PointProgrammerNet: An additive, coordinate-addressed state for point-cloud classification and segmentation

Dian Yu
University of New South Wales, Sydney, NSW 2052, Australia
Corresponding author: dian.yu2@student.unsw.edu.au
ORCID: https://orcid.org/0009-0002-0107-1014

Abstract

A point cloud is rarely delivered all at once: two devices see different parts of a scene, one of them later turns out to be mis-calibrated, and a region is chosen for labelling after the observations are over. This paper asks what a network should keep instead of the points, and answers it for part segmentation and object classification. PointProgrammerNet stores a sum: each point is mapped to a distribution over learned spatial addresses and contributes a second-order value as an outer product, and three fixed 128×170 matrices hold the accumulated products. Nothing that couples one point to another enters the write, so normalization across accumulated observations and nonlinear interpretation of the accumulated statistics are deferred to the read, and what is stored admits arithmetic that a pooled or normalized descriptor does not. Two states add exactly, so several devices can encode locally, exchange a fixed-size matrix, and recover the centralized result. A state can be subtracted, so an unreliable source is withdrawn without re-encoding the rest, and scaling by a retention factor weights evidence by age. Because the sum carries spatial addresses instead of collapsing into one vector, the state also defines a segmentation field over the whole coordinate space: read at labelled coordinates it never wrote, it gives up 0.05 instance mIoU against the coordinates it did write. The model reaches 82.58% instance mIoU on ShapeNetPart with 40.3% fewer parameters than a lightweight PointNet++ control and 86.59% on ModelNet40 from the state alone, and merge, removal and attenuation match direct reconstruction to within 10^{-6}. These results show that useful recognition accuracy can be retained in a fixed-size state that supports exact merging, removal and attenuation after the points are released.

Keywords: point-cloud processing; additive state; state merging and removal; coordinate-conditioned segmentation; multi-sensor fusion; part segmentation
Running title: PointProgrammerNet additive state

1 Introduction

A point cloud is rarely delivered all at once. Observations arrive as frames, a scene is covered in several passes, and several devices may each see a different part of it. Point-cloud encoders are nevertheless written for a cloud that is present in full: features are computed from the current point set, and when that set changes they are computed again. The point list is therefore the object a system has to keep, and what the system can do afterwards is defined in terms of that list.

This paper asks what a network should keep instead, and answers it for the two standard recognition tasks, part segmentation on ShapeNetPart and object classification on ModelNet40. Suppose that once a point has been encoded it is released, and that from then on the system

holds a fixed-size state, together with a coordinate whenever a label is requested at a position. What such a state is worth depends on which operations survive the loss of the points, and those operations are worth naming concretely. Two devices each observe part of a scene, and the encodings of the parts have to combine into the encoding of the whole, or the devices are back to shipping raw points. One of them is later judged unreliable, or its pose is afterwards corrected, and its contribution has to come out of the representation without re-encoding everything else that went in. A scene is observed again later, and evidence from the earlier pass should count for less. A region is designated after the observations are over and labels are wanted on a grid across it, at positions that no point ever occupied.

None of this is hard on its own. A sum combines and can be subtracted; a Fisher or VLAD accumulator has a fixed size; a latent array can be read with a query. Having all of it at once is harder. The usual ways of turning a set into a summary make what one point contributes depend on which other points are present: a softmax normalizes over the observed set, a neighborhood grouping asks which points are nearby, a descriptor is divided by the count it happened to see, and a maximum decides which point wins. Once that coupling enters the stored summary, exact linear editing of an individual contribution is no longer available. PointNet [1], PointNet++ [2] and Fisher- or VLAD-style encoders [3, 4, 5] each provide useful fixed summaries under their published formulations, but do not retain the complete set of operations considered here.

PointProgrammerNet therefore imposes one rule on the architecture: nothing that couples one point to another may enter the stored representation. A point is mapped to a distribution over 128 learned spatial addresses and contributes a second-order value — mass, position, learned content, and their cross and square terms — as an outer product, and what is stored is the sum of those outer products. Normalization across the accumulated observations, centering on the query, and nonlinear interpretation of the accumulated statistics are applied when the state is read.

What the rule buys is that the stored object is a sum, and sums have arithmetic. States of disjoint observations add, exactly and in any order, so the two devices encode independently and a server recovers the centralized result. A state can be subtracted, so the unreliable source is taken out by one subtraction, and a corrected contribution is installed by subtracting the old one and adding the new. Multiplying by a retention factor ρ before the next observation is absorbed makes earlier evidence count for less, which is what the repeated pass needs. The design uses soft spatial assignment, moment statistics and outer-product writes to implement this constraint while preserving the resulting state arithmetic.

Spreading the sum over spatial addresses, rather than collapsing it into one vector, buys something further. The address of a query is a smooth function of its coordinate and is defined everywhere, so a fixed state defines a segmentation field on the whole of $\mathbb{R}^3$, continuous in the query position. A label can therefore be produced on a regular grid, at a resolution other than the one that was recorded, or inside a region chosen after the fact — at coordinates that carry no written point. Once stored point features are released, a readout tied to those features has no anchor at such a position; the fixed state retains one. For classification nothing beyond the three matrices is retained, and for segmentation only the coordinates one actually intends to label.

The same linearity settles where order may enter. Points of one observation are exchangeable and their writes combine by addition alone, so an observation can be divided across chunks or devices however is convenient. Order enters only between observations, through the retention factor of Equation (18), and because the state is a sum of independent writes, multiplying by ρ is not an approximation of forgetting but exactly a reweighting of everything already written by its age. Space composes, and time is the only place where the state carries history.

On ShapeNetPart the segmentation model reaches 82.58% instance mIoU with 40.3% fewer parameters than a lightweight PointNet++ control, and on ModelNet40 the state-only classifier

reaches 86.59%. A controlled comparison shows that the three parallel states carry structure a single wider state does not: at an identical parameter count and identical initialization, removing the two auxiliary reads costs more than a point of instance mIoU. Merge, removal and attenuation agree with direct reconstruction to within 10^{-6}, and a frozen classifier already produces a usable prediction from part of a cloud, reaching 86.02% from half of one and agreeing with the completed prediction on every test object. These results quantify the recognition accuracy retained while the representation remains fixed in size, additive and revisable.

Our contributions are:

- a write-time constraint — no operation that couples one point to another may enter the stored state — realized as a separation between an unnormalized write and a normalized, nonlinear, coordinate-conditioned read;
- the state algebra the constraint leaves available: chunks and devices merge by addition, a contribution whose own state was kept is removed by subtraction, and a partly trusted observation enters with a scaled write;
- a coordinate-addressed read under which a fixed state defines a continuous segmentation field on the whole coordinate space, together with the measurement that labels read at coordinates carrying no written point stay within 0.05 instance mIoU of labels read at the written coordinates;
- benchmark results on ShapeNetPart and ModelNet40, a controlled study isolating the parallel-state design, and numerical verification of merge, removal and attenuation against direct reconstruction.

2 Related Work

Work in *IET Computer Vision* has improved complete-cloud recognition through graph-based global–local feature fusion [6] and efficient semantic segmentation through sparse point–voxel aggregation [7], and a recent survey places point, voxel, graph, Transformer, and multimodal representations within the broader development of 3-D perception [8]. These studies improve how an available cloud is processed. The question here is a different one: what should be left behind when the cloud is not kept, so that evidence can still be merged, a contribution withdrawn, and a position queried? There are several established ways to reduce an unordered point set to a fixed summary, and they differ mainly in what they give up at the moment a point is absorbed. That choice decides which operations survive, so we group them by it.

2.1 Symmetric pooling and metric neighborhoods

PointNet applies pointwise transforms followed by maximum pooling [1], PointNet++ adds hierarchical metric neighborhoods [2], and Deep Sets characterizes permutation-invariant functions built from sums [9]. All three are invariant to point order, and the sum used by Deep Sets is additive in the sense of Equation (14). They differ from the representation studied here in what survives inside the stored summary. A channelwise maximum can be merged by another maximum, but it retains only the winning response, so an individual contribution cannot generally be withdrawn or attenuated. A metric neighborhood is defined relative to the points currently held, so inserting or deleting a point changes the membership of other neighborhoods and the grouping has to be rebuilt. A Deep Sets sum is linearly revisable but forms one global vector without spatial addresses. PointProgrammerNet instead distributes the sum over 128 learned addresses, retaining linear state operations together with a spatially addressed read.

2.2 Fisher-vector and center-based point-cloud descriptors

Fisher vectors encode a sample through gradients of a generative model, providing fixed-size statistics with soft assignment. 3DmFV adapts this principle to point clouds with a structured Gaussian grid and aggregates first- and second-order Fisher components by sum, maximum, and minimum before grid processing [3]. PointProgrammerNet uses related spatial statistics for a different stored object. The published 3DmFV descriptor is built from a complete sample and includes extrema and normalization that are not additive, whereas the state here retains an unnormalized sum that remains mergeable, revisable and coordinate-addressed after the points have gone.

VLAD assigns local descriptors to learned centers and accumulates residuals [4]. NetVLAD makes that assignment soft and differentiable [10], and PointNetVLAD applies it to point-cloud place recognition [5]. Their raw accumulators are sums, but the published descriptors apply blockwise and global normalization for one-shot retrieval. PointProgrammerNet instead keeps the unnormalized pointwise writes as the stored object and moves density normalization to read time. It also accumulates coordinate–content and second-order terms under continuous coordinate addresses, so the retained matrices can be evaluated at a requested position. Pairing an additive write with a coordinate-addressed read is what gives the state its combined algebraic and spatial properties.

2.3 Reading a fixed representation with a query

Set Transformer summarizes a set using inducing points and attention [11], and Perceiver IO maintains a latent array whose size is independent of the input and decodes dense per-element output by attending to it with a query array [12]. Both read a fixed latent with queries; the distinction here is on the encoding side. Softmax cross-attention normalizes over the observed elements, so the latent of two disjoint chunks is not the sum of their latents. Our write applies no normalizer over the observed set, leaving addition and subtraction available on the state. The query also differs in kind: theirs is a learned position embedding, ours a coordinate in the same space as the data.

Implicit neural representations read a coordinate directly. Occupancy Networks and DeepSDF condition a decoder on a coordinate and a latent vector to obtain a continuous field [13, 14], and Convolutional Occupancy Networks attach features to a grid or plane so the field is resolved from the feature nearest the query rather than from one global code [15]. The read studied here is of the second kind, resolved from spatially addressed entries. What differs is the write: these encoders build their features with convolutions or pooling over the observed set, so two partial representations cannot be added, whereas the state here is a sum of independent per-point terms and carries second-order statistics at each address.

2.4 Bounded recurrent state

Kernelized attention can be evaluated recurrently with a fixed-size state [16], and fast-weight programmers express such updates as additive outer products or delta corrections [17]. These are the mechanisms our write uses. Recent point-cloud work also adopts a bounded state through structured state space models; PointMamba serializes points with space-filling curves before the recurrence [18]. Serialization makes the hidden state a function of the traversal order, so it is not permutation invariant and two partial states cannot be added. PointProgrammerNet instead keeps order only between observations, where a retention factor weights evidence by age, while the write within an observation remains a sum.

Table 1: What families of set summaries retain. "Order-free write" means the summary does not depend on the order in which points are absorbed; "linear-additive state" means the summary of a union equals the *sum* of the summaries; "revisable state" means a contribution already written can be subtracted or attenuated; "spatially addressed read" means the stored representation resolves a query through entries indexed by position.

Family	Order-free write	Linear-additive	Revisable state	Spatially addressed read
PointNet, maximum pooling	yes	no	no	no
PointNet++, metric neighborhoods	yes	no	no	no
Deep Sets, global sum	yes	yes	yes	no
Fisher vector, 3DmFV	yes	no[a]	no[a]	no
VLAD, NetVLAD, PointNetVLAD	yes	no[a]	no[a]	no
Set Transformer, inducing points	yes	no[b]	no	no
Perceiver IO, output queries	yes	no[b]	no	no
Linear attention, fast weights	yes[c]	yes	yes	no
State space models on serialized points	no	no	no	no
PointProgrammerNet (this work)	yes	yes	yes	yes

[a]The raw summed components are additive, but the published descriptors apply normalization and, for 3DmFV, extrema. [b]Softmax over the observed elements couples them, so the summary of a union is not determined by the summaries of its parts. [c]The outer-product update is itself order free, although these models are applied to ordered sequences.

2.5 Accumulated volumetric maps

Occupancy grids [19] and truncated signed distance fields [20, 21] also accumulate observations into a fixed structure that can be fused across sensors, attenuated over time, and interpolated at a requested coordinate. They differ in what that structure is. Their addresses are a discretization fixed before any data arrives and each observation is assigned to one cell, so the field is piecewise constant in the query and its memory follows the volume and resolution chosen. What they store is a hand-specified geometric quantity, an occupancy log-odds or a signed distance, and a semantic label needs a separate model on top. Here the addresses are learned, continuous and soft, every point writes to all of them, the stored quantity is the moments of a learned per-point encoding, and the readout that consumes it is trained with it.

2.6 Design constraint and distinction from prior representations

Each formulation above trades one property for another: a maximum gives up revision, a metric neighborhood gives up locality of edits, a normalized descriptor gives up addition, a softmax encoder gives up decomposability, and a serialized recurrence gives up order invariance. The design studied here therefore imposes one constraint at write time: nothing that couples one point's write to another point may enter the stored state. That excludes normalization over the observed set, comparison between points, selection, and ordering. The operations an aggregation method usually performs while absorbing a point — density normalization, centering on a reference position, and nonlinear interpretation — are deferred to the read, where they no longer touch the algebra of the stored matrices. Table 1 sets the resulting properties beside those of the families above.

What the constraint provides and what it costs are both measured: Section 4.2 isolates the parallel-state structure, and Tables 3 and 4 place the resulting accuracy against hierarchical baselines.

We evaluate classification on ModelNet40 [22] and part segmentation on ShapeNetPart [23], which is derived from ShapeNet [24].

3 Materials and Methods

Let $X = \{\boldsymbol{x}_i\}_{i=1}^{N}$, with normalized coordinates $\boldsymbol{x}_i \in \mathbb{R}^3$. PointProgrammerNet replaces the growing point history with three matrices $\mathbf{S}_s \in \mathbb{R}^{128\times 170}$, where $s \in \{f, m, b\}$ denotes branches initialized at radii 0.08, 0.20, and 0.60. The radii, spatial centers, point encoders, and read networks are learned independently. The state is created for each object or stream and is separate from the trained model parameters.

All three branches have the same state dimensions and use the same equations. Their different initial radii only provide distinct starting conditions; the trained network decides what each branch represents. A branch first maps a coordinate to learned content with a two-layer point network,

$$\boldsymbol{g}_s(\boldsymbol{x}) = \mathrm{ReLU}(\mathrm{BN}(W_{s2}\,\mathrm{ReLU}(\mathrm{BN}(W_{s1}\boldsymbol{x} + \boldsymbol{b}_{s1})) + \boldsymbol{b}_{s2})) \in \mathbb{R}^{32}. \tag{1}$$

The matrices W_{s1}, W_{s2} and biases are learned separately in each branch, and both hidden layers have width 32. Consequently, equal state sizes do not imply repeated features. This nonlinear map is evaluated independently for each point. It therefore enriches what a point writes without coupling that write to any other point; additivity is preserved because only the subsequent outer products are accumulated in the persistent state. Figure 1 summarizes the complete write–state–read architecture for segmentation and classification.

3.1 Additive state construction

Each of the 128 state rows has a learned center $\boldsymbol{c}_{sj} \in \mathbb{R}^3$. These centers are not sampled from an input cloud. Before training, a Sobol sequence simply places 128 initial anchors across $[-0.9, 0.9]^3$ so that the rows begin with broad spatial coverage. Every branch receives its own trainable copy, and training may move the anchors through the bounded parameterization $\boldsymbol{c}_{sj} = 0.95\tanh(\boldsymbol{\theta}_{sj})$. For one observed coordinate $\boldsymbol{x}$, the branch measures its distance to all 128 anchors and converts those distances into 128 continuous write fractions. The positive radius $r_s = \exp(-\lambda_s)$ controls how concentrated these fractions are. The normalized address is

$$k_{sj}(\boldsymbol{x}) = \frac{\exp[-\|\boldsymbol{x} - \boldsymbol{c}_{sj}\|^2/(2r_s^2)]}{\sum_{u=1}^{128}\exp[-\|\boldsymbol{x} - \boldsymbol{c}_{su}\|^2/(2r_s^2)]}. \tag{2}$$

The denominator fixes every point's total write mass at one. This prevents dense chunks from changing the scale of an individual write and lets the radius control concentration rather than total magnitude. In practical terms, the centers act like 128 soft spatial bins. A point near several centers gives those rows large fractions and all other rows small fractions; it is never assigned to only one bin. The address determines where the point contributes, while the following 170-dimensional value determines what statistics it contributes:

$$\boldsymbol{v}_s(\boldsymbol{x}) = [1, \boldsymbol{x}, \boldsymbol{g}_s, \mathrm{vec}(\boldsymbol{x}\boldsymbol{g}_s^\top), x^2, y^2, z^2, xy, xz, yz, \boldsymbol{g}_s \odot \boldsymbol{g}_s]. \tag{3}$$

The blocks have dimensions $1+3+32+96+6+32 = 170$ and retain mass, first moments, learned content, coordinate–content products, and second moments. The leading one later supplies the denominator for local averages. The six quadratic coordinate terms and 32 squared feature terms retain spread that cannot be recovered from a mean. Equivalently, the value vector collects every moment of $(\boldsymbol{x}, \boldsymbol{g}_s)$ up to second order, with the feature–feature block restricted to its diagonal so that its width stays linear in the content dimension. Table 2 lists every block and its dimensionality. A point changes state row j by

$$[\Delta\mathbf{S}_s(\boldsymbol{x})]_{j,:} = k_{sj}(\boldsymbol{x})\boldsymbol{v}_s(\boldsymbol{x})^\top. \tag{4}$$

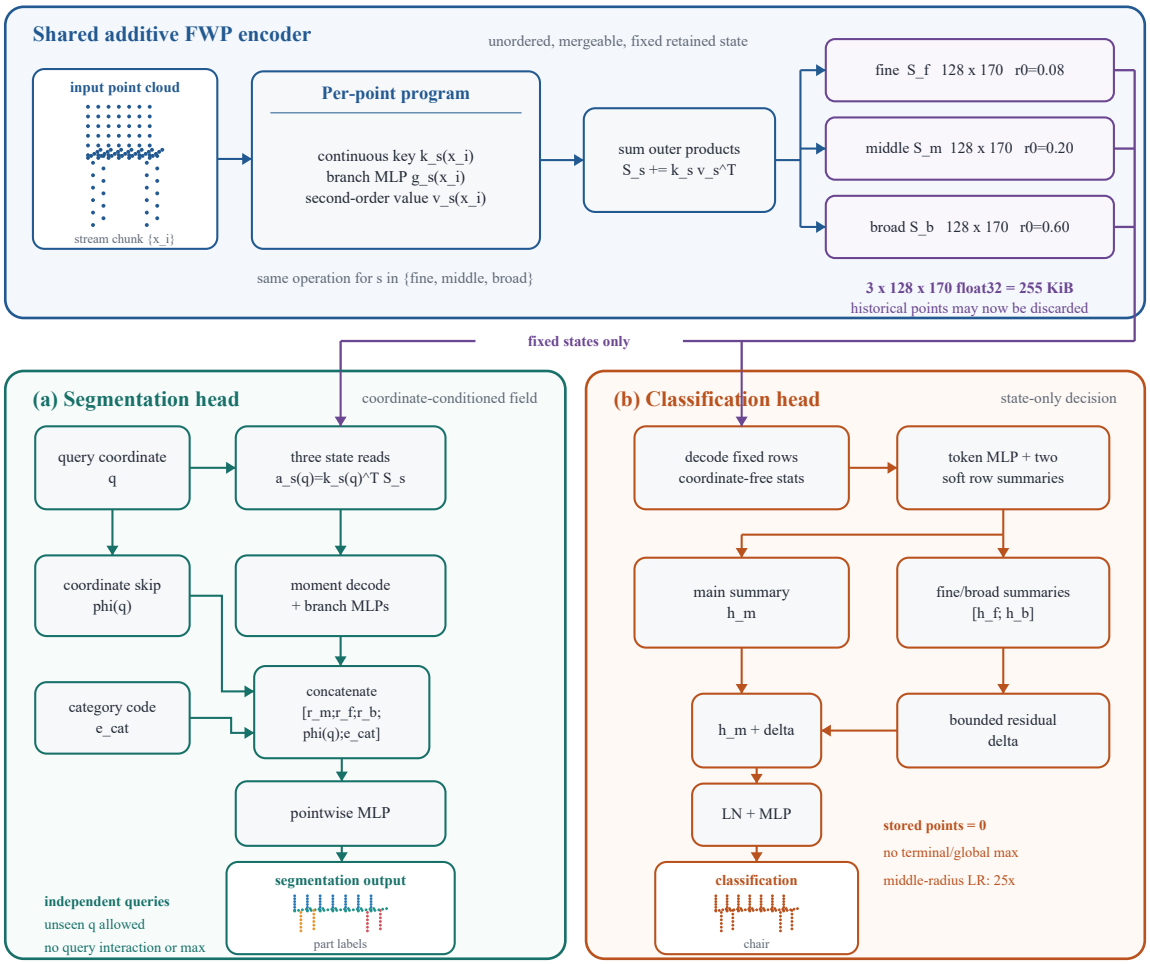

Figure 1: PointProgrammerNet turns a point cloud, or one part of one, into three fixed matrices. Each point writes independently at a learned spatial address and contributes a second-order value, so the matrices stay a sum and remain mergeable after the points are released. The segmentation head produces a part label by reading the matrices at a query coordinate; the classification head produces an object label from the matrix rows. Neither head reconstructs a point set or pools over decoded points.

This rank-one update uses the defining FWP operation: the key selects where to write, the value supplies what to write, and independent writes superpose by addition [17]. PointProgrammerNet preserves this linear superposition, without a write normalization over the observed point set or a nonlinear overwrite of the previous state. Thus a row is a weighted accumulator rather than a slot for one selected point, and summing the outer products gives

$$\mathbf{S}_s(X) = \sum_{i=1}^{N} \boldsymbol{k}_s(\boldsymbol{x}_i)\boldsymbol{v}_s(\boldsymbol{x}_i)^\top = \boldsymbol{K}_s^\top \boldsymbol{V}_s. \tag{5}$$

No point is assigned to a single row and no point is compared with another point. Once its three outer products have been added, it may be released.

For a newly arrived chunk, $\boldsymbol{K}_s \in \mathbb{R}^{M\times 128}$ stacks its addresses and $\boldsymbol{V}_s \in \mathbb{R}^{M\times 170}$ stacks its values. The chunk update is one matrix product $\Delta\mathbf{S}_s = \boldsymbol{K}_s^\top \boldsymbol{V}_s$. The persistent state is updated by $\mathbf{S}_s \leftarrow \mathbf{S}_s + \Delta\mathbf{S}_s$, after which $\boldsymbol{K}_s$, $\boldsymbol{V}_s$, and the chunk points can be discarded. Chunk size changes temporary working memory, but not the retained representation.

Table 2: The value written by one point. Every state row has these 170 columns.

Block	Dim.	Information retained
1	1	accumulated mass
$\boldsymbol{x}$	3	coordinate sum
$\boldsymbol{g}_s$	32	learned-content sum
$\boldsymbol{x}\boldsymbol{g}_s^\top$	96	position–content relation
$\boldsymbol{x}\boldsymbol{x}^\top$	6	geometric second moment
$\boldsymbol{g}_s \odot \boldsymbol{g}_s$	32	feature second moment

The work done for an arriving chunk is therefore short. The encoder (i) normalizes its coordinates in the shared reference frame, (ii) computes the three address and value matrices, (iii) adds the three products $\boldsymbol{K}_s^\top \boldsymbol{V}_s$ to the stored states, and (iv) releases every chunk-dependent tensor. Independent devices perform the same four steps and transmit only their three matrices. At inference, the segmentation head evaluates only requested coordinates, whereas the classification head consumes the matrices directly. No later stage needs to regenerate a point list from the state.

3.2 Task readouts

For segmentation, $\boldsymbol{x}_i$ denotes an observed point that has already been added to the state and may then be discarded; $\boldsymbol{q}$ is only the coordinate at which a label is requested. On ShapeNetPart, queries use the labeled input coordinates, but the decoder receives the coordinate and fixed state rather than the stored point feature. A query produces an address by Equation (2) and reads

$$\boldsymbol{a}_s(\boldsymbol{q}) = \boldsymbol{k}_s(\boldsymbol{q})^\top \mathbf{S}_s = \sum_i \underbrace{\boldsymbol{k}_s(\boldsymbol{q})^\top \boldsymbol{k}_s(\boldsymbol{x}_i)}_{w_{si}(\boldsymbol{q})} \boldsymbol{v}_s(\boldsymbol{x}_i)^\top. \tag{6}$$

The coefficient $w_{si}(\boldsymbol{q})$ is an algebraic interpretation of the matrix read: it is large when point i wrote strongly to rows that $\boldsymbol{q}$ reads strongly. The implementation evaluates the left-hand matrix product directly; it neither reconstructs or revisits point i nor applies a global maximum over decoded points.

The first component of $\boldsymbol{a}_s(\boldsymbol{q})$ is the read mass $\mu_s(\boldsymbol{q})$. The implementation prevents unstable division in nearly empty regions with

$$\epsilon_s = 10^{-3}\,\mathrm{mean}_{\boldsymbol{q}'}\,\mu_s(\boldsymbol{q}') + 10^{-20}, \qquad \widetilde{\mu}_s(\boldsymbol{q}) = \max(\mu_s(\boldsymbol{q}), \epsilon_s). \tag{7}$$

The mean is taken over the query coordinates of one sample and is detached from the gradient. The maximum in Equation (7) is only a scalar numerical floor; it neither compares point features nor performs global maximum pooling. Dividing the stored coordinate, feature, cross-product, and square sums by $\widetilde{\mu}_s$ yields local coordinate and feature means $\bar{\boldsymbol{x}}_s, \bar{\boldsymbol{g}}_s$, cross moment $E_{xg,s}$, geometric moment $E_{xx,s}$, and feature energy $E_{gg,s}$. They form the 170-dimensional descriptor

$$\begin{aligned}\boldsymbol{d}_s(\boldsymbol{q}) = [&\bar{\boldsymbol{g}}_s, \bar{\boldsymbol{x}}_s - \boldsymbol{q}, \log \widetilde{\mu}_s, \mathrm{vec}(E_{xg,s} - \boldsymbol{q}\bar{\boldsymbol{g}}_s^\top),\\ &\mathrm{vech}(E_{xx,s} - \bar{\boldsymbol{x}}_s\bar{\boldsymbol{x}}_s^\top), \sqrt{[E_{gg,s} - \bar{\boldsymbol{g}}_s \odot \bar{\boldsymbol{g}}_s]_+ + 10^{-8}}].\end{aligned} \tag{8}$$

Here vech keeps the six distinct entries of a symmetric 3×3 matrix, and $[\cdot]_+$ clips roundoff-induced negative variances. Its block dimensions are $32, 3, 1, 96, 6, 32$, again totalling 170. The blocks are complementary: the feature mean supplies local content, the relative centroid and geometric covariance describe local position and shape, the log mass records how much evidence supports the read, the coordinate–feature term preserves how content changes around

the query, and the feature standard deviation exposes mixtures and boundaries that a mean would hide. Each branch applies the same $170 \to 160 \to 160$ BatchNorm–ReLU decoder. The final pointwise head is

$$\boldsymbol{z}_{\mathrm{seg}}(\boldsymbol{q}) = h_{\mathrm{seg}}([\boldsymbol{r}_f(\boldsymbol{q}); \boldsymbol{r}_m(\boldsymbol{q}); \boldsymbol{r}_b(\boldsymbol{q}); \phi(\boldsymbol{q}); \boldsymbol{e}_{\mathrm{cat}}]), \tag{9}$$

where $\boldsymbol{r}_s \in \mathbb{R}^{160}$ is a decoded state read, $\phi(\boldsymbol{q}) \in \mathbb{R}^{32}$ is a coordinate feature, and $\boldsymbol{e}_{\mathrm{cat}} \in \mathbb{R}^{16}$ is the ShapeNetPart category code. The segmentation MLP is $528 \to 256 \to 128 \to 50$, with BatchNorm, ReLU, and dropout 0.3 after its first two affine layers. The coordinate path is a U-Net-like skip around state compression. The head acts on each query separately and never pools over decoded points. Per-point features are deleted after the write, and of the coordinates only those that will later be labelled need to remain. By Proposition 2 the same fixed state is also defined at coordinates that were never observed.

For classification, no coordinates are retained. For row j, the 170 stored entries are partitioned as

$$[\mu_{sj}, \boldsymbol{p}_{sj}, \boldsymbol{f}_{sj}, \mathrm{vec}(\boldsymbol{C}_{sj}), \mathrm{vech}(\boldsymbol{Q}_{sj}), \boldsymbol{u}_{sj}],$$

that is, mass, coordinate sum, feature sum, coordinate–feature sum, symmetric coordinate second moment, and squared-feature sum. Row normalization uses

$$\widetilde{\mu}_{sj} = \max\left(\mu_{sj}, 10^{-4} R^{-1} \sum_{u=1}^{R} |\mu_{su}| + 10^{-20}\right), \tag{10}$$

whose lower bound is detached during back-propagation. With $\bar{\mu}_s = R^{-1} \sum_u \widetilde{\mu}_{su}$, division by $\widetilde{\mu}_{sj}$ produces

$$\begin{aligned} \boldsymbol{d}_{sj}^{\mathrm{row}} = [\bar{\boldsymbol{g}}_{sj}, \bar{\boldsymbol{x}}_{sj}, \log(\widetilde{\mu}_{sj}/\bar{\mu}_s), \mathrm{vech}(\mathrm{Cov}_{xx,sj}), \\ \mathrm{Std}_{g,sj}, \mathrm{vec}(\mathrm{Cov}_{xg,sj})] \in \mathbb{R}^{170}, \end{aligned} \tag{11}$$

with block dimensions $32, 3, 1, 6, 32, 96$. An independent $170 \to 128 \to 128$ LayerNorm–GELU network encodes every row, giving $\boldsymbol{T}_s \in \mathbb{R}^{128 \times 128}$. Two learned score columns summarize the fixed rows:

$$\boldsymbol{U}_s = \mathrm{softmax}_{\mathrm{rows}}(\mathrm{score}_s(\boldsymbol{T}_s)), \qquad \boldsymbol{h}_s = H_s(\mathrm{vec}(\boldsymbol{U}_s^\top \boldsymbol{T}_s)) \in \mathbb{R}^{256}. \tag{12}$$

This softmax is over 128 retained state rows, not over the input points, so its cost is fixed after encoding. The middle summary is the reference and the other two form a bounded residual:

$$\begin{aligned} \boldsymbol{\delta} &= \sigma(a) \tanh(W[\boldsymbol{h}_f; \boldsymbol{h}_b] + \boldsymbol{b}), \\ \boldsymbol{z}_{\mathrm{cls}} &= h_{\mathrm{cls}}(\mathrm{LN}(\boldsymbol{h}_m + \boldsymbol{\delta})). \end{aligned} \tag{13}$$

The residual projection starts at zero and $\sigma(a)$ starts at 0.1. The classifier h_{cls} is $256 \to 256 \to 40$, with GELU and dropout 0.3. All radii are learned; only the middle radius uses a $25\times$ learning-rate multiplier and zero weight decay. The classifier uses only the fixed matrices and has no terminal point maximum. Two score columns give each state complementary fixed-cost views instead of collapsing every row through one average, and the zero-started bounded residual keeps the fine and broad branches from perturbing the middle one before each has learned a useful summary, while still letting them in once they help.

3.3 State algebra and the coordinate field

Write the contribution of one point as $F_s(\boldsymbol{x}) = \boldsymbol{k}_s(\boldsymbol{x}) \boldsymbol{v}_s(\boldsymbol{x})^\top$, so that Equation (5) reads $\mathbf{S}_s(X) = \sum_{\boldsymbol{x} \in X} F_s(\boldsymbol{x})$. Everything in this section follows from that one line.

Proposition 1 (Addition, order, and devices)**.** *For arbitrary point multisets A and B,*

$$\mathbf{S}_s(A \uplus B) = \mathbf{S}_s(A) + \mathbf{S}_s(B), \tag{14}$$

and the state does not depend on the order in which points are absorbed. Devices using identical trained encoders, coordinate normalization, and reference frames therefore obtain the centralized state by adding their independently encoded states.

Proof. Both sides expand into the same finite collection of per-point matrices $F_s(\boldsymbol{x})$. Reordering or regrouping a finite sum does not change it in real arithmetic. Float32 results may differ only through accumulation order. □

To make the distributed case explicit, suppose device d observes point multiset $X^{(d)}$ and all devices use the same encoder. A central node forms

$$\mathbf{S}_s^{\text{global}} = \sum_{d=1}^{D} \mathbf{S}_s(X^{(d)}) = \mathbf{S}_s\left(\biguplus_{d=1}^{D} X^{(d)}\right). \tag{15}$$

Both sides expand into the same sum of per-point outer products, so the equality holds whether the partial states arrive simultaneously, asynchronously, or through a tree of intermediate aggregators. Raw point features need not leave the sensing devices.

Addition is not the only operation Equation (14) supplies. If Y is a subset whose own state was retained, then

$$\mathbf{S}_s(X \setminus Y) = \mathbf{S}_s(X) - \mathbf{S}_s(Y), \tag{16}$$

which withdraws the entire contribution of Y without replaying X. Several ordinary situations take this form. A source whose calibration is later found to be wrong, or whose readings should no longer be trusted, is removed by one subtraction while the rest of the accumulated evidence is untouched. A correction is the same operation followed by an addition: when the pose of a device is refined after the fact, its old contribution is subtracted and its re-encoded contribution is added, so only that device's own observations have to be revisited. A source whose data must be withdrawn on request is handled the same way, and because the state holds no point list, nothing else has to be located and deleted. The operation is also a diagnostic: subtracting one contribution and reading the state again reports how much that source was affecting the prediction. Note that a removal acts on aggregates. It removes what Y wrote, and recovering an individual point requires that its own state was kept.

Where time enters

The writes of one observation combine by addition alone into

$$\boldsymbol{A}_{s,t} = \sum_{\boldsymbol{x} \in X_t} F_s(\boldsymbol{x}), \tag{17}$$

so the points of that observation are exchangeable and may be split across chunks or devices in any way. The transition between observations is the only ordered step in the model:

$$\mathbf{S}_{s,t} = \rho\, \mathbf{S}_{s,t-1} + \boldsymbol{A}_{s,t}, \qquad \rho \in [0, 1]. \tag{18}$$

Because the state is a sum of independent rank-one writes, multiplying it by ρ is not an approximate model of forgetting; it reweights every write already stored, exactly. Unrolling Equation (18) gives

$$\mathbf{S}_{s,t} = \boldsymbol{A}_{s,t} + \rho \boldsymbol{A}_{s,t-1} + \rho^2 \boldsymbol{A}_{s,t-2} + \cdots, \tag{19}$$

so the weight an observation carries depends only on its age, and $\rho = 1$ recovers plain accumulation in which every observation counts equally. The arrangement keeps two properties that

a recurrence over serialized points has to trade against each other: the state carries temporal context, and the points inside an observation are still unordered and additively separable. A scene that is observed again and again can therefore let earlier evidence fade, while each individual pass is still encoded by however many devices are convenient.

Proposition 2 (Coordinate field)**.** *For a fixed state* $\mathbf{S}_s$ *and any* $\boldsymbol{q} \in \mathbb{R}^3$*, the read* $\boldsymbol{a}_s(\boldsymbol{q}) = \boldsymbol{k}_s(\boldsymbol{q})^\top \mathbf{S}_s$ *is defined, and the segmentation logits of Equation* (9) *are continuous and piecewise smooth in* $\boldsymbol{q}$*.*

Proof. By Equation (2), $\boldsymbol{k}_s(\boldsymbol{q})$ is a softmax of smooth functions of $\boldsymbol{q}$ over a fixed set of 128 learned centers. It is therefore defined and smooth on all of $\mathbb{R}^3$, and it requires no stored point near $\boldsymbol{q}$. The read is linear in $\mathbf{S}_s$. The descriptor of Equation (8) divides that read by a quantity bounded below by the positive floor of Equation (7), takes the logarithm of the same bounded quantity, and takes square roots stabilized away from zero; the branch decoders and the final head compose affine layers, normalization layers, and piecewise-smooth activations. A composition of such maps is continuous and piecewise smooth. □

A fixed state can accordingly be evaluated on a regular grid, at a resolution other than the one that was recorded, or inside a region chosen after the observations were released. A readout anchored to stored point features has nothing to evaluate at such a position. Section 4.6 measures what a label read this way is worth. What the state does not supply is the query itself: classification retains nothing beyond the three matrices, while segmentation additionally needs the coordinates that are to be labelled — only those, and no point features — together with the object category code on ShapeNetPart.

Proposition 3 (Retained memory and update cost)**.** *With* R_s *rows and* D_s *columns per branch, retained memory is* $O(\sum_s R_s D_s)$*, independent of the number of observations absorbed. Encoding* M *new points costs* $O(M \sum_s R_s D_s)$ *and does not revisit earlier observations.*

Proof. Only the fixed matrices persist. A new chunk contributes new outer products of the same dimensions regardless of the number of earlier chunks. □

A write can also be admitted at partial strength, which is the third thing linearity provides. Scaling one contribution by $t \in [0, 1]$,

$$\mathbf{S}_s(t) = \mathbf{S}_s^{\text{old}} + t\, \boldsymbol{k}_s(\boldsymbol{x}) \boldsymbol{v}_s(\boldsymbol{x})^\top, \tag{20}$$

keeps the state affine in t, and the same holds for a whole chunk because the chunk update is itself a sum. A sensor that is only partly trusted can therefore be admitted at a reduced weight instead of being accepted in full or rejected outright, and what is stored remains the state of a weighted point set. Gaussian addresses, softmax, normalized moment reads with a positive floor, affine layers, BatchNorm or LayerNorm, ReLU or GELU, and stabilized square roots are all continuous, so the classification logits and the segmentation logits at a fixed coordinate move continuously with t.

For the released configuration, $R_s = 128$ and $D_s = 170$ in three branches, so the retained representation holds 65,280 numbers, or 261,120 bytes (255 KiB) in float32. Temporary tensors scale with the arriving chunk and are released immediately.

3.4 Parallel evaluation

The write constraint of Section 2.6 also fixes how the computation can be laid out. Within a chunk the write of one point reads no other point, so all M points are evaluated in parallel and the update is the single matrix product of Equation (5); the address of Equation (2) normalizes over the 128 rows of its own branch rather than over the observed set, so its denominator never

Table 3: ShapeNetPart segmentation with XYZ input. PointProgrammerNet never pools over decoded query points.

Model	Point acc. (%)	Ins. mIoU (%)	Cat. mIoU (%)	Params
Lightweight PointNet++	93.41 ± 0.05	83.88 ± 0.11	79.51 ± 0.24	573,106
PointProgrammerNet	92.95 ± 0.02	82.58 ± 0.15	79.10 ± 0.25	341,909

becomes a reduction across points. Between chunks and devices Proposition 1 makes the combination a sum, which is associative and commutative, so partial states can be produced concurrently and reduced in any order, including a tree of depth $\log D$ over D producers. The three branches share no term during the write, and the segmentation head acts on each query separately, so reading a set of coordinates is again one matrix product followed by pointwise networks.

Three common constructions do not admit this layout. A metric neighborhood needs a data-dependent search over resident points, a recurrence over serialized points is sequential and depends on the traversal, and a normalizer taken over the observed set is itself a reduction across points.

4 Results

The evaluation asks two things of the representation. The first is whether a state written under the constraint of Section 2.6 still supports standard recognition; Section 4.1 reports ShapeNetPart and ModelNet40 accuracy and Section 4.2 isolates the parallel-state design. The second is whether the algebra of Section 3.3 survives a float32 implementation; Sections 4.3 to 4.5 measure chunk equivalence and update cost, compare merge, removal and attenuation against direct reconstruction, and read a task prediction from a partially accumulated state. Section 4.6 then asks what Proposition 2 is worth in accuracy rather than in definedness, by reading a state at labelled coordinates it never wrote.

4.1 Protocol and accuracy

We center each object and divide XYZ by its maximum absolute coordinate, using 1,024 points. ShapeNetPart contains 13,972 training and 2,874 test shapes; ModelNet40 contains 9,840 and 2,468. For the ModelNet40 density-gradient shift, a deterministic random unit direction is generated for each cloud using seed 0, and 1,024 points are sampled without replacement with probabilities proportional to $\exp(2.5\boldsymbol{x}^\top \boldsymbol{d})$. Models train for 20 epochs with AdamW, base learning rate 3×10^{-3}, weight decay 10^{-4}, cosine decay, and batch sizes 16 and 32. Classification uses 0.1 label smoothing. Results are mean $\pm$ population standard deviation over seeds 0, 1, and 2. The segmentation model contains 341,909 trainable parameters and the state-only classifier contains 529,426. In classification, the middle radius alone uses a 25$\times$ learning-rate multiplier without weight decay; the controlled screen in Section 4.7 motivates this robustness-oriented choice. All reported PointProgrammerNet variants use three 128×170 states and no global maximum over input or decoded points. The compact PointNet++ controls use the same protocol and input conditions to isolate the retained representation. Tables 3 and 4 report the final ShapeNetPart and ModelNet40 results, respectively.

With 40.3% fewer parameters, PointProgrammerNet stays within 0.46 points of the lightweight PointNet++ control in point accuracy and within 1.30 points in instance mIoU.

The state-only classifier reaches 86.59% clean accuracy and 82.47% under the density shift, using no coordinates at read time and no pooling over input points. Table 4 places these figures beside both controls.

Table 4: ModelNet40 classification with 1,024 points. Density denotes the fixed density-gradient shift.

Model	Clean (%)	Density (%)	Drop (pp)	Params
Lightweight PointNet++	89.84 ± 0.43	85.47 ± 0.88	4.38 ± 1.18	207,784
Capacity-controlled PointNet	87.90 ± 0.27	87.55 ± 0.38	0.35 ± 0.12	442,838
PointProgrammerNet	86.59 ± 0.23	82.47 ± 0.93	4.12 ± 0.71	529,426

Table 5: Parallel states against a single state. Five-epoch screens, three seeds, on 2,048 training and 512 test shapes. The single-state control retains the full model and initialization but removes both auxiliary reads.

Configuration	States	Params	Ins. mIoU (%)	Cat. mIoU (%)
Main state only (causal control)	1	484,914	74.68	70.50
Main state widened to 256 rows	3	485,298	74.79	70.53
Parallel states, conservative radius rate	3	484,914	75.73	71.03
Parallel states, stronger residual mix	3	484,914	75.88	70.97
Released three-state model	3	341,909	77.22	73.08

4.2 Parallel states versus a single state

The three states are not a redundancy, and their benefit is not a parameter-count effect. We test this causally with a strict pairing: a control that keeps the complete model and its initialization but removes the two auxiliary reads, so that only the main state can influence the output. Both sides then have identical parameter counts and identical initial weights. Table 5 reports the screen, run for five epochs with seeds 0, 1, and 2 on 2,048 training and 512 test shapes.

Removing the two auxiliary states costs 1.05 to 1.20 instance mIoU at an unchanged parameter budget. Adding capacity to one state instead does not help: doubling the main state from 128 to 256 rows, while keeping all three states, reaches 74.79 and stays below every configuration whose states are all 128 rows wide. The released model, which uses three states at 341,909 parameters, reaches 77.22 under the same screening protocol, exceeding the single-state control by 2.54 points with 29.5% fewer parameters. Parallel states contribute usable structure that neither a larger parameter budget nor a wider main state supplies. These short subset screens isolate design effects; final test-split accuracy is reported in Section 4.1.

4.3 Chunk equivalence, retained memory, and update cost

Across 32 objects, states built from 2–32 ordered or shuffled chunks agree with one-shot construction to a relative error below 5.3×10^{-7}, so Proposition 1 survives float32 accumulation. The three states hold 65,280 scalars, or 255 KiB, whatever the number of observations absorbed, which is the bound of Proposition 3; raw XYZ passes that size at about 21,800 observations. Figures 2 and 3 show the numerical equivalence, retained-memory behavior, and measured update-time scaling.

On an RTX 3070 Ti Laptop GPU, four streams receive 256 new points per frame. Incremental latency stays near 1.46–1.53 ms, whereas reconstruction grows with history. Updating becomes 1.52×, 3.04×, and 6.06× faster at 2,048, 4,096, and 8,192 accumulated points. The two paths cost the same below roughly 512 accumulated points, and the update path is ahead from there on.

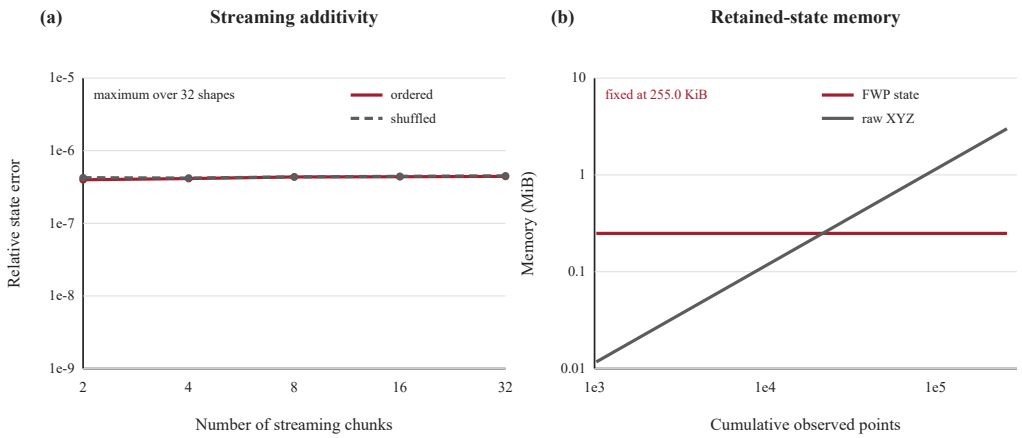

Figure 2: Additive accumulation and fixed retained memory. Writing the same cloud in ordered or shuffled pieces reproduces the state built in one batch, up to floating-point accumulation error. The three 128×170 float32 matrices occupy 255 KiB however many observations have been absorbed, whereas stored XYZ coordinates grow with their number.

4.4 Merge, removal, and attenuation

Figure 4 summarizes the append and merge, removal, and attenuation checks against direct reconstruction. Append, independently encoded merge, stored-frame removal, and exponential decay were compared with direct reconstruction on 32 ShapeNetPart objects. Their mean relative errors were 2.11×10^{-7}, 2.13×10^{-7}, 3.03×10^{-7} and 1.77×10^{-7}, and every maximum stayed below 9.8×10^{-7}. The residual discrepancy is consistent with float32 accumulation order. The retention factor behaves as Equation (19) predicts: after 16 observations plain accumulation reached state norm 14.51, whereas $\rho = 0.9$ held it at 7.42, so the influence of an observation is bounded by its age rather than growing with the number absorbed.

4.5 Predictions from partially accumulated states

The preceding checks establish state equivalence, but not whether a state that holds only part of a cloud already carries enough evidence for a prediction. We therefore accumulate each test object in pieces on the full ModelNet40 test split. For each of the 2,468 objects, the same 1,024 normalized points used in the main evaluation are placed in a deterministic independent random order and written in chunks of 128. After every chunk, the frozen classifier predicts directly from the accumulated state; no prefix-specific training or adaptation is performed. The permutation is fixed across the three reported training seeds. Random arrival isolates the accumulation of evidence from the viewpoint and occlusion pattern of any particular sensor trajectory.

Table 6 shows that a usable prediction is available well before accumulation finishes: after 256 points the mean accuracy is 83.29%, and after half of the cloud it is 86.02%, within 0.57 percentage points of the complete object. Agreement with the completed prediction rises from 84.31% at 128 points to 96.39% at 512. At completion, accumulation in pieces and one-shot encoding agree on the predicted class for every one of the 2,468 test objects, with a maximum relative state error of 6.76×10^{-7} and a maximum absolute logit difference of 7.96×10^{-5}. The equivalence of Section 3.3 therefore reaches the task output, not only the stored matrices.

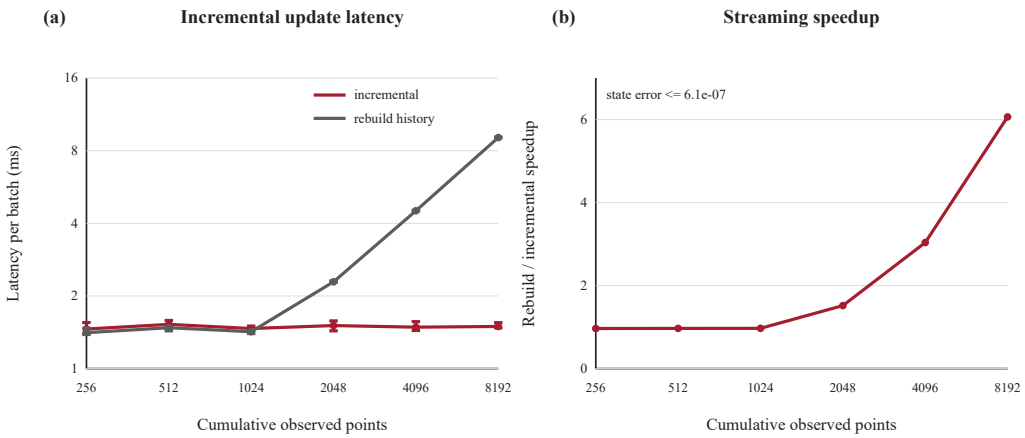

Figure 3: Cost of building the state on a device that receives points in pieces. (a) An incremental update processes one new 256-point piece, whereas reconstruction processes every point accumulated so far. Values are median wall-clock latency over seven interleaved trials of 30 repetitions for four states; whiskers show the observed range. (b) The speedup grows with the number of accumulated points, and the incremental and reconstructed states stay numerically equal.

Table 6: ModelNet40 classification from partially accumulated states, with points arriving in random order. Values are mean ± population standard deviation over three frozen training seeds. Agreement compares each partial prediction with the corresponding one-shot 1,024-point prediction.

Observed points	Fraction	Accuracy (%)	Full-prediction agreement (%)
128	12.5%	77.70 ± 0.15	84.31 ± 0.48
256	25.0%	83.29 ± 0.24	91.72 ± 0.13
512	50.0%	86.02 ± 0.12	96.39 ± 0.09
768	75.0%	86.02 ± 0.32	97.95 ± 0.07
1024	100%	86.59 ± 0.23	100.00 ± 0.00

4.6 Reading at coordinates that carry no written point

Linear-additive summaries support merging, subtraction and attenuation. PointProgrammerNet also spreads the accumulated evidence over spatial addresses, so a read at a position is resolved inside the state rather than left entirely to the decoder that consumes it. Proposition 2 says that read is defined everywhere, but not what a label read away from the written points is worth, and this section measures it. A ShapeNetPart shape supplies the material. Each shape holds between 2,423 and 2,947 labelled points, of which the evaluation protocol writes a fixed subsample of 1,024. The remaining labelled points are coordinates the state never wrote, and they carry ground truth.

The three released states of each test shape are therefore built exactly as in Section 4.1, from the same 1,024 points, and are then read twice. One read uses the written coordinates and reproduces the number in Table 3. The other uses up to 1,024 of the labelled coordinates that were never written. The state, the evidence it holds and the frozen weights are identical across the two reads; only the query coordinates differ. Centering and scaling are taken from the written points alone, so a query never contributes information about the frame in which it is expressed. On 108 of the 2,874 test shapes the protocol subsample exhausts the available points, leaving fewer than 1,024 unwritten coordinates; those shapes are excluded from the comparison and a matched control on the remaining 2,766 is reported beside the full split.

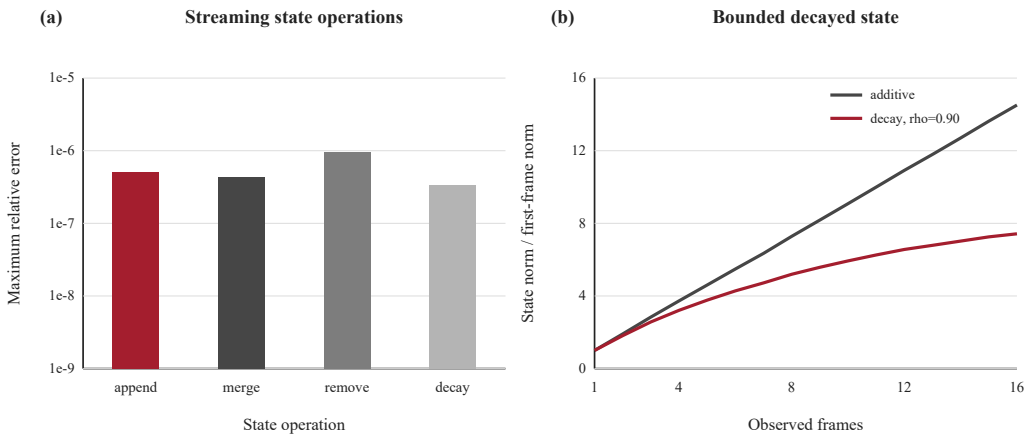

Figure 4: State operations against direct reconstruction. (a) Maximum relative errors for append and merge, for removal of a stored contribution, and for attenuation by a retention factor, measured on 32 ShapeNetPart objects. (b) Plain accumulation grows the state magnitude with the number of observations, while the retention factor bounds the influence of older ones without changing the size of the state.

Table 7: Segmentation read at written and at unwritten coordinates. Every row uses the same released states, built from the standard 1,024-point subsample; only the queried coordinates differ. Values are mean ± population standard deviation over the three released seeds.

Queried coordinates	Shapes	Point acc. (%)	Ins. mIoU (%)	Cat. mIoU (%)
Written, full test split	2,874	92.95 ± 0.02	82.58 ± 0.15	79.10 ± 0.25
Written, matched subset	2,766	93.01 ± 0.03	82.46 ± 0.17	79.02 ± 0.24
Never written, same states	2,766	93.03 ± 0.04	82.41 ± 0.17	79.00 ± 0.28

Figure 5 shows the reads for three shapes, and Table 7 reports the comparison over 2,832,384 queries per seed in each of the matched rows. Moving the query off the written points changes point accuracy by $+0.02$, instance mIoU by -0.05 and category mIoU by -0.02 points, all of them smaller than the spread across seeds. The queries are not coincident with written points: in units where the largest absolute coordinate of a shape is one, a query lies a median of 0.0291 from the nearest written point, and 0.0578 at the ninetieth percentile. A label produced at a coordinate the state never wrote is therefore worth close to what a label produced at a written coordinate is worth, which is what makes a grid read, a change of resolution, or a region selected after the fact usable rather than merely defined.

4.7 Controlled screens for the final heads

Figure 6 visualizes the paired screening effects for the retained segmentation and classification choices. The screens isolate one architectural decision at a time. All of them use the same data split and optimization protocol within a comparison, five epochs, seeds 0, 1, and 2, and a subset of 2,048 training and 512 test shapes. These short runs select structural choices; the main tables report final 20-epoch results on the complete splits. Table 8 reports paired changes of the retained design relative to the named alternative.

Every retained choice is favoured, and the equal segmentation decoders also remove 24,480 parameters. Two entries are worth a word. The middle-reference residual is the largest single effect in the table, and the raised middle-radius learning rate is kept for its consistent gain under the density shift rather than for its unstable clean-accuracy change. The screens motivate the

Figure 5: A state read where it never wrote. Each row is one test shape. The left column separates the two roles a point can play: dark points are the 1,024 the state is built from, which the control also reads, and pale points are labelled coordinates that were never written and are only read. Both sets are drawn in full, in an interleaved order so that neither occludes the other; they cover the same surface, which is why the comparison is a controlled one. The middle column shows the part predicted at the never-written coordinates and the right column the ground truth at the same coordinates. Each shape is the median performer of its category on this measurement, not the best. Hue encodes the part throughout; the categories shown have at most three.

released heads while keeping the state-writing rule symmetric across branches.

5 Discussion and Limitations

Each operation of Section 3.3 answers a question that a batch encoder answers badly, and the value of the representation is that they hold together rather than one at a time. Section 5.1 works through the settings in which each of them lands.

5.1 Application settings

The algebra does not depend on any particular sensor, so the main text names none. The devices in this subsection may be depth cameras, LiDAR units, structured-light scanners, or the edge processors attached to them.

Consider several such nodes observing different parts of a scene. The trained encoder is copied to every node, and all nodes express coordinates in a common frame with the same normalization. Node d encodes its local chunk into $\{\mathbf{S}_f^{(d)}, \mathbf{S}_m^{(d)}, \mathbf{S}_b^{(d)}\}$ and can then delete the local features. A relay may add any subset of these triples; the server adds the relay outputs and obtains Equation (15). The order of arrival and the shape of the communication tree do not affect the real-arithmetic result. Each transmitted triple has the same 255 KiB size whether the local chunk contains hundreds of points or a long accumulation of them.

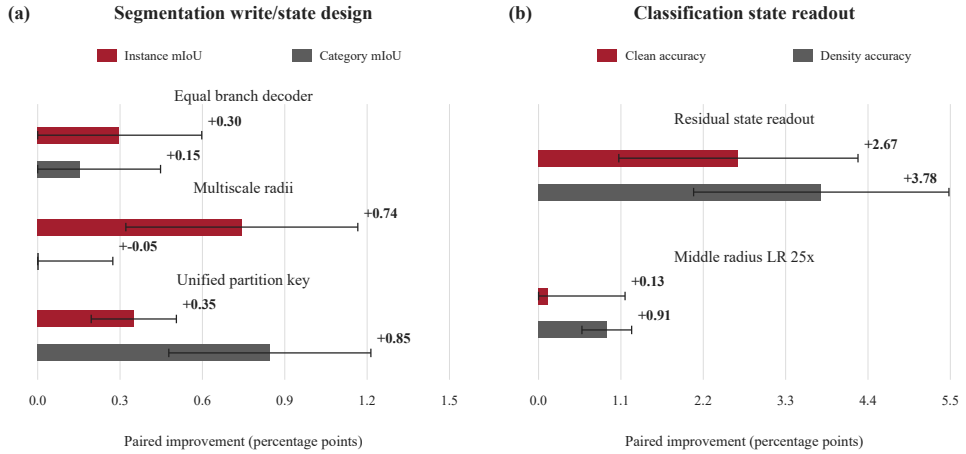

Figure 6: Three-seed, five-epoch architecture screens. Bars show paired mean changes and whiskers show population standard deviations. Segmentation tests equal branch decoders, distinct radius initialization, and a common normalized spatial address. Classification tests residual versus concatenated branch summaries and the middle-radius learning rate.

Table 8: Controlled five-epoch screens. Positive changes favor the retained design.

Retained choice	Paired alternative	Paired change
Seg.: equal decoders	unequal middle decoder	$+0.30 \pm 0.30$ instance; $+0.15 \pm 0.29$ category
Seg.: distinct radius starts	equal radius starts	$+0.74 \pm 0.42$ instance; -0.05 ± 0.33 category
Seg.: common normalized key	mixed key definitions	$+0.35 \pm 0.16$ instance; $+0.85 \pm 0.37$ category
Cls.: middle residual	direct concatenation	$+2.67 \pm 1.60$ clean; $+3.78 \pm 1.71$ shifted
Cls.: 25× middle-radius LR	base radius LR	$+0.13 \pm 1.03$ clean; $+0.91 \pm 0.33$ shifted

Figure 7 carries out this exchange on a single object. For classification, the server passes the merged triple to the state-row readout, and a new device or observation changes the class evidence by adding one more triple. For segmentation, the server supplies a coordinate $\boldsymbol{q}$ to the three matrix reads. By Proposition 2 that read is defined and continuous at every $\boldsymbol{q}$, and the experiment in Section 4.6 places reads at unwritten coordinates within 0.05 instance mIoU of reads at written coordinates. Queries may therefore be the coordinates of selected observed points, a region designated after the observations are over, or positions on a regular grid. Only those coordinates have to be available when the read happens.

The four operations land in different deployments. Merging fits a fixed multi-sensor rig, a moving platform carrying more than one range sensor, and equally a cluster that divides a single large scan across processors. Subtraction fits any deployment whose calibration is revisited: a unit found to be misaligned is withdrawn, a unit whose pose is later refined is reinstalled by a subtraction followed by an addition, and a source whose data must be released on request is removed without searching a point list for its contributions. The retention factor fits a scene captured on a schedule, such as an inspection route or a construction site followed over weeks, where the most recent pass should dominate but earlier passes should not be discarded outright. The coordinate read fits an inspection or annotation service that decides only afterwards which positions matter, and then asks for labels at exactly those positions.

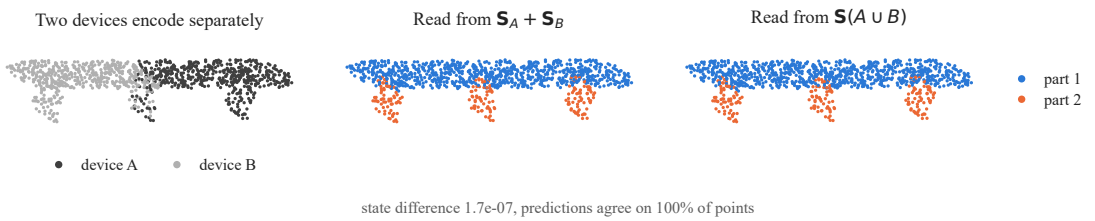

Figure 7: Distributed encoding of one object. Two devices see disjoint halves in a shared frame and encode them independently; the left panel marks which device wrote each point. Adding the two states and reading the result gives the middle panel, and encoding all the points centrally gives the right panel. The two states differ by 1.7×10^{-7} in relative terms and the two reads agree on every point.

5.2 What the fixed state retains and discards

The states are summaries, not compressed point archives: two point sets can write similar moment tables where many addresses overlap, and no decoder can restore a distinction the write already discarded. The state, not the readout, is therefore the binding bottleneck, and the screens show it. Changing the segmentation decoders moves instance mIoU by 0.30 ± 0.30 points and changing their width or sharing pattern produces similar sub-point changes, whereas changing the state configuration in Section 4.2 moves it by more than a point. This is why the design spends its structure on how points are written — the normalized address, the second-order value, the parallel states — rather than on how the state is decoded.

The costs are as concrete as the operations. A state occupies 255 KiB, which is more than bare XYZ below roughly 21,800 observations. Exact merging assumes that every device shares the trained weights, the coordinate normalization and the reference frame, and segmentation reads the state once per requested coordinate. The evaluation uses XYZ input, one dataset per task, one timing GPU and locally trained baselines. The unwritten coordinates of Section 4.6 are labelled points of the same shape, so they lie within the region the state observed; they are not a test of extrapolation into space the state never saw. States are accumulated in random order on static objects rather than along correlated sensor sweeps with motion and occlusion. Temporal tasks under the retention factor and further input attributes are left to future work, as is the adaptive state update sketched below.

Equation (18) weights an observation by its age alone. When the content of an observation should also decide how memory is revised, a frame-level fast-weight delta update [17] takes its place:

$$\mathbf{S}_{s,t} = \rho \mathbf{S}_{s,t-1} + \eta_t \boldsymbol{K}_{s,t}^{\top} (\boldsymbol{V}_{s,t} - \boldsymbol{K}_{s,t} \mathbf{S}_{s,t-1}), \tag{21}$$

where $\boldsymbol{K}_{s,t}$ and $\boldsymbol{V}_{s,t}$ package the keys and values of observation t and η_t is an update rate. Reordering points inside an observation multiplies both matrices on the left by the same permutation P, and the correction is unchanged because $P^{\top} P = I$, so within-observation order invariance survives. Ordered transitions of this kind cost exact addition across observations, and the experiments reported here use Equation (18).

6 Conclusion

PointProgrammerNet stores a point cloud as a sum of independent per-point writes, placed at learned spatial addresses and carrying second-order statistics. Keeping the write free of coupling between points, while deferring normalization across accumulated observations and nonlinear interpretation of the accumulated statistics to the read, makes the stored object algebraic. States of disjoint observations add exactly, so several devices can encode locally and

a server can merge their fixed-size messages. A contribution whose own state was kept can be subtracted, so an unreliable source is withdrawn, and a corrected one installed, without re-encoding the rest. Multiplying by a retention factor reweights everything already written according to its age, which lets a repeatedly observed scene forget while each pass is still encoded by however many devices are convenient. Order therefore never enters inside an observation, and enters between observations only through that factor. Because the sum is spread over spatial addresses rather than collapsed into one vector, a fixed state also defines a segmentation field that is continuous over the whole coordinate space and can be evaluated at coordinates that carry no written point, which costs 0.05 instance mIoU against the written coordinates.

The representation reaches 82.58% instance mIoU on ShapeNetPart with 40.3% fewer parameters than a lightweight PointNet++ control and 86.59% on ModelNet40 from the state alone, and three parallel states carry structure that one wider state does not supply at the same parameter budget. Merge, removal and attenuation agree with direct reconstruction to within 10^{-6}, and a prediction read from half of an object already matches the completed prediction on 96.4% of the test set and on all of it once accumulation finishes. These results measure the recognition information retained by the fixed state. The contribution is the write constraint and the state arithmetic it preserves after the points are released.

Statements and Declarations

Acknowledgments. None.
Competing interests. The author declares no competing financial or non-financial interests related to this work.
Funding. No funding was received for conducting this study.
Ethics approval. Not applicable. This study uses public non-human point-cloud benchmarks and involves no human participants, personal data, or animal experiments.
Patient consent. Not applicable.
Clinical trial registration. Not applicable.
Permission to reproduce material. Not applicable. All figures are original to this manuscript and contain no third-party material.
Author contributions. Dian Yu: conceptualization, methodology, software, validation, formal analysis, investigation, data curation, visualization, writing – original draft, writing – review and editing.
Data and code availability. ShapeNetPart and ModelNet40 are public datasets available from the original sources cited in this article. The implementation, training scripts, partial-state and coordinate-field evaluations, state-operation checks, and exact three-seed reference values are publicly available at `https://github.com/sdsds222/PointProgrammerNet-reproducibility-code`. The repository README specifies the environment, dataset layout, commands, and the state-update interface. Additivity tests should be run in evaluation mode, because BatchNorm then uses the fixed running statistics assumed by Proposition 1.

References

[1] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017.

[2] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, volume 30, pages 5099–5108, 2017.

[3] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3DmFV: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3(4):3145–3152, 2018.

[4] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304–3311, 2010.

[5] Mikaela Angelina Uy and Gim Hee Lee. PointNetVLAD: Deep point cloud based retrieval for large-scale place recognition. In *CVPR*, pages 4470–4479, 2018.

[6] Rui Guo, Yong Zhou, Jiaqi Zhao, Yiyun Man, Minjie Liu, Rui Yao, and Bing Liu. Point cloud classification by dynamic graph CNN with adaptive feature fusion. *IET Computer Vision*, 15(3):235–244, 2021.

[7] Zheng Fang, Binyu Xiong, and Fei Liu. Sparse point–voxel aggregation network for efficient point cloud semantic segmentation. *IET Computer Vision*, 16(7):644–654, 2022.

[8] Bingjie Hao, Wenjing Zhang, Hongjiu Liu, Xiao Dong, Baorong Yang, and Songzhi Su. A comprehensive survey on deep learning-based semantic segmentation of 3d point clouds: Technologies, challenges, and future directions. *IET Computer Vision*, 20(1):e70079, 2026.

[9] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *NeurIPS*, volume 30, pages 3391–3401, 2017.

[10] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *CVPR*, pages 5297–5307, 2016.

[11] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, volume 97 of *PMLR*, pages 3744–3753, 2019.

[12] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver IO: A general architecture for structured inputs and outputs. In *ICLR*, 2022.

[13] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, pages 4460–4470, 2019.

[14] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, pages 165–174, 2019.

[15] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, pages 523–540, 2020.

[16] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *ICML*, volume 119 of *PMLR*, pages 5156–5165, 2020.

[17] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *ICML*, volume 139 of *PMLR*, pages 9355–9366, 2021.

[18] Dingkang Liang, Xin Zhou, Wei Xu, Xingkui Zhu, Zhikang Zou, Xiaoqing Ye, Xiao Tan, and Xiang Bai. PointMamba: A simple state space model for point cloud analysis. In *NeurIPS*, 2024.

[19] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[20] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312, 1996.

[21] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, pages 127–136, 2011.

[22] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920, 2015.

[23] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics*, 35(6):210:1–210:12, 2016.

[24] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012*, 2015.