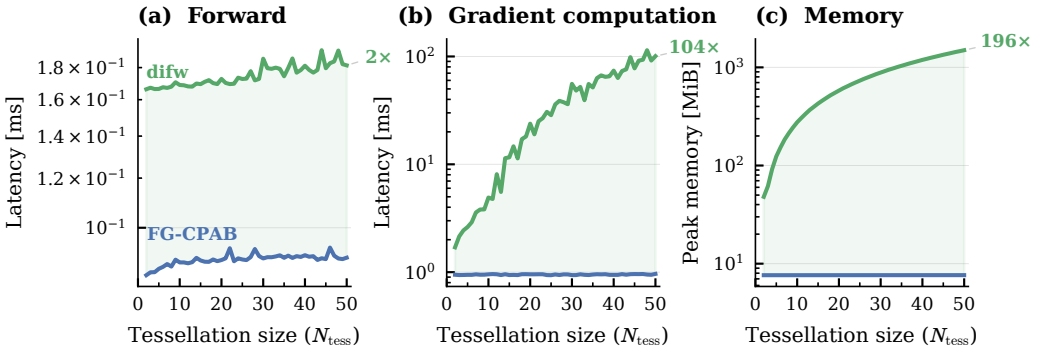


Figure 7: **Performance scaling with tessellation size (N_{tess}).** Comparison of our optimized 1D CPAB vs. the DIFW implementation on a single RTX A6000 GPU. **(a)** Forward pass latency. **(b)** Gradient computation latency. **(c)** Peak GPU memory allocation. Our approach yields up to a $2\times$ speedup in the forward pass and a $104\times$ speedup in the gradient computation, while reducing peak memory by $196\times$.

B Ablations and hyper-parameter analysis

B.1 Tessellation resolution hyper-parameter analysis

We study the effect of tessellation resolution on CPAB performance, since tessellation size is the main factor controlling the trade-off between expressiveness and computational cost.

Setup. We vary the tessellation resolution while keeping the rest of the optimization protocol fixed. We evaluate this both on the IXI brain MRI registration benchmark (Table 3, Figure 3) and on the synthetic fitting task described in § C.1. The purpose of this ablation is to isolate how much of the final performance comes from access to finer tessellations.

Findings. Figure 8 shows a clear and consistent trend: increasing tessellation resolution improves alignment quality up to the largest resolutions tested. This confirms that finer CPAB parameterizations provide additional local flexibility that is useful for modeling complex deformations. At the same time, the results highlight a central practical limitation of previous CPAB implementations: without an efficient backward pass, these larger tessellations are too slow or memory-intensive to optimize reliably, especially in 3D.

Conclusion. The ablation demonstrates that the scalability enabled by gradient factorization is directly tied to model quality. In other words, our method is valuable not only because it makes CPAB faster, but because it enables operation in a more expressive tessellation regime that leads to better registration results.

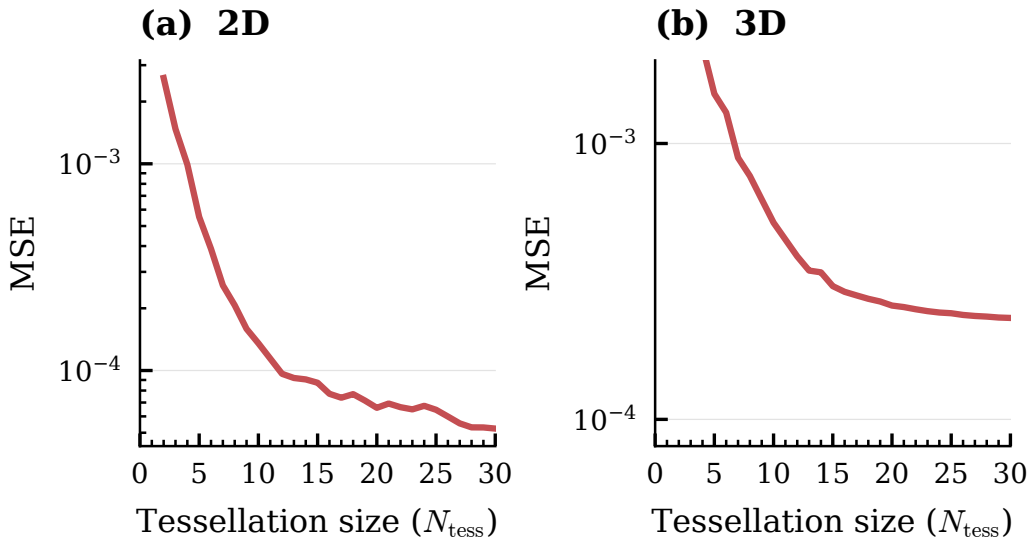


Figure 8: **Effect of tessellation resolution on registration performance.** Increasing the tessellation resolution improves deformation quality by providing a more expressive piecewise-affine parameterization. This trend is observed across both real and synthetic registration settings, supporting the claim that the scalability unlocked by gradient factorization is important not only for efficiency, but also for final task performance.

B.2 Sparse basis ablation

We compare sparse and dense representations of the CPA basis matrix $\mathbf{B}$ for 3D transformations on a 128^3 grid. As tessellation resolution increases, $\mathbf{B}$ grows quickly in both dimensions: the rows correspond to the stacked cell-wise affine coefficients, while the columns correspond to the free CPAB parameters. Since finer tessellations increase both the number of cells and the number of free parameters, a dense basis becomes increasingly expensive, especially in 3D.

Table 5: Sparse vs. Dense CPA basis across tessellation size. Forward/backward are mean runtime in ms (lower is better); peak is memory in MiB (lower is better).

	Sparse basis			Dense basis		
Tess.	Fwd (ms)	Bwd (ms)	Peak (MiB)	Fwd (ms)	Bwd (ms)	Peak (MiB)
2	10.8	51.9	384	10.9	51.9	384
3	11.3	54.0	384	11.3	54.1	384
4	11.5	54.9	385	11.4	55.0	385
5	11.6	55.6	385	11.6	55.7	385
6	11.7	56.1	386	11.7	56.3	386
7	11.9	57.0	388	12.0	57.1	388
8	12.2	57.4	390	12.5	57.7	390
9	12.6	58.0	392	13.0	58.4	392
10	13.8	59.6	395	14.5	60.3	501
11	14.4	59.9	399	15.8	61.5	914
12	15.4	62.8	403	17.8	65.1	1580
13	15.8	62.5	408	19.7	65.7	2607
14	16.9	66.2	414	23.4	71.9	4140
15	17.2	65.1	423	26.9	74.1	6360
16	19.5	72.6	432	34.4	86.2	9492
17	18.4	67.6	438	41.1	87.7	13818
18	20.5	75.4	450	52.0	104.8	19674
19	20.5	72.1	462	64.3	113.4	27467
20	23.5	83.4	474	85.0	140.3	37678

Table 5 shows that the sparse basis mainly improves memory usage. At small tessellations, sparse and dense implementations behave similarly, but at tessellation size 20, peak memory drops from 37678 MiB to 474 MiB. Runtime also improves, though more modestly: forward latency decreases from 85.0 ms to 23.5 ms and backward latency from 140.3 ms to 83.4 ms.

This trend is expected. The main runtime speedup in our method comes from gradient factorization, which removes parameter-dependent integration from the backward pass. In contrast, sparsifying $\mathbf{B}$ mainly reduces the cost of storing and applying the final basis projection. Its primary effect is therefore memory saving, with a secondary but smaller effect on runtime.

B.3 Solver step hyper-parameter analysis

We study how the number of ODE solver steps affects the trade-off between runtime and registration accuracy. Unless otherwise noted, all results in the paper use 50 CPAB solver steps, matching the standard setting used in prior CPAB implementations.

Setup. We use the same synthetic registration protocol as in § C.1 and vary only the number of CPAB solver steps. The sweep is performed for 2D transformations with a 30×30 tessellation, so the goal is to isolate the effect of integration accuracy on optimization behavior without changing the rest of the training setup.

Findings. Table 6 shows a clear runtime-accuracy trade-off. Very small step counts are fastest per iteration, but they incur substantially worse registration error. Increasing the number of solver steps from 5 to 20 reduces the mean MSE from 1.42×10^{-2} to 1.04×10^{-4}, i.e., by more than two orders of magnitude. Beyond 20 steps, however, the curve largely plateaus. The best mean MSE in the full sweep is 8.30×10^{-5} at 65 steps, while 50 steps already achieves 8.89×10^{-5}. Extending the sweep to 100 steps yields 8.31×10^{-5}, essentially the same accuracy, but increases mean fitting time from 0.287 s to 0.468 s and backward latency from 0.415 ms to 0.720 ms.

Conclusion. These results suggest that the paper's default choice of 50 solver steps is a robust operating point: it lies in the accuracy plateau, captures most of the attainable improvement, and avoids the noticeably higher cost of longer integrations.

n_{solver}	**Forward (ms)** ↓	**Backward (ms)** ↓	**Fit time (s)** ↓	**Registration MSE** ↓
5	0.1054	0.1822	0.2559	1.42×10^{-2}
10	0.1068	0.1792	0.2490	2.27×10^{-3}
15	0.1080	0.2080	0.2486	3.23×10^{-4}
20	0.1138	0.2279	0.2438	1.04×10^{-4}
25	0.1206	0.2523	0.2482	9.44×10^{-5}
30	0.1248	0.2910	0.2483	9.18×10^{-5}
35	0.1310	0.3201	0.2482	9.07×10^{-5}
40	0.1376	0.3649	0.2558	9.72×10^{-5}
45	0.1441	0.3805	0.2686	8.78×10^{-5}
50	0.1527	0.4145	0.2869	8.89×10^{-5}
55	0.1584	0.4497	0.3066	9.00×10^{-5}
60	0.1659	0.4748	0.3220	8.60×10^{-5}
65	0.1730	0.5147	0.3433	8.30×10^{-5}
70	0.1765	0.5392	0.3599	8.47×10^{-5}
75	0.1838	0.5673	0.3762	8.60×10^{-5}
80	0.1901	0.6109	0.4003	8.44×10^{-5}
85	0.1960	0.6306	0.4134	8.47×10^{-5}
90	0.2011	0.6623	0.4319	8.70×10^{-5}
95	0.2082	0.6966	0.4520	8.81×10^{-5}
100	0.2115	0.7197	0.4678	8.31×10^{-5}

Table 6: Full CPAB ODE solver-step sweep over $n_{\text{solver}} \in \{5, 10, \ldots, 100\}$. Lower values are better. Accuracy improves dramatically from the very low-step regime and then plateaus, while runtime continues to increase with the number of solver steps.

C Implementation Details

C.1 Optimization Stability and Practical Expressiveness Compared with Dense SVFs

Data and deformation generation. We follow a simulation-based registration protocol where correspondences are known by construction [20]. We evaluate in both 2D and 3D with spatial sizes 512×512 and $128 \times 128 \times 128$, respectively. For each case, we synthesize a moving image as a sum of 9 random Gaussian blobs (blob standard deviations sampled uniformly from $[0.03, 0.12]$), add Gaussian intensity noise with standard deviation 0.01, and normalize intensities to $[0, 1]$. We then sample a smooth random deformation from a coarse control lattice and upsample it to full resolution (FFD-like deformation model [30]), using control sizes 8×8 in 2D and $6 \times 6 \times 6$ in 3D with deformation amplitude 0.03. The fixed image is obtained by warping the moving image with this sampled transformation, yielding synthetic moving/fixed pairs with exact ground-truth deformation fields.

Model configuration. CPAB uses tessellations of size 15×15 in 2D and $15 \times 15 \times 15$ in 3D. The dense SVF baseline is parameterized as a full-resolution velocity field and integrated using scaling-and-squaring with 7 squaring steps. This yields a comparison between a compact low-dimensional diffeomorphic parameterization and a standard dense voxel-wise one.

Optimization protocol. Both methods are optimized against the same reconstruction objective using Adam. For the fixed-budget comparison reported in the main paper, each method is run for 1000 iterations with 5 restarts per target registration. Per dimension, results are aggregated over 100 independently generated target registrations. This fixed-budget setting is intended to compare practical optimization behavior and time-to-accuracy rather than asymptotic best-case fitting after extensive tuning.

Metrics. We report the final mean squared error (MSE) between the recovered image and the target, wall-clock fitting time, and the fraction of sampled spatial locations with non-positive Jacobian determinant after transformation. The Jacobian statistic is included to quantify topology violations.

Fixed-budget behavior. Under this matched optimization budget, FG-CPAB consistently achieves lower residual error, shorter fitting times, and fewer topology violations than dense SVF (Table 2). One possible explanation is the compact CPAB parameterization, which restricts optimization to the low-dimensional space spanned by $\mathbf{B}$. In contrast, dense SVF involves a substantially larger number of free parameters. These results suggest that the reduced parameterization may facilitate more efficient optimization while retaining sufficient expressiveness for the evaluated registration tasks.

Extended-budget and regularized dense SVF. To separate optimization effects from asymptotic fitting behavior, we also evaluate dense SVF under much larger iteration budgets, both with and without explicit smoothness regularization. Figure 9 shows that unregularized dense SVF typically plateaus early at a relatively high error, whereas CPAB rapidly reaches a low-error solution. We then equip dense SVF with standard smoothness regularization and optimize for 100,000 iterations (Figure 10). Regularization substantially improves the optimization stability of dense SVF, but convergence remains much slower than for CPAB.

In 2D, regularized dense SVF still does not match CPAB's reconstruction quality within the tested budget. In 3D, regularized dense SVF eventually attains a lower final MSE, but only after approximately $50{,}000$–$60{,}000$ iterations. Since CPAB is also faster per iteration, this crossover occurs only after a very large wall-clock time budget.

Interpretation. These experiments indicate that dense SVFs may offer greater asymptotic fitting flexibility in some regimes, but that this flexibility is difficult to realize in practice without strong regularization and a substantially extended optimization budget. CPAB occupies a different and practically attractive point in the trade-off: it provides sufficient diffeomorphic expressiveness for complex deformations while yielding substantially better optimization stability, faster time-to-solution, and fewer topology violations out of the box.

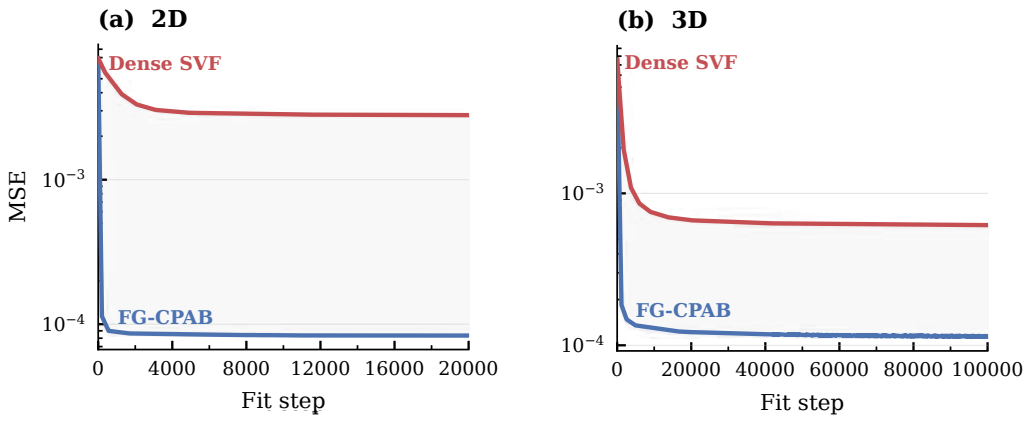


Figure 9: **Convergence without regularization.** Optimization trajectories for CPAB and dense SVF on the synthetic fitting task without additional regularization, shown for both 2D and 3D settings. In both cases, CPAB rapidly reaches a low reconstruction error within the first few hundred iterations, whereas dense SVF plateaus early at a substantially higher MSE. This highlights the optimization difficulty of the dense SVF parameterization under a matched iteration budget, despite its much larger number of degrees of freedom.

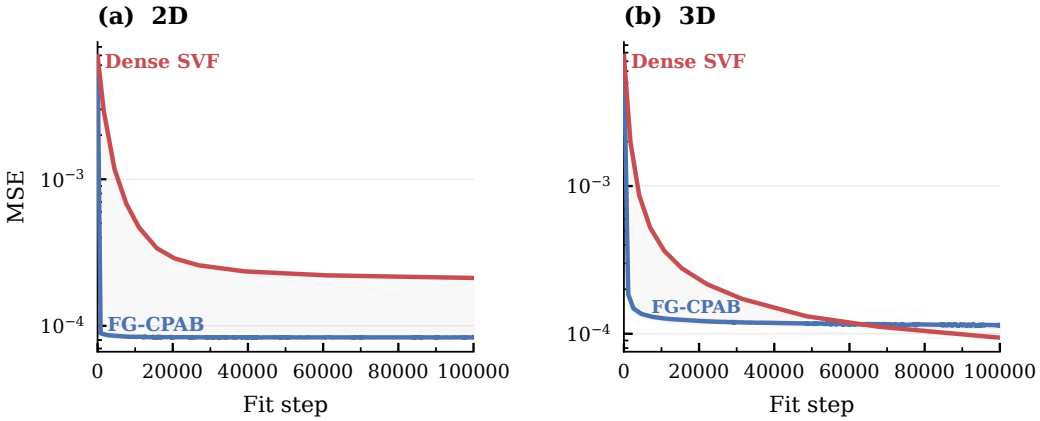


Figure 10: **Convergence with regularization.** Optimization trajectories for CPAB and dense SVF when dense SVF is equipped with standard smoothness regularization, shown for both 2D and 3D settings over an extended budget of 100,000 iterations. CPAB demonstrates extremely rapid initial convergence in both settings, stabilizing within the first few hundred iterations. In 2D (a), CPAB achieves and maintains a consistently lower fitting error. In 3D (b), regularized dense SVF converges much more slowly, but eventually surpasses CPAB to achieve a lower final mean squared error after approximately 50,000 to 60,000 iterations.

D Segmented Checkpointing

Segmented checkpointing is the memory mechanism that makes the $\mathcal{O}(TN)$ adjoint practical on GPU hardware. The reverse sweep requires access to the forward states $\boldsymbol{p}^{(0)}, \ldots, \boldsymbol{p}^{(T)}$, because both the adjoint update and the moment contribution at timestep t depend on the state and active cell at that timestep. In our implementation these trajectories are thread-private. Storing the full trajectory for every point would therefore require $\mathcal{O}(T)$ per-thread storage, increase register pressure, and, once registers are exhausted, spill to local memory, reducing occupancy and increasing memory traffic.

Segmented replay strategy. Let T denote the number of integration steps and S the segment length. We partition the trajectory into

$$K = \left\lceil \frac{T}{S} \right\rceil$$

segments, where segment k covers the timestep range $[kS, \min((k+1)S, T))$. The method then proceeds in two phases.

In the first phase, we integrate forward once and store only the $K + 1$ states at segment boundaries. In the second phase, we process segments in reverse order. For each segment, we restart from its boundary checkpoint, replay the forward dynamics within that segment, and immediately sweep backward through the reconstructed states to propagate the adjoint and accumulate the moments. Once the segment has been processed, its temporary states are discarded.

Memory-compute trade-off. The resulting per-thread storage cost is

$$\mathcal{O}(T/S + S),$$

consisting of $\mathcal{O}(T/S)$ checkpoint states and $\mathcal{O}(S)$ temporary states for the active segment. This replaces the $\mathcal{O}(T)$ storage required by the full trajectory. The trade-off is a constant-factor recomputation overhead: each state is visited once in the checkpointing pass and once again during replay. Although this does not change the asymptotic complexity, the constant matters in practice because reduced register pressure and lower local-memory traffic can substantially improve GPU occupancy.

Relation to the adjoint factorization. Checkpointing changes only how intermediate forward states are materialized; it does not alter the reverse sweep itself. Within each segment, the adjoint recursion and moment accumulation are exactly those of Eq. 8 and Eq. 11. Consequently, segmentation changes memory usage and runtime, but not the factorized gradient in Eq. 12.

Choice of segment length. The segment length S interpolates between two extremes. When $S = 1$, every timestep is checkpointed: checkpoint count is maximal, but replay buffers are minimal. When $S \geq T$, the method reduces to a single-segment replay ("no checkpointing" in Table 7): checkpoint overhead is minimal, but per-thread replay buffers are largest. The runtime trade-off is therefore typically non-monotonic. Small S increases checkpoint bookkeeping, whereas large S increases temporary state, register pressure, and local-memory traffic. Best performance is usually attained at an intermediate S; unless otherwise noted, we use $S = 4$ throughout the paper. Table 7 reports this trade-off empirically.

Table 7: Checkpoint segment size sweep. Forward/backward are mean runtime in ms (lower is better). Bold indicates the fastest value in each metric.

Ckpt seg.	Segments	Fwd (ms)	Bwd (ms)
2	25	0.898	4.161
3	17	0.885	3.977
4	13	**0.882**	**3.860**
5	10	0.885	3.905
6	9	0.891	3.899
7	8	0.889	4.302
8	7	0.893	4.644
9	6	0.889	4.699
10	5	0.887	5.056
11	5	0.891	4.905
12	5	0.887	5.073
13	4	0.887	5.246
14	4	0.885	5.198
15	4	0.892	5.284
16	4	0.885	5.683
17	3	0.893	5.976
18	3	0.905	5.868
19	3	0.901	5.828
20	3	0.898	5.923
21	3	0.897	5.956
22	3	0.897	6.080
23	3	0.900	6.231
24	3	0.898	6.468
25	2	0.897	6.773
50 (no ckpt)	1	0.899	7.339

E Application-Specific Experimental Details

We provide additional experimental details for the two applications considered in the main paper: 3D deformable registration and neural operator learning. These include the datasets, model configurations, and optimization protocols.

E.1 CPAB for 3D deformable registration

Table 3 is produced using an atlas-to-patient registration protocol on the IXI dataset.

Data and preprocessing. We follow the public TransMorph IXI atlas-to-patient protocol [8]. It contains 576 preprocessed T1-weighted MRI volumes, divided into 403 training, 58 validation, and 115 test subjects. Each volume is skull-stripped, affinely aligned, segmented using FreeSurfer, normalized to $[0, 1]$, and represented at a resolution of $160 \times 192 \times 224$. Because FG-CPAB requires no offline training, we evaluate it directly on all 115 test subjects.

The fixed image is the common atlas distributed with the TransMorph IXI data, originally obtained from the CycleMorph protocol, together with its corresponding segmentation. Registration is optimized from the atlas to each patient using MRI intensities; anatomical labels are used only for evaluation. Dice is evaluated over the 30 anatomical labels used by the TransMorph protocol.

Model configuration and optimization. In all CPAB configurations, we use a $20 \times 20 \times 20$ tessellation. For the composed variants, multiple CPAB transformations are applied sequentially during each optimization step. The parameters are fitted independently for every test subject using Adam for 200 iterations.

Loss function. The objective function comprises a local normalized cross-correlation (LNCC) similarity loss (window size 9, sample stride 2, weight 1.0) and a diffusion regularization penalty (L2 gradient of the displacement field, weight 3.0).

E.2 CPAB as a low-dimensional geometry code for operator learning

We follow the reaction–diffusion benchmark of DIMON [39], where solutions on a family of annulus-like domains are transported to a reference circular annulus and a neural operator is conditioned on a low-dimensional geometry code. We keep the DIMON data generation, transport pipeline, and operator architecture unchanged, and vary only the geometry representation.

In the DIMON baseline, geometry is encoded by applying PCA to an LDDMM-based deformation representation. In our variant, we fit a CPAB diffeomorphism from the reference domain to each target domain and use the CPAB parameters directly as the geometry code. We use a 6×6 CPAB tessellation, yielding 122 parameters. For a fair comparison, we also set the DIMON baseline to 122 PCA coefficients. Hence both methods use geometry codes of identical dimension, and differences in performance reflect the quality of the underlying geometry parameterization rather than code size.

F Nodal Sparse Basis

The basis $\mathbf{B}$ introduced in § 2 can be constructed efficiently through a nodal parameterization of the CPA velocity field. Instead of working directly with cell-wise affine coefficients, we first represent the velocity field by its values at the tessellation vertices and then impose the CPAB constraints in this nodal space. This yields a sparse construction of a basis spanning the same space as the constraint-based construction.

Nodal parameterization. Let n_V denote the number of vertices in the tessellation, and let $\boldsymbol{u} \in \mathbb{R}^{Kn_V}$ be the stacked vector of nodal velocities, with one K-dimensional velocity vector per vertex. Let $\boldsymbol{a} \in \mathbb{R}^D$ denote the stacked vector of all cell-wise affine coefficients, where D is the total number of affine coefficients across all cells. Since the velocity field is affine inside each simplex, the affine coefficients on a cell are uniquely determined by the velocities at that cell's $K+1$ vertices. Therefore, the global affine parameter vector can be written as

$$\boldsymbol{a} = \mathbf{N}\boldsymbol{u}, \qquad \mathbf{N} \in \mathbb{R}^{D \times Kn_V}, \tag{14}$$

where $\mathbf{N}$ is a sparse linear map from nodal velocities to cell-wise affine parameters.

Construction of the nodal map. The matrix $\mathbf{N}$ is assembled cell by cell from the mesh geometry. For a simplex c, let

$$\widetilde{\boldsymbol{x}}_{c,0}, \ldots, \widetilde{\boldsymbol{x}}_{c,K} \in \mathbb{R}^{K+1}$$

denote its vertices in homogeneous coordinates, and let

$$\boldsymbol{v}_{c,0}, \ldots, \boldsymbol{v}_{c,K} \in \mathbb{R}^K$$

be the corresponding nodal velocities. Inside the simplex, the CPA velocity field is represented as

$$\boldsymbol{v}_c(\boldsymbol{x}) = \mathbf{A}_c \widetilde{\boldsymbol{x}}, \qquad \mathbf{A}_c \in \mathbb{R}^{K \times (K+1)}.$$

The local affine matrix $\mathbf{A}_c$ is determined by the interpolation conditions

$$\mathbf{A}_c \widetilde{\boldsymbol{x}}_{c,j} = \boldsymbol{v}_{c,j}, \qquad j = 0, \ldots, K.$$

Collecting the homogeneous vertex coordinates and nodal velocities into

$$\mathbf{X}_c = [\widetilde{\boldsymbol{x}}_{c,0} \quad \cdots \quad \widetilde{\boldsymbol{x}}_{c,K}] \in \mathbb{R}^{(K+1)\times(K+1)}, \qquad \mathbf{V}_c = [\boldsymbol{v}_{c,0} \quad \cdots \quad \boldsymbol{v}_{c,K}] \in \mathbb{R}^{K\times(K+1)},$$

gives

$$\mathbf{A}_c \mathbf{X}_c = \mathbf{V}_c.$$

Since the simplex vertices are affinely independent, $\mathbf{X}_c$ is invertible, and thus

$$\mathbf{A}_c = \mathbf{V}_c \mathbf{X}_c^{-1}.$$

This is the local interpolation rule in matrix form. To express it as a linear operator on stacked vectors, we apply the standard identity

$$\operatorname{vec}(\mathbf{UM}) = (\mathbf{M}^\top \otimes \mathbf{I}) \operatorname{vec}(\mathbf{U}),$$

which yields

$$\operatorname{vec}(\mathbf{A}_c) = (\mathbf{X}_c^{-T} \otimes \mathbf{I}_K) \operatorname{vec}(\mathbf{V}_c).$$

Hence the local nodal-to-affine operator is

$$\mathbf{N}_c = \mathbf{X}_c^{-T} \otimes \mathbf{I}_K.$$

The global sparse matrix $\mathbf{N}$ is obtained by assembling these local operators over all cells according to the mesh connectivity. Its sparsity follows from locality: each cell depends only on its own $K+1$ vertices, so each row of $\mathbf{N}$ contains nonzero entries only in the columns corresponding to those local nodal degrees of freedom.

Constraint enforcement in nodal coordinates. In the standard CPAB formulation, valid affine coefficients must satisfy the global linear constraint system

$$\mathbf{L}\boldsymbol{a} = \mathbf{0},$$

which enforces continuity across shared cell boundaries together with any imposed boundary conditions. Substituting the nodal parameterization from Eq. 14 gives

$$\mathbf{LN}\boldsymbol{u} = \mathbf{0}. \tag{15}$$

Defining

$$\mathbf{C} \coloneqq \mathbf{LN}, \tag{16}$$

the admissible nodal configurations are exactly those in the null space of $\mathbf{C}$. Let

$$\mathbf{Z} \in \mathbb{R}^{Kn_V \times d} \tag{17}$$

be a basis for $\mathrm{null}(\mathbf{C})$, so that every valid nodal vector can be written as

$$\boldsymbol{u} = \mathbf{Z}\boldsymbol{\theta}, \qquad \boldsymbol{\theta} \in \mathbb{R}^d. \tag{18}$$

Resulting sparse CPA basis. Combining Eq. 14 and Eq. 18 yields

$$\boldsymbol{a} = \mathbf{NZ}\boldsymbol{\theta}. \tag{19}$$

Hence the CPA basis can be written as

$$\mathbf{B} = \mathbf{NZ}. \tag{20}$$

This shows that the low-dimensional parameter vector $\boldsymbol{\theta}$ can be interpreted as coordinates in a constrained nodal basis, while $\mathbf{B}$ maps these coordinates to the cell-wise affine representation used throughout the paper.

The main advantage of this construction is sparsity: $\mathbf{N}$ is local by design, and the resulting basis inherits the mesh structure of the tessellation. Crucially, on a standard simplex mesh, the nodal formulation automatically satisfies continuity across shared cell boundaries. Consequently, the vast majority of constraints in $\mathbf{C} = \mathbf{LN}$ evaluate exactly to zero. The only remaining active constraints are global boundary conditions (e.g., zero velocity at the domain boundaries). Finding the null space $\mathbf{Z}$ of this drastically reduced system geometrically amounts to identifying the unconstrained interior nodes, yielding a natural sparse basis matrix. In practice, simply thresholding numerical noise from standard dense linear algebra routines fully recovers this exact physical sparsity. This provides an efficient way to assemble $\mathbf{B}$ as a sparse matrix while exactly preserving the same globally continuous CPA velocity fields as the direct algebraic construction.

Remark. The null-space basis $\mathbf{Z}$ can often be constructed directly from the nodal constraints, avoiding an SVD. For more general constraints, one can remove the redundant rows of $\mathbf{C}$ before computing its null space numerically, substantially reducing the cost of the SVD. Although sparsity is not guaranteed in general, the resulting basis can remain sparse in practice for certain constraint structures, enabling efficient storage and computation.

G Proof of the Telescoping Property of the Discrete Adjoint

We prove the claim made in § 3.2 that summing the sensitivity contraction identity Eq. 9 over all timesteps produces a telescoping series in which all propagated terms cancel exactly, leaving only the accumulated direct contributions.

For notational brevity, fix a single point $\boldsymbol{p}_n$ (the argument is identical and independent for every point) and a single parameter component θ_i. Define the scalar quantities

$$s^{(t)} \coloneqq \boldsymbol{\lambda}^{(t)^\top} \frac{\partial \boldsymbol{p}^{(t)}}{\partial \theta_i}, \qquad d^{(t)} \coloneqq \boldsymbol{\lambda}^{(t+1)^\top} \left(h\, \mathbf{B}_{i,c}\, \widetilde{\boldsymbol{p}}^{(t)}_{\text{mid}} + \tfrac{h^2}{2}\, \mathbf{A}_{c,\text{lin}}\, \mathbf{B}_{i,c}\, \widetilde{\boldsymbol{p}}^{(t)} \right),$$

where all quantities on the right-hand side of $d^{(t)}$ are evaluated at the appropriate cell c that the point occupies at timestep t.

The identity Eq. 9 then reads

$$s^{(t+1)} = d^{(t)} + s^{(t)} \qquad \text{for each timestep } t = 0, 1, \ldots, T-1. \tag{21}$$

Summing Eq. 21 from $t = 0$ to $t = T-1$ yields

$$\sum_{t=0}^{T-1} s^{(t+1)} = \sum_{t=0}^{T-1} d^{(t)} + \sum_{t=0}^{T-1} s^{(t)}.$$

The left-hand side expands to

$$s^{(1)} + s^{(2)} + \cdots + s^{(T)},$$

while the second sum on the right-hand side expands to

$$s^{(0)} + s^{(1)} + \cdots + s^{(T-1)}.$$

Rearranging immediately gives the telescoping cancellation

$$s^{(T)} - s^{(0)} = \sum_{t=0}^{T-1} d^{(t)}. \tag{22}$$

Two boundary conditions complete the proof:

1. The initial position $\boldsymbol{p}^{(0)} = \boldsymbol{p}_0$ is the fixed input point and therefore independent of the parameters $\boldsymbol{\theta}$. Consequently,

$$\frac{\partial \boldsymbol{p}^{(0)}}{\partial \theta_i} = \mathbf{0} \quad \Longrightarrow \quad s^{(0)} = 0.$$

2. At the final timestep the adjoint is defined by the loss gradient,

$$\boldsymbol{\lambda}^{(T)} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{p}^{(T)}},$$

so

$$s^{(T)} = \boldsymbol{\lambda}^{(T)^\top} \frac{\partial \boldsymbol{p}^{(T)}}{\partial \theta_i} = \frac{\partial \mathcal{L}}{\partial \theta_i}.$$

Substituting these two facts into Eq. 22 produces

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{t=0}^{T-1} d^{(t)}.$$

The right-hand side is precisely the sum of all direct (local) contributions of θ_i across the trajectory. These contributions are exactly the terms that, after applying the Frobenius inner-product identity and summing over all points n that visit cell c at timestep t, assemble into the moment matrix $\mathbf{M}_c$ of Eq. 11. The final gradient is then recovered by the cheap projection Eq. 12.

Because the derivation relies only on the chain rule and the definition of the exact discrete adjoint of the midpoint integrator, the cancellation is algebraic and holds to machine precision (no discretization error is introduced).

H Why Translation Does Not Enter the State Jacobian

For an arbitrary spatial dimension K, consider a fixed cell c and decompose the affine velocity matrix as

$$\mathbf{A}_c = [\mathbf{L}_c \;\; \boldsymbol{b}_c], \qquad \mathbf{L}_c \in \mathbb{R}^{K\times K}, \quad \boldsymbol{b}_c \in \mathbb{R}^K.$$

For homogeneous coordinates $\widetilde{\boldsymbol{p}} = (\boldsymbol{p}^\top, 1)^\top$, the velocity field is

$$\mathbf{A}_c \widetilde{\boldsymbol{p}} = \mathbf{L}_c \boldsymbol{p} + \boldsymbol{b}_c.$$

Starting from the midpoint integrator in Eq. 5,

$$\boldsymbol{p}_{\text{mid}} = \Big(\mathbf{I} + \frac{h}{2}\mathbf{L}_c\Big)\boldsymbol{p}^{(t)} + \frac{h}{2}\boldsymbol{b}_c,$$

and therefore

$$\boldsymbol{p}^{(t+1)} = \Big(\mathbf{I} + h\mathbf{L}_c + \frac{h^2}{2}\mathbf{L}_c^2\Big)\boldsymbol{p}^{(t)} + \Big(h\mathbf{I} + \frac{h^2}{2}\mathbf{L}_c\Big)\boldsymbol{b}_c.$$

Hence, the state Jacobian of one midpoint step is

$$\frac{\partial \boldsymbol{p}^{(t+1)}}{\partial \boldsymbol{p}^{(t)}} = \mathbf{I} + h\mathbf{L}_c + \frac{h^2}{2}\mathbf{L}_c^2,$$

where $\mathbf{I} \in \mathbb{R}^{K\times K}$. Thus, the Jacobian depends only on the linear block $\mathbf{L}_c = \mathbf{A}_{c,\text{lin}}$, regardless of the spatial dimension. The translation $\boldsymbol{b}_c$ contributes only additive terms independent of $\boldsymbol{p}^{(t)}$, whose derivatives *w.r.t.* $\boldsymbol{p}^{(t)}$ vanish.

I Integrator mismatch

Source of the mismatch and relation to prior CPAB implementations. As discussed in § 3, our forward pass uses the matrix-exponential update, whereas the gradient computation uses the discrete adjoint of the explicit midpoint scheme in Eq. 5. Consequently, the resulting gradient is the exact derivative of the midpoint-discretized dynamics, but only an approximation to the derivative of the matrix-exponential forward computation.

This numerical pairing is not introduced by FG-CPAB. Reference CPAB implementations similarly combine a matrix-exponential forward transformation with a midpoint-discretized sensitivity computation [14, 10]. We retain the same forward and gradient discretizations to ensure a fair comparison with prior CPAB methods. Thus, the comparison evaluates our factorized organization of the gradient computation without introducing a different integration scheme as an additional source of performance or accuracy differences.

Effect of inconsistent cell assignments. We first consider the discrepancy caused by using different cells for the same state. Let $\widetilde{\mathbf{A}}_1$ and $\widetilde{\mathbf{A}}_2$ denote the augmented matrices representing the velocity fields in two neighboring cells, and consider

$$\widetilde{\boldsymbol{p}}_1^{(t+1)} = \exp(h\widetilde{\mathbf{A}}_1)\widetilde{\boldsymbol{p}}^{(t)}, \qquad \widetilde{\boldsymbol{p}}_2^{(t+1)} = \exp(h\widetilde{\mathbf{A}}_2)\widetilde{\boldsymbol{p}}^{(t)}. \tag{23}$$

A Taylor expansion gives

$$\widetilde{\boldsymbol{p}}_1^{(t+1)} - \widetilde{\boldsymbol{p}}_2^{(t+1)} = h\big(\widetilde{\mathbf{A}}_1 - \widetilde{\mathbf{A}}_2\big)\widetilde{\boldsymbol{p}}^{(t)} + \mathcal{O}(h^2). \tag{24}$$

Therefore, for fixed cell matrices and a bounded state (i.e., $\|\widetilde{p}^{(t)}\|$ remains bounded as $h \to 0$), an inconsistent cell choice can introduce an $O(h)$ local discrepancy.

Effect of different integrators within a fixed cell. We next compare the matrix-exponential and midpoint updates when both use the same cell matrix $\widetilde{\mathbf{A}}$. The midpoint update can be written as

$$\widetilde{\boldsymbol{p}}_{\text{mid}}^{(t+1)} = \left(\mathbf{I} + h\widetilde{\mathbf{A}} + \tfrac{h^2}{2}\widetilde{\mathbf{A}}^2\right)\widetilde{\boldsymbol{p}}^{(t)}. \tag{25}$$

Meanwhile,

$$\exp(h\widetilde{\mathbf{A}}) = \mathbf{I} + h\widetilde{\mathbf{A}} + \tfrac{h^2}{2}\widetilde{\mathbf{A}}^2 + \mathcal{O}\left(h^3\|\widetilde{\mathbf{A}}\|^3\right). \tag{26}$$

Hence,

$$\widetilde{\boldsymbol{p}}_{\text{exp}}^{(t+1)} - \widetilde{\boldsymbol{p}}_{\text{mid}}^{(t+1)} = \mathcal{O}\left(h^3\|\widetilde{\mathbf{A}}\|^3\right). \tag{27}$$

For a fixed cell matrix over a finite integration interval, the $\mathcal{O}(h^3)$ local error yields an $\mathcal{O}(h^2)$ global error. Thus, the local integrator mismatch is of higher order than the potential $\mathcal{O}(h)$ discrepancy caused by inconsistent cell assignments.

Empirical gradient validation. The remaining question is whether this approximation is sufficiently accurate for gradient-based optimization. We therefore compared the gradient produced by our backward pass with a central finite-difference estimate of the gradient of the matrix-exponential forward computation. We measured cosine similarity and relative error across multiple tessellation configurations.

For the standard setting used throughout the paper, with 50 solver steps, the cosine similarity was above 0.998, while the relative error was below 0.05. The approximate gradient is therefore closely aligned with the finite-difference reference gradient in the regime used in our experiments.

Conclusion. The forward and backward discretizations are not algebraically identical, and our gradient should accordingly be interpreted as an approximation to the derivative of the matrix-exponential forward computation. However, the mismatch is higher order within a fixed cell, follows the numerical treatment used by prior CPAB implementations, and produces gradients that are strongly aligned with finite-difference estimates. Empirically, it does not hinder stable first-order optimization in any of our experiments.

J Why Gradient-Based Optimization Is Needed: Comparison with MCMC

The original CPAB work considered MCMC-based optimization (Metropolis algorithm with a Gaussian proposal) for standalone registration problems, while its journal extension subsequently introduced gradient-based optimization [13, 14]. Although MCMC can be effective in low-dimensional settings, it is not well suited to the fine-tessellation applications targeted in this work for two main reasons.

End-to-end differentiability. CPAB was originally introduced as a standalone transformation model outside modern automatic-differentiation frameworks. In many recent applications, however, a CPAB transformation is used as a differentiable component within a larger model. Training such a model requires propagating loss gradients through the CPAB parameters to preceding network components. MCMC performs inference over the CPAB parameters but does not provide the vector–Jacobian products required for end-to-end backpropagation.

Scaling with the parameter dimension. Standard MCMC methods, such as Metropolis sampling with a Gaussian proposal, generally become increasingly difficult to apply as the parameter dimension grows. Early CPAB experiments using MCMC involved relatively small parameter spaces; for example, the landmark experiment in the original CPAB work used $d = 38$ parameters. In contrast, our synthetic benchmarks use 842 parameters in 2D and $8{,}232$ parameters in 3D, while the IXI configuration uses $20{,}577$ parameters per CPAB transformation. This fine-tessellation regime strongly favors gradient-based optimization.

Empirical comparison. To verify this empirically, we benchmarked the official CPAB MCMC implementation using the same IXI atlas-to-patient registration setup as FG-CPAB. We allocated MCMC a wall-clock optimization budget $100\times$ larger than that used by FG-CPAB. Nevertheless, MCMC achieved a Dice score of only 0.453, compared with 0.760 for gradient-based FG-CPAB.

These results do not imply that MCMC is unsuitable for every CPAB application. Rather, they show that it is uncompetitive for the high-dimensional, fine-tessellation problems considered here and cannot replace gradient computation when CPAB is used within an end-to-end differentiable model.

K Assets, Licenses, and Terms of Use

This appendix is provided to document the external assets used in this work, including datasets, codebases, and benchmark implementations.

K.1 Public datasets and benchmark data

Table 8: Datasets and benchmark data used in the paper.

Asset	How used in this paper	Citation	Version	License
IXI brain MRI dataset	3D deformable registration benchmark	[21]	link 26.4.26	Creative Commons CC BY-SA 3.0 license
DIMON reaction–diffusion benchmark data / pipeline	Operator-learning experiment in § 4.5	[39]	Commit - a22ed33	MIT License

K.2 External code, software, and baseline implementations

Table 9: External codebases, libraries, and baseline implementations used by the project.

Asset	How used in this paper	Citation	Version / commit	License
`libcpab`	Primary CPAB baseline and reference implementation	[10]	Commit - 22df6cd	MIT license
`DIFW`	1D performance baseline in Appendix A	[28]	Commit - f970dd3	MIT license
VoxelMorph	Dense SVF baseline / runtime comparison	[3]	Commit - 6fea3e8	Apache-2.0 license
`torch-lddmm`	LDDMM runtime baseline	[4]	Commit - 86d5c43	Apache-2.0 license

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the claims made, including the contributions made in the paper and important assumptions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are mentioned in the discussion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Proofs either appear in the main paper or in the supplemental material.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The main paper and appendices describe the algorithm, solver settings, tessellations, losses, optimizers, benchmarks, and evaluation protocols needed to reproduce the reported experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: Code will be released upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Experimental settings are specified in the main paper and appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Results are reported with variability measures (Standard deviation).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports hardware and runtime details in the releven places.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: To the best of our knowledge, the research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The work is foundational and not tied to a deployed system, dataset, or model with clear privacy, fairness, security, or misuse risks; therefore, no direct societal impacts are identified.

    Guidelines:

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Assets, Licenses, and Terms of Use section in the appendix.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The submission does not release a new dataset, model, or code artifact alongside the paper; the implementation is only stated to be released upon acceptance.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject experiments or participant studies requiring IRB review.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not part of the core method, experiments, or scientific claims of this work.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.