PointProgrammerNet: A fixed-size additive state for streaming 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

Fixed-size point-cloud descriptors are well established, but a descriptor for one-shot inference is not necessarily a persistent state. After absorbing a chunk, it should still support merging independent chunks, removing or decaying evidence, and querying spatial context without recovering point history. PointProgrammerNet meets these requirements with three fixed 128×170 matrices. Each observation produces a continuous coordinate-conditioned address and a second-order value whose outer product is accumulated. Writes remain unnormalized across observations; density normalization and nonlinear interpretation are deferred to read time. This separation makes partial states exactly composable by addition and enables subtraction and exponential decay. Segmentation reads the matrices at a requested coordinate, whereas classification uses only the state after all points have been discarded. We prove order invariance, chunk and distributed equivalence, fixed retained memory, and continuous dependence on incoming evidence. With XYZ input, the model achieves 92.95% point accuracy, 82.58% instance mIoU, and 79.10% category mIoU on ShapeNetPart, and 86.59% accuracy on ModelNet40. The three float32 states occupy 255 KiB regardless of stream length. At 8,192 accumulated points, a chunk update is 6.06 times faster than rebuilding the state, with relative error below 10^{-6}.

Keywords: point-cloud processing; streaming inference; additive state; distributed sensing; segmentation; classification
Running title: PointProgrammerNet additive state

1 Introduction

A point cloud is an unordered set, but a visual sensor observes it as a sequence of frames, packets, or independently produced spatial regions. This creates a representation problem that is different from ordinary batch recognition. The retained representation should be independent of point order, fixed in size, exactly composable across chunks or devices, and still spatially queryable after the observed points have been released. Existing point-cloud encoders usually satisfy only a subset. PointNet [1] uses a bounded symmetric maximum, PointNet++ [2] rebuilds metric neighborhoods, and Fisher- or VLAD-style encoders form fixed descriptors for batch inference [3, 4, 5]. None of their published representations is designed to provide all four properties together.

We therefore study a stricter setting: a point may be discarded immediately after it is written, and every later operation accesses only persistent fixed-size matrices, plus a requested coordinate for segmentation. A learned spatial key distributes each point over state rows, while a second-order value records mass, position, learned content, and cross moments. Their

outer product is an independent per-point program, and summing these programs produces the retained state.

The central design choice is where nonlinearity and normalization occur. Soft spatial assignment and moment accumulation themselves are established ideas; we do not claim otherwise. PointProgrammerNet keeps the persistent write as an unnormalized sum and postpones density normalization, moment interpretation, and task-specific nonlinear processing until read time. Consequently, independent chunks can be merged, a stored subset can be subtracted, and old evidence can be decayed without replaying earlier points. The same algebra supports distributed sensing: edge devices encode disjoint regions or time windows, transmit fixed matrices, and a central node adds those matrices before classification or segmentation. Segmentation reads a coordinate-conditioned field; classification reads only state rows.

Our contributions are:

- a persistent-state formulation for point-cloud recognition that jointly provides order-free writes, bounded retained memory, exact state composition, and coordinate-conditioned access;
- a three-scale, second-order state whose pointwise outer-product writes remain unnormalized across observations, so independently produced states can be merged, subtracted, or decayed before normalization at read time;
- a coordinate-conditioned segmentation read and a state-only classification read that operate after historical point features have been discarded and admit parallel evaluation;
- formal guarantees and experiments covering state algebra, accuracy, retained memory, and update cost, including an equal-parameter causal test of the parallel-state structure.

2 Related Work

There are several established ways to reduce an unordered point set to a fixed summary. They differ mainly in what they give up at the moment a point is absorbed, and that choice decides which state operations remain available afterwards. We group the related work by that choice, then state which properties this architecture keeps.

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 [6]. All three are invariant to point order, and the sum used by Deep Sets is additive in the strict sense of Equation (14). A growing stream nevertheless runs into two obstacles. A channelwise maximum discards the information needed to undo it, so a stored summary cannot be corrected once an observation is removed. A metric neighborhood is defined relative to the points currently held; insert or delete one point and the membership of other neighborhoods shifts with it, forcing the grouping to be rebuilt. Deep Sets escapes both, at the price of summing into a single global vector that carries no spatial address and cannot be evaluated at a requested coordinate. We keep the additive sum but spread it over 128 spatially addressed rows, so a coordinate-conditioned read remains possible without giving up additivity.

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]. These are important precedents: soft spatial assignment, fixed-size encoding, and moment statistics are not introduced by this work. The distinction is the representation contract. The published 3DmFV descriptor is constructed for a complete sample and includes non-additive extrema and normalization; it is not retained as an algebraic state for exact chunk merging, subtraction, decay, and coordinate-conditioned reading after the points are released.

VLAD assigns local descriptors to learned centers and accumulates residuals [4]. NetVLAD makes that assignment soft and differentiable [7], 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 preserves unnormalized pointwise writes as the persistent object and moves density normalization to read time. It also accumulates coordinate–content and second-order terms under continuous coordinate addresses, then uses the same retained matrices for both coordinate-conditioned segmentation and state-only classification. Thus the contribution is not the existence of soft assignment or moments, but the combination of an explicitly composable write with spatially queryable task reads.

2.3 Attention over a fixed latent set

Set Transformer summarizes a set using inducing points and attention [8]. Perceiver IO maintains a latent array whose size is independent of the input and decodes arbitrary outputs, including dense per-element predictions, by attending to that latent with an output query array [9]. Reading a fixed latent with a query in order to produce dense output is their contribution, not ours. The difference lies on the encoding side. Softmax cross-attention normalizes over the observed elements, so the latent obtained from two disjoint chunks is not the sum of the two latents; the representation cannot be merged, subtracted, or decayed arithmetically, and no such property is claimed. Our write applies no normalizer over the observed set, so the state remains a sum of independent per-point terms. The meaning of a query also differs: Perceiver queries are learned position embeddings, whereas ours is a coordinate in the same space as the data, so one state can be evaluated at positions that were never observed.

2.4 Bounded recurrent state

Kernelized attention can be evaluated recurrently with a fixed-size state [10], and fast-weight programmers express such updates as additive outer products or delta corrections [11]. 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 [12]. Serialization makes the hidden state a function of the traversal order, so it is not permutation invariant and two partial states cannot be added. We keep the bounded state but not the sequence: chunk and device equivalence rests on the sum being order free.

2.5 What this architecture refuses to do

Every family above trades one property for another: a maximum gives up invertibility, a metric neighborhood gives up locality of edits, a normalized descriptor gives up additivity, a softmax encoder gives up decomposability, and a serialized recurrence gives up order invariance. The design studied here follows from a single refusal at write time: nothing that couples one point's write to another point may enter the persistent state. That excludes normalization over the observed set, comparison between points, selection, and ordering. Operations that aggregation methods usually perform while absorbing a point — density normalization, centring on a

Table 1: Properties retained by families of set summaries. "Order-free write" means the summary does not depend on the order in which points are absorbed; "additive state" means the summary of a union equals the sum of the summaries; "bounded state" means retained memory does not grow with the number of observed points; "coordinate read" means the summary alone can be evaluated at a requested position.

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

[a]Raw summed components can be additive, but the published descriptors include non-additive normalization and, for 3DmFV, extrema. [b]Softmax over the observed elements couples them. [c]Perceiver IO decodes dense outputs from learned position queries, not from a coordinate in the input space. [d]The outer-product update is itself order free, although these models are applied to ordered sequences.

reference position, and nonlinear interpretation — are instead deferred to the read, where they no longer touch the algebra of the stored matrices. Table 1 states the resulting position.

What that refusal provides and what it costs are both measured rather than asserted. Section 4.2 isolates the contribution of the parallel-state structure, and Tables 3 and 4 place the resulting accuracy against hierarchical baselines.

We evaluate classification on ModelNet40 [13] and part segmentation on ShapeNetPart [14], which is derived from ShapeNet [15].

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.

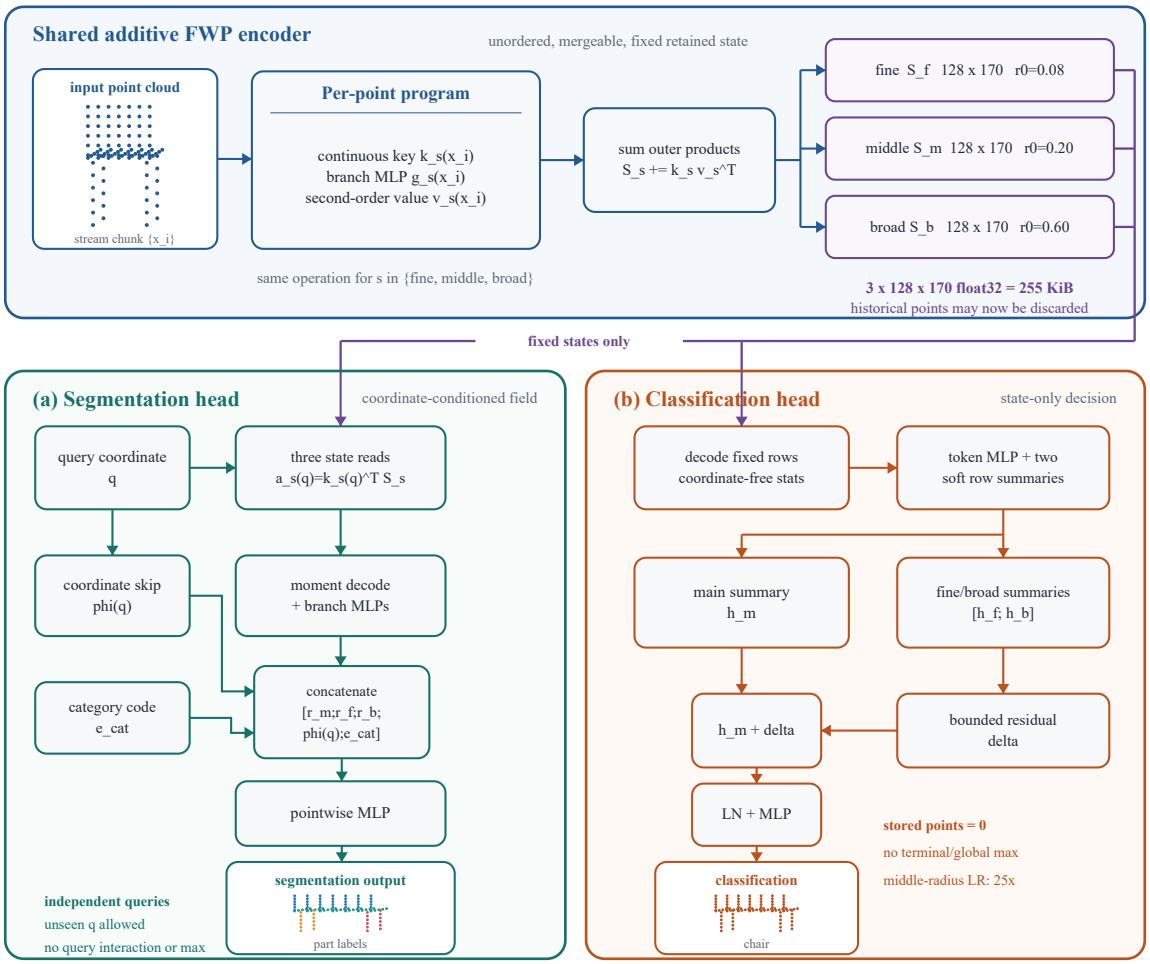

Figure 1: PointProgrammerNet turns an input point cloud or stream chunk into three fixed additive states. Each point independently writes a coordinate-addressed second-order value; the state therefore remains mergeable after historical points are discarded. The segmentation head renders part labels by coordinate-conditioned reads, while the classification head produces an object label from the state alone. Neither head reconstructs or globally pools a decoded point set.

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

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

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. A point changes state row j by

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

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 [11]. PointProgrammer-Net 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.

The complete streaming workflow is therefore short. For each arriving chunk, 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 persistent states, and (iv) releases all chunk-dependent tensors. 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 descriptor deliberately combines complementary statistics. The feature mean supplies local semantic content, the relative centroid and geometric covariance describe local position and shape, and the log mass records how much evidence supports the read. The coordinate–feature term preserves how content changes around the query, while the feature standard deviation exposes mixtures and boundary variation that a mean alone 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. Historical features can be deleted; only coordinates that will later be labelled must remain. The same fixed state can also be queried at an unobserved coordinate.

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 provide complementary fixed-cost views of each state instead of collapsing every row through one average. Using the middle branch as an initially stable reference and admitting the fine and broad summaries through a zero-started bounded residual prevents the three branches from perturbing one another before each has learned a useful summary; the residual can then grow when the auxiliary evidence improves classification.

3.3 Streaming properties

For arbitrary point multisets A and B, Equation (5) gives

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

Finite sums also make the state invariant to point order. Thus separately encoded chunks, devices, or branches of an aggregation tree can be merged by addition. If a removable subset state is available, it can be subtracted; exponential forgetting uses $\mathbf{S}_{s,t} = \rho\mathbf{S}_{s,t-1} + \mathbf{S}_s(X_t)$. Exact distributed equivalence requires common trained weights, coordinate normalization, and reference frame.

Let $F_s(\boldsymbol{x}) = \boldsymbol{k}_s(\boldsymbol{x})\boldsymbol{v}_s(\boldsymbol{x})^\top$ be one point's rank-one FWP write, so that $\mathbf{S}_s(X) = \sum_{\boldsymbol{x}\in X} F_s(\boldsymbol{x})$.

Proposition 1 (Order, chunks, and devices)**.** *For point multisets A, B, $\mathbf{S}_s(A \uplus B) = \mathbf{S}_s(A) + \mathbf{S}_s(B)$, and the state is invariant to point order. 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 claim explicit, suppose device d observes point multiset $X^{(d)}$ and all devices use the same encoder. A central node forms

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

The equality follows by expanding both sides into the same sum of per-point outer products. It holds whether the partial states arrive simultaneously, asynchronously, or through a tree of intermediate aggregators. Classification then uses only the merged matrices. Segmentation

additionally receives the coordinates to be queried and the object category code. Raw point features need not leave the sensing devices.

The same algebra gives reversible state operations when the required partial state is available. If Y is a known subset, then $\mathbf{S}_s(X \setminus Y) = \mathbf{S}_s(X) - \mathbf{S}_s(Y)$. This removes the aggregate contribution of Y without replaying X. Multiplication by $\rho \in [0, 1]$ before an append implements exponential forgetting. These statements concern the representation itself; subtraction cannot recover individual points from a state when their separate subset state was not retained.

Proposition 2 (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 history length. 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. □

To describe a gradually arriving observation, scale its write by evidence strength $t \in [0, 1]$:

$$\mathbf{S}_s(t) = \mathbf{S}_s^{\text{old}} + t\, \boldsymbol{k}_s(\boldsymbol{x}) \boldsymbol{v}_s(\boldsymbol{x})^\top. \tag{16}$$

The state is affine in t. Gaussian addresses, softmax, normalized moment reads with a positive floor, affine layers, BatchNorm or LayerNorm, ReLU or GELU, and stabilized square roots are continuous. Their composition makes the classification logits and fixed-coordinate segmentation logits continuous and piecewise smooth in t. The spatial address is also continuous in the query coordinate $\boldsymbol{q}$, so a fixed state defines a continuous, piecewise-smooth segmentation field. This does not mean a hard argmax label changes continuously: it can switch when two logits cross. It also does not establish calibration or monotonic improvement as evidence arrives.

For the released configuration, $R_s = 128$ and $D_s = 170$ for three branches, so the persistent representation contains 65,280 numbers. In float32 this is 261,120 bytes (255 KiB) per stream. Temporary tensors scale with the arriving chunk and can be released immediately. A segmentation application may retain a small set of coordinates of interest, but those coordinates are external queries rather than part of the historical state. A classification application retains no coordinates at all.

The additive rule treats all frames as exchangeable evidence. When frame order itself should alter memory, a future extension can apply a frame-level fast-weight delta update [11]:

$$\mathbf{S}_{s,t} = \rho \mathbf{S}_{s,t-1} + \eta_t \boldsymbol{K}_{s,t}^\top (\boldsymbol{V}_{s,t} - \boldsymbol{K}_{s,t} \mathbf{S}_{s,t-1}). \tag{17}$$

Here $\boldsymbol{K}_{s,t}$ and $\boldsymbol{V}_{s,t}$ package all keys and values in frame t, ρ retains old state, and η_t is an update rate. Reordering points within a frame multiplies both matrices on the left by the same permutation P; the correction is unchanged because $P^\top P = I$. The extension therefore preserves within-frame point-order invariance while introducing ordered state transitions between frames. It is a proposed direction and is not used in the reported experiments because it intentionally gives up exact additivity across frames.

3.4 Parallel evaluation

The refusal stated in Section 2.5 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 at once and the chunk update is the single matrix product of Equation (5). Between chunks and between devices, Proposition 1 makes the combination a sum, and sums are associative and commutative: partial states can be produced concurrently and reduced in any order, including a tree of depth $\log D$ over D producers, with no synchronization and no fixed schedule. The three branches share no term during the write, so their addresses and values form three independent matrix products that can be issued concurrently or batched. Reading behaves the same way,

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

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
Parameter-matched 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

since the segmentation head acts on each query separately and never pools over decoded queries; a set of requested coordinates is again one matrix product followed by pointwise networks, and the queries can be divided across devices without any exchange.

One detail makes the first of these possible. The address in Equation (2) does normalize, but over the 128 rows of its own branch and not over the observed set, so the denominator stays inside a single point's computation and never becomes a reduction across points. The critical path of encoding a chunk then does not grow with the number of points in it, which is also why the measured incremental latency in Section 4.3 stays flat while reconstruction time grows with history.

Several common constructions do not admit this layout. A metric neighborhood requires a data-dependent search over the points currently held; the point set must be resident and the gather is irregular. A recurrence over serialized points is sequential along the sequence dimension and its result depends on the traversal, so partial results cannot be produced independently and then merged. A normalizer taken over the observed set, as in softmax attention, is itself a reduction across points, and so a synchronization point inside the write.

4 Results

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. 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 4.5 motivates this robustness-oriented choice. All reported PointProgrammerNet variants use three 128×170 states and no global maximum over input or decoded points. Our compact PointNet++ implementations are controlled local baselines rather than claims of canonical reproduction.

PointProgrammerNet is 0.46 points lower in point accuracy and 1.30 points lower in instance mIoU than the lightweight PointNet++ control, while using 40.3% fewer parameters.

The state-only classifier reaches 86.59% clean accuracy and 82.47% under the density shift. Table 4 places these against both controls and quantifies what it costs to discard the point list

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

and classify only from mergeable states.

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 are short screens on a data subset, reported as a controlled comparison and not as final accuracy.

4.3 Additivity and update cost

Across 32 objects, states built from 2–32 ordered or shuffled chunks agree with one-shot construction to relative error below 5.3×10^{-7}. Append, merge, removal, and decay tests remain below 9.8×10^{-7}. The three float32 states contain 65,280 scalars (255 KiB), independent of stream length. Raw XYZ is smaller for short clouds — the measured crossover is about 21,800 observations — and the benefit is bounded retained memory together with direct state operations over long or distributed streams.

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. Below roughly 512 accumulated points the two paths cost the same and rebuilding is marginally faster, so the advantage is asymptotic rather than universal.

4.4 State algebra in practice

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}, respectively, and all maxima were below 9.8×10^{-7}. The remaining discrepancy is consistent with float32 accumulation order. After 16 frames, ordinary addition produced state norm 14.51; decay with $\rho = .9$ limited the norm to 7.42. This experiment validates the state equations and does not claim temporal-task accuracy.

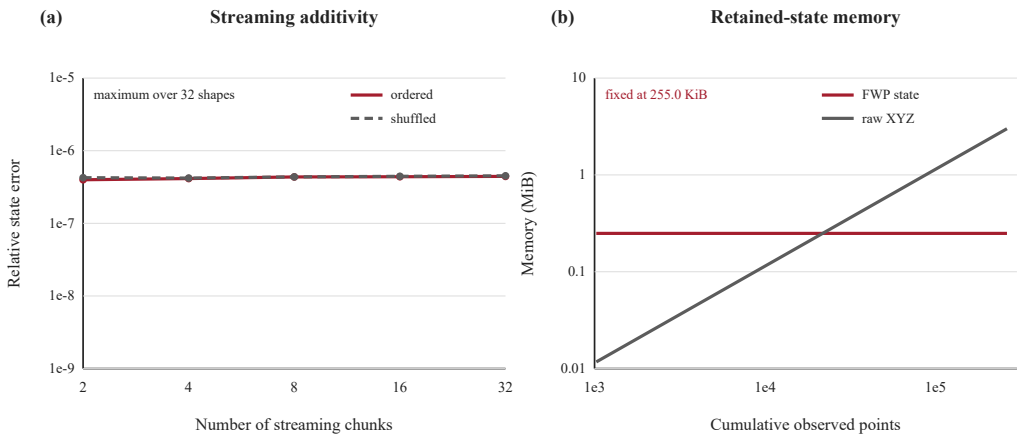

Figure 2: Additive streaming and fixed retained memory. Ordered and shuffled chunk writes reproduce the single-batch FWP state up to floating-point accumulation error. The three 128×170 float32 states occupy 255 KiB regardless of stream length, whereas stored XYZ coordinates grow linearly with the number of observations.

Table 6: 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

4.5 Controlled screens for the final heads

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 6 reports paired changes of the retained design relative to the named alternative.

Equal $170 \rightarrow 160 \rightarrow 160$ segmentation decoders improve instance mIoU by 0.30 ± 0.30 points and remove 24,480 parameters. Distinct radius initialization improves instance mIoU by 0.74 ± 0.42. Applying the same normalized address equation to all three branches improves instance and category mIoU by 0.35 ± 0.16 and 0.85 ± 0.37 over a mixed-key variant. For state-only classification, the middle-reference residual improves clean and density-shift accuracy by 2.67 ± 1.60 and 3.78 ± 1.71 over direct concatenation. Increasing only the middle-radius learning rate gives a variable clean-accuracy change but a consistent 0.91 ± 0.33 shifted-accuracy gain; it is retained for that consistent gain rather than for its clean-accuracy effect. These results motivate the released heads while keeping the state-writing rule symmetric across branches.

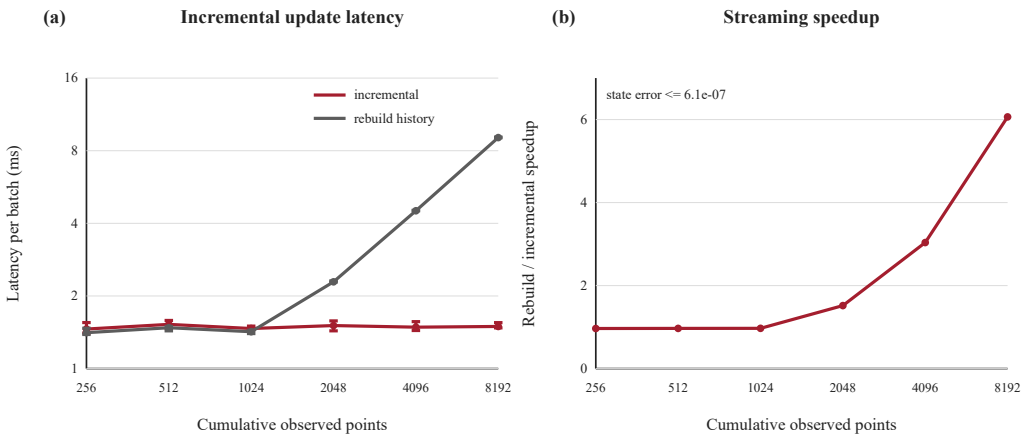

Figure 3: FWP state-construction cost for device-resident point streams. (a) An incremental update processes one new 256-point frame, whereas reconstruction processes every accumulated point. Values are median wall-clock latency over seven interleaved trials of 30 repetitions for four streams; whiskers show the observed range. (b) The speedup grows with history length, while incremental and reconstructed states remain numerically equivalent.

5 Discussion and Limitations

The intended setting is a growing or distributed point stream. A sensor can encode a frame, merge its state, and release the frame. Several cameras, LiDARs, robots, or edge processors can encode disjoint spatial regions or time windows locally and send fixed states to a server. Addition then produces the same representation as centralized encoding, so aggregation can be asynchronous and hierarchical. Raw point sets need not leave the sensing devices. A segmentation service can retain only coordinates it plans to query, while a classifier updates its evidence without replaying earlier points. None of these operations compares a new point with all previously decoded points.

5.1 Worked distributed inference

Consider several LiDAR 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 represents a long accumulated stream.

For classification, the server passes the merged triple directly to the state-row readout. A new device or frame changes the class evidence by adding one more triple, without decoding and pooling the earlier clouds again. For segmentation, the server supplies a coordinate $\boldsymbol{q}$ to the three matrix reads. Queries may be the coordinates of selected observed points, a region requested by an operator, or positions on a regular grid. Only these coordinates need to be available at read time. Because Equation (6) depends smoothly on $\boldsymbol{q}$, nearby queries receive smoothly changing state evidence, including at coordinates that were not explicitly observed. The current experiments evaluate labels only at observed ShapeNetPart points, so accuracy at new coordinates remains an open empirical question.

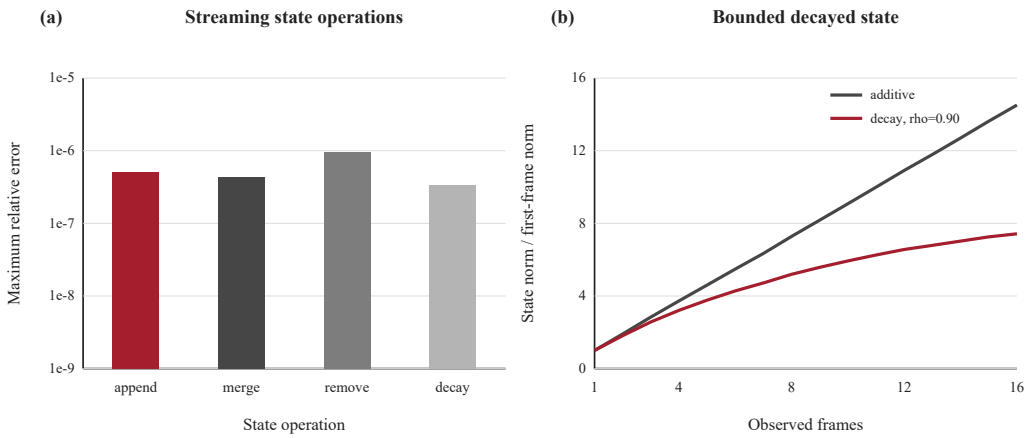

Figure 4: Streaming state operations. (a) Maximum relative errors for append/merge, frame removal, and exponential decay, measured against independently reconstructed FWP states on 32 ShapeNetPart objects. (b) Ordinary addition increases the state magnitude with the number of frames, while exponential decay limits the influence of older frames without changing state size.

5.2 What the fixed state retains and discards

The fixed states are summaries rather than compressed point archives. They are sufficient to evaluate the learned heads, but they are not designed to reconstruct every original observation. Two point sets can produce similar moment tables, especially when many spatial addresses overlap. The learned centers, radii, content networks, and second-order terms determine which distinctions survive this compression. This explains both the bounded-memory benefit and the remaining accuracy gap to a neighborhood method that retains or rebuilds detailed point relations.

More precisely, once a point has been written the state is the only thing any later stage can see, so no decoder can restore a distinction that the write has already discarded. The state, and not the readout, is the binding compression bottleneck of the architecture. The screens support this directly. 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 one point. This is why the design spends its structure on how points are written – the normalized address, the second-order value, and the use of several parallel states – rather than on how the state is decoded, and it is the reason a wider main state is a poor substitute for the parallel structure.

PointNet++ forms neighborhoods around observed points and PointNet preserves channelwise extremes, whereas PointProgrammerNet retains finite moment tables. The state also exceeds bare-XYZ storage for short clouds; its benefit appears when bounded communication, merging, or long streams justify the fixed cost. Distributed equivalence requires identical preprocessing and a shared coordinate frame, and segmentation still reads the state once per requested coordinate. Evaluation is limited to XYZ input, one dataset per task, one timing GPU, and local controlled baselines. Unseen-query accuracy, temporal calibration, additional attributes, and delta adaptation remain future work.

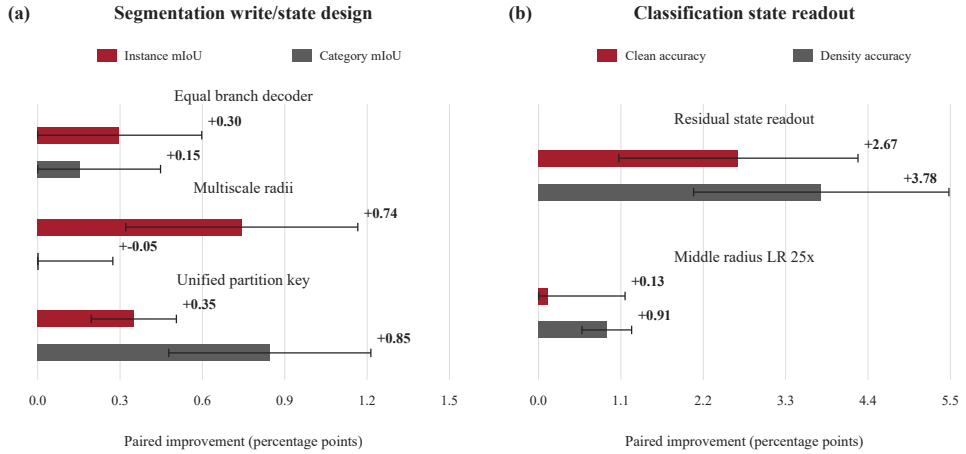

Figure 5: Three-seed, five-epoch architecture screens. Bars show paired mean changes and whiskers show population standard deviations. Segmentation tests equal branch decoders, distinct radius initialization, and a common normalized spatial address. Classification tests residual versus concatenated branch summaries and the middle-radius learning rate.

6 Conclusion

PointProgrammerNet casts point-cloud recognition as persistent-state inference rather than batch descriptor construction. Its methodological contribution is the separation of an independently additive write from normalized, nonlinear task reads: continuous coordinate addresses and second-order values create fixed matrices that can be merged, removed, or decayed before either coordinate-conditioned segmentation or state-only classification. The resulting state is order free, bounded, spatially queryable, and independent of history length at update time. Parallel states retain more usable structure than a single state at the same parameter budget, while the experiments make the resulting accuracy–operability trade-off explicit. This provides a compact, testable representation for centralized and distributed streaming point-cloud processing without claiming novelty for soft assignment, moment statistics, or fast-weight updates in isolation.

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 and consent. Not applicable. This study uses public non-human point-cloud benchmarks and involves no human participants, personal data, clinical trial, or animal experiment.
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 availability. ShapeNetPart and ModelNet40 are public datasets available from the original sources cited in this article. The accompanying reproducibility package contains the 341,909-parameter segmenter and the 529,426-parameter classifier, dataset loaders, training

loops, state-operation checks, and exact three-seed reference values; its `README.md` specifies the environment, dataset layout, commands, and streaming API. 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] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *NeurIPS*, volume 30, pages 3391–3401, 2017.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.