CAPABILITY CARDS FOR COMPILING FEATURE INTERFACES BETWEEN HETEROGENEOUS, INDEPENDENTLY TRAINED COLLABORATIVE PERCEPTION SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Collaborative perception uses complementary observations from multiple agents to mitigate occlusion and limited sensing coverage. In practice, however, systems are often developed independently: they may use different encoders, learn different representations even under the same architecture, or rely on different fusion mechanisms. As a result, features produced by one system may not be directly usable by another. We therefore investigate whether such systems can establish a compatible feature interface from their independently generated capability descriptions. We cast this problem as feature-interface compilation and propose Rosetta, a framework for it. Each system builds a Capability Card from its own model and data alone. The card records how feature restrictions affect the sender's detection performance and how sensitive the receiver's detection is to perturbed features; together these give complementary cues for choosing an interface. When two systems meet, a learned compiler reads both cards to select and configure an interface program, whose pair-specific parameters are then fitted without labels on a small set of public frames. The sender's features are delivered only if the fitted mapping passes a feature-level check; otherwise the receiver detects alone. Both systems stay frozen: each keeps its own representations and fusion module, and the interface translates features between them rather than imposing a common representation space. On OPV2V, direct collaboration degrades the receiver's detection in our heterogeneous settings, whereas Rosetta improves detection on average for LiDAR systems that differ in training seed, fusion module or encoder, including systems unseen in compiler training.

1 INTRODUCTION

Collaborative perception lets vehicles and roadside units share complementary observations, so that each can detect objects that are occluded or beyond its own sensing range (Han et al., 2023; Liu et al., 2023). Exchanging intermediate bird's-eye-view features balances accuracy against bandwidth and is widely studied (Han et al., 2023; Liu et al., 2023; Yazgan et al., 2024), with fusion designs from attention and transformers to sparse communication (Xu et al., 2022d;c; Hu et al., 2022). In practice, collaborating agents come from different developers and manufacturers, with different configurations and models (Huang et al., 2025), and each system is trained by its own team, on its own data, as a complete pipeline. We study collaboration under a deployment constraint: when two such systems meet, both keep their perception pipelines and neither is retrained.

Under this constraint, compatibility depends on more than the encoder. A feature means what the encoder, the fusion module and the detection head learned together, and a receiver's fusion module and head have only learned to read features from their own encoder. Two systems can differ in the encoder, in the fusion module, or only in the training seed, as Figure 1a–c illustrates, and a foreign feature map with the right shape can still be misread. Feeding the sender's features into the receiver's fusion module after aligning their shapes lowers AP70, average precision at an IoU threshold of 0.7, on 141 of the 164 pairs we could evaluate, including pairs whose feature maps already have the same shape, consistent with earlier observations (Yang et al., 2026). Heterogeneity is recognized as

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.

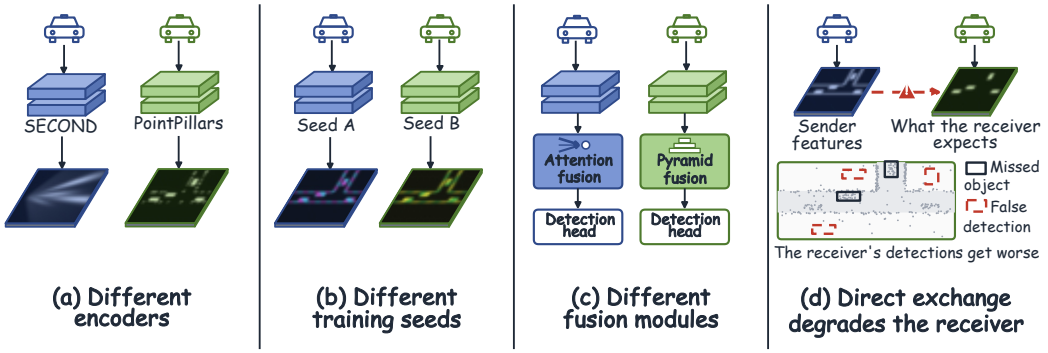

Figure 1: **Separately trained systems do not interoperate by default.** Each vehicle runs a complete detector, and two such systems can differ in (a) the encoder, (b) the training seed of the same architecture, or (c) the fusion module. Whichever of these differs, the receiver's fusion module was never trained to read the sender's features: (d) passing them to it directly degrades the receiver's detections, with missed objects and false detections.

a major challenge for feature-level collaboration (Yazgan et al., 2024; Huang et al., 2025). Existing methods align agents to a shared model or protocol (Lu et al., 2024; Gao et al., 2025), interpret or generate features for the receiver (Xia et al., 2025; Zhou et al., 2025), or self-train an ego-side module during operation (Si et al., 2025; Yang et al., 2026), learning the adaptation from labelled data involving the new partner or from the encounter itself. We study the configuration decision that precedes such adaptation, and ask: can each system prepare a description of itself alone, such that, once two systems pair, their descriptions guide the choice of interface configuration and fitting strategy, and a small amount of public data suffices to fit the interface?

Our starting point is that much of what an interface needs to know about each system can be measured by the system alone. Structural information, the shape and channel layout of the features that enter fusion and the code that consumes them, determines which transformations are possible. Behavioural information adds two complementary cues. The receiver's *sensitivity* says how its detection responds when features from a collaborator of its own type are perturbed, and so which transformations it tolerates. The sender's *information retention* says how feature restrictions affect its own detection, and so which parts of its features carry information worth transmitting. Neither cue alone decides an interface: the first describes how the receiver reads features, the second what the sender has to offer. Each system records these measurements in a *Capability Card* before any partner exists, and the two cards are combined only when a pair forms. Because sensitivity belongs to the receiver, the interface from A to B serves B's use of A's features and need not mirror the one from B to A. The cards guide the choice of an interface rather than predict whether it will help.

Rosetta turns this view into three steps. Each system builds its card from its own frozen model and labelled data. When a sender and a receiver pair, a learned compiler, a language-model controller fine-tuned offline on other pairs, reads both cards and configures an interface program. A numerical backend then fits the program's parameters without labels on a small set of public anchor frames, and the fit results return to the compiler. The fitted interface is delivered only if it passes a certification check on the anchor frames; otherwise the receiver detects alone.

We evaluate on OPV2V with LiDAR systems that differ in training, fusion or encoder. On 148 pairs with a system unseen in compiler training and selection, direct fusion lowers the receiver's detection in every group, whereas Rosetta raises the receiver's AP70 from 0.698 to 0.740 on the 106 pairs on which it collaborates. On development pairs, which were used to select the compiler but not to train it, training the compiler adds 0.042 AP70 over the same compiler with the untuned base language model.

Our contributions are as follows.

- We formulate the problem of building a directed, compatible feature interface for each sender–receiver pair of independently trained collaborative perception systems, using their separately generated capability descriptions.

- We propose Rosetta, in which a learned compiler uses the Capability Cards that each system builds alone to select and configure an interface. The interface parameters are fitted on public anchor frames without labels, while both perception systems remain frozen.
- Experiments on OPV2V show that Rosetta improves detection on average across all evaluated types of heterogeneity, including systems unseen during compiler training. Ablations on development pairs show the benefits of compiler fine-tuning and system-specific capability information.

2 Related Work

Collaborative perception. Collaborative perception has advanced fusion, communication efficiency and robustness to pose errors (Han et al., 2023; Liu et al., 2023): fusing bird's-eye-view features of several agents (Chen et al., 2019; Xu et al., 2022d;c;b; Li et al., 2021; Lu et al., 2024), sending only what is needed (Hu et al., 2022), and fusing robustly under pose errors (Lu et al., 2023). Agents can also exchange detections: late fusion merges received boxes with the ego's (Xu et al., 2022d), and ALF synthesizes ego-compatible features from boxes so that unseen collaborators need no adaptation (Bae & Ahn, 2026). Boxes carry what the sender's detector has already decided, whereas intermediate features carry its evidence before that decision and need a receiver that can read them. Rosetta gives an existing receiver an interface to another system's intermediate features.

Heterogeneous collaborative perception. When collaborators differ in sensors or models, their features must be made compatible before fusion. They can be transformed or interpreted for the receiver: MPDA learns a feature resizer and a sparse cross-domain transformer (Xu et al., 2022a), PnPDA converts features through a standard semantic space (Luo et al., 2024), and PolyInter keeps every perception network fixed and trains only a new agent's prompt and a resizer for an ego-side interpreter (Xia et al., 2025). Agents can be aligned to a shared representation or protocol: HEAL trains a new agent's encoder against a unified fusion module and head (Lu et al., 2024), STAMP trains an adapter and a reverter per agent to and from a shared protocol space (Gao et al., 2025), CodeFilling exchanges codes from a shared codebook (Hu et al., 2024), and HM-ViT fuses camera and LiDAR agents with one jointly trained transformer (Xiang et al., 2023). GenComm generates compatible features on the receiving side and adds a partner by fine-tuning a message extractor for each directed pair of agent types (Zhou et al., 2025). These methods learn compatibility with detection labels on data involving the new partner. PHCP and BOLT instead self-train an ego-side module on the frames of the encounter, with the collaborator's detections or the ego's own predictions as targets (Si et al., 2025; Yang et al., 2026). PHCP, like our setting, connects complete collaborative detectors. Rosetta's numerical fitting is closest to these adapters, but the configuration and fitting strategy of each directed pair are selected from descriptions the two systems measured separately, by a compiler trained offline on other pairs, and the pair's parameters are fitted without labels on public anchor frames, not on its labelled data or the encounter's frames.

Cross-model representation alignment. Model stitching joins the early layers of one network to the later layers of another through a trained stitching layer, and uses the stitched network's accuracy to ask whether two representations can stand in for each other (Lenc & Vedaldi, 2015; Bansal et al., 2021; Csiszárik et al., 2021). Relative representations describe samples by their similarities to shared anchor samples, making independently trained latent spaces comparable (Moschella et al., 2023), and latent space translation estimates a transformation between two latent spaces in closed form from parallel anchors seen by both models (Maiorca et al., 2023). Rosetta uses shared public frames as parallel anchors to fit a mapping between two systems' feature maps, but the descriptions decide which mapping to fit and how, and the mapped features are fused with the receiver's own, so the question is whether they support collaborative detection rather than whether they can replace a representation.

Model descriptions for reuse. Model cards document a model's intended use and evaluation for human readers (Mitchell et al., 2019). Task2Vec embeds tasks through the Fisher information of a probe network to select a pre-trained feature extractor (Achille et al., 2019), and in the learnware paradigm a model's specification lets it be identified and reused, and relates models built on different feature spaces (Tan et al., 2023). These descriptions serve understanding, matching or selection; Rosetta combines two systems' capability cards to configure a directed interface between them.

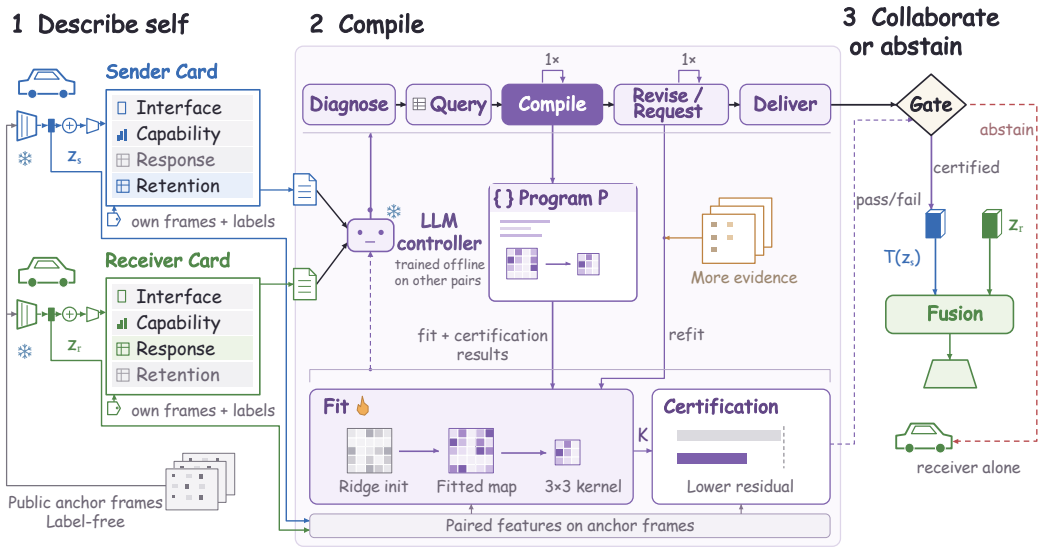

Figure 2: **Rosetta in three steps.** (1) *Describe self*: each system builds its capability card from its own frames and labels, with a response table, which describes how sensitive its detection is to feature transformations, and a retention table, which describes its detection when its features are restricted. (2) *Compile*: the learned compiler, drawn as the LLM controller and trained offline on other system pairs, reads the two cards and configures an interface program. The program is fitted on public anchor frames into an interface instance, which is then certified; the results return to the compiler, which may revise it or request more evidence. (3) *Collaborate or abstain*: if the compiler delivers a fitted interface and it has passed certification, the receiver fuses the transformed sender features with its own; otherwise it detects alone.

3 Method

3.1 Overview and problem setup

An *endpoint* e is a collaborative 3D detector with encoder E_e, fusion module F_e and detection head D_e. The two endpoints of a pair were trained independently, and both stay frozen. The encoder output that enters fusion is the *port* $z_e = E_e(x) \in \mathbb{R}^{C_e \times H \times W}$. For a directed pair $s \to r$, Rosetta produces either abstention or an *interface* $T_{P,\theta}$, determined by an *interface program* P, which fixes the configuration of the transformation and of its fit as Section 3.3 details, and by the parameters θ fitted under that configuration. With an interface, the transformed sender features take the place of a collaborator of the receiver's own type:

$$\hat{y}_r = D_r\big(F_r(z_r,\, T_{P,\theta}(z_s))\big). \tag{1}$$

with abstention the receiver detects alone, $\hat{y}_r = D_r(F_r(z_r))$. The interface converts representations only: it is applied to the collaborating vehicle's port features in that vehicle's own coordinates, and the receiver's fusion pipeline applies the pose transformation between the vehicles, as it does for a collaborator of its own type. Figure 2 gives an overview of the three steps. The learned compiler reads the cards of Section 3.2, chooses P and reacts to fit results in a *session*, one round of interface construction for a directed pair, which runs before any detection; Sections 3.3 and 3.4 describe the session and the compiler's offline training on other pairs with their labels, and Section 3.5 the numerical backend that fits θ without labels on public anchor frames and the certification check that gates delivery. The 64 anchor frames form two tiers of 32: an interface is first fitted on one tier, and the compiler can request the other as further evidence for the fit. Appendices A and C give the card contents and the implementation details of each step.

3.2 CAPABILITY CARDS

Each system builds its card from its own frozen model and labelled training-split data, with no partner involved. The card holds three kinds of information, each bearing on particular choices. *Interface constraints*, the shape and channel layout of the port, the code path that consumes it and its type information, decide which programs are legal, for example whether statistics can be matched channel by channel. The *receiver response table* records how the system's detection changes when features of a collaborator of its own type are perturbed through candidate transformations. Its entries are indexed by a *spatial gain*, a scalar factor applied to the features at every cell of the port, which changes their magnitude but not their resolution or geometric scale, and by a *fit-position rule*, which selects the cells of the port on which an interface is fitted and certified. The *objectness rule* selects the cells where the sender's own detection head predicts objects above a threshold measured on the sender's frames and recorded on its card, and the *energy rule* selects the cells where the sender's features have high energy. The *sender retention table* records how the system's own detection holds up when its features are projected onto their leading principal-component directions or kept only at the cells a fit-position rule selects, so it bears on the choice of that rule. A pair reads the receiver's response table and the sender's retention table.

The cards enter a session along two paths. The compiler reads an anonymized view of each card and reaches the tables through queries, which informs the configuration it chooses. The numerical backend also reads card statistics directly whenever a chosen configuration refers to them: a gain chosen from the response table is picked from the receiver's table, the objectness rule takes its threshold from the sender's card, and *normalization*, which matches the mean and scale of the sender's features to the receiver's, uses the port statistics recorded on both cards.

3.3 INTERFACE CONFIGURATION AND COMPILATION

An interface program fixes how the sender's features are transformed, how the transformation is fitted, on which data, and what happens when the fitted result is not certified, as Table 1 summarizes. We call the interface obtained by one fit of a program an *instance*. One program yields several instances only when it is refitted on further evidence. The receiver-proxy loss addresses a limitation of matching features alone: a map can bring sender rows close to receiver rows and still produce features that the receiver consumes differently, so the loss also compares a receiver-side proxy, such as the port magnitude that the receiver's fusion gate reads, on the mapped sender features and on the receiver's own. The spatial kernel is a scaled identity, whose gain may come from the response table, or a learned kernel.

Table 1: **Configuration of an interface program.** The compiler chooses the options in the second column; the others are fixed in every session.

	Chosen by the compiler	Fixed	What it decides
Feature transformation	normalization; per-channel 3×3 spatial kernel	channel map $K = UV$	how sender channels map onto receiver channels, and how space is weighted
Fitting strategy	fit position; receiver-proxy loss	—	which cells supervise the fit, and what it matches besides the receiver's rows
Data	—	initial evidence tier	which anchor material the fit starts from; a request can add more
Delivery rule	—	certification conditions; fallback set by the receiver	when a fitted instance is delivered, and what happens otherwise

Several of these choices depend on both systems at once, so the compiler makes them in the session, using fit results as feedback, as step (2) of Figure 2 shows. It diagnoses the pair from the two card views, queries the tables and proposes a few programs. Each is fitted on the first anchor tier, and the fit and certification results return to the compiler, because a program that looks suitable from the cards can still fail certification or pass by a narrow margin. The compiler may then revise an instance or request evidence. A *revision* produces a new program that differs only in the receiver-proxy loss

or the spatial kernel, and an instance of it that inherits the channel map of the instance it revises. Because certification checks only the channel map, as Section 3.5 explains, the revised instance keeps that instance's certification verdict. A *request* brings new evidence and refits the instance: the second anchor tier, uncompressed sender features at the fit-position cells, or extra sender features selected by a condition that the sender's card offers. Only a request can change the channel map and, with it, the verdict. Finally the compiler delivers one fitted instance or abstains, within fixed budgets.

3.4 Learning the compiler

The compiler has to learn three abilities: to produce legal actions, to prefer programs that help the receiver, and to judge whether a fit result calls for a revision, a request or delivery. It is Qwen3-4B (Yang et al., 2025), fine-tuned with LoRA adapters (Hu et al., 2021), and at each *compilation step* of a session, namely diagnosis, query, program proposal, revision or request, and delivery, it is decoded under that step's action schema (Willard & Louf, 2023), which admits only well-formed actions of the step. All three training stages optimize supervised fine-tuning on a target action. For a training row with prompt x, the model-visible session state at one compilation step, and target action y, the loss is the mean negative log-likelihood of the target's supervised tokens,

$$\ell(x, y) = -\frac{1}{|\mathcal{S}(y)|} \sum_{t \in \mathcal{S}(y)} \log \pi\big(y_t \mid x, y_{<t}\big), \tag{2}$$

where $\mathcal{S}(y)$ holds the tokens of the action's syntax and of its program, parameter and decision fields, as the compilation step requires, and never free text. The training stages differ in how targets are chosen. *Legality* targets are drawn uniformly from the legal actions, so that the model learns the action schema. *Program preference* fits several programs from the same initial session state on training pairs and scores them by the receiver's detection gain on training-split frames. In every comparison where the gains differ with the same strict sign in AP70 and in an auxiliary score that accumulates detections in the order they are listed instead of ranking them by confidence across frames, and so depends on that order, the winner becomes a target. *Branch preference* runs three branches from a shared session state to the end, delivering without further change, revising or requesting evidence, and the winning branch's action becomes a target, as does the delivery choice between abstention and the certified instances. The learning signal is thus the detection result of training pairs, measured with their labels; a target pair needs none.

3.5 Interface fitting and collaborative inference

Fitting turns a chosen program into an interface. The two systems encode the same anchor frame from the same ego sensor data, their port grids have the same size, and cells with the same index describe the same place, so the sender's and receiver's port rows at a cell form a training pair without labels and without a pose transformation. Only cells at the fit position enter. The N sender rows, after the program's normalization, and the N receiver rows are compressed by one CountSketch (Charikar et al., 2002) shared by both sides, and uncompressed rows obtained by a request are appended as they are. This gives X and Y, and the reconstruction loss $L_{\mathrm{rec}}(K) = \|XK - Y\|_F^2/(NC_r)$, normalized by the number of rows before compression, involves only the channel map. A receiver-proxy loss L_{prx} is added with a weight β that makes the two terms equal in magnitude at the start of the fit:

$$L(K, W) = L_{\mathrm{rec}}(K) + \beta\, L_{\mathrm{prx}}(K, W), \tag{3}$$

where W is the spatial kernel. One further option, unused in the reported sessions, adds a prior on W. The fit starts from the *ridge initialization* K_{init}, the closed-form ridge solution on the rows the fit uses, and runs Adam (Kingma & Ba, 2015) inside a trust region around that start, since the reconstruction loss is not stationary there and the proxy loss can pull the channel map away from the receiver's rows.

Certification compares the fitted channel map $\hat{K}$ with a *certification reference* K_{ref} through the relative residual $e(K; J)$ between mapped sender rows and receiver rows at the fit-position cells of anchor frames J:

$$e(\hat{K}; J) < e(K_{\mathrm{ref}}; J), \qquad J \in \{J_{\mathrm{ho}}, J_{\mathrm{sp}}\}, \tag{4}$$

where J_{ho} holds the anchor frames the evidence tier did not spend and J_{sp} the spent ones. When all frames are spent, only the comparison on J_{sp} remains. The reference is the ridge solution on the sketched rows of the spent frames. It equals K_{init} except after a request for uncompressed rows: the fit then starts from the ridge solution on the mixed rows, while the reference stays, so that requested rows do not move the threshold. The check involves only the channel map at the fit-position cells, so it constrains delivery without measuring detection benefit. If the compiler delivers a certified instance, the receiver inserts $T_{P,\theta}(z_s)$ into its fusion module and detects as in Equation (1), and otherwise it detects alone.

4 EXPERIMENTS

In this section, we evaluate Rosetta on OPV2V with LiDAR systems that differ in training, fusion module or encoder. We show that directly fusing the features of independently trained systems degrades detection, whereas Rosetta's compiled interfaces improve it on average, including for systems unseen in compiler training. We also examine the contributions of compiler training and the capability cards through ablations, and visualize how the interfaces change features and detections.

4.1 EXPERIMENTAL SETUP

We conduct experiments on OPV2V (Xu et al., 2022d). All development data come from its training split: each endpoint measures its card tables on 167 probe frames, 241 frames provide the outcomes for compiler training and the validation readouts, and pair-specific interfaces are fitted and certified without labels on 64 public anchor frames in two tiers of 32. We evaluate detection performance on the complete test split, which contains 2,170 frames from 16 scenes.

Heterogeneous systems. Every endpoint is a complete LiDAR detector. Encoders are PointPillars (Lang et al., 2019), SECOND (Yan et al., 2018) or PIXOR (Yang et al., 2018), and fusion is attentive (Xu et al., 2022d), max (Chen et al., 2019), CoBEVT (Xu et al., 2022b), pyramid (Lu et al., 2024), the multi-scale attentive fusion of CoAlign without its pose correction (Lu et al., 2023), or DiscoNet (Li et al., 2021). Ports have 64 or 256 channels. The two systems of a directed pair differ in one of four ways, a property of the endpoints known in advance; the grouping was drawn after the results were seen and only describes them. In *Training* pairs they share an architecture but were trained separately, with a different seed or training code base. In *Fusion* pairs their fusion modules differ, including CoAlign's multi-scale attentive core against a single-scale one under the same configured name. *Encoder* pairs differ only in the encoder, and *Encoder + Fusion* pairs in both. Eight endpoints form the compiler's training population. Each of the 18 *development pairs* joins a training endpoint with one of three other endpoints kept out of compiler training. The compiler was selected on these pairs' validation frames but never trained on them, so on test only the scenes are new. *Held-out endpoints* are absent from compiler training and selection, and we call a pair with one a pair with an *unseen system*. They use DiscoNet fusion, a new seed of existing configurations, or a new encoder; the new-seed endpoints were used earlier in method development. All pairs except the new-encoder ones were evaluated on the test split once, after the compiler was frozen; the new-encoder pairs were added afterwards, registered before any result was seen, and evaluated once. Every pair was fixed before its test evaluation; reported pairs are selected by endpoint identity alone.

Comparison methods and evaluation. We compare Rosetta with Ego, the receiver detecting alone, and with direct fusion, which feeds the sender's features, aligned in shape without learned parameters, directly into the receiver's fusion module. To assess compiler learning and interface selection, we evaluate two controls: a base-model controller without compiler fine-tuning, and a fixed strategy that fits a default interface program, requests additional sender features and refits the interface. Both controls share Rosetta's capability cards, fitting procedure and certification check. All perception models remain frozen. Compiler selection uses validation data, and the training configuration is fixed before test evaluation. We measure detection performance by AP at IoU thresholds of 0.5 and 0.7, denoted AP50 and AP70. AP is computed on the full test split for each directed pair and then averaged equally across the pairs in each group. Pairs without a delivered interface retain Ego's predictions. Appendices B and C detail the configurations and evaluation protocols.

Table 2: **Detection performance on OPV2V with unseen systems.** Each pair contains a system excluded from compiler training and selection. Rows are grouped by heterogeneity and by Rosetta's final outcome, and all three methods are evaluated on the same pairs of each row, except one pair on which direct fusion could not be evaluated; entries report mean AP over those pairs. Ego denotes the receiver detecting alone.

		Ego		Direct fusion		Rosetta	
Heterogeneity	Rosetta outcome	AP50	AP70	AP50	AP70	AP50	AP70
Training	Collaborate	0.819	0.699	0.647	0.491	**0.887**	**0.767**
	Ego only	**0.791**	**0.686**	0.739	0.625	**0.791**	**0.686**
Fusion	Collaborate	0.813	0.698	0.630	0.516	**0.846**	**0.736**
	Ego only	**0.812**	**0.696**	0.558	0.444	**0.812**	**0.696**
Encoder	Collaborate	0.807	0.690	0.303	0.051	**0.869**	**0.722**
	Ego only	–	–	–	–	–	–
Encoder + Fusion	Collaborate	0.820	0.702	0.548	0.471	**0.875**	**0.744**
	Ego only	**0.826**	**0.711**	0.534	0.296	**0.826**	**0.711**
All	Collaborate	0.813	0.698	0.605	0.475	**0.854**	**0.740**
	Ego only	**0.812**	**0.696**	0.569	0.446	**0.812**	**0.696**

4.2 Quantitative results

Collaboration with unseen systems. Table 2 shows that on systems excluded from compiler training and selection, Rosetta improves average detection performance for every evaluated type of heterogeneity, whereas direct fusion degrades it. Rosetta establishes collaboration on every pair of the Encoder group, showing that compiled interfaces carry over to detectors built on different encoders. When no interface is delivered, the receiver keeps its standalone predictions, so the pairs without collaboration lose no accuracy. Averaged over all pairs, including those without collaboration, the gain over Ego holds across test scenes, with scene-bootstrap intervals above zero for both AP50 and AP70. These results show that compiled interfaces support effective feature-level collaboration between independently trained perception systems.

Interface establishment cost. Each directed pair establishes its interface once, before the receiver processes any test frame. Excluding the time spent building structured-decoding constraints, which we measured separately and subtracted, a session takes an estimated 93 s on average, measured without inference acceleration. Reading the anchor-frame responses before the session adds about 4 s, and a session exchanges a median of 11.8 MB, 2.6 MB of it the two cards. We timed 18 sessions of Rosetta on 6 pairs of validation frames, one at a time on an otherwise idle NVIDIA RTX 6000 Ada GPU, with the host CPUs shared with other jobs. Once established, the interface adds 0.4–1.7 ms per frame, measured at batch 1 with both collaborators' features passing the interface, against 2.2–319 ms for the receiver's own fusion and detection head.

4.3 Ablation studies and analysis

Effect of compiler learning. Table 3 shows that fine-tuning improves detection over the base-model controller on the development pairs. Rosetta also outperforms the fixed strategy, indicating that, on these pairs, learned interface configuration is more effective than following a predefined request-and-refit procedure. These comparisons support training the compiler to guide interface construction.

Table 3: **Effect of compiler learning.** Mean AP on OPV2V test scenes over the same 18 development pairs.

Method	AP50	AP70
Ego	0.811	0.692
Fixed strategy	0.867	0.729
Base-model controller	0.843	0.712
Rosetta	**0.876**	**0.754**

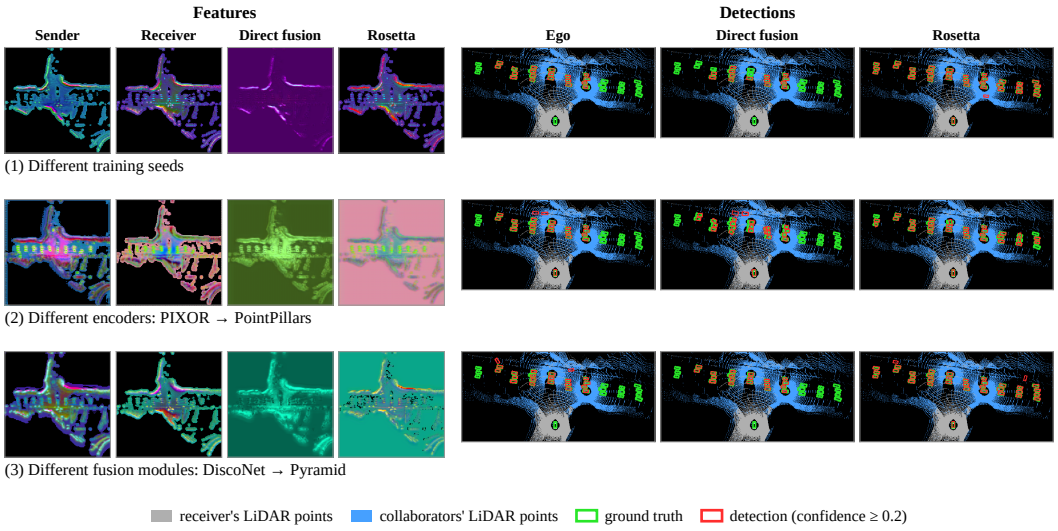

Figure 3: **Visualization of collaborative detection on OPV2V with three sources of heterogeneity.** Rows (1)–(3) show one pair each from the Training, Encoder and Fusion groups of Table 2 on the same test frame. The left four columns show the features of the sender, the receiver, direct fusion and Rosetta, and the right three columns show the detections of Ego, direct fusion and Rosetta. Brightness in the direct-fusion column is stretched for visibility.

Effect of capability descriptions. We replace optional capability fields with those from other endpoints while keeping the compiler and the remaining card fields unchanged. As Table 4 shows, the original fields yield higher detection accuracy, showing that system-specific capability information helps construct more effective interfaces.

Table 4: **Effect of capability-field permutation.** Mean AP on OPV2V over the same 18 development pairs, with the same fine-tuned compiler.

Capability fields	AP50	AP70
Permuted	0.851	0.724
Original	**0.876**	**0.754**

4.4 Qualitative visualizations

Figure 3 compares the three methods on one pair from each of the Training, Encoder and Fusion rows, all on the same test frame. Alone, the receiver misses vehicles in the region that only the collaborating vehicles observe. Direct fusion does not recover them; it also loses some of the receiver's own detections, and where the encoders differ it replaces them with scattered false boxes. With Rosetta, the receiver detects vehicles in the collaborators' region in all three rows without adding false boxes. The feature columns show the difference. The sender's own features carry the road layout and the vehicles, but passed directly to the receiver they fall outside the range of the receiver's own features and take nearly one hue. Rosetta's output keeps the layout visible in the receiver's feature space, even where its background differs from the receiver's own features.

5 Conclusion

We studied whether independently developed collaborative perception systems can establish compatible feature interfaces from capability descriptions that each system generates alone. Rosetta uses a learned compiler to select and configure an interface from the two systems' Capability Cards. Its parameters are fitted without labels on public anchor frames, while both perception systems remain frozen. On OPV2V, where directly fusing features degrades detection, Rosetta improves detection on average across differences in training seed, encoder and fusion module, including systems unseen during compiler training. Ablations on development pairs show the benefits of compiler fine-tuning and system-specific capability information. These findings support capability-guided interface construction as a way to connect independently developed perception systems.

AI USE STATEMENT

In this work, we used generative AI tools (large language model assistants) for the following tasks. *Aiding and polishing writing*: sentence rephrasing, grammar checking and improving the overall flow of the text. *Retrieval and discovery*: searching for related work and summarizing candidate papers, whose relevance and descriptions we checked against the original publications. *Research execution*: writing experiment scripts, data input and output processing and parts of the implementation, running and monitoring experiment jobs, and aggregating results into tables and figures, as well as discussing experimental designs and interpreting results, with the research question, method and conclusions decided by the authors; the core code was checked by the authors, and every reported number was regenerated by scripts from the saved per-pair results and session records. *Drafting sections of the paper*: condensing and rewriting sections at the final typesetting stage to meet the page limit. The authors reviewed all AI-assisted work and take full responsibility for the content of the manuscript, including any text or code generated or polished with the aid of generative AI.

ETHICS STATEMENT

This study follows the ICLR Code of Ethics. All experiments use the publicly available OPV2V dataset, which consists of simulated driving scenes and contains no personal data. The research involves no human participants or animal experiments, and the dataset is used in accordance with its usage guidelines.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our results, we provide detailed implementation information in the appendix, covering the data splits and their uses, the contents of the capability cards and a complete compilation session, and the program options, budgets, fitting and certification details and training settings. Every reported number is regenerated by scripts from the saved per-pair results and session records. We will release the code, the capability cards and the trained compiler weights.

REFERENCES

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Stefano Soatto, and Pietro Perona. Task2Vec: Task embedding for meta-learning. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Hyunchul Bae and Heejin Ahn. Adaptation-free heterogeneous collaborative perception with unseen agent configurations. *arXiv preprint arXiv:2605.26642*, 2026.

Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 225–236, 2021.

Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, pp. 693–703, 2002.

Qi Chen, Xu Ma, Sihai Tang, Jingda Guo, Qing Yang, and Song Fu. F-Cooper: Feature based cooperative perception for autonomous vehicle edge computing system using 3D point clouds. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing (SEC)*, pp. 88–100. ACM, 2019. doi: 10.1145/3318216.3363300.

Adrián Csiszárik, Péter Kőrösi-Szabó, Ákos Matszangosz, Gergely Papp, and Dániel Varga. Similarity and matching of neural network representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Xiangbo Gao, Runsheng Xu, Jiachen Li, Ziran Wang, Zhiwen Fan, and Zhengzhong Tu. STAMP: Scalable task- and model-agnostic collaborative perception. In *International Conference on Learning Representations (ICLR)*, 2025.

Yushan Han, Hui Zhang, Huifang Li, Yi Jin, Congyan Lang, and Yidong Li. Collaborative perception in autonomous driving: Methods, datasets, and challenges. *IEEE Intelligent Transportation Systems Magazine*, 15(6), 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Yue Hu, Shaoheng Fang, Zixing Lei, Yiqi Zhong, and Siheng Chen. Where2comm: Communication-efficient collaborative perception via spatial confidence maps. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Yue Hu, Juntong Peng, Sifei Liu, Junhao Ge, Si Liu, and Siheng Chen. Communication-efficient collaborative perception via information filling with codebook. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Tao Huang, Jianan Liu, Xi Zhou, Dinh C. Nguyen, Mostafa Rahimi Azghadi, Yuxuan Xia, Qing-Long Han, and Sumei Sun. Vehicle-to-everything cooperative perception for autonomous driving. *Proceedings of the IEEE*, 113(5), 2025.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

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

Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast encoders for object detection from point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 991–999, 2015. doi: 10.1109/CVPR.2015.7298701.

Yiming Li, Shunli Ren, Pengxiang Wu, Siheng Chen, Chen Feng, and Wenjun Zhang. Learning distilled collaboration graph for multi-agent perception. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Si Liu, Chen Gao, Yuan Chen, Xingyu Peng, Xianghao Kong, Kun Wang, Runsheng Xu, Wentao Jiang, Hao Xiang, Jiaqi Ma, and Miao Wang. Towards vehicle-to-everything autonomous driving: A survey on collaborative perception. *arXiv preprint arXiv:2308.16714*, 2023.

Yifan Lu, Quanhao Li, Baoan Liu, Mehrdad Dianati, Chen Feng, Siheng Chen, and Yanfeng Wang. Robust collaborative 3D object detection in presence of pose errors. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Yifan Lu, Yue Hu, Yiqi Zhong, Dequan Wang, Yanfeng Wang, and Siheng Chen. An extensible framework for open heterogeneous collaborative perception. In *International Conference on Learning Representations (ICLR)*, 2024.

Tianyou Luo, Quan Yuan, Guiyang Luo, Yuchen Xia, Yujia Yang, and Jinglin Li. Plug and play: A representation enhanced domain adapter for collaborative perception. In *European Conference on Computer Vision (ECCV)*, 2024.

Valentino Maiorca, Luca Moschella, Antonio Norelli, Marco Fumero, Francesco Locatello, and Emanuele Rodolà. Latent space translation via semantic alignment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Conference on Fairness, Accountability, and Transparency (FAT*)*, 2019. doi: 10.1145/3287560.3287596.

Luca Moschella, Valentino Maiorca, Marco Fumero, Antonio Norelli, Francesco Locatello, and Emanuele Rodolà. Relative representations enable zero-shot latent space communication. In *International Conference on Learning Representations (ICLR)*, 2023.

Hao Si, Ehsan Javanmardi, and Manabu Tsukada. You share beliefs, I adapt: Progressive heterogeneous collaborative perception. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Peng Tan, Zhi-Hao Tan, Yuan Jiang, and Zhi-Hua Zhou. Handling learnwares developed from heterogeneous feature spaces without auxiliary data. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning (ICML)*, 2022.

Yuchen Xia, Quan Yuan, Guiyang Luo, Xiaoyuan Fu, Yang Li, Xuanhan Zhu, Tianyou Luo, Siheng Chen, and Jinglin Li. One is plenty: A polymorphic feature interpreter for immutable heterogeneous collaborative perception. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Hao Xiang, Runsheng Xu, and Jiaqi Ma. HM-ViT: Hetero-modal vehicle-to-vehicle cooperative perception with vision transformer. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Runsheng Xu, Jinlong Li, Xiaoyu Dong, Hongkai Yu, and Jiaqi Ma. Bridging the domain gap for multi-agent perception. *arXiv preprint arXiv:2210.08451*, 2022a.

Runsheng Xu, Zhengzhong Tu, Hao Xiang, Wei Shao, Bolei Zhou, and Jiaqi Ma. CoBEVT: Cooperative bird's eye view semantic segmentation with sparse transformers. In *Conference on Robot Learning (CoRL)*, 2022b.

Runsheng Xu, Hao Xiang, Zhengzhong Tu, Xin Xia, Ming-Hsuan Yang, and Jiaqi Ma. V2X-ViT: Vehicle-to-everything cooperative perception with vision transformer. In *European Conference on Computer Vision (ECCV)*, 2022c.

Runsheng Xu, Hao Xiang, Xin Xia, Xu Han, Jinlong Li, and Jiaqi Ma. OPV2V: An open benchmark dataset and fusion pipeline for perception with vehicle-to-vehicle communication. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022d.

Yan Yan, Yuxing Mao, and Bo Li. SECOND: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Bin Yang, Wenjie Luo, and Raquel Urtasun. PIXOR: Real-time 3D object detection from point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Kang Yang, Tianci Bu, Peng Wang, Deying Li, and Yongcai Wang. BOLT: Online lightweight adaptation for preparation-free heterogeneous cooperative perception. *arXiv preprint arXiv:2605.00405*, 2026.

Melih Yazgan, Thomas Graf, Min Liu, Tobias Fleck, and J. Marius Zöllner. A survey on intermediate fusion methods for collaborative perception categorized by real world challenges. In *IEEE Intelligent Vehicles Symposium (IV)*, 2024.

Junfei Zhou, Penglin Dai, Quanmin Wei, Bingyi Liu, Xiao Wu, and Jianping Wang. Pragmatic heterogeneous collaborative perception via generative communication mechanism. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

A Capability cards and one complete session

A.1 One directed pair from cards to detections

Figure 4 follows one development pair through a complete test session of Rosetta, with the certification check enforced. The sender is a LiDAR detector with CoBEVT fusion and a 256-channel port; the receiver is a LiDAR detector with pyramid fusion and a 64-channel port.

A.2 Card contents and the decision view

Layers. A card has four layers.

- **Interface facts and semantics (L0).** Nineteen program facts about the port and the code path that consumes it, execution checks, typed channel keys and semantic fields.
- **Compressed capability state (L1).** Summaries of the port statistics, whose per-channel values accompany the card, an intervention table, and probe-model predictions over a query grid.
- **Small networks with their gate states (L2).** The gate states are read when programs are type-checked; the networks themselves are not executed in a session. This layer also holds the two tables described below.
- **Evidence store (L3).** The card lists what evidence is held and how large it is, not the evidence itself.

The two tables. Each endpoint measures its tables alone, on its own probe frames and with its own ground truth.

- The *receiver response table* records how the receiver's AP70 and its order-dependent variant (Section 3.4) change when a perturbed feature of another vehicle in the same frame reaches it through each candidate action. That feature comes from the receiver's own port.
- The *sender retention table* records how feature restrictions affect the sender's own detection performance: a projection of its port onto its leading principal-component directions, or a restriction to the cells that a fit-position rule selects.

Decision view. The controller sees a deterministic view of each card, capped at 16 KiB and stripped of endpoint names and paths. Long values in the view are replaced by typed placeholders, and fields are dropped in a fixed order when the cap is reached.

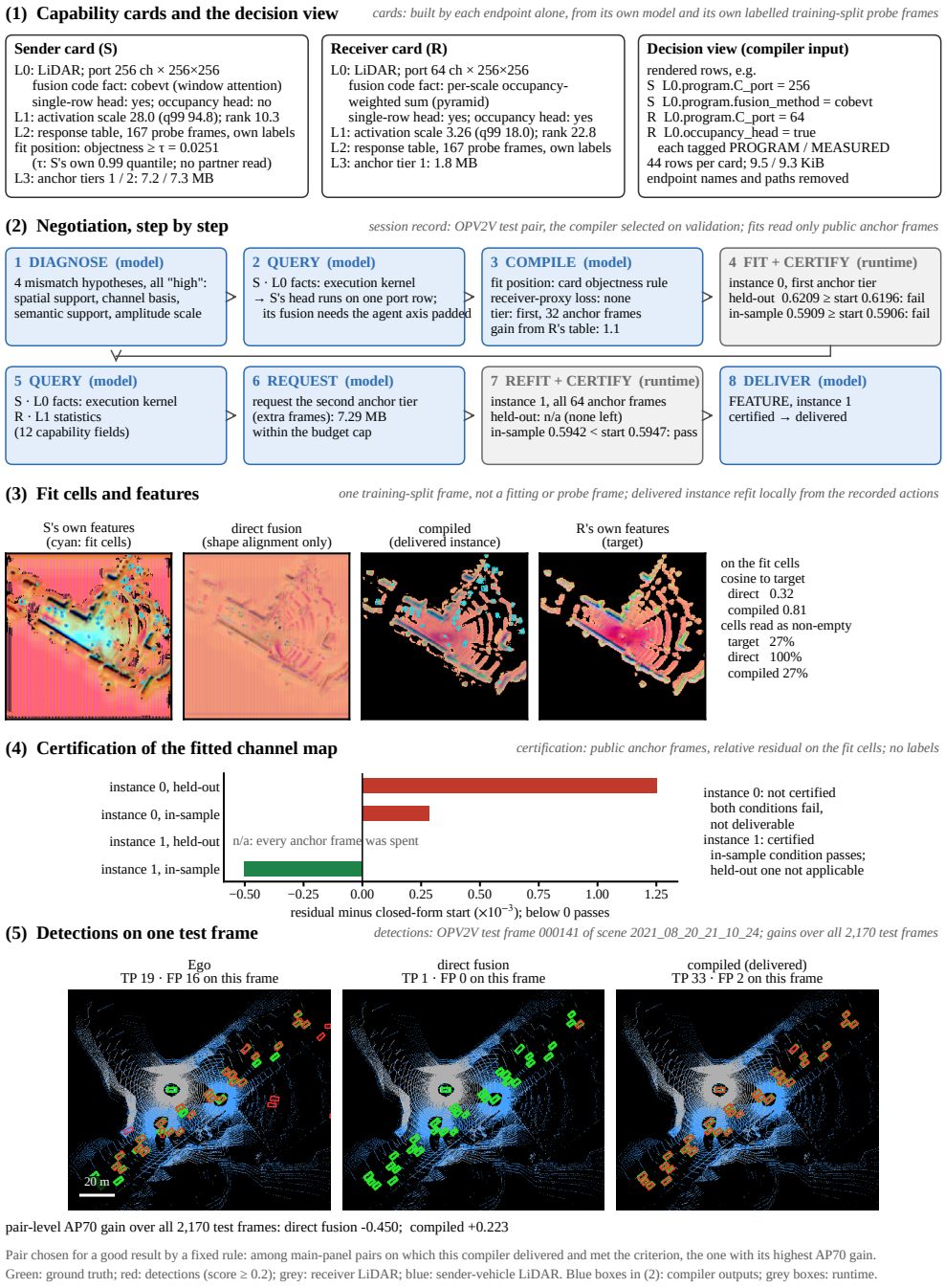

Figure 4: **From Capability Cards to collaborative detection.** A selected development pair illustrates the two systems' Capability Cards, interface compilation, feature transformation, feature-level certification and detection results. Panel (3) shows features from a replayed interface on a separate training-split frame; panel (5) shows detections on a test frame.

B DATA SPLITS

Splits. Table 5 lists the principal OPV2V subsets used in this paper. All development data come from the OPV2V training split, and no number in this paper uses the OPV2V validation split. The validation population excludes the nine anchor frames that an earlier 250-frame sample contained, and it shares scenes with the anchor frames but no frame: the closest validation frame lies two frame indices, one recorded sweep, from an anchor frame of the same scene.

Metrics. AP50 and AP70 count detections with confidence at least 0.2 and rank all detections of the test split by confidence before accumulating precision and recall. The 95% scene intervals resample the 16 test scenes 1,000 times, with the pairs fixed and the same resamples used for every method.

Table 5: Principal OPV2V subsets used in this paper.

Subset	Frames	Scenes	Split	Use; labels read
Anchor frames	64	38	train	fitting, request, certification; none
Probe frames	167	16	train	receiver response and sender retention tables; each endpoint's own
Validation population	241	19	train	compiler-training labels (training pairs), freezing (development pairs); yes
Test	2,170	16	test	all reported test numbers: the main batch, every pair except the new-encoder ones, then the new-encoder pairs; yes

Use of labels. The receiver response and sender retention tables of a card use that endpoint's own labels on the probe frames, and the objectness threshold uses predicted scores only. The small probe networks on a card were trained from a probe run of the endpoint over training-split frames, which reads the endpoint's own labels; the probe-run records confirm this for the 16 endpoints of the main batch, and the registration of the new-encoder pairs records the probe runs of their three added endpoints. The labels of compiler training are outcomes of sessions of the training pairs on the validation population, and no training pair is a development pair.

C IMPLEMENTATION DETAILS

C.1 BUDGETS AND PROGRAM OPTIONS

Algorithm 1 The complete compilation session for a directed pair $s \to r$

Require: decision views V_s, V_r; controller π; anchor tiers A_1, A_2 (32 frames each)
1: $S \leftarrow \{V_s, V_r\} \cup \pi_{\text{DIAGNOSE}}(\cdot)$ ▷ anonymized model-visible state
2: $S \leftarrow S \cup \text{ANSWER}(\pi_{\text{QUERY}}(S))$ while π queries ▷ at most 4 query objects in all
3: $\{P_1, \ldots, P_k\} \leftarrow \pi_{\text{COMPILE}}(S)$, $k \leq 3$; validate the set (one correction allowed)
4: for each valid P_i: $(\theta_i, v_i) \leftarrow \text{FITANDCERTIFY}(P_i, A_1)$; append the record to S
5: $S \leftarrow S \cup \text{ANSWER}(\pi_{\text{QUERY}}(S))$ while π queries ▷ same budget; may name an instance
6: $a \leftarrow \pi_{\text{REVISE-OR-REQUEST}}(S)$ ▷ one correction allowed
7: if a revises instance i: refit the child (only the spatial kernel is trainable); if a requests m: $\text{FITANDCERTIFY}(P_i, A_1 \cup m)$
8: $d \leftarrow \pi_{\text{DELIVER}}(S) \in \{\text{FEATURE}(i), \text{ABSTAIN}\}$
9: **return** ABSTAIN if $d = \text{FEATURE}(i)$ and instance i is not certified, else d ▷ certification gate

Budgets. The controller may request query objects (interface facts, capability statistics or either table) before and after the initial fits, at most four in total, and later queries may refer to a fitted

instance. It proposes up to three programs, which the runtime validates and fits on the first anchor tier, and it takes at most one revision or request, so a session performs at most three initial fits and four fits in total. The controller may correct an invalid action once; if it remains invalid, the session ends without a terminal decision and the receiver detects alone.

Program options. A program sets eight configuration options: normalization (none or one of the two maps below); a channel map $K = UV$, factored at rank $\min(C_s, C_r)$ and started from its ridge initialization; a per-channel 3×3 spatial kernel, fixed at gI with $g = 1$ or $g \in \{0.90, 0.95, 1.00, 1.05, 1.10\}$ (the gain may be chosen from the receiver response table), learned freely inside the trust region, or learned under a receiver-side prior when the receiver's card carries a local target operator (Appendix C.2); the fit position (objectness or energy rule); a receiver-proxy loss (none, one of the two masked losses below, or the receiver's learned proxy); the evidence tier (the first 32 anchor frames at compile time, all 64 once the second tier is requested); the certification conditions; and the fallback. Four options are not chosen freely: the channel map comes from one frozen family, the certification conditions are frozen, a program starts at the first evidence tier, and the fallback is determined by the receiver's capabilities. Requests and abstention are session actions, not options.

Fit position. The objectness rule selects cells where the sender's own head predicts objectness of at least τ_s, the 0.99 quantile of that score on the sender's own frames, recorded on its card. The energy rule selects cells whose sender feature energy, the norm of the port feature vector, exceeds the median cell energy over 12 of the sender's own training-split frames, the first 12 ego frames by scene and frame index of a 250-frame training-split pool, and does not read the card. The default program uses the objectness rule when the sender has a single-row head and its card carries the field.

Requests. A request obtains one kind of extra evidence, whose bytes add to the session's cost, and refits an instance with it: unsketched sender feature rows from the first-tier frames (up to 2 MiB), at positions chosen uniformly or by sender support; the second anchor tier; or a conditional evidence block, extra sender evidence selected by a condition (low or high sender support, or another scene) that can be requested only when the sender's card lists that condition, with the size the card states. The exchange volume in Section 4.2, a median of 11.8 MB per session of which 2.6 MB are the two cards, is the median over the test sessions of the 18 development pairs with the gate enabled; the establishment time there comes from a different set of 18 sessions, run on validation frames.

Normalization. Besides none, the two options are the affine map $(z - \mu_s)/\sigma_s \cdot \sigma_r + \mu_r$, using the port means and standard deviations recorded on the two cards, and the same map with the receiver's foreground-conditioned statistics, in the manner of AdaIN (Huang & Belongie, 2017). Statistics are matched per channel when the two ports have the same channel count. Otherwise a scalar map built from the medians of the statistics is used.

Gain selection. When the controller asks for a gain chosen from the receiver response table, the fitting procedure selects the gain that maximizes the smaller of the two recorded AP differences. It falls back to $g = 1$ when no complete answer exists, and it fixes the gain before fitting. This card read does not count against the controller's query budget.

Receiver-proxy losses. A receiver-proxy loss is a masked squared difference between a receiver-side proxy map computed on the mapped sender features and the same map computed on the receiver's own features. It is evaluated on one block of unsketched port rows from the spent anchor frames, centred on a fit-position cell drawn with a fixed seed, and only on the block's cells whose receptive context lies entirely inside the block. Two proxy maps are available: the port-magnitude map that the receiver's fusion gate uses, and the receiver's per-scale occupancy scores $\mathrm{sigmoid}(h_i(\cdot)) + 10^{-4}$, where h_i is the receiver's single-row head at scale i. These are proxies for how the receiver consumes the features, not predictors of detection accuracy. A third option, a learned proxy of the receiver's task loss, is legal only when the receiver's card admits it; no receiver card admits it in the reported test sessions. The weight β in Equation (3) is $L_{\mathrm{rec}}/L_{\mathrm{prx}}$ at the start for the two masked losses, which are non-negative, and $L_{\mathrm{rec}}/|L_{\mathrm{prx}}|$ for the learned proxy, which is signed, so the two terms start with equal magnitude.

C.2 FITTING DETAILS

Sketch. Let the spent anchor frames contribute n fit-position cells. The sender rows at these cells, after the program's normalization, form $Z_s \in \mathbb{R}^{n \times C_s}$ and the receiver rows at the same cells form $Z_r \in \mathbb{R}^{n \times C_r}$, row i of both taken from the same cell of the same anchor frame. CountSketch (Charikar et al., 2002) gives each row i a bucket $h(i)$ drawn uniformly from $\{1, \ldots, m\}$ and a sign $\sigma(i)$ drawn uniformly from $\{-1, +1\}$, and forms $S \in \mathbb{R}^{m \times n}$ with $S_{h(i),i} = \sigma(i)$ and zeros elsewhere, with $m = 2048$ and one fixed seed. Both sides use the same S, $X = SZ_s$ and $Y = SZ_r$; with a shared S the sketched moments $X^\top X$ and $X^\top Y$ are unbiased for $Z_s^\top Z_s$ and $Z_s^\top Z_r$, whereas independent sketches would lose the pairing of sender and receiver rows.

Ridge solution. For sketched rows X and Y, the ridge solution K_0 solves

$$\left(X^\top X + \lambda I\right) K_0 = X^\top Y, \qquad \lambda = 10^{-2}\ \mathrm{tr}\left(X^\top X\right)/C_s, \tag{5}$$

which has a unique solution whenever $X \neq 0$. The fit starts from K_{init}, the ridge solution on the rows it uses, and certification compares with K_{ref}, the ridge solution on the plain sketch of the spent frames (Section 3.5). When the second anchor tier is requested, its rows are sketched separately and the two sketches are stacked, not added, so the Gram matrices of the two frame sets add without cross terms.

Requested rows. A request for uncompressed sender feature rows obtains a subset B of the n rows. Their contributions are removed from their buckets and the rows are appended uncompressed,

$$X_B = \begin{bmatrix} S_{:,\bar{B}}\, Z_{s,\bar{B}} \\ Z_{s,B} \end{bmatrix}, \tag{6}$$

and likewise for Y_B with the receiver's rows, so each row enters once. The fit then starts from the ridge solution of this mixed object, while K_{ref} stays the ridge solution of the plain sketch, so the requested rows do not move the certification threshold.

Optimizer. Adam runs at learning rate 10^{-3} for at most 300 steps. The adapter applies $K = UV$, then the spatial kernel. The reconstruction loss $\|XK - Y\|_F^2/(nC_r)$ has no ridge term, so its gradient at K_0, $-2\lambda K_0/(nC_r)$, is nonzero and the fit moves away from the ridge solution.

Initialization. The channel map $K = UV$ has rank $r = \min(C_s, C_r)$. With the thin singular value decomposition $K_{\mathrm{init}} = P\Sigma Q^\top$, the factors start at $U_0 = P\Sigma^{1/2}$ and $V_0 = \Sigma^{1/2}Q^\top$, so the adapter starts exactly at the closed-form solution.

Receiver-side prior. When the spatial kernel is learned under the receiver-side prior, the objective of Equation (3) gains a prior term on the kernel, built from the receiver's local target operator and weighted by one outside the β scaling, so the kernel receives gradients even without a receiver-proxy loss. Rosetta's reported sessions do not select this option.

Early stop. No step is executed when no trainable parameter is reachable from the objective, as in a revision that leaves only the spatial kernel trainable while the objective has neither a receiver-proxy loss nor the receiver-side prior.

Trust region. After every step, each parameter block is projected back into a ball around its own initialization, with radius 0.1 times the Frobenius norm of that initialization. A block whose initial norm is zero uses unit root-mean-square as its scale.

Revisions. A revision's child loads the parent's fitted parameters and freezes all of them except the spatial kernel. Because the reconstruction loss depends only on the channel map, the kernel is re-optimized only when it is learned and the objective reaches it, through a receiver-proxy loss or the receiver-side prior. Otherwise the revision sets the kernel (gI in the fixed modes) and takes no optimization step.

Certification. For a channel map K, anchor frames J and fit-position masks M_j, the relative residual is

$$e(K;J) = \left(\frac{\sum_{j\in J} \sum_{u\in M_j} \left\| z_s^{(j)}(u)\, K - z_r^{(j)}(u) \right\|_2^2}{\sum_{j\in J} \sum_{u\in M_j} \left\| z_r^{(j)}(u) \right\|_2^2} \right)^{1/2}, \tag{7}$$

where $z^{(j)}(u)$ is the port row at cell u of anchor frame j, the sender row taken after the program's normalization. With J_{sp} the anchor frames an instance's evidence tier spent and J_{ho} the remaining ones, an instance with fitted channel map $\hat{K}$ is certified if and only if every applicable condition holds:

$$\text{held-out:} \quad e(\hat{K}; J_{\mathrm{ho}}) < e(K_{\mathrm{ref}}; J_{\mathrm{ho}}), \qquad \text{applicable iff } J_{\mathrm{ho}} \neq \emptyset, \tag{8}$$

$$\text{in-sample:} \quad e(\hat{K}; J_{\mathrm{sp}}) < e(K_{\mathrm{ref}}; J_{\mathrm{sp}}), \qquad \text{always applicable.} \tag{9}$$

Once the second tier is requested, all 64 anchor frames are spent, J_{ho} is empty and only the in-sample comparison remains. Both sides of each condition share the frames, the cells and the denominator, so the verdict compares summed squared residuals. Only $\hat{K}$ at the fit-position cells enters, not the spatial kernel, the receiver-proxy loss or the receiver's detections, so the check constrains delivery without measuring detection benefit. A revision keeps its parent's channel map, evidence tier and fit position and therefore receives the parent's condition values and verdict; in the reported test sessions, all 100 revisions whose child could be matched uniquely to its parent confirm this.

C.3 Training details

Decoding and adapters. The compiler is Qwen3-4B with LoRA adapters of rank 16 and $\alpha = 32$ on the query, key, value and output projections, with dropout 0.05, decoded greedily and constrained to each stage's action schema. Generation is capped per stage at 4,096 tokens (diagnose), 512 (query), 3,072 (compile), 512 (revise or request) and 256 (deliver).

Training pairs and targets. The legality and program-preference stages use sessions on 54 directed pairs among the eight training endpoints, and the branch-preference rows come from 46 of these pairs; none of them is a target pair in our evaluation. In program preference a program is a target as often as it wins a comparison, and a program that never wins is never one. Rows are averaged within each optimization step. For revise-or-request and deliver actions, the legality stage supervises only the format of the action, not its decision.

Stages and optimization. Table 6 lists the three stages. The learning rate is constant. In the branch-preference stage, each supervised token of the revise-or-request step carries equal weight, with the step's total weight unchanged. Program preferences are measured on 241 training-split frames from 19 scenes, disjoint from the anchor frames. The last stage runs with three seeds from one selected checkpoint.

Table 6: Compiler training stages.

Stage	Initialized from	Items	Optimization
Legality	base model	324	learning rate 10^{-4}, gradient accumulation 8, at most 40,000 tokens
Program preference	legality	711 (+534 validation)	
Branch preference	selected checkpoint	199	

Checkpoint selection. We select the program-preference checkpoint by preference accuracy on 60 stratified validation comparisons. For each comparison, the preferred target must receive the higher mean per-token log-likelihood. This choice of a training checkpoint is separate from the choice of the compiler we report, which was made on validation frames of development pairs.

Delta averaging. The three branch-preference adapters, each of rank 16, are merged by averaging their weight deltas (Wortsman et al., 2022): their A factors and their B factors are stacked along the rank into one adapter of rank 48 with α unchanged, whose weight delta is exactly the average of the three.