TASK-RELEVANT NULL-SPACE RESIDUALS FOR NON-INJECTIVE NEURAL MAPPINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

Non-injective mappings in neural networks map distinct inputs to the same representation, thereby implicitly inducing equivalence relations in the input space. However, the input differences eliminated by these mappings may still be required by downstream tasks, creating a mismatch between operator-induced indistinguishability and task-required distinctions. For non-injective linear operators realized in the current forward pass, their null spaces exactly characterize these invisible input variations. We propose Task-Relevant Null-Space Residuals (NSR), a general residual framework for non-injective linear mappings. NSR combines null-space component extraction from pre-mapping representations, member-level encoding and gating, and application-specific integration to exploit potentially task-relevant information under downstream supervision while preserving the original aggregation or merging rules. We evaluate NSR in two structurally different settings: token merging and graph aggregation. In token merging, NSR achieves higher semantic segmentation performance than the corresponding compressed baselines in 34 out of 36 evaluated configurations, with a maximum observed gain of 31.51 mIoU points under strong compression. In graph aggregation, NSR achieves 100% training accuracy on Tree-NeighborsMatch at depths $d = 2$–6 across three backbones, alongside gains on heterophilic node classification and molecular graph regression. Together, these results support null-space residuals as a practical complement to non-injective linear mappings, enabling downstream models to learn from input distinctions invisible in the original operator's output.

1 INTRODUCTION

Many representation transformations in neural networks are non-injective: distinct inputs can be mapped to the same output, making them indistinguishable to any subsequent computation that depends only on that output. Such indistinguishability does not necessarily hinder the task; when these inputs require the same prediction, ignoring their differences is appropriate. The issue is that inputs identified by the mapping may still correspond to different task targets Jacobsen et al. (2018a). Therefore, a non-injective mapping not only transforms representations but also determines, through its structure, which inputs are indistinguishable in its output. These operator-induced input equivalences need not be compatible with downstream task requirements (Fig. 1). Making a distinction invisible to an operator does not mean that the task no longer requires it.

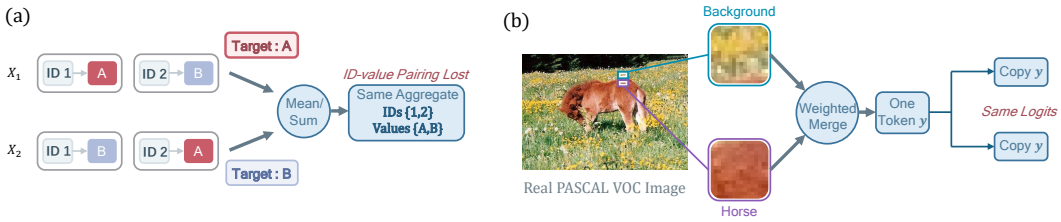


Figure 1: Operator-induced indistinguishability need not align with downstream task requirements: distinct inputs may produce the same mapped representation while requiring different task outputs.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

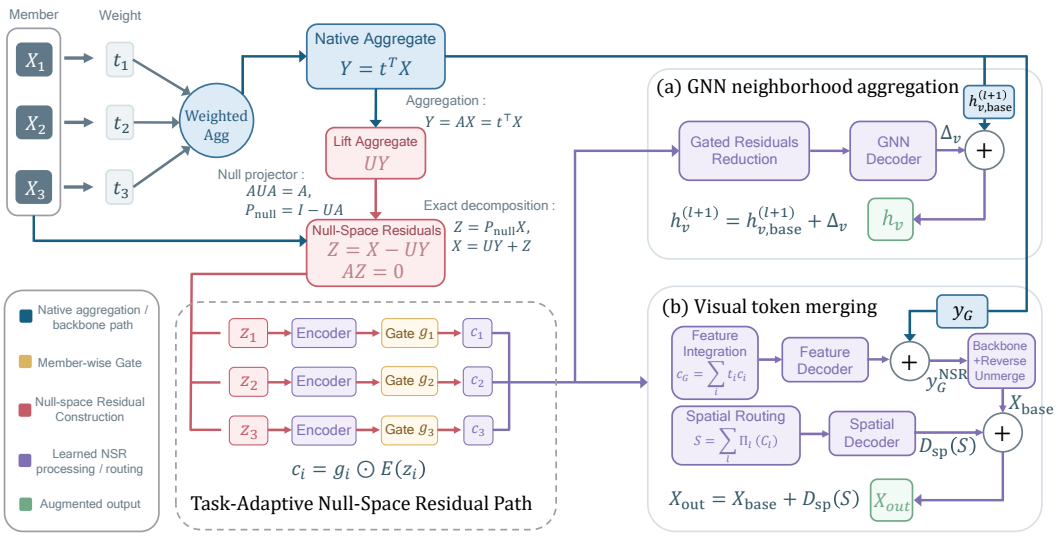


Figure 2: Overview of NSR. Null-space residuals are extracted from pre-mapping representations based on the current linear operator and transformed into usable complementary information through a learnable branch under task supervision. The original mapping branch retains its computation rule.

Starting from this general problem, we focus on non-injective linear mappings, including computational steps in adaptive operations that take a linear form once the current member groups and weights have been determined. The key reason for this scope is that, for a fixed linear operator, two inputs produce the same output if and only if their difference lies in its null space. Thus, the null space provides an exact description of input variations invisible to the operator. For operations with input-dependent grouping or weights, this characterization is conditioned on the linear operator realized in the current forward pass. Linear neighborhood aggregation in graph neural networks and weighted token merging with the current groups and weights fixed are two structurally different instances of this setting Kipf & Welling (2016); Bolya et al. (2023).

This perspective leads to the question studied in this work: Can we retain the original mapping rule while providing the model with a separate pathway to learn to exploit input variations invisible in the original operator's output?

To this end, we propose Task-Relevant Null-Space Residuals (NSR), a general residual framework for non-injective linear mappings, with its overall structure shown in Fig. 2. NSR uses the current operator to explicitly extract null-space components from pre-mapping representations as candidate complementary signals. Within a task-supervised complementary pathway, residual encoding, member-wise gating, and application-specific integration jointly transform these components into task-adaptive updates, enabling downstream models to exploit distinctions invisible in the original operator's output. In graph aggregation and token merging, NSR instantiates this pathway while preserving the original aggregation or merging rules. Our main contributions are as follows:

- **Operator–task mismatch.** We characterize the mismatch between the input equivalence classes induced by the currently realized non-injective linear operator and the distinctions required by downstream tasks, providing an operator-level basis for constructing complementary information.
- **Null-space residual framework.** We propose NSR, a general task-supervised residual framework for non-injective linear mappings. The framework is not limited to null-space extraction but comprises operator-defined null-space component extraction, member-level encoding and gating, and application-specific integration.
- **Experimental evidence across structures and tasks.** Token-merging experiments cover three compression methods, three semantic-segmentation datasets, and multiple compression levels, with NSR achieving higher mIoU than the corresponding compressed baselines in 34 of 36 evaluated configurations. In graph aggregation, the complete NSR pathway

achieves 100% training accuracy on Tree-NeighborsMatch at depths $d = 2$–6 across three backbones, alongside gains on heterophilic node classification and molecular graph regression.

2 Related Work

Representation invariance and null-space methods. Invariance to input variations does not always align with task requirements. Neural networks can be overly sensitive to task-irrelevant variations while exhibiting excessive invariance to task-relevant changes (Jacobsen et al., 2018a). At the architectural level, i-RevNet maintains input reconstructability through invertible intermediate transformations (Jacobsen et al., 2018b); LiftPool uses an invertible subband decomposition and employs the detail subbands produced during downsampling for subsequent upsampling (Zhao & Snoek, 2021). Null spaces have also been used in learning-based inverse problem solving: null-space networks constrain learned reconstruction corrections to the null space of the forward operator to preserve data consistency (Schwab et al., 2019). NSR retains the existing non-injective main mapping without requiring it to be made invertible; its null-space components are extracted directly from accessible pre-mapping representations rather than inferred from mapped outputs. Here, the null space defines the source of candidate complementary signals, while a separate learnable branch learns how to exploit this information under downstream task supervision.

Graph aggregation and residual connections. Neighborhood aggregation is a core operation through which message-passing GNNs construct node representations (Gilmer et al., 2017). GCN uses degree-normalized weighted summation (Kipf & Welling, 2016); GraphSAGE introduces several neighborhood aggregation schemes, with its mean variant combining the neighborhood mean with the central node representation (Hamilton et al., 2017); GIN uses sum aggregation and a multilayer perceptron, relating the discriminative power of multiset aggregation to the expressive power of GNNs (Xu et al., 2019). More generally, Deep Sets studies permutation-invariant functions through shared element-wise transformations and aggregation (Zaheer et al., 2017). PNA combines multiple aggregation statistics with degree-dependent scalers to enrich neighborhood representations (Corso et al., 2020). Another line of work focuses on the propagation and combination of representations across layers: Jumping Knowledge combines node representations from different propagation depths (Xu et al., 2018), while GCNII supports deeper graph convolutional networks through initial residual connections and identity mappings (Chen et al., 2020). NSR differs in how its complementary signal is defined: it retains the backbone's original aggregation branch and uses the null space of the current local aggregation operator to extract operator-invisible member-wise residuals from pre-aggregation representations. These residuals are then transformed into node updates through task-adaptive processing.

Token merging in vision transformers. ViT represents an image as a sequence of embedded image patches (Dosovitskiy et al., 2021). Token merging summarizes multiple tokens into representative tokens to shorten the sequence used in subsequent computation. ToMe progressively merges similar tokens using lightweight matching (Bolya et al., 2023); PiToMe introduces an energy score to select merging candidates, prioritizing the preservation of relatively distinctive or isolated tokens (Tran et al., 2024); ALGM adopts a local-then-global merging strategy (Norouzi et al., 2024); DTEM uses a separate learnable embedding module to produce dedicated representations for token matching (Lee & Hong, 2024); MPM performs mean merging over mutually nearest-neighbor token pairs (Ravé et al., 2026). For dense prediction tasks such as semantic segmentation, merged representations also need to be mapped back to their original spatial positions. ALGM and MPM copy merged representations back to member positions using recorded merge correspondences (Norouzi et al., 2024; Ravé et al., 2026). This operation restores the original token count and spatial arrangement, but copying itself does not restore the original representational differences among members of the same group. NSR focuses on member-wise distinctions eliminated by weighted merging once the current groups and weights have been determined, while preserving the original selection, matching, and weight-computation rules. The same null-space residual source is used through feature and spatial paths to provide complementary information for subsequent token computation and dense prediction at the original positions, respectively.

3 NULL-SPACE RESIDUALS

We characterize operator-induced indistinguishability, construct complementary null-space residuals, and instantiate their task-supervised processing for graph aggregation and token merging.

3.1 OPERATOR-INDUCED INDISTINGUISHABILITY AND TASK RELEVANCE

Let $F : \mathcal{X} \to \mathcal{C}$ map inputs in $\mathcal{X}$ to representations in $\mathcal{C}$. If distinct inputs $x_1, x_2 \in \mathcal{X}$ satisfy $F(x_1) = F(x_2)$, any subsequent computation depending only on $F(x)$ cannot distinguish them. This becomes a limitation when the downstream task requires that distinction; not all collapsed distinctions are task-relevant.

We focus on linear mappings and operations that take a linear form once the current groups and weights are fixed. For a fixed linear operator A and inputs x_1, x_2 in its domain,

$$Ax_1 = Ax_2 \quad \Longleftrightarrow \quad x_1 - x_2 \in \ker(A), \tag{1}$$

where $\ker(A)$ is the null space of vectors mapped to zero by A, exactly characterizing operator-invisible input variations. For input-dependent grouping or weights, all null-space statements are conditioned on the operator A realized in the current forward pass.

3.2 NULL-SPACE COMPLEMENTARITY

Let $X \in \mathbb{R}^{n\times d}$ contain n member representations of dimension d, and let $A \in \mathbb{R}^{k\times n}$ denote the current linear operator, with output $Y = AX$. We choose a linear lift $U \in \mathbb{R}^{n\times k}$ satisfying $AUA = A$ Ben-Israel & Greville (2003), and define the null-space component as

$$Z = (I_n - UA)X, \tag{2}$$

where I_n is the $n \times n$ identity matrix and U maps the output representation back to the member space.

Proposition 1 (Null-space complementarity)**.** *The component defined above satisfies*

$$AZ = 0, \qquad X = UY + Z. \tag{3}$$

Proof. Since $AUA = A$, we have $AZ = (A - AUA)X = 0$. Moreover, the definition directly gives $UY + Z = UAX + (I_n - UA)X = X$. □

Thus, every feature channel of Z lies in $\ker(A)$. For given A and U, Y and Z uniquely determine X; any auxiliary signal $H(X)$ can thus be written as $H(UY + Z)$. At fixed Y, member differences are exactly differences in Z. The lift U may depend on the operator and need not induce an orthogonal projection. NSR uses this decomposition to expose operator-invisible differences for task-supervised complementary processing.

3.3 TASK-SUPERVISED COMPLEMENTARY PATHWAY

Let $z_i \in \mathbb{R}^d$ denote the i-th member residual. NSR constructs its code as

$$c_i = g_i \odot E(z_i), \tag{4}$$

where $E : \mathbb{R}^d \to \mathbb{R}^r$ is a residual encoder with code dimension r, $g_i \in \mathbb{R}^r$ is a learnable gate generated from the member residual and its context, and $\odot$ denotes element-wise multiplication.

Member-level processing precedes reaggregation. For a linear transformation $B \in \mathbb{R}^{d\times r}$ shared across members, reaggregation with the original operator gives $A(ZB) = (AZ)B = 0$. Member-dependent gating can alter their relative contributions, so the processed codes need not satisfy the same cancellation constraint.

Residual extraction specifies the source of candidate complementary information, while member-level processing and application-specific integration make this source usable for downstream prediction. Together, these stages constitute the complete NSR method. Depending on the application, the codes are aggregated or spatially routed and mapped to complementary updates, with the branch trained under the downstream task loss. The null-space properties and exact decomposition apply to the complete residual Z before encoding; member and contextual representations may additionally condition its processing through the gates.

3.4 NSR FOR GRAPH AGGREGATION

At layer l, consider the K_v members participating in the current aggregation at node v. The NSR branch generates messages using a linear map shared across members and stacks them row-wise as $M_v \in \mathbb{R}^{K_v \times d}$. Let $t_v \in \mathbb{R}^{K_v}$ denote the aggregation coefficients actually used by the backbone. The corresponding local operator is $A_v = t_v^\top$.

For $t_v \neq 0$, choosing $U_v = t_v/(t_v^\top t_v)$ gives $A_v U_v = 1$. The resulting residual is

$$Z_v = \left(I_{K_v} - \frac{t_v t_v^\top}{t_v^\top t_v}\right) M_v, \qquad t_v^\top Z_v = 0, \tag{5}$$

which projects along the member dimension and extracts message variations invisible to the current local aggregation. The member set and coefficients follow the corresponding backbone: GCN uses normalized adjacency coefficients including self-loops; GraphSAGE uses neighborhood-mean weights, with its separate root path handled by the backbone; and GIN includes neighbor terms and a root term weighted by $1 + \epsilon^{(l)}$.

The graph branch directly processes residuals in the message space, taking E to be the identity map and $r = d$. Let $z_{v,i}$ and $g_{v,i}$ denote the member residual and its gate, respectively. The node update is

$$\Delta_v = F_{\mathrm{out},v}\left(\sum_{i=1}^{K_v} g_{v,i} \odot z_{v,i}\right), \qquad h_v^{(l+1)} = h_{v,\mathrm{base}}^{(l+1)} + \Delta_v. \tag{6}$$

Here, $h_{v,\mathrm{base}}^{(l+1)}$ is the update produced by the original backbone on the current input. The function $F_{\mathrm{out},v}$ includes neighborhood-dependent scaling and an output transformation that maps the residual to the backbone output dimension, with learnable parameters shared across nodes.

3.5 NSR FOR TOKEN MERGING

Consider a merge group $\mathcal{G}$ of K tokens $x_i \in \mathbb{R}^d$, with original merge weights satisfying $\sum_{i\in\mathcal{G}} t_i = 1$. Collecting these weights in t, the current merge operator is $A_\mathcal{G} = t^\top$.

To match copy-based unmerging, we choose $U_\mathcal{G} = \mathbf{1}_K$, where $\mathbf{1}_K$ is the all-ones vector. Since $A_\mathcal{G} U_\mathcal{G} = 1$, Sec. 3.2 gives

$$y_\mathcal{G} = \sum_{i\in\mathcal{G}} t_i x_i, \qquad z_i = x_i - y_\mathcal{G}, \qquad \sum_{i\in\mathcal{G}} t_i z_i = 0. \tag{7}$$

NSR encodes each member residual z_i as $c_i \in \mathbb{R}^r$ following Sec. 3.3 and uses the same codes through feature and spatial residual paths.

Feature residual. NSR aggregates the codes with the original merge weights and adds the decoded correction to the merged representation:

$$c_\mathcal{G} = \sum_{i\in\mathcal{G}} t_i c_i, \qquad y_\mathcal{G}^{\mathrm{NSR}} = y_\mathcal{G} + D_{\mathrm{feat}}(c_\mathcal{G}). \tag{8}$$

Here, $D_{\mathrm{feat}} : \mathbb{R}^r \to \mathbb{R}^d$ decodes the group code into token features. The updated representation continues through the backbone as a single token, allowing within-group variations to affect subsequent transformations.

Spatial residual. Copy-based unmerging cannot recover within-group distinctions. NSR therefore retains member codes and their correspondence to original patch positions.

For L merge events, let $C_l \in \mathbb{R}^{N_l \times r}$ collect the N_l member codes retained at event l. The routing operation Π_l assigns these codes to the N_0 original patch positions using the recorded correspondence. All events share the same code dimension r, giving

$$S = \sum_{l=1}^{L} \Pi_l(C_l), \qquad X_{\mathrm{out}} = X_{\mathrm{base}} + D_{\mathrm{sp}}(S). \tag{9}$$

Here, $S \in \mathbb{R}^{N_0 \times r}$ accumulates spatial codes, and $X_{\mathrm{base}} \in \mathbb{R}^{N_0 \times d}$ is the representation after standard unmerging, including preceding feature-residual updates. At the end of the backbone, D_{sp}

decodes the accumulated codes into d-dimensional position-specific updates; the codes are not processed as an additional token sequence by the Transformer. Both paths preserve the original token-selection, matching, and weight-computation rules, as well as the predefined merge schedule.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Semantic segmentation is evaluated on the official validation splits of Pascal VOC 2012 Everingham et al. (2012), Cityscapes Cordts et al. (2016), and ADE20K Zhou et al. (2017), using mIoU. All models use an ImageNet-pretrained Russakovsky et al. (2015) DeiT-Tiny/16 backbone Touvron et al. (2021) and a linear segmentation head. We compare ToMe Bolya et al. (2023), PiToMe Tran et al. (2024), and MPM Ravé et al. (2026) with their NSR-augmented counterparts, using the uncompressed Full-ViT as a reference.

Graph experiments use GCN Kipf & Welling (2016), GraphSAGE Hamilton et al. (2017), and GIN Xu et al. (2019). Heterophilic node classification uses Roman-empire, Amazon-ratings, Minesweeper, Tolokers, and Questions Platonov et al. (2023b). The first two use accuracy, while the others use ROC-AUC. Tree-NeighborsMatch Alon & Yahav (2021) evaluates training fit on binary trees of depths 2–8. Molecular graph regression uses the $12{,}000$-graph ZINC subset Dwivedi et al. (2023), evaluated by test MAE. Heterophilic node classification and ZINC use four message-passing layers with a hidden dimension of 128; the tree task at depth d uses $d + 1$ message-passing layers with a hidden dimension of 32.

Each Original–NSR comparison uses the same backbone configuration, optimization settings, and training budget. Segmentation models additionally share pretrained parameter initialization, data ordering, and compression-control configurations. Segmentation models are trained for 100 epochs and evaluated using the final checkpoint; heterophilic node classification and ZINC select checkpoints by validation performance. Specific network configurations, preprocessing, and training protocols are provided in Appendix A. The computation-reduction operating point (WP) is defined as the ratio of Full-ViT GFLOPs to those of the original compressed baseline. Computation is estimated analytically from the actual forward structure. Cityscapes computation is reported per input window, corresponding to one window in full-image sliding-window inference. Data processing, complete training configurations, and computational-accounting conventions are provided in Appendix A, while the construction of compression configurations is described in Appendix F.

4.2 RESULTS

4.2.1 TOKEN MERGING

Table 1 reports semantic-segmentation performance and the absolute gain over the corresponding compressed baseline, $\Delta_{\mathrm{NSR}} = \mathrm{mIoU}_{\mathrm{NSR}} - \mathrm{mIoU}_{\mathrm{Base}}$.

Across the 36 evaluated configurations, NSR achieves higher mIoU than the corresponding compressed baseline in 34 cases. The two lower results occur under mild compression and differ from the corresponding baselines by no more than 0.04 points. The observed gains are generally larger under stronger compression, with ToMe showing maximum gains of 31.51 and 20.83 points on Pascal VOC and Cityscapes, respectively. These results indicate that NSR can provide substantial performance recovery across different merging mechanisms, with the magnitude varying across operators and compression configurations. NSR can also outperform a less compressed original model while requiring less computation. On ADE20K, ToMe with NSR at the $3.4\times$ operating point achieves 34.11 mIoU at 3.443 GFLOPs, exceeding the 31.44 mIoU achieved by the original $2.5\times$ ToMe at 4.202 GFLOPs.

4.2.2 GRAPH AGGREGATION

Tree-NeighborsMatch. This task requires the root to predict the value at the leaf associated with a query key, and therefore depends on key–value associations. Consider the first local aggregation after the concatenated key and value features undergo an initial linear embedding: each member representation consists of a key encoding, a value encoding, and a shared bias. Fix two sibling

Table 1: Semantic-segmentation results across merging methods and compression configurations. Bold marks the higher mIoU in each Original–NSR pair. Δ_{NSR} is computed from unrounded mIoU.

			Original		+NSR		
Dataset	Method	WP	GFLOPs ↓	mIoU ↑	GFLOPs ↓	mIoU ↑	Δ_{NSR} ↑
VOC	Full-ViT	1.0×	1.254	64.85	–	–	–
	ToMe	1.8×	0.698	62.81	0.750	**64.23**	+1.43
		2.5×	0.492	44.58	0.533	**59.54**	+14.95
		3.4×	0.373	18.31	0.407	**49.82**	+31.51
		3.8×	0.333	16.98	0.365	**46.23**	+29.25
	PiToMe	1.8×	0.698	61.48	0.736	**62.44**	+0.96
		2.5×	0.501	55.77	0.536	**59.19**	+3.42
		3.4×	0.368	43.70	0.400	**54.02**	+10.31
		3.8×	0.332	37.32	0.362	**50.66**	+13.34
	MPM	1.8×	0.699	61.99	0.714	**62.64**	+0.65
		2.4×	0.512	58.87	0.530	**59.74**	+0.86
		3.0×	0.429	54.93	0.448	**57.63**	+2.70
		3.3×	0.376	51.34	0.395	**54.85**	+3.51
Cityscapes	Full-ViT	1.0×	30.533	71.83	–	–	–
	ToMe	1.8×	16.983	**71.71**	18.697	71.67	-0.04
		2.5×	12.191	65.83	13.633	**70.59**	+4.76
		3.4×	8.984	54.08	10.155	**69.41**	+15.33
		3.8×	8.037	47.33	9.125	**68.16**	+20.83
	PiToMe	1.8×	16.910	69.70	18.118	**69.80**	+0.10
		2.5×	12.213	69.01	13.370	**71.28**	+2.27
		3.4×	8.962	65.77	10.025	**69.00**	+3.23
		3.8×	8.041	64.21	9.062	**68.95**	+4.74
	MPM	1.8×	16.817	70.86	17.098	**71.34**	+0.48
		2.6×	11.918	67.86	12.328	**69.75**	+1.89
		3.0×	10.067	67.21	10.557	**68.40**	+1.19
		3.4×	9.090	65.20	9.624	**68.02**	+2.82
ADE20K	Full-ViT	1.0×	10.463	37.60	–	–	–
	ToMe	1.8×	5.817	36.60	6.347	**37.41**	+0.81
		2.5×	4.202	31.44	4.643	**37.07**	+5.64
		3.4×	3.085	17.67	3.443	**34.11**	+16.44
		3.8×	2.750	12.44	3.082	**33.74**	+21.31
	PiToMe	1.8×	5.801	36.38	6.181	**37.94**	+1.56
		2.5×	4.180	34.78	4.540	**35.95**	+1.17
		3.4×	3.076	31.39	3.404	**35.65**	+4.26
		3.8×	2.757	30.07	3.071	**34.17**	+4.11
	MPM	1.8×	5.804	**37.49**	5.977	37.48	-0.01
		2.4×	4.443	36.59	4.656	**36.93**	+0.34
		3.2×	3.304	33.57	3.454	**35.74**	+2.16
		3.4×	3.055	32.69	3.213	**35.38**	+2.69

leaves with distinct keys, swap two values with distinct encodings in the message space, and keep all other inputs and aggregation coefficients unchanged. If the two leaves have equal coefficients, the aggregate at their parent remains unchanged; however, when the query is fixed to one of these keys, the correct answer changes. This gives an explicit instance of task-relevant local indistinguishability.

Following the task construction and training-fit evaluation protocol of Alon & Yahav (2021), we use Tree-NeighborsMatch to diagnose whether GNNs can fit long-range key–value associations. Training accuracy is the benchmark's intended metric for detecting fitting failures as tree depth increases. We use $d + 1$ message-passing layers on binary trees of depths $d = 2$–8 and adopt random seed 11 from their reference implementation. Each model is trained once at each depth, and we report the highest training accuracy attained during that run. As shown in Figure 3, the original backbones exhibit degraded fitting as depth increases, whereas all three NSR variants achieve 100% training accuracy at $d = 2$–6 and substantially outperform their counterparts at $d = 7, 8$. These results support the complementary pathway's effectiveness in improving training fit.

Heterophilic node classification. In Table 2, NSR increases the reported metric in 13 of the 15 backbone–dataset combinations. The largest gains occur on Roman-empire, where accuracy improves by 10.85, 6.12, and 8.25 percentage points for GCN, GraphSAGE, and GIN, respectively.

Platonov et al. Platonov et al. (2023b) identify Roman-empire as the only one of these five datasets with Label Informativeness (LI) appreciably above zero. LI quantifies how much information a neighbor's label provides about a node's label Platonov et al. (2023a). The larger NSR gains on Roman-empire are consistent with this label structure: preserving fine-grained distinctions compressed by aggregation may be more valuable when neighborhoods contain stronger task-predictive information. Performance changes vary across datasets and backbones, consistent with the premise

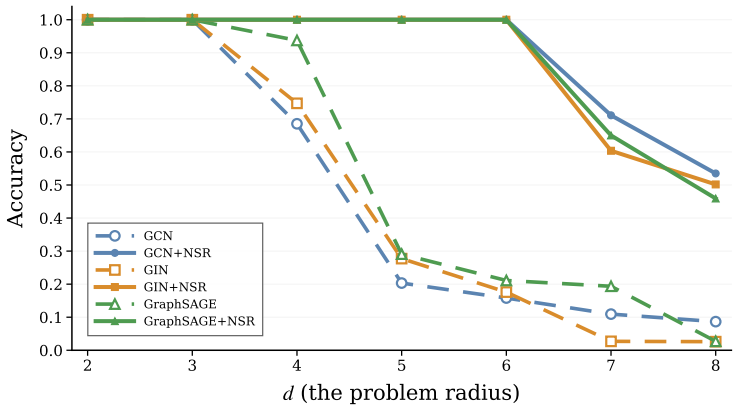


Figure 3: Training accuracy on Tree-NeighborsMatch across tree depths. Dashed and solid curves denote the original GCN, GIN, and GraphSAGE models and their NSR variants, respectively. Each point is the highest training accuracy attained in the corresponding run.

Table 2: Node-classification results on the five heterophilic graph benchmarks. Values are the mean and sample standard deviation (%) over the ten official splits. The higher score within each backbone pair is shown in bold.

Backbone	Variant	Roman-empire Acc. (%) ↑	Amazon-ratings Acc. (%) ↑	Minesweeper ROC-AUC (%) ↑	Tolokers ROC-AUC (%) ↑	Questions ROC-AUC (%) ↑
GCN	Original	72.05 ± 0.64	49.41 ± 0.57	90.20 ± 0.66	$\mathbf{84.90 \pm 0.90}$	75.32 ± 1.35
	+NSR	$\mathbf{82.90 \pm 0.71}$	$\mathbf{49.67 \pm 0.57}$	$\mathbf{92.09 \pm 0.63}$	84.36 ± 0.73	$\mathbf{78.25 \pm 1.21}$
GIN	Original	74.00 ± 0.79	49.56 ± 0.55	86.90 ± 0.57	83.06 ± 0.88	74.81 ± 1.57
	+NSR	$\mathbf{82.25 \pm 0.65}$	$\mathbf{50.47 \pm 0.57}$	$\mathbf{89.41 \pm 1.09}$	$\mathbf{83.18 \pm 0.87}$	$\mathbf{75.06 \pm 1.40}$
GraphSAGE	Original	81.59 ± 0.55	52.33 ± 0.37	$\mathbf{93.37 \pm 0.40}$	83.28 ± 0.56	74.82 ± 0.91
	+NSR	$\mathbf{87.71 \pm 0.61}$	$\mathbf{52.72 \pm 0.75}$	92.75 ± 0.48	$\mathbf{84.50 \pm 0.63}$	$\mathbf{75.07 \pm 2.16}$

that residual utility depends on task relevance and the model's ability to exploit it. LI is defined from graph labels and serves as auxiliary context here, rather than a direct measurement of the hidden-feature residuals processed by NSR.

Label-based null-space diagnostic. We further examine the relationship between NSR gains and label structure in the aggregation null space on Roman-empire. We project member one-hot label encodings onto the null space of the corresponding local aggregation operator, compute normalized projection energies, and average them over the target node's computation tree with weights given by local member counts. This yields a node-level diagnostic score. The score is computed after training and is not used for training, hyperparameter selection, or model selection. Its full definition is provided in Appendix E.

Figure 4 shows that the original backbones' mean accuracy generally decreases across higher score bins, while the NSR variants remain relatively stable, producing an overall widening accuracy gap. This association is consistent with the motivation of learning to exploit aggregation-invisible distinctions. The corresponding gain curves are provided in Appendix E.

ZINC graph regression. To test whether NSR can serve as a more general mechanism for compensating information discarded by aggregation, we evaluate graph-level molecular property regression on ZINC. This setting differs from the preceding experiments in data semantics, prediction granularity, and learning objective: the model predicts a continuous molecular property through repeated local aggregation and graph-level readout. To keep aggregation backbones consistent across graph tasks, we retain the original node-feature-based formulations of GCN, GraphSAGE, and GIN, using only atom types and molecular connectivity without bond-type attributes.

Table 3 shows that NSR lowers the mean test MAE of all three backbones, with absolute reductions of 0.1785, 0.0360, and 0.0542 for GCN, GIN, and GraphSAGE, respectively. This supports the

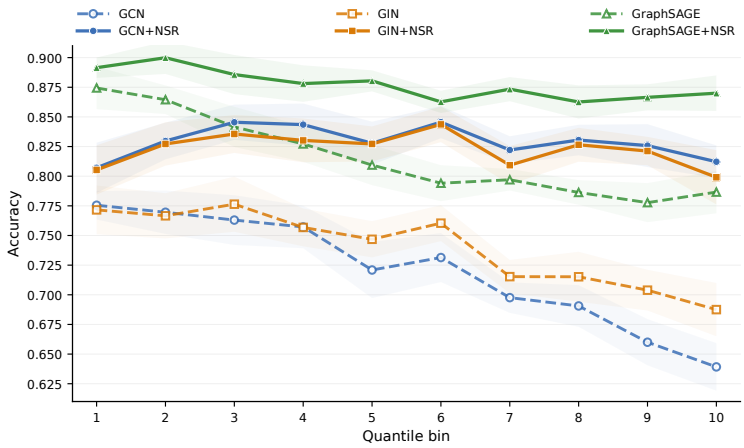


Figure 4: Within each official split, test nodes are divided into ten approximately equal-frequency bins, with the horizontal axis ordering the bins from low to high scores. Lines show the mean accuracy of the corresponding bin across the ten official splits, and shaded bands of the same color indicate the mean $\pm$ one sample standard deviation.

Table 3: Test MAE on the ZINC subset. Values are the mean and standard deviation over ten random seeds; lower values are better.

Backbone	Original	+NSR
GCN	0.4732 ± 0.0067	$\mathbf{0.2947 \pm 0.0050}$
GIN	0.3451 ± 0.0093	$\mathbf{0.3091 \pm 0.0093}$
GraphSAGE	0.4373 ± 0.0086	$\mathbf{0.3831 \pm 0.0125}$

effectiveness of the complete NSR pathway beyond matching and node classification, extending to graph-level regression.

These graph experiments provide complementary evidence: Tree-NeighborsMatch examines fitting under explicit local mismatch; the heterophily experiments extend evaluation to real graph data, with the label-based diagnostic examining how predictive gains relate to label structure; and ZINC extends validation to graph-level regression. Together, they support using null-space residuals to complement the original aggregation.

Overall, experiments across token merging and graph aggregation support the practical value of NSR in exploiting operator-invisible member distinctions to improve task performance while preserving the original mapping rules. Comparisons and ablations in both settings provide further evidence for the framework's design (Appendix G).

5 Conclusion

Motivated by the potential mismatch between operator-induced input equivalences and downstream task requirements, we propose Task-Relevant Null-Space Residuals (NSR), a general residual framework for non-injective linear mappings. NSR combines operator-defined null-space residual extraction, member-level encoding and gating, and application-specific integration into a task-supervised complementary pathway while preserving the original aggregation or merging rules. The null space of the currently realized linear operator defines candidate complementary information, while downstream supervision guides its use. Experiments in graph aggregation and token merging demonstrate improvements across multiple models and configurations. These results support a practical design principle: for many-to-one neural computation, explicitly providing access to operator-invisible variations has practical value.

AI USE STATEMENT

We used generative AI tools for language polishing and translation, and to assist with writing and debugging portions of the code. The authors reviewed the AI-assisted text and revised it as needed, and verified the relevant code through manual inspection and experimental execution. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Section 3 defines NSR and its application-specific constructions for token merging and graph aggregation, with the associated assumptions and derivations provided in Appendix B. Appendix A documents the model implementations, residual-branch architectures, training and evaluation protocols, and computational-accounting conventions. The construction and configurations of the token-merging operating points are specified in Appendix F, and numerical verification of the token-merging residuals is provided in Appendix C. Appendix E specifies the label-based diagnostic and binning procedure.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

Adi Ben-Israel and Thomas N. E. Greville. *Generalized Inverses: Theory and Applications*. Springer, 2 edition, 2003. doi: 10.1007/b97366.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token Merging: Your ViT but Faster. In *International Conference on Learning Representations*, 2023.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735. PMLR, 2020.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes challenge 2012 (VOC2012) results. PASCAL Visual Object Classes Challenge, 2012.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Jörn-Henrik Jacobsen, Jens Behrmann, Richard Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. *arXiv preprint arXiv:1811.00401*, 2018a.

Jörn-Henrik Jacobsen, Arnold W. M. Smeulders, and Edouard Oyallon. i-RevNet: Deep invertible networks. In *International Conference on Learning Representations*, 2018b.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Dong Hoon Lee and Seunghoon Hong. Learning to Merge Tokens via Decoupled Embedding for Efficient Vision Transformers. In *Advances in Neural Information Processing Systems*, volume 37, pp. 54079–54104, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Narges Norouzi, Svetlana Orlova, Daan de Geus, and Gijs Dubbelman. ALGM: Adaptive Local-then-Global Token Merging for Efficient Semantic Segmentation with Plain Vision Transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Oleg Platonov, Denis Kuznedelev, Artem Babenko, and Liudmila Prokhorenkova. Characterizing graph datasets for node classification: Homophily-heterophily dichotomy and beyond. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *International Conference on Learning Representations*, 2023b.

Simon Ravé, Pejman Rasti, and David Rousseau. MPM: Mutual Pair Merging for Efficient Vision Transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition – Findings Track (CVPRF)*, 2026.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Johannes Schwab, Stephan Antholzer, and Markus Haltmeier. Deep null space learning for inverse problems: Convergence analysis and rates. *Inverse Problems*, 2019. doi: 10.1088/1361-6420/aaf14a.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 2021.

Hoai-Chau Tran, Duy M. H. Nguyen, Duy M. Nguyen, TrungTin Nguyen, Ngan Le, Pengtao Xie, Daniel Sonntag, James Zou, Binh T. Nguyen, and Mathias Niepert. Accelerating Transformers with Spectrum-Preserving Token Merging. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30772–30810, 2024.

Ross Wightman. PyTorch Image Models. GitHub repository, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5453–5462. PMLR, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Jiaojiao Zhao and Cees G. M. Snoek. LiftPool: Bidirectional ConvNet pooling. In *International Conference on Learning Representations*, 2021.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ADE20K dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

APPENDIX

A IMPLEMENTATION AND EXPERIMENTAL DETAILS

This appendix describes the implementation of NSR for token merging and graph aggregation and supplements the experimental settings in the main text. In both applications, NSR constructs member residuals using the members and weights of the current aggregation operation and transforms them into task-adaptive updates through a learnable branch.

A.1 DATASETS, BASELINES, AND EXPERIMENTAL PROTOCOLS

This section provides the datasets, comparison settings, training and evaluation protocols, and computational-accounting conventions for the experiments in the main text. The implementations of NSR for token merging and graph aggregation are described in Appendices A.2 and A.3, respectively.

Semantic-segmentation data and preprocessing. Pascal VOC 2012 Everingham et al. (2012) contains 20 foreground classes and one background class, with $1{,}464$ training images and $1{,}449$ validation images. Cityscapes Cordts et al. (2016) uses the fine-annotation split, containing $2{,}975$ training images and 500 validation images with 19 evaluated semantic classes. ADE20K Zhou et al. (2017) contains $20{,}210$ training images and $2{,}000$ validation images covering 150 semantic classes. All experiments use the official training and validation splits.

VOC 2012 retains its original class encoding. ADE20K maps valid class labels to 0–149, and Cityscapes maps the original annotations to its 19 training classes. Invalid labels are assigned the ignore index 255 and excluded from both the loss and evaluation metrics. All images are normalized using the ImageNet channel means and standard deviations.

For VOC and ADE20K, training augmentation consists of aspect-ratio-preserving random resizing, random cropping, and horizontal flipping with probability 0.5. The resizing scale is sampled from 0.5–2.0, and the crop sizes are 224×224 and 512×512, respectively. Validation resizes the shorter image side to 224 and 512, respectively, followed by a center crop of the corresponding size. Cityscapes uses random resizing, photometric distortion, 512×1024 random cropping, and horizontal flipping. Validation performs sliding-window inference on full images, using a 512×1024 window and a 341×682 stride. Class logits are averaged in overlapping regions. Geometric transformations are applied jointly to images and labels, with nearest-neighbor interpolation used for labels.

Compared methods and controlled comparisons. The segmentation experiments compare ToMe, PiToMe, and MPM with their respective NSR-augmented versions and include the uncompressed Full-ViT as a reference. All models use the same DeiT-Tiny/16 segmentation architecture. ToMe and PiToMe retain their token-size-based merging weights, while MPM retains mutual-nearest-neighbor pairing and arithmetic-mean merging. The merge locations and within-group weights are summarized in Table 5.

Graph experiments use GCN, GraphSAGE, and GIN, with NSR branches added to their neighborhood aggregation layers. Residual construction follows the member sets and aggregation coefficients used by the corresponding backbone, as specified in Table 6. Each pair shares the same backbone configuration, task inputs, loss function, and training protocol.

Before segmentation training, shared backbone and segmentation-head parameters are copied across the compared models. Random seeds are reset and data loaders are reconstructed for each model, ensuring consistent data ordering and training budgets on the same dataset. Each Original–NSR pair uses the same compression-control configuration; adding NSR preserves the original token-selection, matching, and merging rules.

Semantic-segmentation training and evaluation. All segmentation models use an ImageNet-pretrained DeiT-Tiny/16 backbone. After the representations are restored to the original patch grid, LayerNorm and a linear classification layer produce class logits, which are bilinearly upsampled to the input image resolution. The backbone, segmentation head, and NSR parameters are trained jointly under downstream segmentation supervision.

Training uses pixel-wise cross-entropy and AdamW Loshchilov & Hutter (2019), with an initial learning rate of 10^{-4} and weight decay of 10^{-4}. The learning rate follows polynomial decay with power 0.9, updated at each training iteration. Configurations with warm-up first increase the learning rate linearly to its initial value. Training runs for 100 epochs on all three datasets. CUDA training uses automatic mixed precision and gradient scaling.

For each semantic-segmentation configuration, we report validation mIoU from the final checkpoint of a single training run. The mIoU is computed from the confusion matrix accumulated over the entire validation set and averaged over classes with nonzero union. The mIoU is computed from the confusion matrix accumulated over the entire validation set and averaged over classes with nonzero union. The main results and GFLOPs are reported in Table 1, with additional performance and parameter results in Table 8.

Heterophilic node classification. We use Roman-empire, Amazon-ratings, Minesweeper, Tolokers, and Questions Platonov et al. (2023b). Roman-empire is a word-dependency graph derived from an English Wikipedia article, Amazon-ratings is a product co-purchasing network, and Minesweeper is a synthetic graph based on the corresponding game. Tolokers and Questions are interaction networks constructed from a crowdsourcing platform and a question-answering service, respectively. Dataset statistics are provided in Table 16. We follow the ten fixed splits supplied with the benchmark. Roman-empire and Amazon-ratings use classification accuracy, while the remaining three datasets use ROC-AUC.

The models use four message-passing layers, a hidden dimension of 128, and dropout of 0.5. Training uses the full graph, with cross-entropy loss computed only on training nodes. Adam Kingma & Ba (2015) uses an initial learning rate of 10^{-3} and weight decay of 5×10^{-4}, with the gradient norm clipped to 1. Each official split uses random seed 42, and evaluation is performed every ten epochs.

The learning-rate scheduler monitors the validation metric, with a reduction factor of 0.5, a patience of 50 evaluations, and a minimum learning rate of 10^{-6}. Training runs for at most $3{,}000$ epochs and stops after 100 validation evaluations without improvement. For each split, we report the test score of the checkpoint with the best validation metric. Final results are the mean and sample standard deviation over the ten splits. Results are reported in Table 2.

Tree-NeighborsMatch. We follow the task construction of Alon and Yahav Alon & Yahav (2021) and use training accuracy to examine fitting ability on this controlled matching task. The root must predict the value stored at the leaf associated with its query key. Each leaf contains a key–value pair, and node inputs concatenate one-hot encodings of keys and values. We use complete binary

Table 4: Tree-NeighborsMatch configurations. Effective batches larger than 256 are implemented using gradient accumulation.

Tree depth	Generated samples	Message-passing layers	Nominal effective batch size
2	96	3	64
3	8,000	4	64
4	16,000	5	1,024
5	32,000	6	1,024
6	32,000	7	1,024
7	32,000	8	2,048
8	32,000	9	2,048

trees of depths 2–8, with edges directed from children to parents and self-loops included. Examples are divided into 80% training and 20% test sets using label-stratified splitting, without a separate validation set.

For a tree of depth d, the model uses $d + 1$ message-passing layers and a hidden dimension of 32. Table 4 lists the generated sample counts, message-passing depths, and nominal effective batch sizes.

Training uses cross-entropy and Adam with an initial learning rate of 10^{-3}, without dropout or weight decay, for at most $50{,}000$ epochs. Learning-rate scheduling and early stopping both monitor training accuracy, with patience values of $1{,}000$ and $2{,}000$ epochs, respectively. All models use random seed 11 and are trained once at each tree depth. We report the highest training accuracy attained during each run. Figure 3 shows training fit across tree depths.

ZINC. We use the $12{,}000$-graph ZINC subset Dwivedi et al. (2023), with fixed training, validation, and test splits containing $10{,}000$, $1{,}000$, and $1{,}000$ graphs, respectively. The task is molecular property regression, evaluated by MAE, for which lower values indicate better performance. All models use atom types and graph connectivity without bond-type features.

Atom types are mapped to 128-dimensional learnable embeddings. The models use four message-passing layers, a hidden dimension of 128, and dropout of 0.5. Training minimizes the L_1 loss using Adam, with an initial learning rate of 10^{-3}, weight decay of 10^{-5}, and a batch size of 128. The gradient norm is clipped to 1, and training runs for at most 400 epochs.

Validation MAE determines both learning-rate scheduling and checkpoint selection. The scheduler uses a patience of 20 epochs, a reduction factor of 0.5, and a minimum learning rate of 10^{-5}. Early stopping uses a patience of 100 epochs. After training, the checkpoint with the lowest validation MAE is restored and evaluated once on the test set. Table 3 reports the mean and standard deviation of test MAE over random seeds 0–9.

Operating points and configuration consistency. Token-merging experiments cover operating points from mild to aggressive compression. The computation-reduction factor is defined as $C = F_{\mathrm{Full}}/F_{\mathrm{Base}}$, where F_{Full} and F_{Base} denote the GFLOPs of the uncompressed Full-ViT and the original compressed baseline, respectively. Each method adjusts only its native compression controls to approach the target computation budget.

ToMe varies the planned number of tokens merged per layer, PiToMe varies the token-retention ratio, and MPM varies the Transformer blocks at which merging is applied. The original matching rules, weight computations, and merging mechanisms remain unchanged. Because MPM pairing depends on input content and model representations, the compression factors achieved during calibration and final evaluation may differ. We retain the selected insertion schedule and report the computation corresponding to final evaluation.

Each Original–NSR pair uses the same compression-control configuration: ToMe shares the planned merge count and layer-wise merge configuration; PiToMe shares the retention ratio, margin schedule, and matching mode schedule; and MPM shares the insertion-block schedule. NSR does not increase compression to compensate for its additional computational overhead, and its total GFLOPs

Table 5: Integration of NSR with the three token-merging methods. Each method retains its merge location and weighting rule. Token size denotes the number of original patches represented by a token.

Method	Merge location	Within-group weights
ToMe	After the attention residual update and before the MLP	Normalized token-size weights
PiToMe	After the attention residual update and before the MLP	Normalized token-size weights
MPM	Before a designated Transformer block	Equal weights of $1/2$ within each pair

are reported separately. The configurations for the three methods are provided in Tables 9, 10, and 11, respectively. The complete configuration-search procedure is described in Appendix F.

Computational accounting. Computation is estimated analytically from the model's forward structure, counting one multiply–accumulate as one FLOP. The calculation includes patch embedding, attention, MLPs, the main matching and merging operations, and the linear segmentation head. For NSR models, it additionally includes residual encoding, gating, feature decoding, spatial write-back, and spatial decoding. Normalization, activation, indexing operations, and final upsampling are excluded. Parameter counts include all trainable parameters of the complete model.

For ToMe and PiToMe, layer-wise token counts are determined by the compression configurations. For MPM, computation is calculated from each sample's valid pair counts and then averaged across samples, excluding the additional cost of batch padding. Cityscapes GFLOPs correspond to a single 512×1024 input window, and MPM pair statistics are collected from center windows of validation images. Total computation for full-image sliding-window inference also depends on the number of windows.

Computing environment. Experiments were conducted on four compute instances, each equipped with 16 CPU cores, 128 GB of RAM, and one NVIDIA A100 GPU. The implementation is based on PyTorch Paszke et al. (2019). Graph experiments use PyTorch Geometric Fey & Lenssen (2019) and torch-scatter, while vision experiments use timm Wightman (2019) and torchvision.

A.2 Token-Merging Implementation

All semantic-segmentation models use an ImageNet-pretrained DeiT-Tiny/16 backbone. For different input resolutions, the pretrained patch positional embeddings are resized to the target grid using bicubic interpolation, while the positional embedding of the classification token is retained. After the backbone representations are restored to the original patch grid, LayerNorm and a linear classification layer produce class logits, which are bilinearly upsampled to the input image resolution.

ToMe, PiToMe, and MPM use the same NSR branch architecture. For each method, residuals are constructed from the member groups and weights produced by its merging operation. Table 5 summarizes the merge locations and within-group weights.

ToMe and PiToMe use proportional attention based on token size. MPM uses mutual-nearest-neighbor pairing and arithmetic-mean merging. The classification token participates in backbone attention but is excluded from matching and merging. The operating-point configurations of all three methods are provided in Appendix F.

For a merge group $\mathcal{G}$, NSR computes the merged representation using the original within-group weights and extracts the residual of each member relative to that representation. For ToMe and PiToMe, all source tokens assigned to the same destination are organized into a complete merge group. Each valid MPM group contains two members. Valid-member masks exclude padded entries introduced by batching, and singleton groups produce no residual update.

The residual encoder consists of RMSNorm Zhang & Sennrich (2019) followed by a bias-free linear layer that maps the d-dimensional residual to a code of dimension $d_c = 64$. The gating network receives the concatenation of the normalized member residual, member representation, and group

Table 6: Member sets and aggregation coefficients used by the NSR graph branches.

Backbone	Members	Aggregation coefficients
GCN	Incoming messages in the normalized backbone graph, including self-loops	Degree-normalized GCN coefficients
GraphSAGE	Neighbor messages specified by the input edge list	Neighborhood-mean weights
GIN	Neighbor messages and a separate root message	1 for neighbors and $1 + \epsilon$ for the root

representation. It uses two linear layers with dimensions $3d \to 32 \to 64$, a GELU activation between them, and a sigmoid output to generate member-dependent, channel-wise weights. The member and group representations condition the gate, while the information encoded by the residual encoder comes from the member residual.

The same member codes support the feature and spatial residual paths. The feature path aggregates the member codes using the original merge weights and applies a bias-free $64 \to d$ linear decoder. The resulting update is added to the current merged token, which continues through the backbone. Different merge locations have separate encoders, gating networks, and feature decoders.

The spatial path maintains the correspondence between current tokens and original patch positions. At each merge event, a member code is written to the original positions covered by that member and accumulated in a fixed $N_0 \times 64$ code tensor. After the backbone computation, the merge events are reversed to restore the original patch ordering. A single shared bias-free linear decoder maps the accumulated codes to position-specific updates, which are added to the restored patch representations. The spatial code tensor is initialized separately for each forward pass and is not processed as an additional token sequence by the Transformer.

Both the feature decoders and the final spatial decoder are zero-initialized. Consequently, with identical shared parameters, the NSR-augmented model initially preserves the output of its corresponding merging baseline. The original and NSR-augmented models use the same matching rules, weight-computation procedures, and predefined compression configurations. NSR constructs its residuals from the groups realized at each merge event.

The feature-only and spatial-only ablations retain the corresponding update path. The post-merge-only control generates a feature update from the merged group representation without using member residuals or spatial correspondences. The associated results are reported in Appendix G.

A.3 Graph-Aggregation Implementation

The graph experiments use GCN, GIN, and GraphSAGE as backbones. All models first map node inputs into the hidden representation space and then apply multiple message-passing layers. Each message-passing layer has a separate NSR branch, with no parameter sharing across layers.

The graph branch applies its own bias-free $d \to d$ linear layer to the node representations participating in the current aggregation and stacks the resulting messages row-wise to form the member-message matrix M_v in Section 3.4. The parameters of this linear layer are shared across members. Using the member set and aggregation coefficients t_v of the corresponding backbone, the branch applies the null-space projection in Eq. (5) to M_v to extract member-level residuals. The projection properties are derived in Appendix B. The implementation uses scatter operations grouped by destination node and does not explicitly materialize local projection matrices.

Table 6 specifies the members and coefficients used by each backbone.

Each branch follows the incoming-edge and self-loop handling of its corresponding backbone. The separate root branch of GraphSAGE is retained in the backbone. For GIN, the residual construction includes both the neighbor aggregation terms and the separate root term. The evaluated GIN configuration uses a fixed $\epsilon = 0$.

Member residuals are modulated by channel-wise gates. The gating network concatenates the source-node representation, destination-node representation, and member residual. It consists of

two linear layers with dimensions $3d \to d \to d$, a GELU activation, and a sigmoid output. The bias of its final linear layer is initialized to -2. The gated residuals are summed by destination node and scaled by the inverse square root of the number of participating members. The resulting node-level representation is passed through LayerNorm, Dropout, and a $d \to d \to d$ output network with a GELU activation between its linear layers to produce the correction.

For GCN and GraphSAGE, the baseline layer applies graph convolution, LayerNorm, GELU, and Dropout, followed by a residual addition to the layer input. The GIN convolution contains two successive Linear–BatchNorm–ReLU stages. Its output is passed through Dropout and added to the layer input. In all cases, the NSR correction is added to the corresponding baseline layer update, retaining the original backbone structure.

Node classification applies a linear classifier to the final node representations. Tree-NeighborsMatch reads out only the root representation. ZINC uses global mean pooling to obtain a graph representation, followed by a linear regression head.

B Supplementary Derivations for NSR

This section explains why the residual operator in the main text is a null-space projection and verifies that the lifts used for graph aggregation and token merging satisfy the conditions of the general construction. All derivations concern the linear operator realized in a single computation event. When group assignments or weights depend on the input, they are held fixed at the values realized for that event.

B.1 Null-Space Projection Properties

Following the notation in Sec. 3.2, let $A \in \mathbb{R}^{k\times n}$ denote the current linear operator, where n and k are the numbers of input and output members, respectively. Let $U \in \mathbb{R}^{n\times k}$ be a linear lift from the output representation space to the input member space, satisfying $AUA = A$ Ben-Israel & Greville (2003). Define the residual operator as $P = I_n - UA$, where I_n is the $n \times n$ identity matrix.

We verify that P maps inputs into the null space of A and leaves vectors in that null space unchanged. Write $\ker(A) = \{q \in \mathbb{R}^n : Aq = 0\}$. Using $AUA = A$, we obtain

$$\begin{aligned} AP &= A - AUA = 0, \\ P^2 &= I_n - 2UA + U(AUA) = I_n - UA = P. \end{aligned} \tag{10}$$

The identity $AP = 0$ implies that the range of P is contained in $\ker(A)$. Conversely, for any $q \in \ker(A)$, we have $Pq = q - UAq = q$, so $\ker(A)$ is also contained in the range of P. Together with $P^2 = P$, these relations establish that P is a projection onto $\ker(A)$, though not necessarily an orthogonal projection. Thus, the residual operator in the main text projects each feature channel onto directions invisible to the original operator and acts as the identity on those directions.

B.2 Specialization to Graph Aggregation and Token Merging

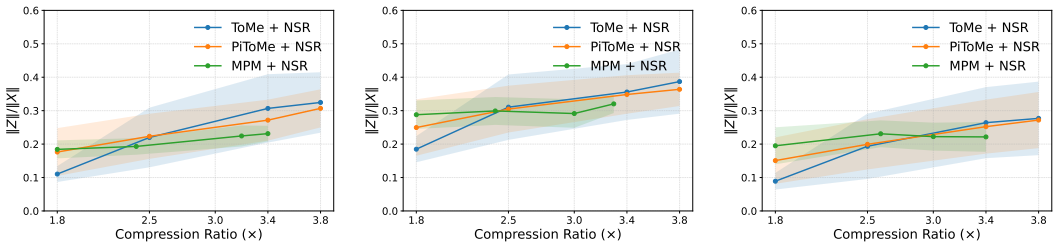
The two applications use different lifts: graph aggregation lifts along the aggregation coefficient vector, whereas token merging copies the merged representation back to the member positions. We verify below that both choices satisfy $AUA = A$, allowing direct application of Proposition 1.

Graph aggregation. Fix a local aggregation at node v. Let $K_v \geq 1$ be the number of participating members and let $t_v \in \mathbb{R}^{K_v}$ be the nonzero vector of aggregation coefficients. This operation reduces the member messages to a single output through a weighted sum, with operator $A_v = t_v^\top \in \mathbb{R}^{1\times K_v}$. Following the construction in Sec. 3.4, choose the lift $U_v = t_v/(t_v^\top t_v) \in \mathbb{R}^{K_v\times 1}$.

Since $t_v^\top t_v > 0$, this lift is well defined, and

$$A_v U_v = \frac{t_v^\top t_v}{t_v^\top t_v} = 1, \qquad A_v U_v A_v = A_v. \tag{11}$$

This choice therefore satisfies the condition of the general construction, and the graph residual in the main text is obtained by a null-space projection for the current local aggregation operator. The



(a) ADE20K (b) Pascal VOC (c) Cityscapes

Figure 5: Relative null-space residual magnitude $\rho_Z = \|Z\|_F / \|X\|_F$ for the trained +NSR token-merging models on (a) ADE20K, (b) Pascal VOC, and (c) Cityscapes. For each compression setting, the solid line reports the mean across merging blocks in which NSR is constructed, and the shaded region shows the corresponding mean $\pm$ one standard deviation.

member set and coefficients are those of the aggregation actually performed by the corresponding backbone.

Token merging. Consider a realized merge group $\mathcal{G}$ containing $K \geq 1$ tokens. Let $t \in \mathbb{R}^K$ be the merge weights, satisfying $t^\top \mathbf{1}_K = 1$, where $\mathbf{1}_K$ is the K-dimensional all-ones column vector. The merge operator is $A_\mathcal{G} = t^\top \in \mathbb{R}^{1\times K}$. Following the copy-based unmerging used in Sec. 3.5, choose $U_\mathcal{G} = \mathbf{1}_K \in \mathbb{R}^{K\times 1}$, which copies the same merged representation to every member position in the group.

The weight normalization gives

$$A_\mathcal{G} U_\mathcal{G} = t^\top \mathbf{1}_K = 1, \qquad A_\mathcal{G} U_\mathcal{G} A_\mathcal{G} = A_\mathcal{G}. \tag{12}$$

Thus, the copy-based lift also satisfies the condition of the general construction, and the difference between the member representations and the copied merged representation is the corresponding null-space residual. This conclusion depends only on the normalization of the weights within the current group and does not require uniform weights.

These verifications connect both residual constructions to the unified formulation in the main text. Their null-space properties and complementarity follow directly from Proposition 1.

C Numerical Verification of NSR for Token Merging

To verify that the implemented NSR construction is consistent with its theoretical null-space definition, we examine both the null-space constraint and the magnitude of the resulting residuals. For each realized token-merging operation, let A denote the corresponding linear aggregation operator, X the input token representations, and Z the residual defined in Eq. equation 7. We measure

$$\epsilon_{\text{null}} = \|AZ\|_\infty, \qquad \rho_Z = \frac{\|Z\|_F}{\|X\|_F}. \tag{13}$$

Here, ϵ_{null} measures the numerical violation of the null-space constraint $AZ = 0$, while ρ_Z characterizes the residual magnitude relative to the input representation.

Figure 5 shows that the relative residual magnitude generally increases with stronger compression for ToMe and PiToMe across the three datasets, whereas MPM varies more mildly and is occasionally non-monotonic. These results show that the null-space residual is numerically non-negligible and that its magnitude depends on both the merging operator and the compression strength.

Table 7 reports the complementary constraint check. Across all datasets and merging methods, the maximum observed violation is on the order of 10^{-6} or below, with an overall maximum of 1.76×10^{-6}. Thus, the constructed residuals satisfy $AZ \approx 0$ to a small numerical tolerance across the evaluated settings. Together with Figure 5, these results verify that NSR constructs residual components with non-negligible magnitude while remaining effectively invisible to the original merging operator.

Table 7: Maximum null-space violation $\epsilon_{\text{null}} = \|AZ\|_\infty$ for the trained +NSR models. Each entry is the maximum observed value over all evaluated compression settings and merging blocks.

Dataset	ToMe	PiToMe	MPM
ADE20K	9.02×10^{-7}	9.54×10^{-7}	9.54×10^{-7}
Pascal VOC	7.38×10^{-7}	6.28×10^{-7}	4.77×10^{-7}
Cityscapes	1.76×10^{-6}	9.54×10^{-7}	4.77×10^{-7}

D Additional Performance and Parameter Results for Token Merging

Table 8 supplements the main token-merging results with pixel accuracy (Pixel Acc.), model parameter counts (Params), and Compression Gap Closure for ToMe, PiToMe, and MPM across different compression settings on PASCAL VOC, Cityscapes, and ADE20K. These results provide an additional view of the performance recovery achieved by NSR and its associated parameter overhead.

In terms of model size, all Original Token Merging models have the same number of parameters as the corresponding uncompressed Full-ViT, consistent with the design of these methods, which introduce no additional learnable parameters. This agreement also provides an auxiliary check on the correct implementation of the Token Merging baselines from the perspective of model size. After introducing NSR, the relative parameter increases are 3.56%–10.23%, 1.78%–9.61%, and 4.25%–9.90% on PASCAL VOC, Cityscapes, and ADE20K, respectively, indicating that NSR introduces only limited additional parameter overhead. In particular, MPM incurs lower overhead in some settings; for example, at the $1.8\times$ setting on Cityscapes, its parameter count increases by only 1.78%.

In terms of segmentation performance, NSR achieves higher Pixel Acc. than the corresponding compressed baseline in 35 of the 36 evaluated settings, with larger observed recovery under stronger compression. For example, for ToMe at the $3.8\times$ setting on PASCAL VOC and ADE20K, NSR increases the parameter count by only 8.56% and 9.90%, while improving Pixel Acc. by 0.1376 and 0.2543, respectively. These results further show that NSR can effectively mitigate the segmentation-performance degradation caused by Token Merging under strong compression while introducing limited additional parameter overhead.

To complement the absolute mIoU gains in Table 1, we report Compression Gap Closure, which measures the fraction of the performance gap between the original compressed baseline and Full-ViT that is closed by NSR. For the evaluated configurations, where $\mathrm{mIoU}_{\mathrm{Full}} > \mathrm{mIoU}_{\mathrm{Base}}$, it is defined as

$$\text{Gap Closure} = \frac{\mathrm{mIoU}_{\mathrm{NSR}} - \mathrm{mIoU}_{\mathrm{Base}}}{\mathrm{mIoU}_{\mathrm{Full}} - \mathrm{mIoU}_{\mathrm{Base}}} \times 100\%, \tag{14}$$

where $\mathrm{mIoU}_{\mathrm{Full}}$, $\mathrm{mIoU}_{\mathrm{Base}}$, and $\mathrm{mIoU}_{\mathrm{NSR}}$ denote the mIoU of Full-ViT, the corresponding original compressed baseline, and its NSR-augmented counterpart, respectively. A Gap Closure of 100% indicates that the NSR-augmented model matches Full-ViT; values above 100% indicate that it surpasses Full-ViT, while negative values indicate a decrease relative to the compressed baseline.

The final column of Table 8 reports Gap Closure for all 36 compression configurations. NSR closes a substantial fraction of the performance gap across multiple merging methods and datasets. For example, ToMe at the $3.4\times$ operating point on PASCAL VOC gains 31.51 mIoU points, closing 67.7% of the gap to Full-ViT. On ADE20K, PiToMe at $1.8\times$ reaches a Gap Closure of 127.1%, consistent with its mIoU exceeding Full-ViT.

Because the metric normalizes by the Full-ViT–baseline performance gap, it is sensitive to small reference gaps. For example, ToMe at $1.8\times$ on Cityscapes has a Gap Closure of -33.3%, although its absolute mIoU change is only -0.04. Gap Closure is therefore interpreted together with the absolute mIoU values and gains in Table 1.

Table 8: Additional semantic-segmentation results for token merging with and without NSR. WP denotes the computation-reduction operating point of the original compressed baseline relative to Full-ViT. Pixel Acc. and Params are reported separately for Original and +NSR. Within each Original–NSR pair, the higher Pixel Acc. is shown in bold. Gap Closure is computed from unrounded mIoU values according to Eq. equation 14 and reported to one decimal place.

			Pixel Acc. ↑		Params		
Dataset	Method	WP	Original	+NSR	Original	+NSR	Gap Closure ↑
VOC	Full-ViT	1.0×	0.9062	–	5,528,853	–	–
	ToMe	1.8×	0.8988	**0.9030**	5,528,853	6,094,485	69.7%
		2.5×	0.8248	**0.8873**	5,528,853	6,094,485	73.8%
		3.4×	0.7131	**0.8607**	5,528,853	6,048,373	67.7%
		3.8×	0.7111	**0.8487**	5,528,853	6,002,261	61.1%
	PiToMe	1.8×	0.8934	**0.8978**	5,528,853	6,094,485	28.5%
		2.5×	0.8722	**0.8869**	5,528,853	6,094,485	37.6%
		3.4×	0.8241	**0.8696**	5,528,853	6,094,485	48.8%
		3.8×	0.7978	**0.8595**	5,528,853	6,094,485	48.5%
	MPM	1.8×	0.8969	**0.8991**	5,528,853	5,725,589	22.7%
		2.4×	0.8866	**0.8893**	5,528,853	5,817,813	14.5%
		3.0×	0.8735	**0.8827**	5,528,853	6,002,261	27.2%
		3.3×	0.8601	**0.8753**	5,528,853	6,094,485	26.0%
Cityscapes	Full-ViT	1.0×	0.9517	–	5,884,051	–	–
	ToMe	1.8×	**0.9508**	0.9507	5,884,051	6,449,683	−33.3%
		2.5×	0.9390	**0.9497**	5,884,051	6,449,683	79.3%
		3.4×	0.9046	**0.9463**	5,884,051	6,449,683	86.4%
		3.8×	0.8860	**0.9446**	5,884,051	6,449,683	85.0%
	PiToMe	1.8×	0.9489	**0.9495**	5,884,051	6,449,683	4.7%
		2.5×	0.9454	**0.9492**	5,884,051	6,449,683	80.5%
		3.4×	0.9375	**0.9459**	5,884,051	6,449,683	53.3%
		3.8×	0.9333	**0.9448**	5,884,051	6,449,683	62.2%
	MPM	1.8×	0.9493	**0.9501**	5,884,051	5,988,563	49.5%
		2.6×	0.9443	**0.9476**	5,884,051	6,219,123	47.6%
		3.0×	0.9429	**0.9469**	5,884,051	6,265,235	25.8%
		3.4×	0.9379	**0.9453**	5,884,051	6,449,683	42.5%
ADE20K	Full-ViT	1.0×	0.7708	–	5,712,726	–	–
	ToMe	1.8×	0.7687	**0.7693**	5,712,726	6,278,358	81.4%
		2.5×	0.7272	**0.7661**	5,712,726	6,278,358	91.5%
		3.4×	0.5661	**0.7548**	5,712,726	6,278,358	82.5%
		3.8×	0.4963	**0.7506**	5,712,726	6,278,358	84.7%
	PiToMe	1.8×	0.7662	**0.7704**	5,712,726	6,278,358	127.1%
		2.5×	0.7545	**0.7636**	5,712,726	6,278,358	41.4%
		3.4×	0.7332	**0.7601**	5,712,726	6,278,358	68.6%
		3.8×	0.7160	**0.7532**	5,712,726	6,278,358	54.5%
	MPM	1.8×	0.7685	**0.7713**	5,712,726	5,955,574	−7.0%
		2.4×	0.7653	**0.7659**	5,712,726	6,001,686	33.6%
		3.2×	0.7486	**0.7622**	5,712,726	6,186,134	53.7%
		3.4×	0.7432	**0.7583**	5,712,726	6,278,358	54.8%

E Label-Based Null-Space Diagnostic: Definition and Supplementary Analysis

This appendix provides the full definition of the diagnostic score used in the main text, the binning procedure, and supplementary results on accuracy gains.

For a local aggregation event u, let $S_u \in \mathbb{R}^{k_u \times C}$ contain the one-hot label encodings of the participating members, where k_u is the number of members and C is the number of classes. Let $t_u \in \mathbb{R}^{k_u}$ be the nonzero aggregation coefficient vector actually used by the corresponding backbone. Follow-

ing the graph-aggregation residual projection in Eq. equation 5, we define

$$P_u = I_{k_u} - \frac{t_u t_u^\top}{t_u^\top t_u}, \qquad r_u = \frac{\|P_u S_u\|_F^2}{\|S_u\|_F^2 + \epsilon}, \tag{15}$$

where $\epsilon > 0$ is a numerical stabilizer. Since P_u is the orthogonal projection onto $\ker(t_u^\top)$, it satisfies $t_u^\top P_u S_u = 0$. Thus, r_u measures the normalized projection energy of the label encodings in the null space of this local aggregation. Both the member set and the aggregation coefficients follow the aggregation definition of the corresponding backbone. This score operates on label encodings and does not directly measure task-relevant information loss in the model's hidden features.

For a model with L message-passing layers, let $\mathcal{T}_v^{(L)}$ denote the set of local aggregation events in the unfolded computation tree of target node v. We define the node-level diagnostic score as

$$R_v^{(L)} = \frac{\sum_{u \in \mathcal{T}_v^{(L)}} k_u r_u}{\sum_{u \in \mathcal{T}_v^{(L)}} k_u}, \tag{16}$$

where u denotes a local aggregation event in the computation tree and k_u is the number of members in that event. This weighted average summarizes the local diagnostic scores within the target node's computation tree. All scores are computed from ground-truth labels after training and are not used for training, hyperparameter selection, or model selection.

On Roman-empire, we compute $R_v^{(L)}$ for the test nodes separately for each official split and each of the GCN, GIN, and GraphSAGE backbones. Bin boundaries are determined by score deciles, with nodes having identical scores kept in the same bin. The Original and +NSR models are compared on the same nodes within each bin. Bins are constructed independently within each split, and cross-split statistics are aggregated by bin index, ordered from low to high scores.

Figure 4 in the main text reports the mean accuracy in each bin. Figure 6 further shows the accuracy gain within the same bins. For bin b in split s, we define

$$\Delta\mathrm{Acc}_{s,b} = \mathrm{Acc}_{+\mathrm{NSR},s,b} - \mathrm{Acc}_{\mathrm{Original},s,b}, \tag{17}$$

and average $\Delta\mathrm{Acc}_{s,b}$ over the ten official splits. For all three backbones, the mean gain exhibits an overall increasing trend with the score-bin index, with local fluctuations. This figure presents the differences underlying the accuracy results in the main text and provides a complementary view of how NSR gains are distributed across score bins in this experiment.

F Operating-Point Construction

We define the computation-reduction factor as

$$C = \frac{F_{\mathrm{Full}}}{F_{\mathrm{compressed}}}, \tag{18}$$

where F_{Full} and $F_{\mathrm{compressed}}$ denote the GFLOPs of the uncompressed Full-ViT and the corresponding token-merging baseline, respectively. For the token-merging experiments, the reported WP values serve as nominal labels for configurations constructed using this ratio; the achieved computation-reduction factors, which need not equal these labels, are reported in the Achieved columns of the configuration tables below. For a target reduction factor C_{target}, each method searches only within its native compression-control space to match the target computation budget as closely as possible, without modifying its original matching or merging mechanism.

ToMe. ToMe controls compression through the planned number of tokens r merged per layer. For each target computation budget, we search over all valid integer values of r and select

$$r^* = \arg\min_r \left| \frac{F_{\mathrm{Full}}}{F_{\mathrm{ToMe}}(r)} - C_{\mathrm{target}} \right|, \tag{19}$$

where $F_{\mathrm{ToMe}}(r)$ denotes the GFLOPs of the ToMe baseline with compression parameter r. The actual number of merged tokens r_l at layer l follows the original ToMe merging rules and is constrained by the number of remaining tokens.

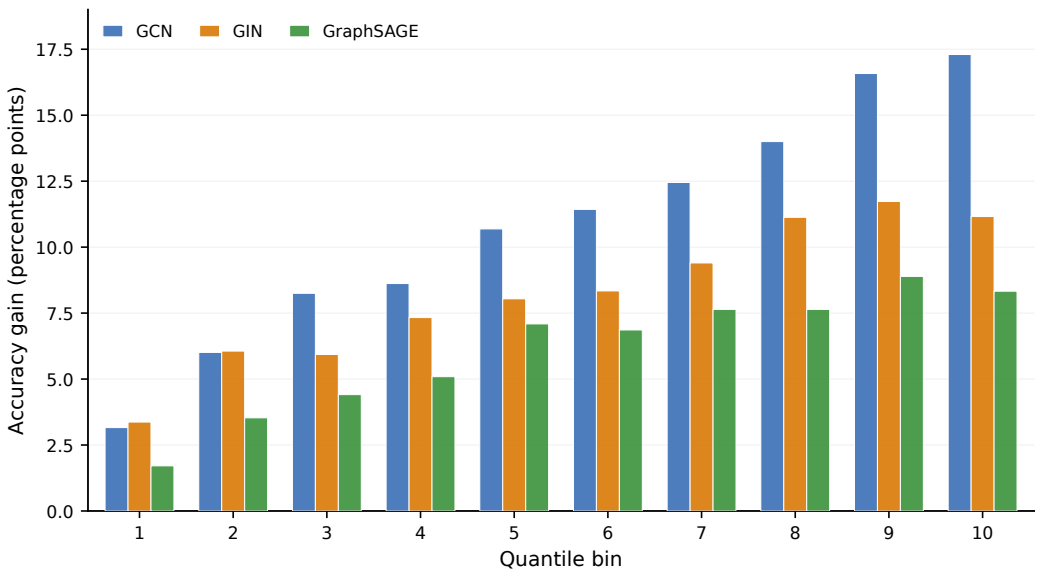


Figure 6: Mean accuracy gain from NSR across diagnostic-score bins on Roman-empire. Bins are the same as those in Figure 4. Within each official split, the gain is computed as the accuracy of the NSR-augmented model minus that of the Original model. The figure reports the mean over the ten official splits.

PiToMe. PiToMe controls compression through the retention ratio q. The number of tokens merged at Transformer block l is

$$r_l = \lfloor T_l(1-q) \rfloor , \tag{20}$$

where T_l denotes the number of patch tokens before merging at that block, excluding the CLS token. We deterministically search q over $[0.5, 0.999]$ with a step size of 0.001, while keeping the original energy-based token selection, margin schedule, and matching mode schedule unchanged.

MPM. Since the actual number of merged tokens in MPM depends on the input features, we construct different operating points by searching the insertion-block schedule without modifying its mutual-nearest-neighbor matching or arithmetic-mean merging mechanism. We use 32 and 200 images from the training split for coarse search and refinement, respectively. Candidate schedules are generated by adding, removing, or moving insertion blocks, and the final schedule is selected according to its distance from the target computation-reduction factor, without using validation mIoU.

Because MPM matching depends on the input content and model representations, the compression factors estimated during calibration and achieved during final validation may differ. We report both values and keep the selected insertion schedule fixed during final evaluation.

Configuration consistency. For each operator and operating point, the baseline and its NSR-augmented counterpart share the same compression-control configuration. ToMe shares r and the layer-wise merge schedule; PiToMe shares the retention ratio, layer-wise merge schedule, margin schedule, and matching mode schedule; and MPM shares the insertion-block schedule. NSR does not search for a more aggressive compression configuration to compensate for its additional computational overhead.

All Transformer block indices reported below are zero-based.

F.1 ToMe Compression Configurations

Table 9 reports the selected r, achieved computation-reduction factor, and GFLOPs for ToMe on the three datasets. Since the actual layer-wise merge count is constrained by the number of remaining tokens, the selected r may not be fully applied at every layer. The corresponding layer-wise compression distributions are visualized in Fig. 7.

Table 9: ToMe compression configurations on the three semantic segmentation datasets. Target and Achieved denote the target and realized computation-reduction factors, respectively. Selected r is the planned number of tokens merged per layer. GFLOPs denotes the computation of the corresponding baseline.

Dataset	Target	Selected r	Achieved	GFLOPs
VOC2012	1.8×	14	1.797×	0.6980
	2.5×	21	2.550×	0.4918
	3.4×	29	3.367×	0.3725
	3.8×	33	3.768×	0.3329
ADE20K	1.8×	66	1.799×	5.8169
	2.5×	98	2.490×	4.2015
	3.4×	140	3.392×	3.0850
	3.8×	161	3.805×	2.7500
Cityscapes	1.8×	124	1.798×	16.9827
	2.5×	187	2.505×	12.1911
	3.4×	266	3.399×	8.9841
	3.8×	304	3.799×	8.0367

Table 10: PiToMe compression configurations on the three semantic segmentation datasets. q denotes the retention ratio. All operating points use the same margin and matching mode schedules. GFLOPs denotes the computation of the corresponding baseline.

Dataset	Target	Retention ratio q	Achieved	GFLOPs
VOC2012	1.8×	0.895	1.796×	0.6983
	2.5×	0.829	2.503×	0.5010
	3.4×	0.754	3.407×	0.3681
	3.8×	0.725	3.783×	0.3316
ADE20K	1.8×	0.908	1.804×	5.8011
	2.5×	0.849	2.503×	4.1799
	3.4×	0.781	3.401×	3.0765
	3.8×	0.752	3.795×	2.7572
Cityscapes	1.8×	0.913	1.806×	16.9101
	2.5×	0.859	2.500×	12.2127
	3.4×	0.794	3.407×	8.9615
	3.8×	0.767	3.797×	8.0407

F.2 PiToMe Compression Configurations

PiToMe uses the same margin schedule across all datasets and operating points:

$$\begin{aligned}\boldsymbol{m} = [&0.75,\ 0.6875,\ 0.625,\ 0.5625,\\ &0.5,\ 0.4375,\ 0.375,\ 0.3125,\\ &0.25,\ 0.1875,\ 0.125,\ 0.0625].\end{aligned} \tag{21}$$

Here, $\boldsymbol{m}$ specifies the margin parameter of each Transformer block in execution order. The first six blocks use BSM mode, while the remaining six blocks use standard mode. The CLS token never participates in matching or merging.

Table 10 summarizes the retention ratio, achieved computation-reduction factor, and GFLOPs for each operating point. The corresponding layer-wise compression distributions are shown in Fig. 7.

F.3 MPM Compression Configurations

MPM constructs different computation operating points by adjusting its insertion-block schedule. Table 11 summarizes the target reduction factors, selected insertion blocks, and compression factors achieved during calibration and final validation.

Table 11: MPM compression configurations on the three semantic segmentation datasets. Target denotes the search target, while Reported point is the operating-point label used in the final experiments. Search and Achieved denote the computation-reduction factors obtained during calibration and final validation, respectively. GFLOPs denotes the computation of the final baseline. A range i–j in Insertion blocks includes all consecutive block indices from i to j, inclusive.

Dataset	Target	Reported point	Insertion blocks	Search	Achieved	GFLOPs
VOC2012	1.8×	1.8×	[0,4,5,9]	1.801×	1.793×	0.6994
	2.5×	2.4×	[0–1,3,5–6,10]	2.494×	2.449×	0.5122
	3.0×	3.0×	[0–2,4–6,8–11]	3.004×	2.926×	0.4286
	3.8×	3.3×	[0–11]	3.438×	3.332×	0.3764
ADE20K	1.8×	1.8×	[2,4,7,9,11]	1.803×	1.803×	5.8041
	2.5×	2.4×	[0–1,4,7,9,11]	2.507×	2.355×	4.4431
	3.4×	3.2×	[0–4,7–11]	3.412×	3.167×	3.3035
	3.8×	3.4×	[0–11]	3.691×	3.425×	3.0553
Cityscapes	1.8×	1.8×	[0,2]	1.800×	1.816×	16.8170
	2.5×	2.6×	[1–6,10]	2.487×	2.562×	11.9181
	3.0×	3.0×	[0–2,4–7,10]	2.966×	3.033×	10.0667
	3.4×	3.4×	[0–11]	3.271×	3.359×	9.0903

As shown in Table 11, the final computation-reduction factor of MPM may differ from its calibration estimate. For example, the $2.5\times$ target on ADE20K achieves $2.507\times$ during calibration but $2.355\times$ during final validation. This difference is associated with the dependence of content-adaptive matching on the input and model representations, while the selected insertion schedule remains unchanged during final evaluation.

For the $3.8\times$ targets on VOC2012 and ADE20K, and the $3.4\times$ target on Cityscapes, all 12 Transformer blocks are used as MPM insertion points. These configurations exhaust the available insertion positions under the current experimental protocol. We therefore report their achieved compression factors rather than treating the original targets as exactly matched.

F.4 Layer-wise Token Merging Distributions

To further illustrate the layer-wise compression behavior across operators and operating points, Fig. 7 presents the normalized token-merging distributions of ToMe and PiToMe on the three datasets. The normalized number of merged tokens at Transformer block l is defined as

$$M_l = \frac{r_l}{T_0}, \tag{22}$$

where r_l denotes the actual number of tokens merged at block l, and T_0 denotes the initial number of patch tokens entering the ViT, excluding the CLS token. This normalization enables comparison of layer-wise merging quantities across different input resolutions on a common scale.

As shown in Fig. 7, stronger target compression leads to greater token reduction in the early layers for both ToMe and PiToMe, while their layer-wise merging patterns differ. ToMe maintains a relatively stable merge count in the initial blocks before the remaining token count imposes a tighter upper bound. In contrast, PiToMe uses a shared retention ratio, resulting in progressively decreasing merge counts as the remaining token sequence becomes shorter. Similar patterns are observed across the three datasets, indicating that the NSR evaluation covers token-merging configurations with different layer-wise compression patterns rather than a single reduction pattern.

G Additional Comparisons and Ablations

G.1 Comparison with Post-Merge-Only Refinement

NSR is motivated by a potential mismatch between operator-induced indistinguishability and task-required distinctions: member-level variations invisible to an operator may still be needed for downstream prediction. This comparison examines whether additional processing of operator-visible merged representations can reproduce the gains of the complete NSR pathway.

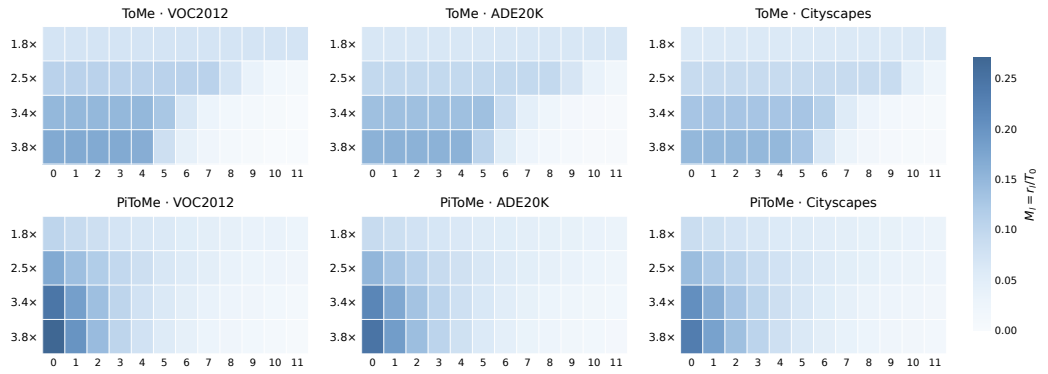


Figure 7: Layer-wise token-merging distributions of ToMe and PiToMe on VOC2012, ADE20K, and Cityscapes. The top and bottom rows correspond to ToMe and PiToMe, respectively, while the three columns correspond to VOC2012, ADE20K, and Cityscapes from left to right. The horizontal axis indicates the Transformer block index, and the vertical axis indicates the target computation-reduction factor. Color represents the normalized number of merged tokens, $M_l = r_l/T_0$. All subplots share the same color scale, with darker colors indicating a larger fraction of the initial patch tokens merged at the corresponding block.

To this end, we introduce a post-merge-only control based on merged representations. It takes the output y_G of each merge group as input, generates a feature update through encoding, gating, and decoding, and adds this update back to the corresponding merged representation. This branch only processes the group representations already produced by the operator. It does not access pre-merge member representations or null-space residuals, nor does it introduce spatial memory or a write-back path. All models use the same backbone architecture, training protocol, merging rules, and predefined compression configurations.

In NSR, the null space of the currently realized linear operator defines the source of candidate complementary residuals, while the encoder, member-wise gating, and application-specific integration jointly provide a task-supervised mechanism for exploiting these residuals. Accordingly, this comparison evaluates the complete pathway against local refinement of the merged representations.

As shown in Table 12, the effect of the post-merge-only refinement varies across compressors and operating points: it provides small improvements in some settings but reduces baseline performance in others. In contrast, Full NSR outperforms both the corresponding baseline and the post-merge-only control in every setting reported in the table, with more pronounced gains in several aggressive compression settings. For example, at the $3.4\times$ operating point of ToMe, the post-merge-only refinement decreases mIoU from the baseline value of 0.1831 to 0.1778, whereas Full NSR increases it to 0.4982.

These results show that, in the evaluated settings, additional local processing of the merged representations alone does not reproduce the gains of the complete NSR pathway. Spatial routing is specific to the token-merging instantiation for dense prediction and is absent from the graph-aggregation instantiation, where NSR also improves performance across multiple backbones and tasks, providing complementary evidence for the generality of the same design principle across structurally different settings. Section G.2 examines how the residuals are integrated through feature updates and spatial routing, separately and jointly. Section G.3 further compares explicitly extracted residuals with raw member features while retaining both integration paths.

G.2 Ablation of information-integration pathways within NSR

We examine Feature Integration and Spatial Routing as two application-specific ways of using null-space residual information in token merging. Copy-based unmerging restores the original token layout by copying each group representation to its member positions, but does not recover the original feature differences between those members. Both NSR variants extract member null-space residuals and process them through the residual encoder and member-dependent gates. Feature Integration

Table 12: Comparison with a local residual correction based only on merged representations. Best and second-best results at each operating point are shown in bold and underlined, respectively.

Compressor	Compression	Base	Post-merge-only	Full NSR
ToMe	1.8×	<u>0.6281</u>	0.6274	**0.6423**
	2.5×	<u>0.4458</u>	0.4395	**0.5954**
	3.4×	<u>0.1831</u>	0.1778	**0.4982**
	3.8×	<u>0.1698</u>	0.1683	**0.4623**
PiToMe	1.8×	0.6148	<u>0.6181</u>	**0.6244**
	2.5×	0.5577	<u>0.5593</u>	**0.5919**
	3.4×	<u>0.4370</u>	0.4198	**0.5402**
	3.8×	<u>0.3732</u>	0.3683	**0.5066**
MPM	1.8×	0.6199	<u>0.6203</u>	**0.6264**
	2.4×	<u>0.5887</u>	0.5837	**0.5974**
	3.0×	0.5493	<u>0.5609</u>	**0.5763**
	3.3×	0.5134	<u>0.5225</u>	**0.5485**

Table 13: Ablation of Feature Integration and Spatial Routing. Best and second-best results at each operating point are shown in bold and underlined, respectively.

Compressor	Compression	Base	NSR-Feature	NSR-Spatial	Full NSR
ToMe	1.8×	0.6281	<u>0.6351</u>	0.6308	**0.6423**
	2.5×	0.4458	0.4347	**0.5985**	<u>0.5954</u>
	3.4×	0.1831	0.1693	<u>0.4823</u>	**0.4982**
	3.8×	0.1698	0.1632	<u>0.4576</u>	**0.4623**
PiToMe	1.8×	0.6148	0.6138	**0.6258**	<u>0.6244</u>
	2.5×	0.5577	0.5687	<u>0.5873</u>	**0.5919**
	3.4×	0.4370	0.4162	<u>0.5008</u>	**0.5402**
	3.8×	0.3732	0.3783	<u>0.4810</u>	**0.5066**
MPM	1.8×	0.6199	0.6187	<u>0.6211</u>	**0.6264**
	2.4×	<u>0.5887</u>	0.5837	0.5778	**0.5974**
	3.0×	0.5493	0.5468	<u>0.5647</u>	**0.5763**
	3.3×	0.5134	0.5325	<u>0.5367</u>	**0.5485**

uses the resulting codes to update merged tokens, allowing member differences to influence subsequent backbone computation. Spatial Routing uses the member codes to generate feature corrections at the corresponding original patch positions. We evaluate the two paths separately in NSR-Feature and NSR-Spatial, respectively, and jointly in Full NSR. All three NSR variants use the same null-space residual construction. Comparisons among these variants therefore examine how the residuals are integrated, with the original baseline providing a reference without the NSR pathway.

As shown in Table 13, The strong performance of NSR-Spatial in several configurations shows that spatial utilization of null-space residuals can provide substantial benefits. Full NSR achieves the highest mIoU in ten of the twelve settings, indicating an advantage from jointly using the two paths in most evaluated configurations. For example, at the $3.4\times$ and $3.8\times$ operating points of PiToMe, jointly enabling the two paths improves mIoU from 0.5008 and 0.4810 with the spatial path alone to 0.5402 and 0.5066, respectively.

At the $2.4\times$ operating point of MPM, the feature-only and spatial-only variants achieve mIoU values of 0.5837 and 0.5778, respectively, neither exceeding the baseline value of 0.5887, whereas Full NSR reaches 0.5974. These results support a complementary role for the two paths in some configurations: jointly using null-space residual information through local feature correction and compensation at the corresponding spatial positions can yield gains not achieved by either path alone.

Table 14: Input-source ablation within the token-merging instantiation of NSR on Pascal VOC. RawSource directly encodes pre-merge member features while retaining the complete NSR architecture. Base denotes the corresponding original merging baseline. WP follows the operating-point labels used in the main experiments. Values are mIoU on a 0–1 scale; higher is better. The best result at each operating point is shown in bold.

Compressor	WP	Base	RawSource	Full NSR
ToMe	1.8×	0.6281	0.6349	**0.6423**
	2.5×	0.4458	0.5930	**0.5954**
	3.4×	0.1831	0.4912	**0.4982**
	3.8×	0.1698	0.4490	**0.4623**
PiToMe	1.8×	0.6148	0.6201	**0.6244**
	2.5×	0.5577	0.5899	**0.5919**
	3.4×	0.4370	0.5034	**0.5402**
	3.8×	0.3732	0.4972	**0.5066**
MPM	1.8×	0.6199	0.6226	**0.6264**
	2.4×	0.5887	0.5839	**0.5974**
	3.0×	0.5493	0.5658	**0.5763**
	3.3×	0.5134	0.5435	**0.5485**

G.3 Ablation of the Input Source within NSR

We examine the input-source choice within the token-merging instantiation of the complete NSR framework. To this end, we construct a raw-member-source control, denoted RawSource, by replacing $z_i = x_i - y_G$ with x_i as the source supplied to both the encoder and the source input of the gating network. The member and group representations that additionally condition the gate remain unchanged. RawSource retains the same normalization modules, encoder, gating network, feature decoder, spatial routing, and final spatial decoder as Full NSR. The two variants also use the same valid-member masks and singleton handling, parameterization, initialization, training protocol, and predefined compression configurations. RawSource retains within-group variations. Full NSR versus RawSource evaluates this internal source choice, whereas Full NSR versus Base evaluates the complete NSR method.

As shown in Table 14, Full NSR achieves higher mIoU than RawSource in all twelve evaluated configurations, with improvements ranging from 0.20 to 3.68 mIoU points. For example, at the $3.4\times$ operating point of PiToMe, Full NSR achieves 0.5402 mIoU compared with 0.5034 for RawSource. RawSource also improves over the corresponding original baseline in eleven configurations, showing that directly retaining and processing member information is beneficial in most evaluated settings. The additional gains of Full NSR support the practical value of explicit null-space residual extraction within the same complete pathway architecture.

A possible explanation is that explicit residual extraction reduces the need for the complementary encoder to represent components already available through the current merged representation. This may allow the finite-dimensional codes to focus more effectively on operator-invisible member variations, while the member and group representations continue to provide context for gating.

G.4 Complementary Pathway and Input-Source Ablation for Graph Aggregation

We compare Base, Post-aggregation-only, RawSource, and Full NSR on Roman-empire using GCN, GraphSAGE, and GIN. Post-aggregation-only generates corrections solely from the current aggregate, without accessing member representations or residuals, while matching the trainable parameter count of Full NSR. RawSource replaces Z_v in Eq. equation 5 with the unprojected messages M_v, including the corresponding gate input, while retaining the member set, contextual inputs, remaining branch structure, and initialization. All configurations share the same data splits, backbone settings, optimization protocol, training budget, and validation-based model selection. Table 15 reports the results; Base and Full NSR are the same as in Table 2.

Table 15: Graph-aggregation ablations on Roman-empire. Values are test accuracy (%), reported as the mean and sample standard deviation over the ten official splits. The best result in each column is shown in bold.

Configuration	GCN	GraphSAGE	GIN
Base	72.05 ± 0.64	81.59 ± 0.55	74.00 ± 0.79
Post-aggregation-only	75.30 ± 0.69	84.00 ± 0.46	76.01 ± 0.62
RawSource	79.21 ± 0.41	85.57 ± 0.64	80.69 ± 0.51
Full NSR	$\mathbf{82.90 \pm 0.71}$	$\mathbf{87.71 \pm 0.61}$	$\mathbf{82.25 \pm 0.65}$

All three backbones exhibit the same mean-performance ordering: Base $<$ Post-aggregation-only $<$ RawSource $<$ Full NSR. Additional post-aggregation processing helps, but Full NSR exceeds this control by 7.60, 3.71, and 6.24 percentage points for GCN, GraphSAGE, and GIN, respectively. RawSource's advantage over Post-aggregation-only supports the value of retaining and processing member information. Full NSR further improves over RawSource by 3.69, 2.14, and 1.56 percentage points. Since RawSource also retains member differences, this comparison supports the practical value of explicitly extracting operator-defined residuals within the same branch architecture.

Unlike the visual instantiation, the graph branch contains no spatial memory or write-back path (Figure 2). These results therefore complement the token-merging ablations: both the complete pathway and its residual-source choice remain beneficial without visual spatial integration. Together, the findings support the NSR framework of combining operator-defined complementary information with member-level processing and application-specific integration.

H Heterophilic Dataset Statistics

To complement the heterophilic graph experiments in the main text, Table 16 presents the statistics of the five main heterophilic graph benchmarks, including graph structural properties, node attributes, and label-related metrics. These statistics are taken from Platonov et al. (2023b) and provide additional context for understanding the experimental results across different datasets.

Table 16: Statistics of the five heterophilic graph benchmarks, as reported by Platonov et al. (2023b).

Statistic	Roman-empire	Amazon-ratings	Minesweeper	Tolokers	Questions
Nodes	22662	24492	10000	11758	48921
Edges	32927	93050	39402	519000	153540
Avg. degree	2.91	7.60	7.88	88.28	6.28
Global clustering	0.29	0.32	0.43	0.23	0.02
Avg. local clustering	0.39	0.58	0.44	0.53	0.03
Diameter	6824	46	99	11	16
Node features	300	300	7	10	301
Classes	18	5	2	2	2
Edge homophily	0.05	0.38	0.68	0.59	0.84
Adjusted homophily	-0.05	0.14	0.01	0.09	0.02
Label informativeness	0.11	0.04	0.00	0.01	0.00

Notably, Roman-empire exhibits relatively high Label Informativeness (LI), and NSR achieves more pronounced performance gains on this dataset. This observation is consistent with the discussion in the main text that task-relevant information may influence the performance benefits of NSR, providing additional context for understanding the experimental results across different datasets.