TASKHC: FUNCTION-STRUCTURED PERSISTENT STATES WITH OPERATION-DEPENDENT ACCESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Classification and localization require different visual evidence, yet standard detection transformers accumulate and integrate information for both tasks in a single shared state per query. We introduce TaskHC, which extends Hyper-Connections by organizing persistent residual streams into functional groups. Task-specific working states are reconstructed from designated groups, and task updates are written back to them; shared modules reconstruct a joint working state from the full stream bank and mix residual information across groups. Architectural ablations show that group-specific reconstruction and writeback improve detection over access to the full stream bank with the task-specific modules held fixed, and that shared modules benefit from cross-group mixing. Sampling analyses and frozen-state probes reveal specialization in evidence acquisition and task-information accessibility. On COCO, TaskHC improves matched DEIM–D-FINE and DEIM–RT-DETRv2 baselines. For DEIM–D-FINE-L, AP increases from 54.45 to 54.68 with 0.53 ms additional latency on an NVIDIA T4 using TensorRT FP16.

1 INTRODUCTION

Classification and localization describe the same object but emphasize different visual evidence. Classification relies on category-discriminative cues, whereas localization requires precise information about spatial extent and geometry. These different requirements have motivated task-specific features, prediction heads, and attention mechanisms in object detection (Jiang et al., 2018; Wu et al., 2020; Song et al., 2020; Li et al., 2020; Zhang et al., 2021; Feng et al., 2021). At the same time, both tasks contribute information about the object that shared computation can use. The representational challenge is to preserve task specialization while allowing information accumulated by the two tasks to be integrated.

Detection transformers make this challenge explicit through iterative query refinement. Each object query carries a state across decoder layers, accumulating information that supports classification and localization. Existing task-specialized decoders introduce task-aware queries and separate computation paths while retaining shared operations for interaction (He & Todorovic, 2022; Zhang et al., 2023; Zhou et al., 2023). This motivates a complementary question: how should the persistent representation be organized so that task-specific modules can draw on their own accumulated information while shared modules integrate information from both tasks?

Hyper-Connections (HC) provide a basis for studying this organization (Zhu et al., 2025). HC expands a residual pathway into a bank of persistent streams, reconstructs a module's working state from learned combinations of these streams, and distributes its update back to them. This separates the representation carried across modules from the working state used for each computation. Within such a bank, separate task controllers can reconstruct distinct working states from all streams and distribute both tasks' updates throughout the bank. Explicit functional grouping makes a further assignment: designated streams serve as each task's reconstruction source and update destination, allowing task information to be accumulated and reused within the same group across decoder layers.

We introduce **TaskHC**, which combines this persistent functional grouping with reconstruction and updates determined by each module's computational role. The stream bank is organized into classification and localization groups. Task-specific working states are reconstructed from the corre-

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.

sponding group, and newly computed task updates return to that group. Shared modules reconstruct a joint working state from the full stream bank and distribute their updates across it; residual mixing also combines existing stream states across functional groups. Thus, task-specific processing acquires and accumulates information within designated groups, while shared processing integrates information maintained by both tasks. Each module receives a reconstructed working state at the original hidden width.

Our architectural studies establish how these two forms of processing work together. Task-specific cross-attention and prediction are most effective alongside shared self-attention and feed-forward transformation. With the task-specific modules held fixed, group-specific reconstruction and writeback improve detection over access to the full stream bank. Cross-group residual mixing at shared modules further improves detection. These findings connect explicit task-group reconstruction and writeback with shared integration of previously accumulated information. Sampling analyses and frozen-state probes reveal corresponding specialization in visual-evidence acquisition and task-information accessibility.

We evaluate TaskHC on COCO with matched DEIM–D-FINE and DEIM–RT-DETRv2 baselines. The gains across both decoder architectures demonstrate the applicability of this state organization to real-time detection. On DEIM–D-FINE-L, TaskHC improves AP from 54.45 to 54.68 with an additional 0.53 ms of latency on an NVIDIA T4 using TensorRT FP16.

Our contributions are threefold:

- **Function-structured persistent states.** We introduce TaskHC, which assigns persistent HC streams to functional groups that jointly specify task-state reconstruction sources and update destinations across decoder layers.
- **Module-dependent specialization and interaction.** We establish how group-specific reconstruction and writeback work with joint-state reconstruction and cross-group residual mixing to combine task-specific processing with shared computation.
- **Evidence of learned functional specialization.** Sampling analyses and frozen-state probes characterize task-related differences in evidence acquisition and information accessibility, connecting the designed functional organization to the behavior of the trained detector.

2 Related Work

2.1 Real-time Detection Transformers

DETR formulates object detection as end-to-end set prediction, and RT-DETR and RT-DETRv2 adapt this paradigm to real-time detection through efficient architectures and improved training strategies (Carion et al., 2020; Zhao et al., 2024; Lv et al., 2024). D-FINE improves localization through fine-grained distribution refinement, while DEIM improves matching and training efficiency (Peng et al., 2025; Huang et al., 2025b). DEIMv2 and RT-DETRv4 further use stronger pretrained representations or knowledge transfer (Huang et al., 2025a; Liao et al., 2026). TaskHC complements these developments by organizing the persistent decoder representation to support task-specific processing and shared computation.

2.2 Classification–localization specialization

Task-specific features and heads accommodate the different information needs of classification and localization, while localization-quality estimation and task-aligned learning improve their prediction consistency (Wu et al., 2020; Song et al., 2020; Jiang et al., 2018; Li et al., 2020; Zhang et al., 2021; Feng et al., 2021). In transformer detectors, DESTR introduces separate classification and regression cross-attention branches (He & Todorovic, 2022). Decoupled DETR combines task-aware queries for shared self-attention before separate cross-attention and feed-forward processing; D2Q-DETR combines shared self-attention with task-specific branches and sums their outputs before the next layer (Zhang et al., 2023; Zhou et al., 2023).

These designs provide task-specific representations and mechanisms for interaction. TaskHC organizes persistent residual streams into functional groups from which modules reconstruct their work-

ing states. Each task reads from and returns updates to its designated group, while shared modules reconstruct one joint state from the full bank. A functional group can retain several streams without requiring a computation branch for every stream. This organization makes the persistent sources and destinations of task information explicit alongside the division of decoder computation.

2.3 Hyper-Connections and Structured Persistent States

HC expands residual pathways into multiple streams and learns mappings for input reconstruction, residual propagation, and output distribution (Zhu et al., 2025). mHC constrains residual mixing to stabilize propagation, and mHC-lite provides a direct parameterization of this transport (Xie et al., 2026; Yang & Gao, 2026). Structured variants use known input structure to organize streams, as in White-Box mHC and mHC-HSI (Zhu et al., 2026b;a). IntTravel introduces task-conditioned read and residual-mixing mappings, aggregating task contributions before shared computation and updating a common stream bank (Yan et al., 2026). TaskHC assigns classification and localization responsibilities to persistent stream groups and uses those groups to specify the support of task-specific reconstruction and writeback. Joint operations retain full-bank access and residual mixing, combining functional state organization with shared processing across groups.

3 Method

3.1 Function-Structured Persistent States

TaskHC organizes each query's persistent representation into functional stream groups and reconstructs working states according to the role of each decoder module (Figure 1).

A conventional detection decoder propagates a hidden state $h \in \mathbb{R}^C$ for each object query. Following Hyper-Connections (Zhu et al., 2025; Xie et al., 2026), we maintain a bank of n residual streams,

$$X = [x_1; \ldots; x_n] \in \mathbb{R}^{n \times C}, \qquad x_r \in \mathbb{R}^C. \tag{1}$$

We omit batch and query indices for clarity. Each query maintains its own bank X, which is propagated and updated across decoder modules. A working state $z \in \mathbb{R}^C$ is reconstructed from this bank to serve the current computation. Self-attention operates across the working states reconstructed for different queries.

For an even number of streams, TaskHC assigns equal-sized groups to classification and localization,

$$\mathcal{G}_{\text{cls}} = \{1, \ldots, n/2\}, \qquad \mathcal{G}_{\text{loc}} = \{n/2 + 1, \ldots, n\}. \tag{2}$$

The group assignment persists throughout the decoder. Each group's responsibility is defined by the task-specific computations that read from it, the updates returned to it, and the prediction path that consumes its reconstructed state. A group can contain multiple streams, providing several persistent carriers for its designated function. Shared modules reconstruct a joint working state from the full bank, integrating information from both functional groups.

Before the first decoder layer, we initialize the bank from the incoming query representation using Learned Stream Scaling (Alimaskina et al., 2026),

$$X^{(0)} = [s_1 \odot h^{(0)}; \ldots; s_n \odot h^{(0)}], \qquad s_r \in \mathbb{R}^C, \tag{3}$$

where the learnable stream-specific scales break the symmetry of replicated initialization. Our default configuration uses four streams, with two assigned to each functional group.

3.2 Joint and Task-Specific State Updates

For a module that updates the bank, TaskHC uses the HC/mHC reconstruction, writeback, and residual mappings (Xie et al., 2026), with functional groups determining their scopes (Figure 2):

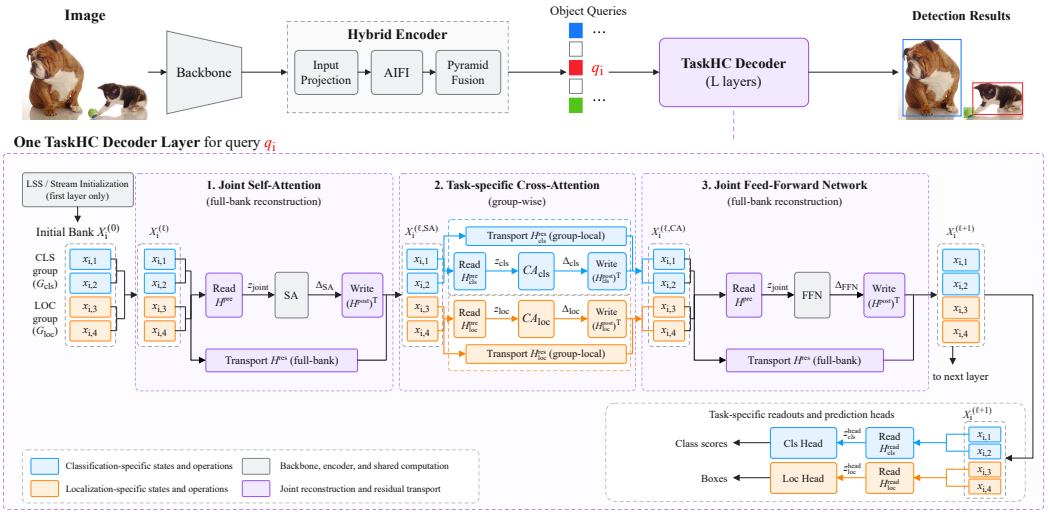

Figure 1: TaskHC architecture. Persistent streams are organized into classification and localization groups. Cross-attention and prediction heads use group-specific working states, while shared self-attention and feed-forward modules use joint working states reconstructed from the full stream bank.

$$z = H^{\text{pre}} X, \qquad \Delta = F(z), \qquad X^{+} = H^{\text{res}} X + (H^{\text{post}})^{\top} \Delta. \tag{4}$$

Here, $H^{\text{pre}}, H^{\text{post}} \in \mathbb{R}^{1\times n}$ and $H^{\text{res}} \in \mathbb{R}^{n\times n}$. The function F produces the module-specific increment Δ to be written back. The pre-mapping reconstructs the module input, the post-mapping distributes the resulting update, and the residual mapping recomposes the existing streams. Residual transport carries previously stored information, while writeback incorporates the module update. Prediction readouts use only the reconstruction step.

Joint reconstruction and update. A shared operation uses the full bank as its scope. It reconstructs one working state $z_{\text{joint}} = H^{\text{pre}}_{\text{joint}} X$, applies one shared module, and distributes the resulting update across the bank. The residual mapping also operates across the full bank. The shared module operates at the original hidden width C.

Task-specific reconstruction and update. A specialized operation uses one controller and one computational branch for each functional group. For $t \in \{\text{cls}, \text{loc}\}$,

$$z_t = H^{\text{pre}}_t X, \qquad \Delta_t = F_t(z_t), \qquad \text{supp}(H^{\text{pre}}_t), \text{supp}(H^{\text{post}}_t) \subseteq \mathcal{G}_t. \tag{5}$$

Here, the group-local mappings are written in full-bank coordinates with zeros outside their designated groups. Both branches read from the incoming bank and return their updates to the corresponding groups:

$$X^{+} = H^{\text{res}} X + \sum_{t\in\{\text{cls},\text{loc}\}} (H^{\text{post}}_t)^{\top} \Delta_t. \tag{6}$$

For this mode, residual transport is also computed within each group, giving a block-diagonal H^{res}. Cross-group interaction is provided by the joint operations. The module's computational role determines which streams may contribute to reconstruction and receive updates; the controller learns input-dependent weights within that scope.

Connection parameterization. Following mHC-lite (Yang & Gao, 2026), controllers generate connection coefficients from the flattened, RMS-normalized streams within the selected scope. Sigmoid and scaled-sigmoid gates produce the pre- and post-mappings, while softmax weights combine fixed permutation matrices into H^{res}. These mappings act on the original stream values. Reconstruction

and writeback use one scalar coefficient per stream, shared across channels. Figure 2(c) and Appendix A.1 detail the parameterization; Appendix B explores channel-wise extensions.

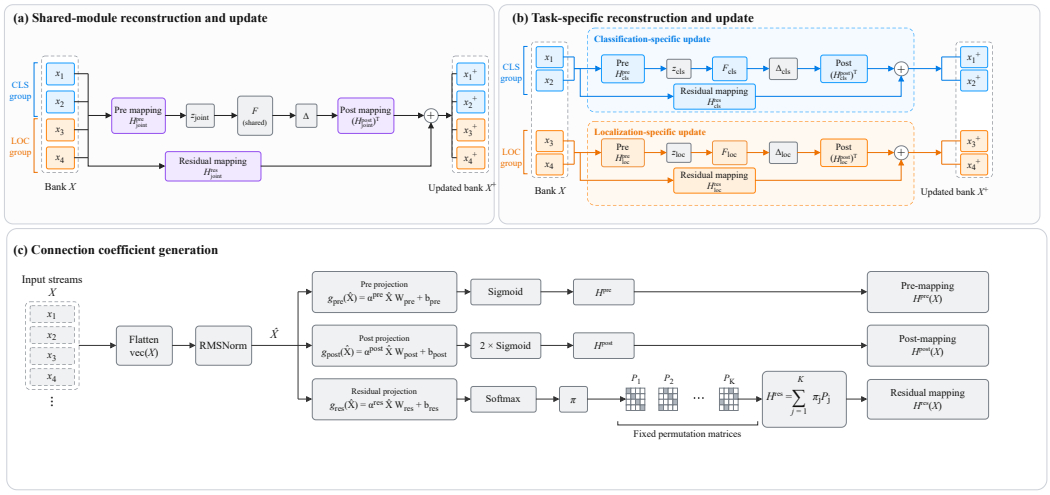

Figure 2: State reconstruction and update in TaskHC. Shared modules reconstruct working states and update residual streams across the full stream bank (a), whereas task-specific modules operate within their designated functional groups (b). Reconstruction, writeback, and residual-transport coefficients are generated from the corresponding stream context (c).

3.3 Instantiation in the Detection Decoder

We instantiate the joint and task-specific modes in the decoder sequence shown in Figure 1. Self-attention and FFN use joint reconstruction and updates; cross-attention uses task-specific reconstruction and updates, while prediction uses group-specific readouts.

Joint query interaction. Each object query reconstructs a joint working state from its full bank. Self-attention operates across these states, allowing query interactions to use information accumulated by both tasks. Its output updates the bank through joint writeback and full-bank residual transport.

Task-specific evidence acquisition. The classification and localization cross-attention branches attend to the same encoder memory using their respective task-specific working states, reconstructed from the corresponding functional groups. Each branch acquires visual evidence for its task and writes the resulting update back to its designated group. Appendix A.2 describes how D-FINE's cross-attention gate is incorporated into this update formulation.

Joint feature transformation. After the task-specific updates, a joint working state is reconstructed from the full bank and processed by a shared FFN. The transformation therefore uses information acquired by both task paths. Joint writeback and residual transport update the bank for subsequent computation.

Task-specific prediction. At each decoder layer, group-specific readouts supply the native classification and localization prediction paths without writing an update back to the bank. The detector's native prediction mechanisms, including localization-quality estimation and iterative box refinement, are retained.

4 Experiments

4.1 Experimental Setup

We evaluate TaskHC on DEIM–D-FINE and DEIM–RT-DETRv2. We train on COCO train2017 and evaluate on val2017, using ImageNet-pretrained backbones and matched training settings within

Table 1: Detection accuracy and deployment cost on COCO `val2017`. Baseline and TaskHC models in each pair use the same training settings.

Model	Params (M)	GFLOPs	Lat. (ms)	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
DEIM–D-FINE									
DEIM–D-FINE-N	3.75	7.16	2.52	42.80	60.32	45.83	23.21	46.93	61.42
+ TaskHC	4.18	7.43	2.63	43.04	60.33	46.01	23.44	47.34	61.53
DEIM–D-FINE-S	10.24	25.17	3.73	48.69	65.81	52.86	29.49	52.51	66.00
+ TaskHC	11.48	25.94	4.04	48.94	65.91	53.05	30.74	52.22	66.11
DEIM–D-FINE-M	19.25	56.68	5.41	52.60	69.97	56.99	33.92	56.77	69.97
+ TaskHC	20.91	57.71	6.32	52.74	70.20	57.24	34.56	56.86	69.83
DEIM–D-FINE-L	30.77	91.14	8.37	54.45	72.20	59.18	37.11	58.98	71.78
+ TaskHC	33.26	92.69	8.90	54.68	72.20	59.38	37.67	59.30	71.97
DEIM–D-FINE-X	61.70	202.67	13.95	56.24	73.80	61.16	38.77	60.93	73.88
+ TaskHC	64.19	204.21	14.69	56.33	73.85	61.46	37.63	60.99	73.77
DEIM–RT-DETRv2									
DEIM–RT-DETRv2-R18	19.98	60.21	4.84	49.06	66.06	53.24	32.23	52.53	64.09
+ TaskHC	20.83	63.93	5.41	49.43	66.53	53.74	32.93	52.66	64.36
DEIM–RT-DETRv2-R34	31.24	92.26	6.66	51.10	68.50	55.72	33.53	54.74	66.88
+ TaskHC	32.37	97.22	7.37	51.14	68.67	55.54	33.80	54.22	68.11
DEIM–RT-DETRv2-R50-M*	32.99	99.53	7.13	53.09	71.17	57.63	36.06	57.73	69.53
+ TaskHC	33.85	103.25	7.70	53.05	70.90	57.76	36.08	57.59	69.64
DEIM–RT-DETRv2-R50-L	42.14	133.49	9.57	54.24	72.04	59.01	37.31	58.63	70.96
+ TaskHC	43.78	136.52	10.23	54.45	72.32	59.12	36.98	59.09	70.97
DEIM–RT-DETRv2-R101	74.87	251.43	14.60	55.41	73.47	60.22	38.17	59.87	72.58
+ TaskHC	76.58	258.88	15.40	55.56	73.51	60.70	37.73	59.87	72.91

each baseline/TaskHC pair. We report validation AP, deployment parameters and FLOPs, and inference latency on an NVIDIA T4 with TensorRT FP16. Appendix C details the training and measurement protocols.

4.2 Detection Performance and Deployment Cost

Under matched training settings, TaskHC improves detection across both DEIM–D-FINE and DEIM–RT-DETRv2 (Table 1). All five evaluated D-FINE scales gain 0.09–0.25 AP, while gains on DEIM–RT-DETRv2 reach 0.37 AP with the ResNet-18 backbone. These results support using the same functional grouping and module-dependent state access across different decoder architectures and model capacities. Across seeds 0–4 on DEIM–D-FINE-L, the paired AP improvement has a mean of 0.18 and a sample standard deviation of 0.19. Appendix D reports the individual runs and summary statistics.

Across the evaluated configurations, TaskHC adds 0.43–2.49M parameters, 0.27–7.45 GFLOPs, and 0.11–0.91 ms of inference latency. We reduce deployment overhead by jointly executing the two cross-attention branches and fusing operators for residual-stream mixing and writeback. Appendix C details these optimizations.

4.3 Functional Organization of Persistent States

Placement of task-specific computation We examine how decoder modules should use persistent information for task-specific and shared computation (Table 2). Introducing a generic four-stream mHC-lite bank without functional grouping lowers AP by 0.30 relative to the baseline: every module still reconstructs a single joint working state from all streams, leaving the additional persistent capacity without an explicit task assignment. Although effective in earlier DETR architectures, a Decoupled DETR-like design with separate task states and concatenation lowers AP by 0.21 when adapted to DEIM–D-FINE-L, while adding 9.91M parameters and 5.94 GFLOPs. These outcomes motivate revisiting how task information is retained and combined in a real-time decoder.

Within the four-stream bank, we assign classification and localization to separate functional groups and vary which modules reconstruct task-specific working states. Task-specific prediction readouts alone improve over the generic bank but remain below the baseline. Extending task-specific reconstruction and writeback to cross-attention adds 0.40 AP. Task specialization now governs both how

Table 2: Task-specific state access in DEIM–D-FINE-L. In module columns, checkmarks denote task-specific access and dashes denote joint access.

Method	Stream bank	Prediction inputs	Cross-attention	Self-attention	Feed-forward	Params (M)	GFLOPs	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
DEIM–D-FINE-L baseline	–	–	–	–	–	30.77	91.14	54.45	72.20	59.18
Decoupled DETR-like	–	✓	✓	–	✓	40.68	97.08	54.24	71.76	58.91
mHC-lite, shared modules	✓	–	–	–	–	31.41	91.59	54.15	71.78	58.84
Prediction only	✓	✓	–	–	–	31.40	91.58	54.28	71.91	59.07
Prediction + CA	✓	✓	✓	–	–	33.26	92.69	**54.68**	72.20	59.38
Prediction + CA + FFN	✓	✓	✓	–	✓	36.26	94.47	54.53	71.91	59.28
Prediction + CA + SA	✓	✓	✓	✓	–	37.83	95.42	54.54	72.14	59.22
Prediction + CA + SA + FFN	✓	✓	✓	✓	✓	40.82	97.20	54.27	71.86	58.95

visual evidence is acquired and how it is used for prediction: each task's cross-attention acquires encoder information that accumulates in the same functional group supplying its prediction readout.

Extending task specialization to self-attention or FFN reduces accuracy, with a larger decline when both are specialized (Table 2). These results show that broader task separation does not necessarily improve detection; instead, access to persistent states should reflect each module's computational role. Cross-attention and prediction use task-specific information to acquire visual evidence and make task-specific predictions, whereas shared self-attention and FFN jointly process information accumulated by both tasks for query interaction and feature transformation. TaskHC thus maintains persistent task responsibility through functional groups while allowing each module to access and process their information according to its role.

Functional grouping for task-specific reconstruction and writeback With task-specific cross-attention and prediction fixed, we test why their reconstruction sources and update destinations should be assigned to explicit functional groups (Table 3a). Under full-bank access, independent controllers can already reconstruct distinct task states and distribute their updates across all streams, learning the task-to-stream association through reconstruction and writeback weights. Explicit grouping specifies the streams that supply each task's state and receive its update, while retaining learned weights within those scopes.

Group-specific reconstruction and writeback improve AP by 0.23 over full-bank access with the same task-specific modules. This comparison connects the benefit of grouping to the organization of accumulated information: each task repeatedly reconstructs its state from the group that also retains its new evidence. The functional group thus carries task responsibility across decoder layers, linking the information accumulated by a task to its subsequent computation.

Restricting only reconstruction or only writeback to the designated group performs worse than restricting both. With group-specific reconstruction and full-bank writeback, task updates are distributed beyond the source of subsequent task states; with full-bank reconstruction and group-specific writeback, updates have a designated destination but task states still draw on all streams. The two scopes serve complementary roles in accumulation and reuse: the group retains newly acquired task information and supplies it when that task next reconstructs a working state.

Cross-group fusion through residual mixing We next examine how shared modules integrate information accumulated by the two functional groups. Keeping working-state reconstruction and writeback fixed, we compare identity transport (IHC), intra-group mixing, and full-bank mixing (Table 3b). IHC carries each incoming stream unchanged through the residual path. The two learned variants retain group-local transport at task-specific cross-attention and differ in the mixing scope of shared self-attention and FFN.

Full-bank mixing improves AP by 0.10 over IHC and by 0.24 over intra-group mixing. Intra-group mixing replaces identity carry with weighted combinations of streams from the same task, but leaves the two groups disconnected along the residual path. It therefore changes the information carried by each stream without gaining access to the other task's accumulated information, which may explain its lower accuracy than IHC. Full-bank mixing allows this cross-task exchange, enabling shared modules to combine the complementary information retained by the two functional groups.

Table 3: Functional-group access and residual mixing in DEIM–D-FINE-L.

(a) Task-state reconstruction and writeback.

Reconstruction source	Writeback destination	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
Task group	Task group	**54.68**	72.20	59.38
Full bank	Task group	54.51	72.31	59.16
Task group	Full bank	54.32	71.91	58.93
Full bank	Full bank	54.45	71.89	59.38

(b) Residual mixing at shared modules.

Residual mixing	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
Identity (IHC)	54.58	72.24	59.33
Intra-group mixing	54.44	72.17	59.29
Full-bank mixing	**54.68**	72.20	59.38

These results support combining explicit task responsibility with cross-group information exchange. Task-specific modules reconstruct from and write back to their designated groups, connecting task-specific evidence acquisition and reuse. At shared modules, joint reconstruction brings information from both groups into the current computation, while full-bank residual mixing integrates their accumulated information into the persistent states carried forward.

Multiple persistent streams per function A functional group can retain several residual streams even though each task-specific operation consumes one reconstructed working state. We vary the number of streams per group at a fixed working-state width to examine the benefit of this additional storage capacity (Table 4). Increasing from one to two streams per group improves AP from 54.39 to 54.68 with mHC-lite and from 54.41 to 54.58 with IHC. The improvement under both transport mechanisms supports retaining multiple streams for each function. With IHC, further expansion to four streams per group lowers AP to 54.32. Two streams per group provide the most effective capacity in these comparisons with stream-wise scalar routing.

Further expansion is constrained by the cost of the enumerated mHC-lite residual mapping. With eight total streams, the twelve joint residual predictors alone require at least 991.39M parameters (Appendix A). Together with the stream-count comparisons, this supports our choice of four persistent streams, with two assigned to each functional group.

Table 4: Persistent streams per functional group.

		mHC-lite		IHC	
Total streams	Streams per group	Params (M)	AP^{val}	Params (M)	AP^{val}
2	1	32.86	54.39	32.84	54.41
4	2	33.26	54.68	32.95	54.58
8	4	$\geq 991.39^{\dagger}$	–	33.39	54.32

†: Parameter-count lower bound; not trained due to excessive parameter and computational costs.

4.4 Learned Task Specialization

Task-specific evidence acquisition. We examine whether the classification and localization branches acquire visual evidence suited to their respective tasks. Figure 3 shows that classification sampling emphasizes object interiors, whereas localization sampling places greater emphasis on peripheral regions. These patterns are consistent with the different evidence needed to recognize object categories and establish spatial extent. Task-specific working states thus support differentiated evidence acquisition: each branch gathers information relevant to its prediction task, and the resulting updates accumulate in its designated functional group for subsequent computation.

Task-information accessibility. We examine whether the two prediction representations make information for their assigned tasks more readily accessible. We freeze the trained detector and fit classification and localization probes to both the CLS and LOC prediction inputs, using the same probe architecture and training protocol within each task. The preferred representation depends on the prediction task: classification probes perform better on the CLS input, whereas localization probes perform better on the LOC input. Both advantages emerge early in probe fitting and persist as the learning curves plateau (Figure 4). On COCO val2017, the corresponding gains are 1.39 per-

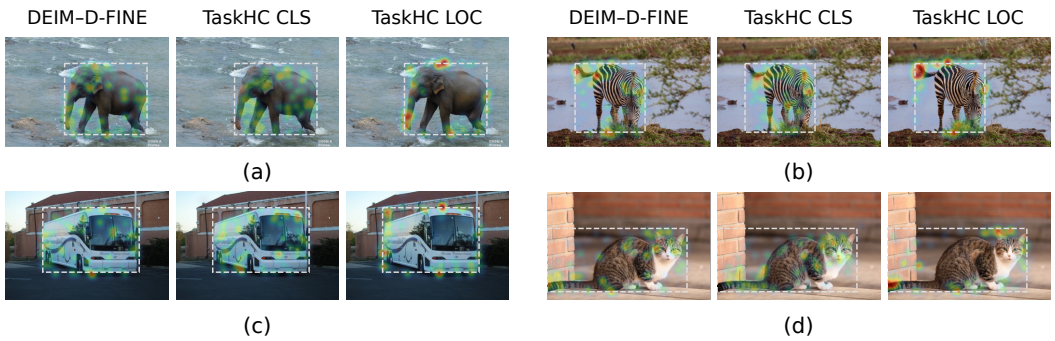

Figure 3: Task-specific evidence acquisition. Sampling locations for DEIM–D-FINE and the classification and localization branches of TaskHC on the same objects.

centage points in classification macro accuracy and 3.87 points in localization accuracy at IoU ≥ 0.5 (Appendix E.2). These task-aligned advantages show that functional specialization is reflected in the information recoverable from the frozen representations. Together with the sampling analysis, the probes connect differentiated evidence acquisition to task-specialized representations available for prediction.

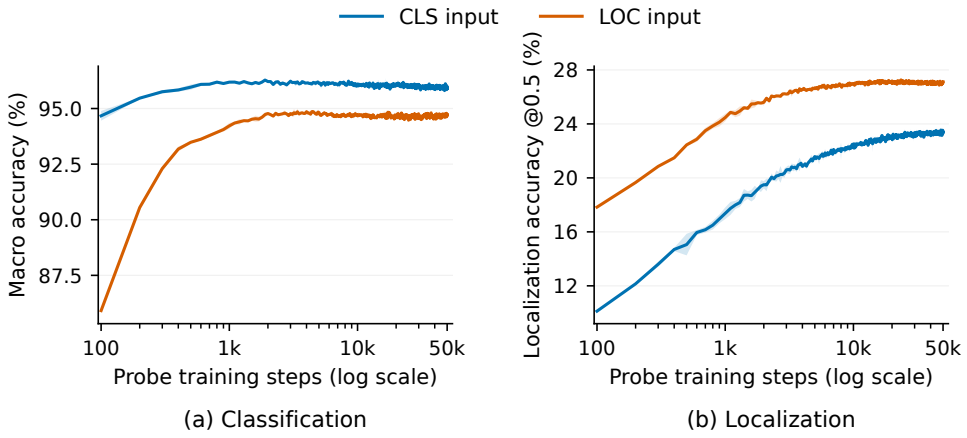

Figure 4: Task-information accessibility in frozen prediction inputs. Classification (a) and localization (b) probes use either classification or localization inputs. Curves show calibration-set means and shaded bands indicate one standard deviation across three probe seeds.

5 Conclusion

We introduced TaskHC, which organizes persistent residual streams into functional groups and uses module-dependent reconstruction and writeback to connect task-specific information acquisition, accumulation, and prediction. Shared modules integrate information across groups through joint-state reconstruction and residual mixing. Architectural ablations and representation analyses support this combination of functional specialization and shared processing, while matched evaluations show improved detection accuracy in two real-time detector architectures. Our study instantiates this representation mechanism with classification and localization in real-time detection transformers. More broadly, this approach may also be applicable to settings with multiple interdependent tasks that require different information for acquisition and prediction, while benefiting from shared processing. Functional grouping and module-dependent access offer a way to organize how such information is retained, used, and updated across computations.

Reproducibility statement

The appendix specifies the training recipes, module configurations, static-complexity convention, and frozen-state probe protocol. The method equations distinguish raw residual states, controller coefficients, native module updates, and prediction-head inputs.

AI use statement

The authors developed the research ideas, formulated the core claims, and designed the experiments. Under the authors' direction, generative AI tools assisted with implementing and reviewing experimental code, literature retrieval, preparing and editing the manuscript, translation, and articulating and checking interpretations of experimental results. The authors take responsibility for the final manuscript, code, and scientific conclusions.

References

Ekaterina Alimaskina, Gleb Molodtsov, and Aleksandr Beznosikov. Analyzing Stream Collapse in Hyper-Connections: From Diagnosis to Mitigation. In *Workshop on Weight-Space Symmetries at the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2606.03483. Workshop paper; arXiv:2606.03483.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pp. 213–229, 2020. doi: 10.1007/978-3-030-58452-8_13. URL https://www.ecva.net/papers/eccv_2020/papers_ECCV/html/832_ECCV_2020_paper.php.

Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R. Scott, and Weilin Huang. TOOD: Task-Aligned One-Stage Object Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3510–3519, October 2021. doi: 10.1109/iccv48922.2021.00349. URL https://openaccess.thecvf.com/content/ICCV2021/html/Feng_TOOD_Task-Aligned_One-Stage_Object_Detection_ICCV_2021_paper.html.

Shasha Han, Chong Li, Xinning Wang, and Xuebo Li. YOLO-ULM: Ultra-Lightweight Models for Real-Time Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/papers/Han_YOLO-ULM_Ultra-Lightweight_Models_for_Real-Time_Object_Detection_CVPR_2026_paper.pdf.

Liqiang He and Sinisa Todorovic. DESTR: Object Detection With Split Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9377–9386, June 2022. doi: 10.1109/CVPR52688.2022.00916. URL https://openaccess.thecvf.com/content/CVPR2022/html/He_DESTR_Object_Detection_With_Split_Transformer_CVPR_2022_paper.html.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html.

Jiannan Huang, Aditya Kane, Fengzhe Zhou, Yunchao Wei, and Humphrey Shi. Revisiting Real-Time Detection Transformer with Efficient Encoder Design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings*, pp. 6859–6868, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026F/html/Huang_Revisiting_Real-Time_Detection_Transformer_with_Efficient_Encoder_Design_CVPRF_2026_paper.html.

Shihua Huang, Yongjie Hou, Longfei Liu, Xuanlong Yu, and Xi Shen. Real-Time Object Detection Meets DINOv3. arXiv preprint arXiv:2509.20787, 2025a. URL https://arxiv.org/abs/2509.20787v4. Version 4, revised January 26, 2026.

Shihua Huang, Zhichao Lu, Xiaodong Cun, Yongjun Yu, Xiao Zhou, and Xi Shen. DEIM: DETR with Improved Matching for Fast Convergence. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15162–15171, June 2025b. doi: 10.1109/

CVPR52734.2025.01412. URL https://openaccess.thecvf.com/content/CVPR2025/html/Huang_DEIM_DETR_with_Improved_Matching_for_Fast_Convergence_CVPR_2025_paper.html.

Borui Jiang, Ruixuan Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. Acquisition of Localization Confidence for Accurate Object Detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–799, September 2018. doi: 10.1007/978-3-030-01264-9_48. URL https://openaccess.thecvf.com/content_ECCV_2018/html/Borui_Jiang_Acquisition_of_Localization_ECCV_2018_paper.html.

Glenn Jocher, Jing Qiu, Mengyu Liu, Shuai Lyu, Fatih Cagatay Akyon, and Muhammet Esat Kalfaoglu. Ultralytics YOLO26: Unified Real-Time End-to-End Vision Models. arXiv preprint arXiv:2606.03748, 2026. URL https://arxiv.org/abs/2606.03748.

Mengqi Lei, Siqi Li, Yihong Wu, Han Hu, You Zhou, Xinhu Zheng, Guiguang Ding, Shaoyi Du, Zongze Wu, and Yue Gao. YOLOv13: Real-Time Object Detection with Hypergraph-Enhanced Adaptive Visual Perception. arXiv preprint arXiv:2506.17733, 2025. URL https://arxiv.org/abs/2506.17733.

Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection. In *NeurIPS*, 2020.

Zijun Liao, Yian Zhao, Xin Shan, Yu Yan, Chang Liu, Lei Lu, Xiangyang Ji, and Jie Chen. RT-DETRv4: Painlessly Furthering Real-Time Object Detection with Vision Foundation Models. In *European Conference on Computer Vision*, 2026. URL https://eccv.ecva.net/Conferences/2026/AcceptedPapers.

Wenyu Lv, Yian Zhao, Qinyao Chang, Kui Huang, Guanzhong Wang, and Yi Liu. RT-DETRv2: Improved Baseline with Bag-of-Freebies for Real-Time Detection Transformer. arXiv preprint arXiv:2407.17140, 2024. URL https://arxiv.org/abs/2407.17140.

Yansong Peng, Hebei Li, Peixi Wu, Yueyi Zhang, Xiaoyan Sun, and Feng Wu. D-FINE: Redefine Regression Task of DETRs as Fine-grained Distribution Refinement. In *International Conference on Learning Representations*, 2025.

Guanglu Song, Yu Liu, and Xiaogang Wang. Revisiting the Sibling Head in Object Detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11563–11572, June 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Song_Revisiting_the_Sibling_Head_in_Object_Detector_CVPR_2020_paper.html.

Yunjie Tian, Qixiang Ye, and David Doermann. YOLOv12: Attention-Centric Real-Time Object Detectors. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 78433–78457. Curran Associates, Inc., 2025. doi: 10.52202/085713-2627. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/7103444259031cc58051f8c9a4868533-Abstract-Conference.html.

Ultralytics. Ultralytics YOLOv5 v7.0. Software repository, 2022. URL https://github.com/ultralytics/yolov5/tree/v7.0.

Ultralytics. Ultralytics YOLOv8. Software documentation, 2023. URL https://docs.ultralytics.com/models/yolov8/.

Ultralytics. Ultralytics YOLO11. Official model documentation, 2024. URL https://docs.ultralytics.com/models/yolo11/.

Shuo Wang, Chunlong Xia, Feng Lv, and Yifeng Shi. RT-DETRv3: Real-Time End-to-End Object Detection with Hierarchical Dense Positive Supervision. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pp. 1628–1636, February 2025. URL https://openaccess.thecvf.com/content/WACV2025/html/Wang_RT-DETRv3_Real-Time_End-to-End_Object_Detection_with_Hierarchical_Dense_Positive_Supervision_WACV_2025_paper.html.

Yue Wu, Yinpeng Chen, Lu Yuan, Zicheng Liu, Lijuan Wang, Hongzhi Li, and Yun Fu. Rethinking Classification and Localization for Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10186–10195, June 2020. doi: 10.1109/cvpr42600.2020.01020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Wu_Rethinking_Classification_and_Localization_for_Object_Detection_CVPR_2020_paper.html.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Mingyu Xu, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mHC: Manifold-Constrained Hyper-Connections. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026. URL https://openreview.net/forum?id=mDhyxu8WRb.

Huimin Yan, Longfei Xu, Junjie Sun, Zheng Liu, Wei Luo, Kaikui Liu, and Xiangxiang Chu. IntTravel: A real-world dataset and generative framework for integrated multi-task travel recommendation. arXiv preprint arXiv:2602.11664, 2026. URL https://arxiv.org/abs/2602.11664.

Yongyi Yang and Jianyang Gao. mHC-lite: You Don't Need 20 Sinkhorn-Knopp Iterations. In *ICLR 2026 Workshop on Geometry-grounded Representation Learning and Generative Modeling*, 2026. URL https://openreview.net/forum?id=5IJX6kvOif.

Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sunderhauf. VarifocalNet: An IoU-Aware Dense Object Detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8514–8523, June 2021. doi: 10.1109/cvpr46437.2021.00841. URL https://openaccess.thecvf.com/content/CVPR2021/html/Zhang_VarifocalNet_An_IoU-Aware_Dense_Object_Detector_CVPR_2021_paper.html.

Manyuan Zhang, Guanglu Song, Yu Liu, and Hongsheng Li. Decoupled DETR: Spatially Disentangling Localization and Classification for Improved End-to-End Object Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6601–6610, October 2023. doi: 10.1109/iccv51070.2023.00607. URL https://openaccess.thecvf.com/content/ICCV2023/html/Zhang_Decoupled_DETR_Spatially_Disentangling_Localization_and_Classification_for_Improved_End-to-End_ICCV_2023_paper.html.

Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. DETRs Beat YOLOs on Real-time Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16965–16974, June 2024. doi: 10.1109/cvpr52733.2024.01605. URL https://openaccess.thecvf.com/content/CVPR2024/html/Zhao_DETRs_Beat_YOLOs_on_Real-time_Object_Detection_CVPR_2024_paper.html.

Qiang Zhou, Chaohui Yu, Zhibin Wang, and Fan Wang. D2Q-DETR: Decoupling and Dynamic Queries for Oriented Object Detection with Transformers. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1–5, 2023. doi: 10.1109/ICASSP49357.2023.10095341.

Wuyang Zhou, Yuxuan Gu, Giorgos Iacovides, and Danilo Mandic. KromHC: Manifold-constrained hyper-connections with kronecker-product residual matrices. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2601.21579.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-Connections. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/f1e8ff97057e0c3f6bcb7018c78d4df1-Abstract-Conference.html.

Yimin Zhu, Zack Dewis, Quinn Ledingham, Saeid Taleghanidoozdoozan, Mabel Heffring, Zhengsen Xu, Motasem Alkayid, Megan Greenwood, and Lincoln Linlin Xu. mHC-HSI: Clustering-Guided Hyper-Connection Mamba for Hyperspectral Image Classification. arXiv preprint arXiv:2603.03418, 2026a. URL https://arxiv.org/abs/2603.03418.

Yimin Zhu, Lincoln Linlin Xu, Zhengsen Xu, Zack Dewis, Mabel Heffring, Saeid Taleghanidoozdoozan, Motasem Alkayid, Quinn Ledingham, and Megan Greenwood. White-Box mHC: Electromagnetic Spectrum-Aware and Interpretable Stream Interactions for Hyperspectral Image Classification. arXiv preprint arXiv:2601.15757, 2026b. URL https://arxiv.org/abs/2601.15757.

A Implementation of Functional State Updates

Stream-bank initialization. Following Learned Stream Scaling (LSS; Alimaskina et al., 2026), we initialize the persistent bank from the incoming query representation $h^{(0)}$ using learnable stream-specific scales:

$$x_r^{(0)} = s_r \odot h^{(0)}, \qquad X^{(0)} = [x_1^{(0)}; \ldots; x_n^{(0)}], \qquad s_r \in \mathbb{R}^C.$$

The scales differentiate the initial streams through channel-wise modulation of the same query representation. This initialization is applied once, before the first decoder layer; subsequent modules reconstruct and update the resulting bank as described below.

A.1 Scalar Reconstruction, Writeback, and Residual Transport

TaskHC follows the connection parameterization of mHC-lite (Yang & Gao, 2026) to reconstruct working states, distribute module updates, and combine residual streams. We detail these computations below to explain how the mappings in Figure 2 act on the persistent bank. The default formulation uses **one scalar read coefficient and one scalar write coefficient per participating stream**, each shared across all channels. Appendix B extends this formulation to channel-wise vector coefficients.

Generating read and write coefficients. For one object query, let $X = [x_1; \ldots; x_k] \in \mathbb{R}^{k \times C}$ denote the streams used by the current controller. Here, X contains either the full bank for a shared module or the designated functional group for a task-specific module. We omit batch and query indices. Following mHC-lite, the controller flattens and RMS-normalizes these streams to obtain a descriptor:

$$\widehat{X} = \mathrm{RMSNorm}(\mathrm{vec}(X)) \in \mathbb{R}^{1 \times kC}.$$

Separate learned projections generate the read and write coefficients:

$$H^{\mathrm{pre}} = \sigma\left(\alpha^{\mathrm{pre}} \widehat{X} W^{\mathrm{pre}} + b^{\mathrm{pre}}\right), \qquad H^{\mathrm{post}} = 2\sigma\left(\alpha^{\mathrm{post}} \widehat{X} W^{\mathrm{post}} + b^{\mathrm{post}}\right).$$

Both mappings have shape $1 \times k$, with $W^{\mathrm{pre}}, W^{\mathrm{post}} \in \mathbb{R}^{kC \times k}$. The learned biases provide static connection terms, while the scaled projections provide input-dependent terms. Sigmoid gates act independently on each coefficient; the read and write weights are not constrained to sum to one. Normalization is used to predict the coefficients, which are then applied to the original stream values.

Reconstructing the working state and distributing new information. The pre-mapping weights each participating stream and sums their contributions:

$$z = H^{\mathrm{pre}} X = \sum_{j=1}^{k} h_j^{\mathrm{pre}} x_j \in \mathbb{R}^{1 \times C}.$$

Thus, the module receives one C-dimensional working state reconstructed from k persistent streams. Let $\Delta \in \mathbb{R}^{1 \times C}$ denote the update produced by processing this state. The post-mapping distributes that update to the participating streams:

$$w_i = h_i^{\mathrm{post}} \Delta, \qquad i = 1, \ldots, k.$$

Each destination receives a separately weighted contribution from the same module update. In the default formulation, both read and write coefficients are predicted from the incoming streams; the write coefficients do not depend on the newly computed module output.

Combining existing information and completing the update. In parallel with module computation, the residual mapping combines the old stream states. Following mHC-lite, we construct this mapping as a convex combination of fixed permutation matrices:

$$\pi = \mathrm{softmax}\Big(\alpha^{\mathrm{res}}\widehat{X}W^{\mathrm{res}} + b^{\mathrm{res}}\Big), \qquad H^{\mathrm{res}} = \sum_{p=1}^{K} \pi_p P_p \in \mathbb{R}^{k\times k},$$

where $W^{\mathrm{res}} \in \mathbb{R}^{kC\times K}$, and the enumerated construction uses all $K = k!$ permutations. Each fixed matrix P_p rearranges the streams; the input-dependent weights π_p determine their mixture. This produces a doubly stochastic mapping.

For two streams, the two permutations respectively preserve and exchange their positions, giving

$$\widetilde{x}_1 = \pi_1 x_1 + \pi_2 x_2, \qquad \widetilde{x}_2 = \pi_2 x_1 + \pi_1 x_2.$$

The updated stream combines this transported history with its share of the new information:

$$x_i^{+} = \underbrace{\sum_{j=1}^{k} H_{ij}^{\mathrm{res}} x_j}_{\text{combined existing information}} + \underbrace{h_i^{\mathrm{post}}\Delta}_{\text{newly written information}}.$$

In TaskHC's default four-stream bank, shared modules apply these computations across all four streams. Task-specific modules apply them separately within the two-stream classification and localization groups. Prediction readouts use only the reconstruction step. The same connection mechanism therefore supports joint and task-specific working states through the streams assigned to each computation.

A.2 Adapting D-FINE Gated Updates

D-FINE combines the cross-attention output with its input through a learned gate. Given the reconstructed working state z and the attention output y after dropout, the gate predicts two channel-wise coefficient vectors from their concatenation and computes

$$[g_z, g_y] = \sigma([z, y]W_g + b_g), \qquad z_{\mathrm{gate}} = \mathrm{LayerNorm}(g_z \odot z + g_y \odot y),$$

where $[z, y]$ denotes concatenation along the channel dimension. This produces a complete updated working state. To use it within TaskHC's residual-stream update, we form the increment $\Delta = z_{\mathrm{gate}} - z$ and distribute it through the post-mapping, while the residual mapping transports the existing bank. The native gate controls the fusion of the working state and attention output; TaskHC's scalar write coefficients control how the resulting increment is distributed among streams. This adaptation preserves D-FINE's native gating mechanism and keeps the TaskHC integration aligned with its decoder architecture.

A.3 Parameter Cost of Full-Permutation Residual Mixing

The parameter cost of COMB depends on how the residual mapping is constructed. mHC uses Sinkhorn iterations to obtain a doubly stochastic mapping, introducing iterative normalization overhead ((Xie et al., 2026)). The mHC-lite construction adopted here replaces this iterative procedure with a convex combination of fixed permutation matrices ((Yang & Gao, 2026)). Enumerating all permutations makes this construction inexpensive at small stream counts, but its coefficient predictor grows factorially with the number of streams.

For k participating streams of width C, the normalized descriptor $\widehat{X}$ has kC entries. The residual projection predicts one mixture logit for each of the $K = k!$ permutation matrices. Its weight matrix and bias have shapes $kC \times k!$ and $1 \times k!$, respectively, giving

$$N_{\mathrm{res}}(k, C) = kC \cdot k! + k! = (kC + 1)k!.$$

This counts the residual-logit projection and bias. In an L-layer TaskHC decoder, SA and FFN each use one full-bank controller per layer. For n total streams, their combined parameter count is

$$N_{\text{shared}} = 2L\, N_{\text{res}}(n, C) = 2L(nC + 1)n!.$$

With $C = 256$ and $L = 6$, this gives:

Total streams n	Permutations $n!$	Parameters per residual projection	Parameters in 12 shared-module projections
4	24	24,600	295,200
6	720	1,106,640	13,279,680
8	40,320	82,615,680	991,388,160

Increasing the bank from four to eight streams raises the parameter count of the twelve full-bank residual projections from 0.30M to 991.39M. This subtotal excludes task-group controllers, reconstruction and writeback mappings, and the rest of the detector. Table 4 reports it as a theoretical lower bound for the untrained eight-stream mHC-lite configuration. The factorial growth caused by full permutation enumeration makes this configuration impractical for the real-time detector studied here.

Yang and Gao explicitly discuss this scaling issue: the commonly used four-stream setting requires only 24 permutations, while sampling a subset can bound the mixture size and still preserve double stochasticity. More recently, KromHC (Zhou et al., 2026) constructs the residual mapping as a Kronecker product of smaller doubly stochastic matrices, avoiding full-bank permutation enumeration. These approaches offer ways to scale residual mixing beyond the present construction. Within our experimental budget, we retain full-permutation mHC-lite as the reference implementation and leave these alternatives outside the comparisons reported here.

B Controller Granularity, Conditioning, and Capacity

HC and mHC use stream-wise scalar coefficients to combine residual streams and distribute module outputs, applying each coefficient uniformly across feature channels. This provides a compact parameterization of the connections between streams ((Zhu et al., 2025); (Xie et al., 2026)).

Channel-wise feature modulation is an established mechanism in visual representation learning, motivating us to investigate finer control over how each stream contributes to reconstruction and writeback ((Hu et al., 2018)). We extend the scalar mappings to channel-wise coefficients while retaining TaskHC's functional groups and residual-mixing mechanism. Because finer control increases the number of predicted coefficients, we use compact bottleneck predictors and examine their capacity, the information available when predicting writes, and their interaction with stream count. These studies characterize the accuracy and deployment costs of different controller designs within the same functional organization.

B.1 Scalar, Vector, and Scalar–Vector Gates

Stream-wise scalar coefficients. Following Appendix A.1, let $X \in \mathbb{R}^{k\times C}$ contain the k streams participating in an operation. The controller constructs the normalized descriptor

$$\widehat{X} = \mathrm{RMSNorm}(\mathrm{vec}(X)) \in \mathbb{R}^{1\times kC}.$$

For $q \in \{\mathrm{pre}, \mathrm{post}\}$, a learned projection produces the scalar logits

$$\ell^q = \alpha^q \widehat{X} W^q + b^q \in \mathbb{R}^{1\times k},$$

where $W^q \in \mathbb{R}^{kC\times k}$. The corresponding mappings are

$$H^{\mathrm{pre}} = \sigma(\ell^{\mathrm{pre}}), \qquad H^{\mathrm{post}} = 2\sigma(\ell^{\mathrm{post}}).$$

Each mapping supplies one coefficient per stream. During reconstruction, this coefficient weights all channels of a source stream equally; during writeback, it scales the entire update assigned to a destination stream.

Channel-wise vector coefficients. We retain the same normalized stream descriptor but predict a separate coefficient for each channel of each participating stream. The output therefore changes from a length-k vector to a $k \times C$ coefficient matrix.

A direct projection from kC inputs to kC outputs would require k^2C^2 weights per mapping, making its cost grow quadratically with the hidden width. We instead use separate bottleneck predictors for reconstruction and writeback:

$$G^q(\widehat{X}) = \mathrm{reshape}_{k\times C}\Big[\mathrm{SiLU}\Big(\widehat{X}W_1^q + b_1^q\Big)\, W_2^q + b_2^q\Big],$$

where $W_1^q \in \mathbb{R}^{kC\times d}$ and $W_2^q \in \mathbb{R}^{d\times kC}$. The bottleneck width d controls predictor capacity; Appendix B.2 examines its parameter cost and empirical effect.

The channel-wise mappings are

$$\mathbf{H}^{\mathrm{pre}} = \sigma\Big(b_0^{\mathrm{pre}} + G^{\mathrm{pre}}(\widehat{X})\Big), \qquad \mathbf{H}^{\mathrm{post}} = 2\sigma\Big(b_0^{\mathrm{post}} + G^{\mathrm{post}}(\widehat{X})\Big),$$

with $\mathbf{H}^{\mathrm{pre}}, \mathbf{H}^{\mathrm{post}} \in \mathbb{R}^{k\times C}$. The fixed biases b_0^{pre} and b_0^{post} preserve the stream-wise initialization and are broadcast across channels. The channel predictors learn the input-dependent coefficients without a separate learned scalar read/write factor.

Let $\mathbf{h}_i^{\mathrm{pre}}$ and $\mathbf{h}_i^{\mathrm{post}}$ denote the i-th rows of these matrices. Reconstruction and writeback become

$$z = \sum_{i=1}^{k} \mathbf{h}_i^{\mathrm{pre}} \odot x_i, \qquad x_i^+ = \sum_{j=1}^{k} H_{ij}^{\mathrm{res}} x_j + \mathbf{h}_i^{\mathrm{post}} \odot \Delta.$$

Each channel of the working state can thus combine the streams with different weights, while each destination stream receives a channel-wise modulated version of the module update. As in the scalar formulation, normalization is used to predict the coefficients, which act on the original stream values and update.

Scalar–vector composition. We also examine whether channel-wise control benefits from retaining an explicit stream-level factor. This formulation preserves the original scalar controller and combines its coefficients with a separate vector factor:

$$\mathbf{H}_{ic}^{\mathrm{pre}} = \sigma(\ell_i^{\mathrm{pre}})\, 2\sigma\Big(G_{ic}^{\mathrm{pre}}(\widehat{X})\Big), \qquad \mathbf{H}_{ic}^{\mathrm{post}} = 2\sigma(\ell_i^{\mathrm{post}})\, 2\sigma\Big(G_{ic}^{\mathrm{post}}(\widehat{X})\Big).$$

The two factors pass through their respective sigmoid gates before multiplication. The scalar factor provides a common scaling across the stream's channels, while the vector factor adjusts individual channel contributions relative to that scaling. The channel predictors' final projections initialize to zero, so $2\sigma(0) = 1$ and the composition initially recovers the scalar mapping.

All three formulations operate within the same functional scopes: shared modules use the full bank, whereas task-specific modules use their designated groups. Residual mixing remains unchanged and applies the same stream-to-stream coefficients across channels. Appendix B.3 examines the information used to predict write coefficients.

Table B.1: Detection accuracy and inference cost of scalar, vector, and scalar–vector gating.

Read/write gates	Params (M)	GFLOPs	Latency (ms)	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
Scalar	33.26	92.69	8.90	54.68	72.20	59.38	37.67	59.30	71.97
Vector	33.91	93.05	9.92	54.73	72.27	59.45	37.09	59.35	72.27
Scalar $\times$ vector	34.07	93.14	11.33	54.55	72.18	59.18	36.95	58.59	72.06

DEIM–D-FINE-L with four streams, two per functional group. Each controller form applies to both reconstruction and writeback; vector and scalar–vector controllers use bottleneck width $d = 8$.

Accuracy and deployment cost. We compare the complete scalar, vector, and scalar–vector controller configurations with four persistent streams (Table B.1). The vector configuration improves AP by 0.05 over scalar control, while adding 0.65M parameters and 1.02 ms of latency. Scalar control therefore offers a more favorable accuracy–latency trade-off at this bank size and remains the default in TaskHC.

Adding a separate scalar factor to the vector formulation reduces AP by 0.18 and increases latency by 1.41 ms relative to direct vector control. The additional stream-level factor thus increases controller complexity without improving detection in this comparison. We use the direct vector formulation for the subsequent studies, with scalar control as the reference. We first determine how much predictor capacity is needed to generate effective channel-wise coefficients, then examine whether observing the module output improves writeback. Finally, we investigate how reducing the number of persistent streams affects accuracy under each control formulation.

B.2 Bottleneck Design for Channel-Wise Control

Channel-wise control predicts kC coefficients for each reconstruction or writeback mapping. Directly projecting a kC-dimensional bank descriptor to these coefficients requires $O((kC)^2)$ parameters, making dense prediction costly when repeated across decoder modules and layers. We introduce an intermediate width d, giving the bank-based predictor a $kC \to d \to kC$ structure and reducing its parameter cost to $O(kCd)$. The bottleneck compresses the predictor's intermediate representation while retaining a separate output coefficient for every channel of every participating stream. We vary d with the bank size, functional groups, and writeback conditioning fixed to examine how much predictor capacity is needed for effective channel-wise control (Table B.2).

Table B.2: Bottleneck width and parameter cost.

Bottleneck width d	Params (M)	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
2	33.36	54.43	71.99	59.27
4	33.54	54.67	72.30	59.38
8	33.91	54.73	72.27	59.45
16	34.65	54.47	71.93	59.20
32	36.13	54.54	72.36	59.13
64	39.08	—	—	—
Dense (no bottleneck)	72.50	—	—	—

DEIM–D-FINE-L; four total streams, two streams per group, $C = 256$, and output-conditioned vector writing. Params counts the complete deployment model, including the untrained variants. We omit training for $d = 64$ and dense prediction because of their larger parameter costs; — denotes unavailable AP.

Increasing the bottleneck width from $d = 2$ to $d = 8$ improves AP by 0.30, indicating that the narrowest predictor limits the effectiveness of channel-wise control. Further widening to $d = 16$ or 32 increases parameter cost but reduces accuracy. A compact predictor can therefore support fine-grained reconstruction and writeback across all stream channels, while additional predictor capacity does not necessarily improve detection. We select $d = 8$ for the vector configurations, which achieves the highest AP among the tested widths.

B.3 Conditioning Writeback on the Module Output

In HC and mHC, the pre-, post-, and residual mappings are generated from the incoming residual states without observing the module output. The writeback coefficients are thus determined without access to the new information they will distribute. Viewing writeback as residual gating motivates conditioning these coefficients on the newly computed output, allowing the controller to use its content when deciding how to write the update into the stream bank. We evaluate this output-conditioned formulation against the original state-only formulation under both scalar and channel-wise vector control.

We compare bank-only prediction with prediction conditioned on both the bank and the module output. Using the normalized descriptors $\widehat{X}$ and $\widehat{y} = \mathrm{RMSNorm}(y)$, the write-controller inputs are

$$\xi_{\mathrm{bank}} = \widehat{X}, \qquad \xi_{\mathrm{bank+output}} = [\widehat{X}; \widehat{y}].$$

The second variant predicts the write coefficients after obtaining y, allowing them to depend on the new features they will distribute. We keep four persistent streams and compare the two inputs separately for scalar and vector controllers (Table B.3). Reconstruction and residual mixing retain their original conditioning.

Table B.3: Output-conditioned writeback.

Read/write gates	Write-gate input	$\mathrm{AP}^{\mathrm{val}}$	$\mathrm{AP}^{\mathrm{val}}_{50}$	$\mathrm{AP}^{\mathrm{val}}_{75}$	$\mathrm{AP}^{\mathrm{val}}_{S}$	$\mathrm{AP}^{\mathrm{val}}_{M}$	$\mathrm{AP}^{\mathrm{val}}_{L}$
Scalar	Bank only	54.68	72.20	59.38	37.67	59.30	71.97
Scalar	Bank + output	54.31	71.81	59.03	36.41	58.65	72.12
Vector	Bank only	54.35	71.93	59.05	36.42	58.94	71.59
Vector	Bank + output	54.73	72.27	59.45	37.09	59.35	72.27

DEIM–D-FINE-L; four streams, two per functional group; vector predictors use $d = 8$. Bank + output concatenates normalized bank and module-output descriptors. For D-FINE CA, y is the attention output before native gating; the written update remains Δ from Appendix A.2.

Conditioning writeback on the module output has opposite effects at the two granularities: it reduces AP by 0.37 with scalar coefficients but increases AP by 0.38 with channel-wise coefficients. The value of observing new output therefore depends on how the controller can distribute it. A scalar coefficient scales the entire update assigned to a stream, whereas channel-wise coefficients can use the output's channel responses to modulate individual components. These results support designing the controller's information input together with its control granularity. We retain bank-only conditioning for scalar gates and output-conditioned writing for vector gates in the subsequent stream-count comparison.

B.4 Stream Count under Scalar and Channel-Wise Control

We next examine how reducing the number of persistent streams affects accuracy under the two control formulations. Following B.3, we use bank-only conditioning for scalar gates and bank-plus-output conditioning for vector gates, keeping each choice fixed when comparing two and four streams (Table B.4a,b). The comparison reduces each task group from two persistent states to one.

Table B.4: Accuracy and deployment cost at different stream counts.

(a) Scalar gates: bank-only conditioning.

Total streams	Streams per group	Params (M)	GFLOPs	Latency (ms)	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
2	1	32.86	92.41	8.87	54.39	72.02	59.25	37.61	58.82	71.70
4	2	33.26	92.69	8.90	54.68	72.20	59.38	37.67	59.30	71.97

(b) Vector gates: bank-plus-output conditioning.

Total streams	Streams per group	Params (M)	GFLOPs	Latency (ms)	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
2	1	33.24	92.62	8.90	54.68	72.32	59.66	37.13	58.91	72.19
4	2	33.91	93.05	9.92	54.73	72.27	59.45	37.09	59.35	72.27

DEIM–D-FINE-L; streams are divided equally between the two functional groups; vector predictors use $d = 8$. Conditioning is fixed within each panel, so the comparison changes the stream count.

Reducing the bank from four streams to two lowers AP by 0.29 with scalar control, compared with 0.05 with output-conditioned vector control. The vector configuration therefore retains more of its accuracy as the bank becomes smaller. With one stream per task, channel-wise gates still determine how individual components of that state contribute to reconstruction and how new information is written back. Fine-grained control remains useful even when task-specific reconstruction no longer combines multiple streams within a group.

The two-stream vector model (Table B.4b) reaches 54.68 AP, matching the four-stream scalar model (Table B.4a), with a similar deployment cost of 33.24M parameters, 92.62 GFLOPs and 8.90 ms latency. This provides an alternative balance between persistent-state capacity and control granularity: fewer states with channel-wise reconstruction and writeback can achieve comparable accuracy and cost to a larger bank with scalar control.

B.5 Summary and Implications

This study extends stream-wise scalar hyper-connections to channel-wise reconstruction and writeback, allowing different feature components within each stream to be combined and updated with different weights. The experiments show that compact bottleneck predictors can support this finer control; increasing predictor width does not consistently improve accuracy. They also reveal an interaction between control granularity and writeback conditioning: access to the module output improves vector writeback but reduces accuracy with scalar coefficients. The benefit of observing newly computed information therefore depends on how the controller can distribute it.

The stream-count study further shows that output-conditioned vector control retains more accuracy when the bank is reduced from four streams to two. Together, these findings motivate designing hyper-connections through both the granularity of their coefficients and the information used to predict them. Channel-wise reconstruction and writeback provide a promising extension of HC/mHC, motivating further investigation of efficient coefficient predictors, deployment operators, and applications beyond the detector architectures studied here.

C Training and Deployment Protocols

C.1 Model Configurations and Training Settings

We follow the DEIM training framework for DEIM–D-FINE and DEIM–RT-DETRv2, retaining their native prediction heads and losses. Models are trained on COCO train2017 with ImageNet-pretrained backbones and evaluated on val2017. Tables C.1 and C.2 list the architecture and training settings shared by each baseline–TaskHC pair. The default TaskHC configuration uses four streams, with two assigned to each task, and scalar reconstruction and writeback coefficients.

We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, 2,000 warm-up iterations, automatic mixed precision, and EMA with a base decay of 0.9999. Evaluation uses 640×640 inputs; the default random seed is 0.

Table C.1: Architecture and training settings for DEIM–D-FINE and its TaskHC counterparts.

Setting	N	S	M	L	X
Architecture					
Backbone	HGNetv2-B0	HGNetv2-B0	HGNetv2-B2	HGNetv2-B4	HGNetv2-B5
Encoder hidden dimension	128	256	256	256	384
Decoder hidden dimension	128	256	256	256	256
Decoder FFN dimension	512	1024	1024	1024	1024
Decoder layers	3	3	4	6	6
Object queries	300	300	300	300	300
Feature strides	16, 32	8, 16, 32	8, 16, 32	8, 16, 32	8, 16, 32
Sampling points per level	6, 6	3, 6, 3	3, 6, 3	3, 6, 3	3, 6, 3
Optimization and augmentation					
Training epochs	160	132	102	58	58
Total batch size	128	32	32	32	32
Base learning rate	8e-4	4e-4	4e-4	5e-4	5e-4
Backbone learning rate	4e-4	2e-4	4e-5	2.5e-5	5e-6
Weight decay	1e-4	1e-4	1e-4	1.25e-4	1.25e-4
Learning-rate schedule	Constant	Flat cosine	Flat cosine	Flat cosine	Flat cosine
Cosine decay starts (epoch)	—	64	49	29	29
Minimum / base learning rate	1	0.5	0.5	0.5	0.5
MixUp interval	[4, 78)	[4, 64)	[4, 49)	[4, 29)	[4, 29)
Strong augmentation stops (epoch)	148	120	90	50	50

Epoch indices are zero-based; intervals are half-open. D-FINE-N keeps a constant learning rate after warm-up. The displayed L/X settings also apply to the additional paired runs; the original S/M/X recipes retain downward EMA restarts, while the additional paired recipes use upward restarts.

Table C.2: Architecture and training settings for DEIM–RT-DETRv2 and its TaskHC counterparts.

Setting	R18	R34	R50-M*	R50-L	R101
Architecture					
Backbone	ResNet-18-D	ResNet-34-D	ResNet-50-D	ResNet-50-D	ResNet-101-D
Encoder hidden dimension	256	256	256	256	384
Decoder hidden dimension	256	256	256	256	256
Decoder FFN dimension	1024	1024	1024	1024	1024
Decoder layers	3	4	3	6	6
Object queries	300	300	300	300	300
Feature strides	8, 16, 32	8, 16, 32	8, 16, 32	8, 16, 32	8, 16, 32
Sampling points per level	4, 4, 4	4, 4, 4	4, 4, 4	4, 4, 4	4, 4, 4
Optimization and augmentation					
Training epochs	120	120	60	60	60
Total batch size	16	16	16	16	16
Base learning rate	2e-4	2e-4	2e-4	2e-4	2e-4
Backbone learning rate	2e-4	1e-4	2e-5	2e-5	2e-6
Weight decay	1e-4	1e-4	1e-4	1e-4	1e-4
Learning-rate schedule	Flat cosine	Flat cosine	Flat cosine	Flat cosine	Flat cosine
Cosine decay starts (epoch)	64	64	34	34	34
Minimum / base learning rate	0.5	0.5	0.5	0.5	0.5
MixUp interval	[4, 64)	[4, 64)	[4, 34)	[4, 34)	[4, 34)
Strong augmentation stops (epoch)	117	117	58	58	58

Both families use the DEIM augmentation pipeline and EMA evaluation, selecting the checkpoint with the highest validation AP. R50-M* and R50-L follow the official DEIM names for the two ResNet-50 configurations. The earlier RT-DETRv2-L comparison uses the R50-L architecture and the same displayed training hyperparameters.

C.2 Deployment Optimization and Measurement

Deployment optimization. We optimize the two cross-attention branches and the small operators used for residual-stream mixing and writeback. After ONNX export, these operations are implemented with TensorRT plugins to reduce kernel launches and intermediate memory traffic while preserving the models computation and parameter-sharing structure. The same implementations are reused for matching computational structures across model scales and ablation configurations.

Dual cross-attention. A dual-output kernel performs sampling and aggregation for both task branches in a single launch. Each branch retains its own sampling locations, attention weights, and output. The kernel accepts either a shared projected value tensor or separate value tensors, following the configurations parameter-sharing structure.

Residual mixing and transport. We fuse softmax with the weighted combination of fixed permutation matrices to construct the residual-mixing map without storing intermediate tensors. For a fixed stream count, we unroll its application to the stream bank, reducing small-matrix and tensor-rearrangement overhead.

Writeback and normalization. We fuse weighted writeback, residual addition, and adjacent LayerNorm or RMSNorm where these operations occur consecutively, reducing repeated memory accesses to the feature tensors. Numerically sensitive normalization statistics and accumulations use FP32 within the FP16 engine.

Latency measurement. We benchmark configurations sequentially on the same NVIDIA Tesla T4 using TensorRT 10.4, FP16, batch size one, and 640×640 inputs, with CUDA Graph disabled. Baseline and TaskHC models use the same general engine-building and timing settings. Following the official DEIM implementation (Huang et al., 2025b), we use its synchronous whole-engine timing script with 1,000 warm-up executions, followed by three rounds of 1,000 executions, and report the mean per-image latency. Timing includes the model forward pass and exported decoding/Top-K operations; preprocessing, host–device transfers, and external NMS are excluded.

Complexity measurement. We count parameters and FLOPs on the evaluation/deployment graph with batch size one and 640×640 inputs. Parameter counts include frozen parameters and exclude buffers. FLOPs are measured using calflops with FP32 operator tracing, including matrix multiplications and counting each multiply-add as two operations. External postprocessing is excluded.

D Repeated Training and Additional Comparisons

D.1 Results Across Training Repetitions

We report the original seed-0 comparison and four additional single-GPU repetitions (seeds 1–4) on DEIM–D-FINE-L. Each run uses 58 epochs and total batch size 32. Tables D.1 and D.2 report single-GPU and four-GPU DDP results separately; $\Delta\mathrm{AP}^{\mathrm{val}}$ is TaskHC minus the baseline at the same seed.

Table D.1: Single-GPU detection results and variation across seeds.

(a) Individual runs.

Model	Seed	Epoch	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}	ΔAP^{val}
DEIM–D-FINE-L	0	54	54.45	72.20	59.18	37.11	58.98	71.78	—
TaskHC-L	0	56	54.68	72.20	59.38	37.67	59.30	71.97	+0.23
DEIM–D-FINE-L	1	56	54.51	72.13	59.31	37.58	58.92	71.88	—
TaskHC-L	1	55	54.54	72.15	59.06	36.94	58.95	71.87	+0.03
DEIM–D-FINE-L	2	57	54.19	71.92	59.07	36.53	58.90	71.54	—
TaskHC-L	2	52	54.35	72.03	59.01	36.92	58.63	71.51	+0.16
DEIM–D-FINE-L	3	52	54.47	72.15	59.23	37.13	58.94	71.41	—
TaskHC-L	3	54	54.48	72.00	59.41	37.64	59.07	72.07	+0.00
DEIM–D-FINE-L	4	57	54.16	71.67	58.83	35.60	58.72	71.80	—
TaskHC-L	4	51	54.64	72.14	59.56	36.76	58.96	72.29	+0.48

(b) Mean $\pm$ sample standard deviation.

Seeds	Baseline AP^{val}	TaskHC AP^{val}	ΔAP^{val}
0–4 (all five)	54.36 ± 0.17	54.54 ± 0.13	$+0.18 \pm 0.19$

Epoch denotes the best-AP checkpoint epoch. Statistics and paired differences are computed before rounding.

Table D.2: Four-GPU DDP comparison (seed 0, total batch size 32).

Model	Seed	Epoch	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}	ΔAP^{val}
DEIM–D-FINE-L	0	57	54.39	71.93	58.99	36.33	58.88	71.83	—
TaskHC-L	0	55	54.51	72.04	59.20	36.71	59.05	71.54	+0.12

D.2 Published Real-Time Detector Comparisons

Table D.3: Published detection accuracy and deployment cost on COCO `val2017`. VFM: foundation-model-assisted models; YOLO26: Objects365 pretraining; r: reported complexity. Deployment protocol: Appendix C.2.

Model	Params (M)	GFLOPs	Latency (ms)	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv5-M (Ultralytics, 2022)	21.2	49.0	3.53	45.4	64.1	–	–	–	–
YOLOv5-L	46.5	109.1	6.06	49.0	67.3	–	–	–	–
YOLOv5-X	86.7	205.7	11.04	50.7	68.9	–	–	–	–
YOLOv8-M (Ultralytics, 2023)	25.9	82.6	5.05	50.2	67.2	54.6	32.0	55.7	66.4
YOLOv8-L	43.7	172.2	8.23	52.9	69.8	57.5	35.3	58.3	69.8
YOLOv8-X	68.2	266.7	13.23	53.9	71.0	58.7	35.7	59.3	70.7
YOLO11-M (Ultralytics, 2024)	20.1	74.9	4.56	51.5	68.1	55.8	–	–	–
YOLO11-L	25.3	100.8	6.21	53.4	69.7	58.3	–	–	–
YOLO11-X	56.9	215.7	11.27	54.7	71.3	59.7	–	–	–
YOLOv12-M (v1.0) (Tian et al., 2025)	20.2	76.8	6.48	52.5	69.6	57.1	35.7	58.2	68.8
YOLOv12-L (v1.0)	26.4	107.3	10.45	53.7	70.7	58.5	36.9	59.5	69.9
YOLOv12-X (v1.0)	59.1	226.7	17.79	55.2	72.0	60.2	39.6	60.7	70.9
YOLOv13-L (Lei et al., 2025)	27.6	101.5	13.20	53.4	70.9	58.1	–	–	–
YOLOv13-X	64.0	214.2	21.69	54.8	72.0	59.8	–	–	–
YOLO-ULM-M (Han et al., 2026)	16.0^{r}	70.2^{r}	–	52.8	70.2	57.6	35.2	58.6	69.5
YOLO-ULM-L	22.8^{r}	95.1^{r}	–	54.1	71.5	59.0	37.4	60.0	70.3
YOLO-ULM-X	51.0^{r}	213.1^{r}	–	55.6	72.8	60.8	40.6	61.0	71.8
YOLO26-M E2E (Jocher et al., 2026)	20.4	73.1	4.57	52.5	–	–	–	–	–
YOLO26-L E2E	24.8	96.7	6.14	54.4	–	–	–	–	–
YOLO26-X E2E	55.7	209.3	11.04	56.9	–	–	–	–	–
RT-DETRv3-M (R34) (Wang et al., 2025)	31.2	93.2	6.29	49.9	67.7	53.9	31.7	54.0	66.2
RT-DETRv3-M* (R50m)	36.0^{r}	100.0^{r}	–	51.7	–	–	–	–	–
RT-DETRv3-L (R50)	42.1	134.3	8.83	53.4	71.7	57.3	35.4	57.4	69.8
RT-DETRv3-X (R101)	76.0^{r}	259.0^{r}	–	54.6	73.1	–	–	–	–
D-FINE-M (Peng et al., 2025)	19.2	56.7	5.47	52.3	69.8	56.4	33.2	56.5	70.2
D-FINE-L	30.7	91.0	8.44	54.0	71.6	58.4	36.5	58.0	71.9
D-FINE-X	61.6	202.5	13.44	55.8	73.7	60.2	37.3	60.5	73.4
DEIM–D-FINE-M (Huang et al., 2025b)	19.2	56.7	5.43	52.7	70.0	57.3	35.3	56.7	69.5
DEIM–D-FINE-L	30.8	91.1	8.37	54.7	72.4	59.4	36.9	59.6	71.8
DEIM–D-FINE-X	61.7	202.7	14.02	56.5	74.0	61.5	38.8	61.4	74.2
DEIM–RT-DETRv2-M (Huang et al., 2025b)	31.2	92.3	6.27	50.9	68.6	55.2	34.3	54.4	67.1
DEIM–RT-DETRv2-M*	33.0	99.5	6.95	53.2	71.2	57.8	35.3	57.6	70.2
DEIM–RT-DETRv2-L	42.1	133.5	8.91	54.3	72.3	58.8	37.5	58.7	70.8
DEIM–RT-DETRv2-X	74.9	251.4	14.21	55.5	73.5	60.3	37.9	59.9	73.0
Le-DETR-M (Huang et al., 2026)	31.4^{r}	114.1^{r}	–	52.9	70.0	57.4	35.4	56.9	68.6
Le-DETR-L	41.5^{r}	124.3^{r}	–	54.3	71.7	58.9	37.1	58.7	71.4
Le-DETR-X	44.9^{r}	196.9^{r}	–	55.1	72.5	59.8	38.1	59.6	71.4
RT-DETRv4-M^{VFM} (Liao et al., 2026)	19.4	56.7	5.25	53.5	71.1	58.1	34.9	57.7	72.1
RT-DETRv4-L^{VFM}	31.0	91.1	8.26	55.4	73.0	60.3	37.1	60.1	72.9
RT-DETRv4-X^{VFM}	62.0	202.7	13.46	57.0	74.6	62.1	39.5	61.9	74.8
DEIMv2-M^{VFM} (Huang et al., 2025a)	18.1	52.2	9.54	53.0	70.2	57.6	34.2	57.4	71.5
DEIMv2-L^{VFM}	32.2	96.7	12.29	56.0	73.4	60.9	37.5	60.8	75.2
DEIMv2-X^{VFM}	50.3	151.6	16.41	57.8	75.4	63.2	39.2	62.9	75.9

VFM denotes foundation-model-assisted models; YOLO26 uses Objects365 pretraining. Superscript r marks source-reported complexity. Other complexity and measured latency follow Appendix C.2. Published AP retains its source precision; — denotes unavailable measurements. Citations on the first row of each family apply to the family's subsequent rows.

E Task Specialization in Evidence Acquisition and Prediction States

E.1 Cross-Attention Sampling Geometry

We characterize how the classification and localization paths acquire visual evidence by measuring their sampling distributions across decoder layers. We analyze 36,335 matched objects from all 5,000 COCO validation images, using final-layer Hungarian assignments consistently across layers. For a sampling point p_i, we normalize its displacement from the ground-truth box center (c_x, c_y) by the box width w and height h:

$$r_i = \sqrt{\left(\frac{p_{i,x} - c_x}{w}\right)^2 + \left(\frac{p_{i,y} - c_y}{h}\right)^2}.$$

For each object and task path, we compute the attention-weighted mean radius over sampling points, heads, and feature levels, then average equally over objects. Level-specific statistics use the same calculation within each feature level. We report 95% confidence intervals from 2,000 paired bootstrap samples at the image level.

Figure E.1 shows that localization has a larger attention-weighted mean normalized center distance than classification in layers 1–4. From layer 2 to layer 5, the localization mean decreases, while the classification mean remains relatively stable, bringing the two paths to similar distances. In the final layer, the classification mean increases markedly and exceeds the localization mean, which changes only slightly. The reversal therefore primarily reflects an outward shift in classification sampling.

A plausible interpretation is that early localization draws on peripheral cues to establish object extent, while classification initially emphasizes discriminative appearance cues closer to the object center. As box estimates stabilize, localization can build on accumulated geometric information for further corrections, consistent with its reduced center distances in later layers. The late outward shift in classification may supplement previously acquired appearance information with peripheral object features or surrounding context that helps resolve category ambiguity. These later updates build on the task representations accumulated across preceding layers, allowing each path to acquire additional evidence suited to its remaining prediction needs.

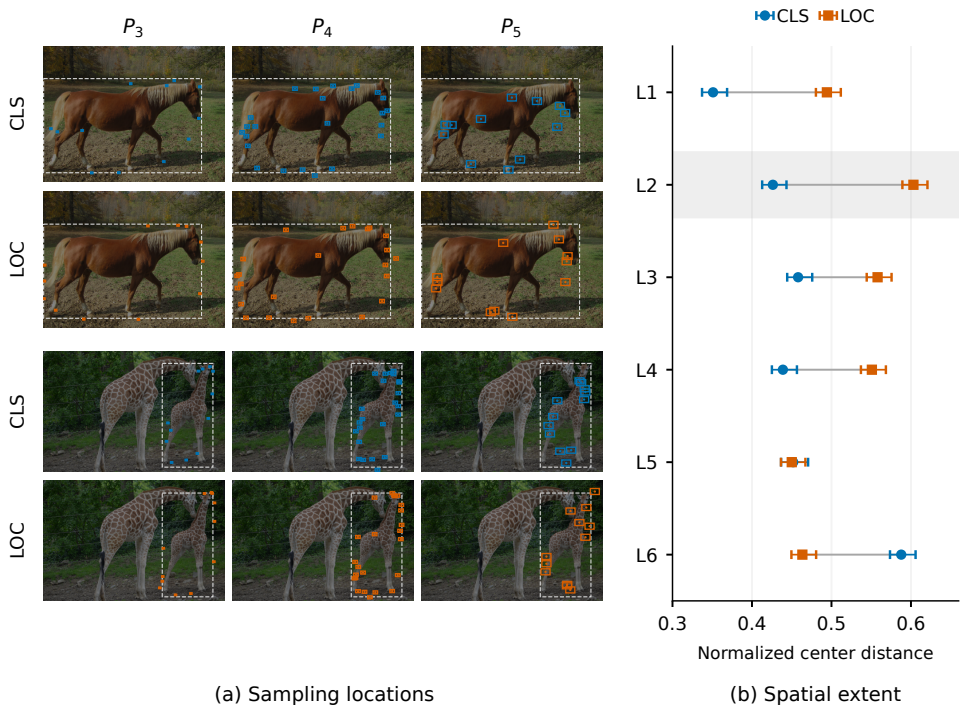

Figure E.1: Task-specific sampling geometry. (a) Sampling locations at P3–P5. (b) Attention-weighted distance from the box center, normalized by box width and height, over 36,335 validation objects. Error bars: 95% paired image-bootstrap confidence intervals.

E.2 Task Information in Frozen Prediction Inputs

We examine how functional specialization is reflected in the information accessible from the two prediction paths. We freeze the trained TaskHC detector and train classification and localization

probes separately on the native final-layer prediction inputs. The localization input includes D-FINE's cross-layer state accumulation. Within each probe task, the two input representations are evaluated using the same probe architecture, optimization settings, and matched objects.

Probe training. We use disjoint subsets of COCO train2017 for fitting and checkpoint selection, containing 807,119 and 42,828 matched objects, respectively. Final evaluation uses the 36,335 matched objects in val2017. Features are standardized using per-channel statistics estimated on the fitting set. The probe settings are summarized below; initialization and sample order are paired across the two input representations for seeds 0, 1, and 2.

Setting	Classification probe	Localization probe
Architecture	Linear, $256 \to 80$	MLP, $256 \to 256 \to 256 \to 4$
Hidden/output activation	—	ReLU / sigmoid
Prediction target	Object category	Image-normalized (c_x, c_y, w, h)
Training loss	Cross-entropy	$5L_1 + 2(1 - \mathrm{GIoU})$
Optimizer	AdamW	AdamW
Learning rate / weight decay	10^{-3} / 10^{-4}	10^{-3} / 10^{-4}
Training steps / batch size	50,000 / 2,048 objects	50,000 / 2,048 objects
Checkpoint selection	Minimum calibration loss, evaluated every 100 steps	Same

Both probes are randomly initialized. The localization probe predicts boxes directly from the frozen features. Classification is evaluated by macro accuracy over the 80 categories; Loc@0.5 and Loc@0.75 measure the percentage of predicted boxes reaching the corresponding IoU threshold.

Table E.1: Task information recovered from frozen prediction inputs on COCO val2017.

Input	Classification macro accuracy	Loc@0.5	Loc@0.75
CLS prediction input	91.70 ± 0.11	22.91 ± 0.06	7.71 ± 0.14
LOC prediction input	90.31 ± 0.08	26.78 ± 0.04	12.85 ± 0.12

Values are percentages, reported as mean $\pm$ sample standard deviation over three probe-training seeds with the detector fixed. Figure 4 shows calibration-set learning curves; this table reports final validation performance.

The LOC input also improves localization accuracy at IoU ≥ 0.75 by 5.14 percentage points, showing that it supports more precise box recovery under the same probe architecture and training protocol.