FastAdapter: Codeword-Conditioned Adapter Generation for Cross-Encoder Alignment in CSI Feedback

Anonymous authors
Paper under double-blind review

Abstract

Independently trained CSI feedback encoders can produce incompatible codeword representations, degrading reconstruction by a fixed base-station decoder. Training an alignment model for each new encoder incurs repeated optimization. This paper proposes FastAdapter, which generates a complete adaptation module from unlabeled codewords received at the base station while preserving both endpoints. Offline, codeword refinement and reconstruction constraints prepare decoder-compatible target Adapters, and hidden-unit realignment gives functionally equivalent parameters consistent coordinates. A Gram–Raw condition encoder combines codeword coordinates, magnitudes, and inter-sample geometry to construct position-specific context. Contrastive pretraining associates this context with target parameters, and conditional diffusion generates reusable Adapters for unseen encoders. Evaluations on COST2100, WAIR-D, and DeepMIMO cover unseen training configurations and unseen architectures under a shared CSI distribution and feedback interface. Against the strongest generative baseline in each setting, FastAdapter lowers NMSE by 0.34–2.07 dB for unseen training configurations and 0.17–1.17 dB for unseen architectures. The generated Adapter reconstructs subsequent feedback without paired CSI supervision or gradient updates during adaptation, transferring offline adaptation experience across encoder models. Code is available at `https://anonymous.4open.science/r/FastAdapter-91AA`.

1 Introduction

Massive multiple-input multiple-output (MIMO) systems address this need through spatial multiplexing and beamforming, which rely on channel state information (CSI) at the base station (BS). Conveying high-dimensional CSI from user equipment (UE) incurs substantial feedback overhead. Learning-based CSI compression feedback reduces this overhead by encoding the channel into a compact codeword at the UE and reconstructing it with a BS decoder (Wen et al., 2018; Guo et al., 2022). However, compatible codeword representations are typically established through paired training. In deployments involving different vendors or model versions, an independently trained or updated UE encoder may produce representations that differ from those expected by the fixed decoder, even when the codeword dimension is unchanged. This *codeword drift* degrades reconstruction when the endpoints are connected directly (Figure 1). Reusing both models therefore requires adaptation that transforms the new encoder's codewords into representations compatible with the existing decoder.

Existing studies establish endpoint compatibility primarily through training collaboration (Saini et al., 2024; Bathala et al., 2023; Korpi et al., 2026) and interface design (Huang et al., 2024; Chen et al., 2024). Network-First (Saini et al., 2024) coordinates separate endpoint training through raw-data sharing, whereas CSI-PPPNet (Chen et al., 2024) uses linear UE projection with learned BS recovery. These approaches establish compatibility through advance coordination or prescribed encoding operations, which limits their applicability to endpoints that have already been trained independently and must remain frozen. Fitting a transformation for each new encoder introduces repeated optimization. A mechanism is therefore needed to obtain a compatible adaptation function from feedback already received at the BS.

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.

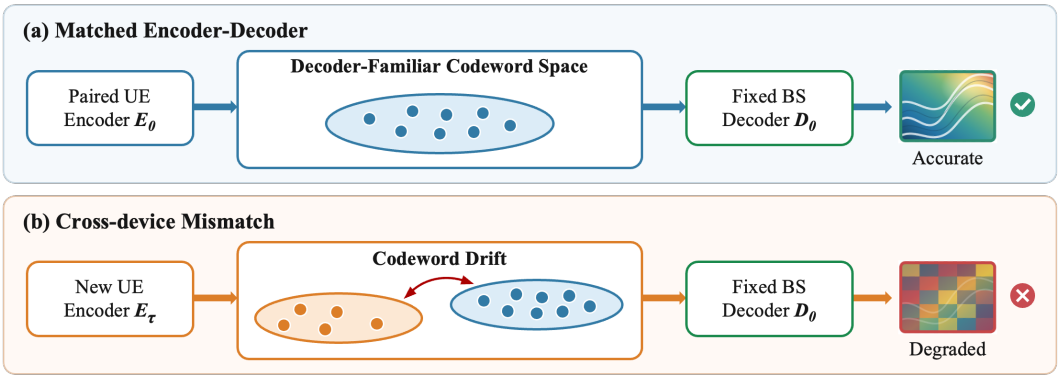


Figure 1: Codeword drift: the paired encoder E_0 produces representations compatible with the fixed decoder D_0, whereas an independently trained E_τ can change these representations and degrade reconstruction.

Conditional parameter generation learns the relationship between task observations and model weights, offering a way to transfer adaptation experience across encoders. Hypernetworks predict weights directly from task conditions (Ha et al., 2017; Zhmoginov et al., 2022), while parameter diffusion conditions iterative denoising on sample sets or behavioral observations (Soro et al., 2025; Liang et al., 2024). Applying these approaches here requires connecting codeword observations to decoder-compatible transformations. On the observation side, uploaded codewords contain both channel content and the encoder's representation, so useful adaptation information must be extracted from a finite support set. On the target side, independently trained Adapters can implement equivalent functions with different hidden-unit orderings (Ainsworth et al., 2023), creating inconsistent parameter targets. Effective generation thus needs both informative codeword conditions and compatible targets with consistent parameter coordinates.

This paper proposes FastAdapter, which generates a reusable BS-side Adapter from unlabeled support codewords. Its design connects target preparation, condition encoding, and parameter generation. Target preparation uses codeword refinement and reconstruction constraints to learn decoder-compatible Adapters, then applies ParamRealign to align hidden-unit orderings while preserving their functions. Condition encoding combines raw codeword coordinates and magnitudes with inter-sample geometry captured by a Gram matrix. Parameter-position queries extract relevant information from these features for different parts of the Adapter. Contrastive pretraining associates the resulting conditions with target parameters, and conditional diffusion learns to generate complete Adapters. These components map codeword observations to the transformation needed by the fixed decoder.

When a new encoder connects, the BS samples from codewords it has already received and generates an Adapter while keeping both endpoints frozen. Subsequent feedback reuses this Adapter, requiring neither paired CSI nor gradient optimization during adaptation. Experiments on COST2100, WAIR-D, and DeepMIMO cover unseen training configurations and unseen architectures. The main contributions are:

- **A codeword-driven adaptation paradigm.** The proposed paradigm generates a reusable BS-side Adapter for an independently trained encoder from unlabeled feedback, preserving the encoder and fixed decoder without per-encoder optimization at deployment.
- **The FastAdapter framework.** Codeword refinement prepares decoder-compatible targets; ParamRealign aligns functionally equivalent parameters; Gram–Raw conditions and contrastive alignment guide diffusion to generate complete Adapters for unseen encoders.
- **Reconstruction gains with measured adaptation cost.** FastAdapter improves over the strongest generative baseline by 0.34–2.07 dB under unseen training configurations and 0.17–1.17 dB under unseen architectures. Its COST2100/in adaptation pipeline takes 5.99 s.

2 Related Work

2.1 Endpoint Compatibility in CSI Feedback

CSI feedback compatibility studies primarily follow two routes: training collaboration, which shares training information or coordinates optimization to establish compatible representations (Saini et al., 2024; Bathala et al., 2023; 3rd Generation Partnership Project, 2025; Korpi et al., 2026), and interface design, which constrains representations or encoding operations to reduce endpoint coupling (Huang et al., 2024; Chen et al., 2024).

For training collaboration, Network-First (Saini et al., 2024) uses raw-data sharing to train compatible UE and BS models separately. Korpi et al. (2026) further train interoperable endpoints without disclosing either party's model and validate them in a prototype system. For interface design, CSI-PPPNet (Chen et al., 2024) combines linear UE projection with BS-side plug-and-play recovery, concentrating learning at the BS and reusing a learned denoiser across compression ratios.

These approaches establish compatibility through coordination during training or a prescribed feedback interface. Their dependence on these arrangements limits direct reuse of endpoints that have already been trained independently and must remain frozen. Adapting a new encoder to a fixed decoder using only received codewords therefore requires a mechanism for obtaining the transformation between their representations.

2.2 Conditional Neural Parameter Generation

Conditional parameter generation converts task information into model weights to reduce deployment-time optimization. The main approaches are hypernetworks that predict weights directly (Ha et al., 2017; Zhmoginov et al., 2022; Scott et al., 2024; Chen et al., 2025; Charakorn et al., 2025) and parameter diffusion that generates weights or their latent representations through iterative denoising (Jin et al., 2024; Soro et al., 2025; Wang et al., 2025; Liang et al., 2024).

Hypernetworks learn a shared mapping from conditions to task-dependent parameters. HyperTransformer (Zhmoginov et al., 2022), for example, generates convolutional network weights from support samples. In CSI feedback, AdapCsiNet (Liu et al., 2025) generates reconstruction-layer parameters from scene graphs of the propagation environment for environmental adaptation. Diffusion methods instead learn conditional parameter distributions. D2NWG (Soro et al., 2025) aggregates sample-set features to guide weight generation, while Make-An-Agent (Liang et al., 2024) learns contrastive representations of behavioral trajectories to condition policy-parameter generation.

These methods establish sample-conditioned weight generation, but do not specifically connect unlabeled feedback codewords to transformations compatible with a fixed decoder. Codeword conditions must account for both channel content and encoder representations, while equivalent Adapter functions can have different parameter coordinates. FastAdapter addresses this relationship through codeword conditioning, decoder-compatible target preparation, and function-preserving realignment, generating complete Adapters that can be reused for subsequent feedback.

3 Problem Formulation

Let $\mathbf{H} \sim \mathcal{P}_{\mathbf{H}}$ be preprocessed CSI in $\mathcal{H} = \mathbb{R}^{2\times N_c\times N_t}$, where $\mathcal{P}_{\mathbf{H}}$ is the CSI distribution, the first dimension contains real and imaginary components, N_c is the number of retained delay/frequency bins, and N_t is the number of antennas. The reference encoder $E_0 : \mathcal{H} \to \mathbb{R}^C$ and fixed decoder $D_0 : \mathbb{R}^C \to \mathcal{H}$ are trained as a pair, with codeword dimension C and reconstruction $\widehat{\mathbf{H}}_0 = D_0(E_0(\mathbf{H}))$. A task $\tau = (a, s)$ identifies an encoder E_τ by architecture a and training configuration s. For the same $\mathbf{H}$, its codeword $\mathbf{z}_\tau = E_\tau(\mathbf{H})$ can differ from $E_0(\mathbf{H})$ even when both have dimension C. Since D_0 was trained on the latter representation, direct decoding $D_0(\mathbf{z}_\tau)$ can increase reconstruction error. This codeword mismatch arises from encoder models under a shared CSI distribution, codeword dimension, and Adapter architecture.

A BS-side Adapter $A_\theta : \mathbb{R}^C \to \mathbb{R}^C$ transforms the new encoder's codewords while both endpoints remain frozen. With single-sample reconstruction error ε and parameter space Θ, the ideal supervised adaptation objective is

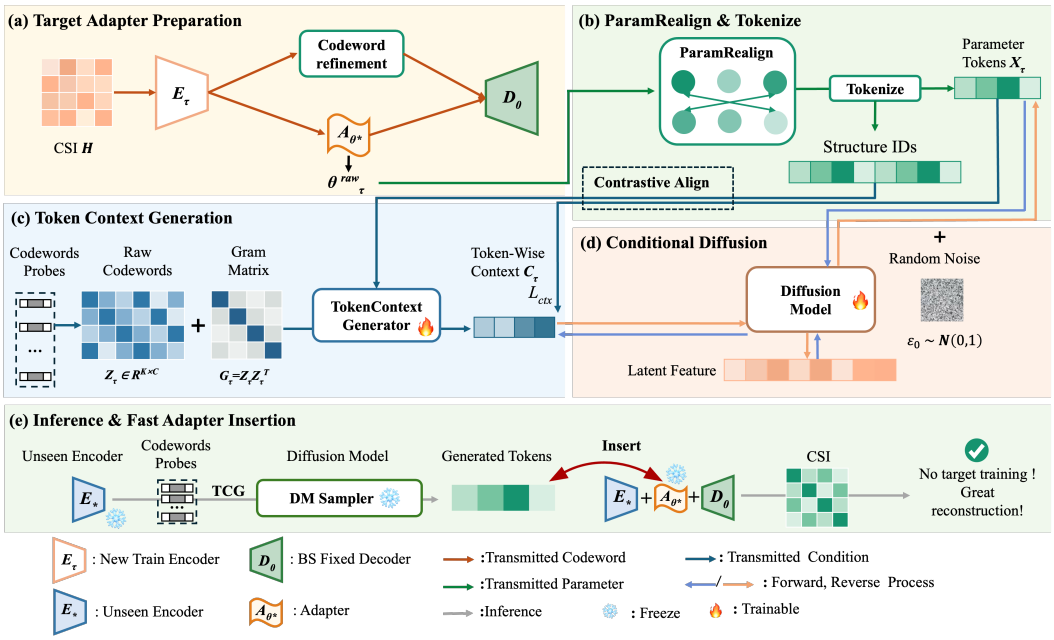


Figure 2: FastAdapter: offline learning pairs codeword conditions with compatible, realigned Adapter parameters. During deployment, a fixed generator uses support codewords received at the BS to generate an Adapter, which is reused between the frozen encoder and decoder.

$$\theta_\tau^* \in \arg\min_{\theta\in\Theta} \underbrace{\mathbb{E}_{\mathbf{H}\sim\mathcal{P}_\mathbf{H}}[\varepsilon(D_0(A_\theta(E_\tau(\mathbf{H}))),\mathbf{H})]}_{\mathcal{E}_{\mathrm{adp}}(\theta;\tau)}. \tag{1}$$

Per-encoder optimization approximates this objective using paired CSI. FastAdapter instead learns to generate parameters from a finite support set S_τ selected from the N_τ uploaded codewords $\mathcal{B}_\tau = (\mathbf{z}_{\tau,n})_{n=1}^{N_\tau}$ when an encoder is registered or updated. The BS has D_0, but corresponding raw CSI, encoder weights, and architecture labels are unavailable as adaptation inputs. A mapping learned on seen encoders produces

$$\widehat{\theta}_\tau = F_\phi(S_\tau), \qquad \widehat{\mathbf{H}} = D_0(A_{\widehat{\theta}_\tau}(\mathbf{z})). \tag{2}$$

where $\mathbf{z}$ is a subsequent codeword from the same encoder and ϕ collectively denotes the learned condition-encoder and denoiser parameters. The mapping F_ϕ is trained for the fixed decoder D_0 and remains fixed for unseen encoders; the generated Adapter is stored for subsequent feedback. Supervision from seen encoders prepares offline targets, and support codewords drive parameter generation at deployment.

4 Method

FastAdapter links codeword observations to complete Adapter parameters through target preparation, condition encoding, and conditional generation (Figure 2). Offline preparation establishes compatible targets and consistent parameter coordinates. The condition encoder and generator learn from these targets and remain fixed when adapting unseen encoders.

4.1 Adapter Parameter Preparation

Decoder-compatible targets. Let D_τ be the decoder paired with a seen encoder E_τ, and let MSE denote elementwise mean squared error. Pointwise regression to reference codewords $E_0(\mathbf{H})$ over-

looks the reconstruction capabilities learned by the source pair. FastAdapter instead refines a candidate codeword in the input space of D_0 to reproduce the source reconstruction $\widehat{\mathbf{H}}_\tau^{\mathrm{src}} = D_\tau(E_\tau(\mathbf{H}))$:

$$\widetilde{\mathbf{z}}_\tau = \arg\min_{\mathbf{u}\in\mathbb{R}^C} \mathrm{MSE}\Big(D_0(\mathbf{u}), \widehat{\mathbf{H}}_\tau^{\mathrm{src}}\Big). \tag{3}$$

Finite optimization yields an approximate refinement target. The Adapter applies affine alignment followed by residual feedforward blocks with layer normalization, learning to map $\mathbf{z}_\tau$ to this target. With B_{A} residual blocks indexed by $b = 0, \ldots, B_{\mathrm{A}} - 1$, the transformation is

$$\begin{aligned} \mathbf{h}^{(0)} &= \mathbf{W}_a \mathbf{z}_\tau + \mathbf{b}_a, \\ \mathbf{h}^{(b+1)} &= \mathbf{h}^{(b)} + \rho\ \mathrm{FFN}_b\Big(\mathrm{LN}_b(\mathbf{h}^{(b)})\Big). \end{aligned} \tag{4}$$

Here, $\mathbf{W}_a \in \mathbb{R}^{C\times C}$ and $\mathbf{b}_a \in \mathbb{R}^C$ are the affine parameters, ρ scales the residual update, and the Adapter output is $\mathbf{h}^{(B_{\mathrm{A}})}$. The affine layer handles a global coordinate transformation, while the residual blocks provide nonlinear corrections. Its objective combines codeword alignment, CSI reconstruction, and reference re-encoding:

$$\begin{aligned} \mathcal{L}_{\mathrm{A}} = \lambda_{\mathrm{align}}\,\mathrm{MSE}(A_\theta(\mathbf{z}_\tau), \widetilde{\mathbf{z}}_\tau) + \lambda_{\mathrm{recon}}\,\mathrm{MSE}(\widehat{\mathbf{H}}_\theta, \mathbf{H}) \\ + \lambda_{\mathrm{ref}}\,\mathrm{MSE}(E_0(\widehat{\mathbf{H}}_\theta), E_0(\mathbf{H})), \quad \widehat{\mathbf{H}}_\theta = D_0(A_\theta(\mathbf{z}_\tau)). \end{aligned} \tag{5}$$

The nonnegative λ coefficients weight the three terms, which respectively align the output with the refined codeword, constrain reconstruction error, and preserve consistency after re-encoding through E_0. Paired CSI, source reconstructions, and E_0 prepare offline targets for seen encoders: refinement updates the candidate codeword through the frozen decoder, and Adapter fitting updates the transformation parameters. The source reconstruction determines the refined target, while the CSI and reference re-encoding terms constrain the fitted Adapter, producing $\theta_\tau^{\mathrm{raw}}$ for generation learning. A newly connected encoder uses the trained generator directly. Appendix E gives the Adapter architecture and expanded losses.

Function-preserving parameter realignment. Independently trained Adapters can represent similar transformations with different hidden-unit orderings (Ainsworth et al., 2023). ParamRealign compares centered, normalized hidden activations on offline calibration samples and uses Hungarian matching to align each residual block to a shared reference. For centered, normalized activation vectors $\overline{\mathbf{a}}_{\tau,b,i}$ of source unit i and $\overline{\mathbf{a}}_{\mathrm{ref},b,j}$ of reference unit j, matching minimizes cosine distance over the set $\mathcal{P}$ of $d_b \times d_b$ permutation matrices, where d_b is the hidden width of block b; the matrix inner product sums the selected costs:

$$\begin{aligned} \mathbf{D}_{\tau,b}(i,j) &= 1 - \langle \overline{\mathbf{a}}_{\tau,b,i}, \overline{\mathbf{a}}_{\mathrm{ref},b,j} \rangle, \\ \mathbf{P}^*_{\tau,b} &= \arg\min_{\mathbf{P}\in\mathcal{P}} \langle \mathbf{P}, \mathbf{D}_{\tau,b} \rangle. \end{aligned} \tag{6}$$

The matched permutation jointly reorders the first-layer weights and bias and the second-layer input dimension, preserving the Adapter's function. The shared reference is updated by averaging aligned activations from training Adapters and is then frozen. The realigned target $\theta_\tau^\dagger$ retains the learned transformation in consistent parameter coordinates. Appendix A gives the reference updates, function-preserving transformations, and numerical checks.

Structured parameter tokens. For each of the N parameter tensors indexed by j, training Adapters provide fixed scalar statistics μ_j and $\sigma_j > 0$ for standardization. The shared parameter manifest divides standardized tensors into tokens $\mathbf{X}_{\tau,0} \in \mathbb{R}^{L\times d_{\mathrm{tok}}}$; matrix tokens stay within output rows, and vector tokens contain contiguous segments. A mask $\mathbf{M}$ excludes padding. Each token has tensor, local-position, block, role, and weight/bias identifiers, whose embeddings sum to $\mathbf{e}_l$. Validation and deployment reuse the same manifest and statistics. Appendix E gives the standardization and tokenization formulas.

4.2 Condition Encoding from Codeword Support Sets

Gram–Raw representation. The BS randomly selects K distinct complete codewords from $\mathcal{B}_\tau$, constructs a support matrix $\mathbf{Z}_\tau \in \mathbb{R}^{K\times C}$, and computes its Gram matrix:

$$S_\tau = (\mathbf{z}^{\mathrm{s}}_{\tau,k})_{k=1}^{K}, \qquad \mathbf{Z}_\tau = [\mathbf{z}^{\mathrm{s}}_{\tau,1}, \ldots, \mathbf{z}^{\mathrm{s}}_{\tau,K}]^{\mathsf{T}}, \qquad \mathbf{G}_\tau = \mathbf{Z}_\tau \mathbf{Z}_\tau^{\mathsf{T}}. \tag{7}$$

The superscript s identifies support codewords; $\mathbf{Z}_\tau$ has K rows and $\mathbf{G}_\tau \in \mathbb{R}^{K\times K}$. The raw branch retains codeword coordinates and magnitudes; the Gram branch exposes inter-sample geometry. For an orthogonal coordinate change $\mathbf{Z}_\tau\mathbf{Q}$ with $\mathbf{Q} \in \mathbb{R}^{C\times C}$ and $\mathbf{Q}\mathbf{Q}^{\mathsf{T}} = \mathbf{I}$, the Gram matrix remains unchanged while the raw representation changes.

Position-specific context. The condition encoder projects the two branches into row features, retaining raw means and magnitudes through a bypass. Row-position embeddings preserve within-task correspondence between branches. Learned-query bilinear pooling produces a global condition $\mathbf{g}_\tau$, while unpooled features form $\mathbf{U}_\tau$. The global condition modulates parameter-position embeddings, which retrieve support information through cross-attention:

$$\begin{gathered}(\boldsymbol{\gamma}_{\tau,l}, \boldsymbol{\beta}_{\tau,l}) = P_l(\mathbf{g}_\tau), \qquad \mathbf{q}_{\tau,l} = \mathbf{e}_l \odot (1 + \tanh \boldsymbol{\gamma}_{\tau,l}) + \boldsymbol{\beta}_{\tau,l}, \\ \mathbf{C}_\tau = \mathrm{Seq}(\mathrm{CrossAttn}(\mathbf{q}_{\tau,1:L}, \mathbf{U}_\tau)) \in \mathbb{R}^{L\times d_{\mathrm{ctx}}}.\end{gathered} \tag{8}$$

Here, P_l is the learned map predicting scale and shift at token position l, $\odot$ denotes elementwise multiplication, d_{ctx} is the context width, and Seq is a bidirectional sequence model coordinating parameter tokens. Each context token $\mathbf{c}_{\tau,l}$ combines task information with its structural role. Under the shared CSI distribution, the network learns codeword statistics associated with adaptation requirements.

Condition–parameter alignment. Because all Adapters share a token layout, position information alone cannot distinguish tasks. Contrastive pretraining associates context with target parameters using

$$\mathcal{L}_{\mathrm{ctx}} = \mathcal{L}_{\mathrm{task}} + \lambda_{\mathrm{cos}}\mathcal{L}_{\mathrm{cos}} + \lambda_{\mathrm{pos}}\mathcal{L}_{\mathrm{pos}}. \tag{9}$$

The coefficients λ_{cos} and λ_{pos} weight the auxiliary losses. A frozen projection P_{p} maps masked target tokens to the context width. The symmetric InfoNCE loss $\mathcal{L}_{\mathrm{task}}$ (van den Oord et al., 2018) matches pooled condition and parameter embeddings from the same encoder task against other tasks. The cosine loss $\mathcal{L}_{\mathrm{cos}}$ further brings positive pairs together. The local loss $\mathcal{L}_{\mathrm{pos}}$ aligns each context token with the corresponding parameter feature after subtracting the batch mean at that position. Let $\mathbf{y}_{i,l}$ be the projected target feature and $m_{i,l}$ indicate a valid token. For a minibatch of B encoder tasks indexed by i, j, the residual used by the local cosine objective is

$$\Delta\mathbf{y}_{i,l} = \mathrm{sg}\left(\mathbf{y}_{i,l} - \frac{\sum_{j=1}^{B} m_{j,l}\mathbf{y}_{j,l}}{\sum_{j=1}^{B} m_{j,l}}\right), \tag{10}$$

Here, sg stops gradients. These residual targets use stopped gradients, emphasizing task-dependent variation beyond the common parameter layout. Pretraining updates the condition network while the parameter projection remains fixed. Appendix E gives all loss definitions and masking rules.

4.3 Conditional Parameter Generation

Denoising training. Conditional diffusion (Ho et al., 2020) learns the distribution of standardized Adapter tokens given $\mathbf{C}_\tau$. With cumulative signal-retention coefficient $\bar{\alpha}_t$ and Gaussian noise $\boldsymbol{\epsilon}$, training uses

$$\begin{aligned}\mathbf{X}_{\tau,t} &= \sqrt{\bar{\alpha}_t}\mathbf{X}_{\tau,0} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \\ \mathcal{L}_{\mathrm{diff}} &= \mathbb{E}_{\tau,t,\boldsymbol{\epsilon}}\left[\omega_t\big((1-\kappa)\mathcal{L}_{\mathrm{elem}} + \kappa\mathcal{L}_{\mathrm{tensor}}\big)\right].\end{aligned} \tag{11}$$

Here, t is the diffusion timestep, ψ denotes denoiser parameters, κ balances masked elementwise and per-tensor noise errors, and ω_t is a timestep weight. The mask excludes padding; Appendix E defines both losses and the weighting. The condition generator, initialized by contrastive pretraining, is jointly optimized with denoiser ϵ_ψ, linking support observations to the parameter distribution.

Generation and deployment. For an unseen encoder, the BS constructs the support condition and samples from Gaussian initial tokens. Classifier-free guidance (Ho & Salimans, 2022) combines conditional and unconditional predictions, and 50 DDIM steps (Song et al., 2021) produce Adapter tokens. Sampling updates are deterministic given the initial noise, but different initial draws can produce different parameters. For the unseen encoder τ_{new}, the final standardized tokens $\widehat{\mathbf{X}}_{\tau_{\mathrm{new}},0}$ recover the complete Adapter as

$$\widehat{\theta}_{\tau_{\mathrm{new}}} = \mathrm{Unstd}\Big(\mathrm{Tok}^{-1}\Big(\widehat{\mathbf{X}}_{\tau_{\mathrm{new}},0}\Big); \{\mu_j, \sigma_j\}_{j=1}^{N}\Big). \tag{12}$$

Here, Tok^{-1} reverses tokenization using the parameter manifest, and Unstd restores each tensor with its fixed training-side statistics. The BS loads $\widehat{\theta}_{\tau_{\mathrm{new}}}$ between $E_{\tau_{\mathrm{new}}}$ and D_0 and reuses it for subsequent codewords. Appendix E provides the sampling updates and recovery details.

5 Experiments

5.1 Experimental Setup

Datasets and generalization protocols. The experiments evaluate fixed-decoder CSI reconstruction on COST2100 indoor/outdoor (Liu et al., 2012), WAIR-D (Huangfu et al., 2022), and DeepMIMO (Alkhateeb, 2019), with one fixed TransNet decoder per dataset. The encoder library contains seven families: CsiNet (Wen et al., 2018), CRNet (Lu et al., 2020), CLNet (Ji & Li, 2021), TransNet (Cui et al., 2022), ResNet, MLP-AE, and Attention-CNN. The unseen-training-configuration protocol groups encoders by training configuration and splits them into training, validation, and test sets at an $8{:}1{:}1$ ratio. The unseen-architecture protocol uses seven leave-one-architecture-out folds. Each group and its derived data remain in one split. Codeword dimension, compression ratio, and Adapter architecture are shared within each adaptation task.

Support and supervision. Each encoder supplies $K = 128$ support codewords randomly sampled from its training CSI split. Encoders held out during generator training provide support codewords in the same way. The condition encoder reads only codewords, and NMSE is evaluated on independent test CSI.

No Adapter, Mean Adapter, Nearest Adapter, and Affine Adapter compare direct connection, mean parameters, parameter retrieval, and affine transformation. Nearest Adapter retrieves a seen encoder's Adapter by cosine similarity over complete support-codeword representations, without query CSI or gradient updates. The generative baselines are HyperTransformer (Zhmoginov et al., 2022), Cond P-Diff (Jin et al., 2024), D2NWG (Soro et al., 2025), and RPG (Wang et al., 2025). They use the same task splits, target Adapters, codeword conditions, and parameter manifest. Individual Adapter separately optimizes an Adapter for each test encoder using paired CSI and serves as a supervised reference. Appendix F details the model library, baseline implementations, and training settings. Generation uses 50 DDIM steps. Validation selects target-loss weights, with the search reported in Appendix B.

Implementation. The input shapes are $2 \times 32 \times 32$ for COST2100, $2 \times 64 \times 64$ for WAIR-D, and $2 \times 32 \times 64$ for DeepMIMO. Parameter tokens have width 512. AdamW trains the generator for 1000 epochs with a peak learning rate of 10^{-4}, linear warmup over the first 5% of training, and cosine decay thereafter. Validation selects target-loss weights $(\lambda_{\mathrm{align}}, \lambda_{\mathrm{recon}}, \lambda_{\mathrm{ref}}) = (1, 1000, 2)$. Training-task statistics determine standardization and the ParamRealign reference, which remain fixed for held-out encoders.

Metric. For test sample i, let $\mathbf{H}_i$ and $\widehat{\mathbf{H}}_i$ be the true and reconstructed CSI. Reconstruction is measured by normalized mean squared error (NMSE, lower is better):

$$\mathrm{NMSE} = 10\log_{10}\left(\frac{\sum_i \|\widehat{\mathbf{H}}_i - \mathbf{H}_i\|_F^2}{\sum_i \|\mathbf{H}_i\|_F^2}\right). \tag{13}$$

Table 1: NMSE (dB, ↓) across encoder-generalization settings. Bold and underlined values mark the best and second-best generative methods. Individual Adapter uses additional paired CSI supervision.

Method	Unseen training configuration				Unseen architecture			
	COST2100 in	COST2100 out	WAIR-D	DeepMIMO	COST2100 in	COST2100 out	WAIR-D	DeepMIMO
Mismatch diagnostics and supervised comparison								
No Adapter	23.07	24.15	29.32	25.80	22.21	17.62	20.25	31.03
Mean Adapter	0.47	1.26	-0.71	1.50	0.08	-0.45	1.49	-1.61
Nearest Adapter	13.11	19.38	11.77	7.91	15.47	20.38	12.76	17.02
Affine Adapter	-3.75	0.32	-2.14	-5.92	0.85	4.17	-3.66	-2.70
Individual Adapter	-23.18	-16.81	-22.90	-22.07	-23.31	-15.12	-22.06	-22.52
Parameter generation at deployment								
HyperTransformer	-16.07	-9.69	-17.04	-17.56	-14.31	-7.01	-16.05	-12.91
Cond P-Diff	-19.90	-12.47	-19.20	-17.13	-18.38	-9.33	-17.72	-17.06
D2NWG	<u>-20.69</u>	-12.77	-18.34	<u>-20.47</u>	<u>-20.01</u>	-11.60	<u>-18.86</u>	<u>-19.34</u>
RPG	-20.28	<u>-13.55</u>	<u>-20.71</u>	-20.45	-19.85	<u>-12.57</u>	-18.19	-18.22
FastAdapter	**-21.49**	**-15.62**	**-21.05**	**-21.11**	**-21.18**	**-12.74**	**-19.99**	**-19.87**

Errors and channel energy are accumulated over each encoder's test samples before conversion to dB. Scores are then macro-averaged over encoders.

5.2 Cross-Encoder Reconstruction

Table 1 shows that direct connection yields 23.07–29.32 dB under unseen training configurations and 17.62–31.03 dB under unseen architectures. Mean and Affine Adapters reduce the error but remain well above the supervised Individual Adapter. Nearest Adapter yields 7.91–19.38 and 12.76–20.38 dB, respectively, showing that support-codeword similarity alone does not reliably identify a compatible transformation.

Under unseen training configurations, FastAdapter has the lowest NMSE among generative methods in all four settings, improving over the strongest alternative in each by 0.34–2.07 dB. The largest gain is on COST2100/out (-15.62 versus RPG's -13.55 dB). Under unseen architectures, it again leads in all four settings by 0.17–1.17 dB; on DeepMIMO, it reduces direct-connection NMSE from 31.03 to -19.87 dB.

The gap to the supervised Individual Adapter is 0.96–1.85 dB under unseen training configurations and 2.07–2.65 dB under unseen architectures. The larger gap in every corresponding architecture setting suggests that cross-architecture parameter generation is harder here. Appendix C gives baseline-by-baseline comparisons.

5.3 Component Ablations

Table 2 shows that every ablation raises NMSE in both protocols. Under unseen training configurations, removing ParamRealign or the global condition increases NMSE by 7.76–11.17 or 9.62–13.66 dB. Under unseen architectures, the corresponding increases are 6.68–11.02 and 8.85–12.96 dB. Gram and Raw conditioning, position identifiers, position-specific context, and contrastive pretraining also benefit every setting. Appendix A confirms that realignment preserves the target Adapter's function.

5.4 Adaptation Computation

Under unseen training configurations, FastAdapter takes 5.87–6.08 s, faster than Online Affine in three of four settings, with 15.19–18.91 dB lower NMSE (Tables 3 and 1). Under unseen architectures, it takes 5.84–6.21 s, is faster in two of four settings, and lowers NMSE by 16.33–22.03

Table 2: Component ablations across eight settings (NMSE in dB; lower is better).

Configuration	Unseen training configuration				Unseen architecture			
	COST2100 in	COST2100 out	WAIR-D	DeepMIMO	COST2100 in	COST2100 out	WAIR-D	DeepMIMO
ParamRealign								
w/o ParamRealign	-10.56	-7.86	-9.88	-11.66	-11.83	-6.06	-8.97	-10.84
Condition construction								
w/o Gram	-17.22	-11.77	-16.65	-17.33	-17.24	-10.47	-16.04	-15.04
w/o Raw	-16.45	-11.25	-15.49	-16.15	-16.13	-9.29	-15.35	-15.01
Context generation								
Global broadcasting	-14.94	-10.39	-15.51	-14.56	-14.77	-9.02	-13.12	-13.09
w/o position IDs	-9.76	-7.51	-8.63	-10.03	-9.22	-6.27	-8.90	-9.16
w/o global condition	-8.18	-6.00	-7.39	-8.53	-8.78	-3.89	-7.75	-6.91
w/o contrastive	-15.82	-12.01	-16.13	-15.39	-15.65	-8.64	-14.50	-14.31
FastAdapter (full)	**-21.49**	**-15.62**	**-21.05**	**-21.11**	**-21.18**	**-12.74**	**-19.99**	**-19.87**

Table 3: Adaptation time (s); † marks gradient updates. Individual Adapter times are rounded to seconds.

Method	Unseen training configuration				Unseen architecture			
	COST2100 in	COST2100 out	WAIR-D	DeepMIMO	COST2100 in	COST2100 out	WAIR-D	DeepMIMO
Individual Adapter†	17415	18265	23410	21832	17167	16959	24841	22249
Online Affine†	5.51	6.84	8.92	7.65	4.86	6.12	9.45	8.18
Nearest Adapter	1.27	1.15	2.14	1.86	1.94	1.38	2.45	1.92
FastAdapter	5.99	5.87	5.91	6.08	5.91	6.21	5.88	5.84

dB. Nearest Adapter is faster but much less accurate; Individual Adapter requires much longer per-encoder optimization with paired CSI.

Timing covers the support-sample encoder pass through Adapter loading, excluding codeword collection, transmission, and offline training. On COST2100/in, DDIM sampling takes 4.48 s (74.8% of the 5.99 s total; Appendix D). Subsequent feedback reuses the generated Adapter.

6 Limitations

The experiments isolate encoder mismatch by keeping the CSI distribution, feedback interface, and BS decoder fixed. Their results therefore do not yet cover channel-domain shifts or RF hardware mismatch. Generator training uses supervised target Adapters, whereas adaptation to a new encoder uses only unlabeled codewords. The effects of support-set composition, quantized or noisy feedback, and generation variability remain to be measured. A deterministic generator trained with the same conditions would further clarify the contribution of diffusion.

7 Conclusion

This paper studies codeword drift when independently trained CSI encoders connect to a fixed decoder. FastAdapter learns to generate reusable BS-side Adapters from unlabeled codewords: offline target preparation and ParamRealign establish consistent training targets, while Gram–Raw conditioning guides parameter diffusion. On COST2100, WAIR-D, and DeepMIMO, the generated Adapters achieve lower NMSE than every tested generative baseline under both unseen training configurations and unseen architectures. Each Adapter is generated for a new encoder and reused for subsequent feedback, with both endpoints frozen and no deployment-time gradient updates.

AI USE STATEMENT

OpenAI Codex was used to assist with manuscript writing and polishing, including improvements to clarity, organization, and wording. It was also used for literature retrieval and discovery to identify potentially relevant work. The authors retain full responsibility for the accuracy and integrity of the manuscript, including its scientific claims, experimental results, and references.

REPRODUCIBILITY STATEMENT

The method and evaluation protocols are specified in Sections 4 and 5. The appendix provides the ParamRealign algorithm and numerical checks (Appendix A), the Adapter loss-weight search (Appendix B), and implementation details, including model-library splits, support-codeword sampling, optimization settings, and evaluation (Appendix F). The anonymized code repository linked in the abstract provides the implementation.

REFERENCES

3rd Generation Partnership Project. Study on Artificial Intelligence (AI)/Machine Learning (ML) for NR Air Interface. Technical Report TR 38.843, Version 19.0.0, 3GPP, 2025. URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3983.

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git Re-Basin: Merging Models Modulo Permutation Symmetries. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CQsmMYmlP5T.

Ahmed Alkhateeb. DeepMIMO: A Generic Deep Learning Dataset for Millimeter Wave and Massive MIMO Applications, 2019. URL https://arxiv.org/abs/1902.06435.

Sharan Mourya Bathala, Saidhiraj Amuru, and Kiran Kumar Kuchi. Multi-Task Learning for Massive MIMO CSI Feedback. In *The Third International Conference on Artificial Intelligence and Machine Learning Systems*, pp. 1–4, 2023. doi: 10.1145/3639856.3639898.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant Transformer Adaption. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 7485–7514, 2025. URL https://proceedings.mlr.press/v267/charakorn25a.html.

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Benjamin Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative Adapter: Contextualizing Language Models in Parameters with a Single Forward Pass. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=bc3sUsS6ck.

Wei Chen, Weixiao Wan, Shiyue Wang, Peng Sun, Geoffrey Ye Li, and Bo Ai. CSI-PPPNet: A One-Sided One-for-All Deep Learning Framework for Massive MIMO CSI Feedback. *IEEE Transactions on Wireless Communications*, 23(7):7599–7611, 2024. doi: 10.1109/TWC.2023.3342735. URL https://arxiv.org/abs/2211.15851.

Yaodong Cui, Aihuang Guo, and Chunlin Song. TransNet: Full Attention Network for CSI Feedback in FDD Massive MIMO System. *IEEE Wireless Communications Letters*, 11(5):903–907, 2022. doi: 10.1109/LWC.2022.3149416.

Jiajia Guo, Chao-Kai Wen, Shi Jin, and Geoffrey Ye Li. Overview of Deep Learning-Based CSI Feedback in Massive MIMO Systems. *IEEE Transactions on Communications*, 70(12):8017–8045, 2022. doi: 10.1109/TCOMM.2022.3217777. URL https://arxiv.org/abs/2206.14383.

David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance, 2022. URL https://arxiv.org/abs/2207.12598.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Yihang Huang, Chenhui Ye, and Yijia Feng. Codebook-Agnostic Separate Training for DL-Based CSI Feedback Enhancement. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, pp. 1–6, 2024. doi: 10.1109/WCNC57260.2024.10570565.

Yourui Huangfu, Jian Wang, Shengchen Dai, Rong Li, Jun Wang, Chongwen Huang, and Zhaoyang Zhang. WAIR-D: Wireless AI Research Dataset, 2022. URL https://arxiv.org/abs/2212.02159.

Sijie Ji and Mo Li. CLNet: Complex Input Lightweight Neural Network Designed for Massive MIMO CSI Feedback. *IEEE Wireless Communications Letters*, 10(10):2318–2322, 2021. doi: 10.1109/LWC.2021.3100493.

Xiaolong Jin, Kai Wang, Dongwen Tang, Wangbo Zhao, Yukun Zhou, Junshu Tang, and Yang You. Conditional LoRA Parameter Generation, 2024. URL https://arxiv.org/abs/2408.01415.

Dani Korpi et al. Demonstrating Interoperable Channel State Feedback Compression with Machine Learning. *IEEE Wireless Communications*, 33(3):126–133, 2026. doi: 10.1109/MWC.2025.3630622.

Yongyuan Liang, Tingqiang Xu, Kaizhe Hu, Guangqi Jiang, Furong Huang, and Huazhe Xu. Make-An-Agent: A Generalizable Policy Network Generator with Behavior-Prompted Diffusion. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/223eb69a2f2fb97fde58eaa958babb7a-Abstract-Conference.html.

Jiayi Liu, Jiajia Guo, Yiming Cui, Chao-Kai Wen, and Shi Jin. AdapCsiNet: Environment-Adaptive CSI Feedback via Scene Graph-Aided Deep Learning. In *2025 IEEE Global Communications Conference (GLOBECOM)*, pp. 6388–6393, 2025. doi: 10.1109/GLOBECOM59602.2025.11431744.

Lingfeng Liu, Claude Oestges, Juho Poutanen, Katsuyuki Haneda, Pertti Vainikainen, Frédéric Quitin, Fredrik Tufvesson, and Philippe De Doncker. The COST 2100 MIMO Channel Model. *IEEE Wireless Communications*, 19(6):92–99, 2012. doi: 10.1109/MWC.2012.6393523.

Zhilin Lu, Jintao Wang, and Jian Song. Multi-Resolution CSI Feedback with Deep Learning in Massive MIMO System. In *Proceedings of the IEEE International Conference on Communications*, pp. 1–6, 2020. doi: 10.1109/ICC40277.2020.9149229.

Aakash Saini, Jee Hyun Kim, Amir Ahmadian Tehrani, Yunchou Xing, and Wolfgang Gerstacker. Network-First Separate Training with Raw Dataset Sharing: A Training Approach for AI/ML-Driven CSI Feedback. In *Proceedings of the IEEE International Conference on Communications Workshops*, pp. 1950–1955, 2024. doi: 10.1109/ICCWorkshops59551.2024.10615736.

Jonathan Scott, Hossein Zakerinia, and Christoph H. Lampert. PeFLL: Personalized Federated Learning by Learning to Learn. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=MrYiwlDRQO.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

Bedionita Soro, Bruno Andreis, Hayeon Lee, Wonyong Jeong, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-Based Neural Network Weights Generation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=j8WHjM9aMm.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding, 2018. URL https://arxiv.org/abs/1807.03748.

Kai Wang, Dongwen Tang, Wangbo Zhao, Konstantin Schürholt, Zhangyang Wang, and Yang You. Scaling Up Parameter Generation: A Recurrent Diffusion Approach. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/85de6d4171114a24a083b5cdbc49091f-Abstract-Conference.html.

Chao-Kai Wen, Wan-Ting Shih, and Shi Jin. Deep Learning for Massive MIMO CSI Feedback. *IEEE Wireless Communications Letters*, 7(5):748–751, 2018. doi: 10.1109/LWC.2018.2818160.

Andrey Zhmoginov, Mark Sandler, and Maksym Vladymyrov. HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27075–27098, 2022. URL https://proceedings.mlr.press/v162/zhmoginov22a.html.

APPENDIX

TABLE OF CONTENTS

A ParamRealign Algorithm and Diagnostics **14**

A.1 Constructing the Offline Reference 14

A.2 Function-Preserving Permutations 14

A.3 Matching Costs and Numerical Checks 14

B Target Adapter Loss-Weight Search **15**

C NMSE Differences from Generative Baselines **16**

C.1 Reconstruction by Method 16

D Computational Runtime Breakdown for New-Encoder Adaptation **18**

E Method Implementation Details **18**

E.1 Target Adapter training 18

E.2 Parameter standardization and tokenization 19

E.3 Per-token context generation 19

E.4 Contrastive alignment 20

E.5 Denoising training 21

E.6 Adapter generation at inference 21

F Experimental Protocol and Implementation **22**

A PARAMREALIGN ALGORITHM AND DIAGNOSTICS

This appendix expands the parameter realignment in Section 4.1, specifying its reference construction and function-preserving transformations.

A.1 CONSTRUCTING THE OFFLINE REFERENCE

ParamRealign standardizes the hidden-unit ordering of target Adapters to provide comparable coordinates for parameter generation. Offline calibration uses codeword samples from the training tasks and compares activations in the row order defined by calibration indices. For each residual block, activations are extracted after the first linear layer and GELU. Let the activation matrix be $\mathbf{A}_{\tau,b} \in \mathbb{R}^{M_{\text{cal}} \times d_b}$, where M_{cal} is the number of calibration samples, d_b is the hidden width, and each column corresponds to a hidden unit. Each column is centered and divided by its norm, with the denominator lower-bounded by 10^{-8}. Denote the resulting matrix by $\overline{\mathbf{A}}_{\tau,b}$.

The reference $\mathbf{R}_b^{(r)} \in \mathbb{R}^{M_{\text{cal}} \times d_b}$ at iteration r has one normalized activation column per hidden unit. The initial reference is taken from an Adapter in the training set. At each iteration, cosine distances to the current reference are computed using Equation (6), and the Hungarian algorithm finds a one-to-one assignment of hidden units. At iteration r, let $\mathbf{P}_{\tau,b}^{(r)}$ solve Equation (6) with source units as rows and reference units as columns. Its transpose $\mathbf{\Pi}_{\tau,b}^{(r)} = (\mathbf{P}_{\tau,b}^{(r)})^\mathsf{T}$ reorders source hidden units into reference order. For training tasks $\mathcal{T}_{\text{tr}}$, the reference update is

$$\mathbf{R}_b^{(r+1)} = \frac{1}{|\mathcal{T}_{\text{tr}}|} \sum_{\tau \in \mathcal{T}_{\text{tr}}} \overline{\mathbf{A}}_{\tau,b} (\mathbf{\Pi}_{\tau,b}^{(r)})^\mathsf{T}. \tag{14}$$

At the next iteration, reference columns are centered and normalized again before matching costs are computed. The barycenter averages aligned activation features from training Adapters. The diagnostic run uses a fixed 10 iterations; after the reference is frozen, validation and test Adapters are matched to it for target diagnostics.

A.2 FUNCTION-PRESERVING PERMUTATIONS

For a two-layer feedforward transformation within a residual block, with block input $\mathbf{x} \in \mathbb{R}^C$, first-layer weights $\mathbf{W}_1 \in \mathbb{R}^{d_b \times C}$ and bias $\mathbf{b}_1 \in \mathbb{R}^{d_b}$, second-layer weights $\mathbf{W}_2 \in \mathbb{R}^{C \times d_b}$ and bias $\mathbf{b}_2 \in \mathbb{R}^C$, and elementwise GELU activation σ,

$$f(\mathbf{x}) = \mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2, \tag{15}$$

let $\mathbf{\Pi}$ reorder the source hidden units into reference order; its rows index reference units and its columns index source units. The joint transformations

$$\mathbf{W}_1' = \mathbf{\Pi} \mathbf{W}_1, \qquad \mathbf{b}_1' = \mathbf{\Pi} \mathbf{b}_1, \qquad \mathbf{W}_2' = \mathbf{W}_2 \mathbf{\Pi}^\mathsf{T} \tag{16}$$

preserve the input–output function. Elementwise activation satisfies $\sigma(\mathbf{\Pi}\mathbf{v}) = \mathbf{\Pi}\sigma(\mathbf{v})$, and $\mathbf{\Pi}^\mathsf{T}\mathbf{\Pi} = \mathbf{I}$, hence $f'(\mathbf{x}) = f(\mathbf{x})$. The operation changes only the ordering of FFN hidden units. The input affine transformation, LayerNorm, second-layer bias, and residual main path remain unchanged. The realigned $\theta_\tau^\dagger$ is therefore functionally equivalent to the original target θ_τ^{raw}.

A.3 MATCHING COSTS AND NUMERICAL CHECKS

The diagnostic collection contains 1,213 Adapters, with 973, 121, and 119 in the training, validation, and test splits, respectively, and uses 4,096 samples for activation calibration. Table 4 summarizes this collection.

Figure 4 reports a separate COST2100/in training-only run of reference iteration: the mean matching cost decreases from 0.6360 to 0.4233, or approximately 33.4%. In contrast, Figure 3 and Table 4 compare identity assignments with optimal matching under the frozen reference in the three-split diagnostic collection, yielding mean cost reductions of approximately 47%. The former compares

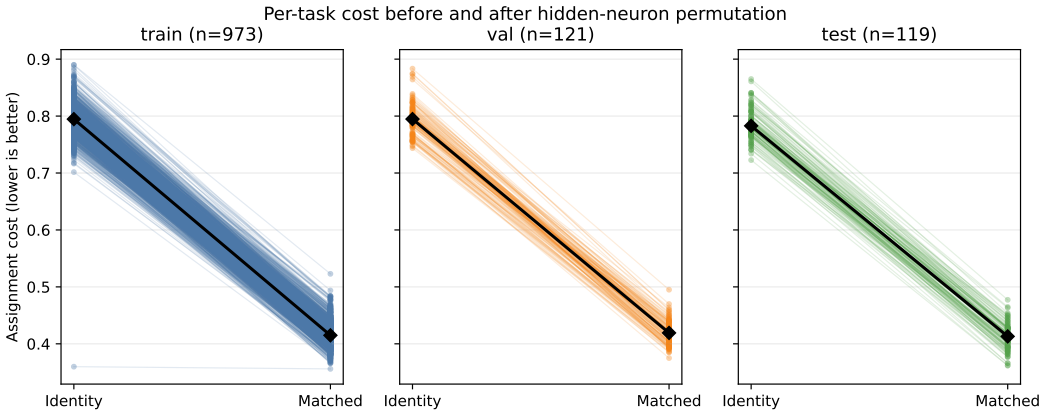


Figure 3: Identity-assignment costs before ParamRealign and costs after Hungarian matching. All three splits share a frozen barycenter constructed using training Adapters only.

Table 4: Activation-based matching costs and functional-equivalence checks. The maximum absolute difference is computed between the outputs of the complete Adapter before and after alignment, using identical inputs for the function check.

Split (n)	Identity	Realigned	Abs. reduction	Rel. reduction	Pass rate	Max. abs. diff.
Training (973)	0.7948	0.4169	0.3779	47.5%	100%	5.72×10^{-6}
Validation (121)	0.7950	0.4193	0.3757	47.3%	100%	5.72×10^{-6}
Test (119)	0.7858	0.4124	0.3734	47.5%	100%	7.63×10^{-6}
All (1,213)	0.7940	0.4167	0.3773	47.5%	100%	7.63×10^{-6}

references at different iterations, while the latter compares whether units are reordered under the same reference; the two reductions have different baselines and task collections.

Each complete Adapter is evaluated on identical inputs before and after alignment. All numerical checks satisfy a maximum output-difference threshold of 10^{-4}. The global maximum absolute difference is 7.63×10^{-6}, and the mean relative MSE is 2.52×10^{-14}. These diagnostics check functional consistency in the finite-precision implementation; the main-text ablation evaluates realignment's effect on generation learning.

By iteration 3, the mean cost reaches 0.4493, accounting for 87.8% of the total absolute improvement in this run.

B Target Adapter Loss-Weight Search

The final loss weights used in the main text are $(\lambda_{\text{align}}, \lambda_{\text{recon}}, \lambda_{\text{ref}}) = (1, 1000, 2)$. This section reports the full two-dimensional hyperparameter grid search. With $\lambda_{\text{align}} = 1$ fixed, the search ranges are

$$\lambda_{\text{recon}} \in \{0, 200, 400, 600, \ldots, 2000\},$$
$$\lambda_{\text{ref}} \in \{0, 0.5, 1, 1.5, 2\}.$$

The grid contains 55 configurations that compare the weights of the target Adapter's reconstruction loss and reference re-encoding constraint. Figure 5 shows NMSE for both evaluation protocols on a shared color scale, with exact values in each cell and the minima outlined in orange. Both protocols attain their lowest NMSE at $(\lambda_{\text{recon}}, \lambda_{\text{ref}}) = (1000, 2)$, with values of -21.49 dB and -21.18 dB, respectively. The selected weight pair therefore remains unchanged.

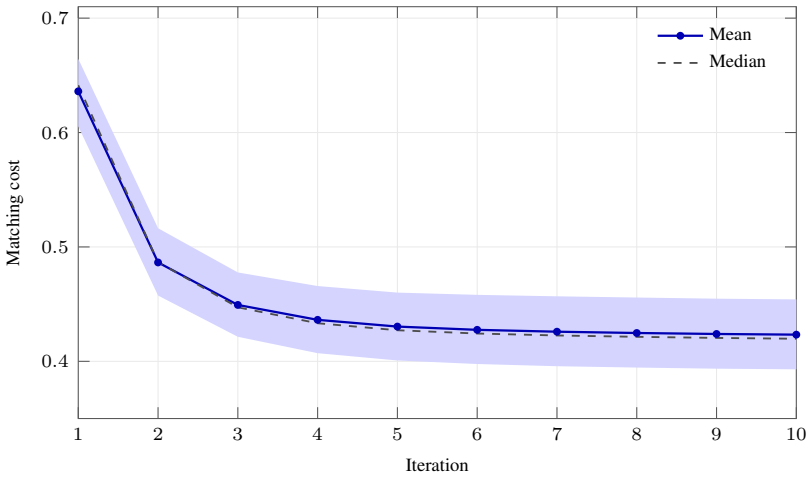


Figure 4: ParamRealign reference updates for 946 COST2100/in training Adapters using 4,096 activation-calibration samples. Shading shows the 10th–90th percentile range across tasks.

Table 5: Best loss weights selected by the two-dimensional search under the two evaluation protocols.

Protocol	λ_{align}	λ_{recon}	λ_{ref}	NMSE (dB)
Unseen training configuration	1	1000	2	-21.49
Unseen architecture	1	1000	2	-21.18

Table 5 reports configuration selection for target training, whereas Table 1 evaluates cross-encoder reconstruction after complete parameter generation.

C NMSE Differences from Generative Baselines

This section compares FastAdapter with each of four generative baselines in NMSE. The difference relative to baseline b is

$$\text{Gain}(b) = \text{NMSE}(b) - \text{NMSE}(\text{FastAdapter}), \tag{17}$$

where b is a generative baseline; positive values mean lower NMSE for FastAdapter. Tables 6 and 7 report differences from HyperTransformer, Cond P-Diff, D2NWG, and RPG.

FastAdapter has lower NMSE than all four generative baselines under each protocol. Relative to the strongest alternative in each setting, its gains are 0.34–2.07 dB for unseen training configurations and 0.17–1.17 dB for unseen architectures. The smallest difference occurs on COST2100/out with an unseen architecture. Each entry directly subtracts the NMSE values in the main table.

C.1 Reconstruction by Method

Mismatch and simple adaptation. In Table 1, direct connection yields 23.07–29.32 dB for unseen training configurations and 17.62–31.03 dB for unseen architectures. Mean and Affine Adapters reduce the error but remain above the supervised Individual Adapter. Nearest Adapter yields 7.91–19.38 and 12.76–20.38 dB, respectively, showing that support-codeword similarity does not reliably identify a transferable Adapter.

Parameter generation for unseen encoders. Under unseen training configurations, FastAdapter improves over the strongest competing generator by 0.34–2.07 dB and trails Individual Adapter

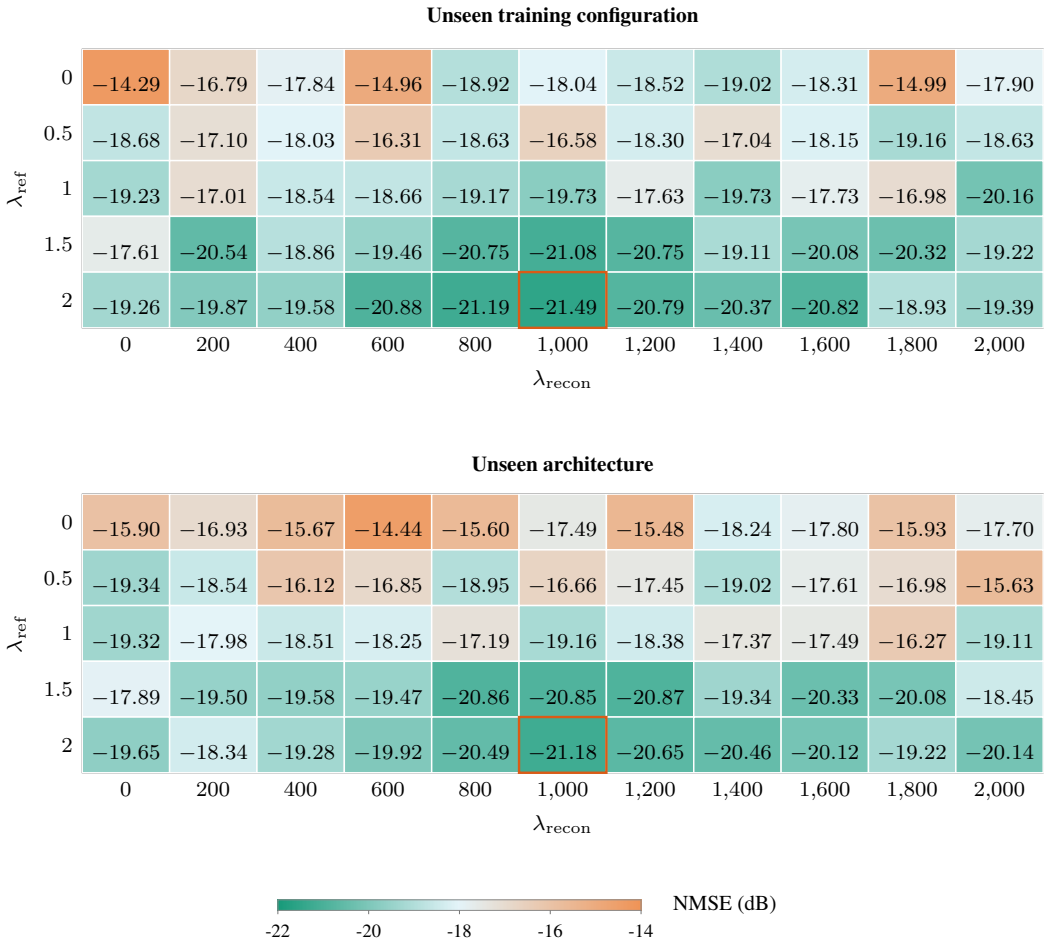


Figure 5: Two-dimensional search over target Adapter loss weights with $\lambda_{\text{align}} = 1$ fixed. Both protocols share an orange-to-teal NMSE scale, with teal indicating lower error. Orange outlines mark the minima, and cell labels report NMSE in dB.

Table 6: NMSE differences (baseline minus FastAdapter) under the unseen-training-configuration protocol (dB; higher is better).

Baseline	COST2100/in	COST2100/out	WAIR-D	DeepMIMO
HyperTransformer	5.42	5.93	4.01	3.55
Cond P-Diff	1.59	3.15	1.85	3.98
D2NWG	0.80	2.85	2.71	0.64
RPG	1.21	2.07	0.34	0.66

by 0.96–1.85 dB. Under unseen architectures, these ranges are 0.17–1.17 and 2.07–2.65 dB. On DeepMIMO with an unseen architecture, FastAdapter reduces NMSE from 31.03 dB under direct connection to −19.87 dB. Tables 6 and 7 give the differences from each generative baseline.

Table 7: NMSE differences (baseline minus FastAdapter) under the unseen-architecture protocol (dB; higher is better).

Baseline	COST2100/in	COST2100/out	WAIR-D	DeepMIMO
HyperTransformer	6.87	5.73	3.94	6.96
Cond P-Diff	2.80	3.41	2.27	2.81
D2NWG	1.17	1.14	1.13	0.53
RPG	1.33	0.17	1.80	1.65

D Computational Runtime Breakdown for New-Encoder Adaptation

Table 8 breaks down adaptation computation on COST2100/in. The experiment measures the pipeline from the encoder forward pass on support samples to Adapter parameter loading, totaling 5.99 seconds. Measurements use the same hardware, precision, and batch settings with CUDA synchronization. Codeword accumulation, wireless transmission, target preparation for seen encoders, and generator training are excluded.

At deployment, the BS forms its support set from previously received feedback codewords. The table assigns 1.05 seconds to the encoder forward pass and 4.94 seconds to the five subsequent stages. Both are measured stages of the experimental computation pipeline.

Table 8: Stage runtimes for FastAdapter adaptation computation. Percentages use the 5.99-second computational pipeline total as the denominator and are rounded to one decimal place.

Stage	Time (s)	Share
Support-codeword encoding (128 samples)	1.05	17.5%
Gram–Raw condition construction	0.06	1.0%
Per-token context generation	0.21	3.5%
50-step DDIM sampling	4.48	74.8%
Inverse standardization and detokenization	0.07	1.2%
Adapter parameter loading	0.12	2.0%
Computational pipeline total	5.99	100.0%

Table 9: Exact Individual Adapter training times (s), corresponding to the rounded values in Table 3.

Protocol	COST2100/in	COST2100/out	WAIR-D	DeepMIMO
Unseen training configuration	17414.73	18264.50	23409.76	21832.48
Unseen architecture	17166.67	16959.47	24840.66	22248.61

E Method Implementation Details

E.1 Target Adapter training

A_θ learns a samplewise mapping from $\mathbf{z}_\tau$ to $\widetilde{\mathbf{z}}_\tau$. It first performs affine alignment, then learns nonlinear corrections through residual feedforward network (FFN) blocks with layer normalization (LN), following Equation (4).

Here, $b = 0, \ldots, B_{\mathrm{A}} - 1$ indexes residual blocks, LN_b and FFN_b are the corresponding operations, ρ is a residual scaling coefficient, and θ contains all Adapter parameters. The output is $A_\theta(\mathbf{z}_\tau) = \mathbf{h}^{(B_{\mathrm{A}})}$. Let

$$\mathbf{z}_0 = E_0(\mathbf{H}), \qquad \overline{\mathbf{z}}_\tau = A_\theta(\mathbf{z}_\tau), \qquad \widehat{\mathbf{H}}_\theta = D_0(\overline{\mathbf{z}}_\tau), \qquad \mathbf{z}_\theta^{\mathrm{ref}} = E_0(\widehat{\mathbf{H}}_\theta),$$

denote the reference codeword, Adapter output, fixed decoder reconstruction, and re-encoding by the reference encoder, respectively. The training objective is

$$\begin{aligned}
\mathcal{L}_{\mathrm{align}} &= \mathrm{MSE}(\overline{\mathbf{z}}_\tau, \widetilde{\mathbf{z}}_\tau)\,, \\
\mathcal{L}_{\mathrm{recon}} &= \mathrm{MSE}\Big(\widehat{\mathbf{H}}_\theta, \mathbf{H}\Big)\,, \\
\mathcal{L}_{\mathrm{ref}} &= \mathrm{MSE}\big(\mathbf{z}_\theta^{\mathrm{ref}}, \mathbf{z}_0\big)\,, \\
\mathcal{L}_{\mathrm{A}}(\theta;\tau) &= \lambda_{\mathrm{align}}\mathcal{L}_{\mathrm{align}} + \lambda_{\mathrm{recon}}\mathcal{L}_{\mathrm{recon}} + \lambda_{\mathrm{ref}}\mathcal{L}_{\mathrm{ref}}.
\end{aligned} \tag{18}$$

where the λ coefficients weight the losses. The codeword alignment term supervises the Adapter in the fixed decoder's coordinate system; the reconstruction term directly constrains CSI error; and the reference re-encoding term enforces consistency between the reconstruction encoded by E_0 and the reference codeword. Optimizing this composite objective yields the trained parameters $\theta_\tau^{\mathrm{raw}}$ for subsequent generation learning. Paired CSI, source reconstructions, and the reference encoder E_0 are used for offline preparation. Adapting a new encoder requires only condition encoding, parameter generation, and decoding with the fixed D_0.

E.2 Parameter standardization and tokenization

Parameter tensors differ in shape and numerical scale. FastAdapter computes a mean and standard deviation for each tensor across training Adapters, then organizes the parameters into fixed-width tokens according to the parameter manifest:

$$\widetilde{\boldsymbol{\theta}}_{\tau,j} = \frac{\boldsymbol{\theta}^\dagger_{\tau,j} - \mu_j}{\sigma_j}, \qquad \mathbf{X}_{\tau,0} = \mathrm{Tok}\Big(\widetilde{\boldsymbol{\theta}}_{\tau,1:N}\Big) \in \mathbb{R}^{L\times d_{\mathrm{tok}}}. \tag{19}$$

Here, N is the number of parameter tensors, Tok applies the shared manifest, and μ_j and $\sigma_j > 0$ summarize the elements of tensor j across training Adapters; validation and inference reuse these fixed statistics. Two-dimensional weight matrices are split within output rows, while vector parameters are split into contiguous segments. Each segment shorter than d_{tok} is zero-padded. Tokens cross neither parameter-tensor boundaries nor output-row boundaries. The total number of tokens is L, and the mask $\mathbf{M} \in \{0,1\}^{L\times d_{\mathrm{tok}}}$ marks valid elements. The structural identifiers and embeddings of token l are

$$\begin{aligned}
\mathbf{r}_l &= \big(i_l^{\mathrm{tensor}}, i_l^{\mathrm{local}}, i_l^{\mathrm{block}}, i_l^{\mathrm{role}}, i_l^{\mathrm{kind}}\big)\,, \\
\mathbf{e}_l &= \sum_q \mathrm{Emb}_q(i_l^q).
\end{aligned} \tag{20}$$

These five indices identify the tensor, local position, residual block, role, and weight/bias type, respectively; q ranges over these five fields, Emb_q is a learned embedding lookup, and $\mathbf{e}_l$ is the sum of the five embeddings.

E.3 Per-token context generation

Different parameter tensors perform affine alignment, normalization, and nonlinear transformation, requiring information relevant to their respective positions from the same codeword support set. TokenContextGenerator $\mathcal{G}$ combines the Gram–Raw representation with structural identifiers to construct context for each parameter token:

$$\mathbf{C}_\tau = \mathcal{G}(\mathbf{Z}_\tau, \mathbf{G}_\tau, \mathbf{r}_{1:L}) = [\mathbf{c}_{\tau,1}, \ldots, \mathbf{c}_{\tau,L}]^{\mathsf{T}} \in \mathbb{R}^{L\times d_{\mathrm{ctx}}}. \tag{21}$$

Here, d_{ctx} is the context width, and each $\mathbf{c}_{\tau,l}$ combines parameter-position information with adaptation information in the support codewords. The condition encoder first projects each row of Gram features and applies a coordinate-sensitive projection to each raw codeword, retaining its mean and magnitude through a bypass. Sample-position embeddings identify rows of the current support matrix and preserve correspondence between the two branches within the task. Bilinear pooling with learned queries then combines row-level information into a global condition vector $\mathbf{g}_\tau$, while unpooled features are retained as $\mathbf{U}_\tau = [\mathbf{U}_\tau^{\mathrm{G}}; \mathbf{U}_\tau^{\mathrm{Z}}]$, where the semicolon concatenates Gram and Raw rows.

The global vector predicts scale and shift coefficients for each structural position. These coefficients modulate the position embeddings, which retrieve relevant support-sample features through cross-attention. A bidirectional sequence model then coordinates dependencies among parameter tokens:

$$\begin{aligned}(\boldsymbol{\gamma}_{\tau,l}, \boldsymbol{\beta}_{\tau,l}) &= P_l(\mathbf{g}_\tau),\\ \mathbf{q}_{\tau,l} &= \mathbf{e}_l \odot (1 + \tanh \boldsymbol{\gamma}_{\tau,l}) + \boldsymbol{\beta}_{\tau,l},\\ \mathbf{C}_\tau &= \mathrm{Seq}(\mathrm{CrossAttn}(\mathbf{q}_{\tau,1:L}, \mathbf{U}_\tau))\,. \end{aligned} \tag{22}$$

Thus, the raw-codeword branch retains coordinate-dependent drift, while the Gram branch provides geometry among support samples. The global and row-level pathways jointly determine how codeword observations inform each parameter position.

E.4 Contrastive alignment

All Adapters share the same token layout, so parameter positions alone cannot distinguish the adaptation requirements of different encoders. Contrastive pretraining on conditions and target parameters learns their correspondence. For training task i, the target tokens and global embeddings are defined as

$$\begin{aligned}\mathbf{Y}_i &= P_{\mathrm{p}}(\mathbf{M} \odot \mathbf{X}_{i,0}) = [\mathbf{y}_{i,1}, \ldots, \mathbf{y}_{i,L}]^{\mathsf{T}}\,,\\ \mathbf{c}_i &= \mathrm{norm}(\mathrm{Pool}_{\mathrm{c}}(\mathbf{C}_i))\,,\\ \mathbf{p}_i &= \mathrm{norm}(\mathrm{Mean}_m(\mathbf{Y}_i))\,. \end{aligned} \tag{23}$$

P_{p} is a frozen parameter projection whose output dimension equals the context width d_{ctx}. The local vector $\mathbf{c}_{i,l}$ is the context token in Equation (21). Here, $\mathrm{Pool}_{\mathrm{c}}$ denotes condition pooling, Mean_m is the masked mean over valid tokens, and norm normalizes a vector. Global embeddings $\mathbf{c}_i$ and $\mathbf{p}_i$ form a positive pair when they come from the same task and a negative pair otherwise. Letting $s_{ij} = \mathbf{c}_i^{\mathsf{T}} \mathbf{p}_j / \tau_{\mathrm{c}}$, the InfoNCE loss (van den Oord et al., 2018) is

$$\begin{aligned}\mathcal{L}_{\mathrm{task}} = -\frac{1}{2B} \sum_{i=1}^{B} \Bigg[&\log \frac{\exp(s_{ii})}{\sum_{j=1}^{B} \exp(s_{ij})}\\ &+ \log \frac{\exp(s_{ii})}{\sum_{j=1}^{B} \exp(s_{ji})} \Bigg], \end{aligned} \tag{24}$$

where B is the batch size in encoder tasks and τ_{c} is the temperature. Let $m_{i,l} \in \{0, 1\}$ indicate whether token l of task i is valid. The loss also uses token residuals centered across the batch with stopped gradients as defined in Equation (10).

Here, sg stops gradients through the residual targets, and $m_{i,l} = 1$ indicates a token containing valid parameter elements. Both $\Delta\mathbf{y}_{i,l}$ and $\mathbf{c}_{i,l}$ belong to $\mathbb{R}^{d_{\mathrm{ctx}}}$. The three pretraining losses are

$$\mathcal{L}_{\cos} = \frac{1}{B}\sum_{i=1}^{B}\left(1 - \mathbf{c}_i^{\top}\mathbf{p}_i\right),$$
$$\mathcal{L}_{\text{pos}} = \frac{\sum_{i=1}^{B}\sum_{l=1}^{L} m_{i,l}\left[1 - \cos(\mathbf{c}_{i,l}, \Delta\mathbf{y}_{i,l})\right]}{\sum_{i=1}^{B}\sum_{l=1}^{L} m_{i,l}},$$
$$\mathcal{L}_{\text{ctx}} = \mathcal{L}_{\text{task}} + \lambda_{\cos}\mathcal{L}_{\cos} + \lambda_{\text{pos}}\mathcal{L}_{\text{pos}}. \tag{25}$$

$\mathcal{L}_{\text{task}}$ learns condition–parameter correspondence across tasks, $\mathcal{L}_{\cos}$ brings positive pairs closer, and $\mathcal{L}_{\text{pos}}$ directs local context toward task-dependent variation at the same parameter position. Freezing P_{p} keeps parameter targets stable, while pretraining updates the condition-side network. Vector normalization and cosine computation impose a positive lower bound on norms to avoid undefined operations at zero norm.

E.5 Denoising training

After pretraining, a conditional Gaussian diffusion model (Ho et al., 2020) learns $p(\mathbf{X}_{\tau,0} \mid \mathbf{C}_\tau)$ in token space. Given target tokens, the forward process constructs a noisy state at step t:

$$\mathbf{X}_{\tau,t} = \sqrt{\bar{\alpha}_t}\,\mathbf{X}_{\tau,0} + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{26}$$

Here, $\bar{\alpha}_t$ is the cumulative signal-retention coefficient and $\boldsymbol{\epsilon}$ is Gaussian noise. The condition generator $\mathcal{G}$, initialized from contrastive pretraining, is jointly trained with the denoiser ϵ_ψ parameterized by ψ:

$$\widehat{\boldsymbol{\epsilon}} = \epsilon_\psi(\mathbf{X}_{\tau,t}, t, \mathbf{C}_\tau). \tag{27}$$

Let $\mathbf{M}_j$ mark the valid token elements belonging to parameter tensor j, so that $\sum_{j=1}^{N}\mathbf{M}_j = \mathbf{M}$. With $\mathbf{E} = \boldsymbol{\epsilon} - \widehat{\boldsymbol{\epsilon}}$, the elementwise and tensor-balanced losses are

$$\mathcal{L}_{\text{elem}} = \frac{\|\mathbf{M}\odot\mathbf{E}\|_F^2}{\|\mathbf{M}\|_1},$$
$$\mathcal{L}_{\text{tensor}} = \frac{1}{N}\sum_{j=1}^{N}\frac{\|\mathbf{M}_j\odot\mathbf{E}\|_F^2}{\|\mathbf{M}_j\|_1}. \tag{28}$$

The denoising objective is

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{\tau,t,\boldsymbol{\epsilon}}\left[\omega_t\big((1-\kappa)\mathcal{L}_{\text{elem}} + \kappa\mathcal{L}_{\text{tensor}}\big)\right]. \tag{29}$$

Here, $\kappa \in [0,1]$ controls tensor balancing. For noise prediction, the optional Min-SNR weight is $\omega_t = \min(\text{SNR}_t, \gamma_{\text{snr}})/\text{SNR}_t$, where $\text{SNR}_t = \bar{\alpha}_t/(1-\bar{\alpha}_t)$; γ_{snr} is the clipping threshold, and taking $\gamma_{\text{snr}} = \infty$ gives $\omega_t = 1$.

This objective learns the conditional parameter distribution by predicting the noise added during training; the valid-element mask excludes padding from the accumulated error.

E.6 Adapter generation at inference

For an unseen encoder τ_{new}, the BS samples a support set from its uploaded codewords, constructs the condition using Equation (7), and starts sampling from $\mathbf{X}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where T is the first timestep in the selected reverse schedule. The task subscript τ_{new} on sampling states is omitted below. Condition dropout enables unconditional prediction. Classifier-free guidance (Ho & Salimans, 2022) combines the unconditional prediction $\widehat{\boldsymbol{\epsilon}}_\emptyset$ and conditional prediction $\widehat{\boldsymbol{\epsilon}}_{\text{cond}}$:

$$\widehat{\boldsymbol{\epsilon}}_{\mathrm{cfg}} = \widehat{\boldsymbol{\epsilon}}_{\emptyset} + w\left(\widehat{\boldsymbol{\epsilon}}_{\mathrm{cond}} - \widehat{\boldsymbol{\epsilon}}_{\emptyset}\right),$$
$$\widehat{\mathbf{X}}_{0,t} = \frac{\mathbf{X}_t - \sqrt{1-\bar{\alpha}_t}\,\widehat{\boldsymbol{\epsilon}}_{\mathrm{cfg}}}{\sqrt{\bar{\alpha}_t}},$$
$$\mathbf{X}_{t'} = \sqrt{\bar{\alpha}_{t'}}\,\widehat{\mathbf{X}}_{0,t} + \sqrt{1-\bar{\alpha}_{t'}}\,\widehat{\boldsymbol{\epsilon}}_{\mathrm{cfg}}. \tag{30}$$

Here, w is the guidance strength, $\widehat{\mathbf{X}}_{0,t}$ is the estimated clean state, and $t' < t$ is the next selected sampling timestep; at the terminal step, $\bar{\alpha}_{t'} = 1$. The last two equations describe deterministic DDIM updates along the sampling timesteps (Song et al., 2021). The update trajectory is deterministic given the initial noise, but different initial noise samples can still produce different Adapters. Let $\widehat{\mathbf{X}}_{\tau_{\mathrm{new}},0}$ denote the final generated standardized tokens for the unseen encoder. The complete Adapter parameters are recovered by Equation (12).

Here, Unstd reverses per-tensor standardization using the fixed μ_j and σ_j, and Tok^{-1} reverses tokenization according to the same manifest. The Adapter parameterized by $\widehat{\theta}_{\tau_{\mathrm{new}}}$ is inserted between the frozen $E_{\tau_{\mathrm{new}}}$ and D_0. The support set is used to generate parameters once during adaptation, and subsequent codewords reuse the Adapter. The fixed decoder reconstructs CSI from $A_{\widehat{\theta}_{\tau_{\mathrm{new}}}}(\mathbf{z}_{\tau_{\mathrm{new}}})$.

F Experimental Protocol and Implementation

Datasets and encoder splits. The evaluation covers CSI reconstruction on COST2100 indoor and outdoor scenarios (Liu et al., 2012), WAIR-D (Huangfu et al., 2022), and DeepMIMO (Alkhateeb, 2019). The encoder model library contains CsiNet (Wen et al., 2018), CRNet (Lu et al., 2020), CLNet (Ji & Li, 2021), TransNet (Cui et al., 2022), ResNet, a multilayer perceptron autoencoder (MLP-AE), and an attention-based convolutional neural network (Attention-CNN). Each dataset uses a fixed TransNet decoder as the target. Codeword dimension, compression ratio, and Adapter architecture remain consistent within each adaptation task.

The model library is partitioned by groups defined by training configurations or architectures. The unseen-training-configuration protocol includes all encoder architectures and uses training configurations as the split unit, with an $8{:}1{:}1$ training/validation/test ratio. Encoders in the same group and their derived data remain in the same split. The unseen-architecture protocol performs seven leave-one-architecture-out evaluations, each holding out an entire architecture family for testing and using the remaining families for generator training and validation. The two protocols assess adaptation to changes in training configurations and encoder architectures, respectively.

Support codewords and query data. For each task, $K = 128$ samples are randomly selected from the codewords produced by that encoder on its training CSI split to form the support set in Equation (7). Encoders held out during generator training also provide support codewords from their training CSI split. Condition construction takes codewords as input; NMSE is computed on independent test CSI.

Baselines and supervised reference. No Adapter directly connects a new encoder to the fixed decoder. Mean Adapter and Nearest Adapter use the mean training-side parameters and a retrieved target Adapter, respectively, while Affine Adapter provides an affine-transformation baseline. Nearest Adapter computes cosine similarity over the full-dimensional support-codeword representation, selects the closest seen encoder's Adapter, and loads its parameters. Generative baselines include HyperTransformer (Zhmoginov et al., 2022), Cond P-Diff (Jin et al., 2024), D2NWG (Soro et al., 2025), and RPG (Wang et al., 2025). They use the same model-library splits, target Adapters, codeword conditioning information, and parameter manifest.

Individual Adapter trains a separate adapter using paired CSI for each test encoder, providing a reconstruction reference with additional supervision to measure the gap between generation and per-encoder optimization.

Metric and implementation. NMSE follows Equation (13).

Reconstruction errors and channel energy are first accumulated over each encoder's test samples. The ratio is converted to dB, and scores are then macro-averaged across encoders.

The input sizes for COST2100, WAIR-D, and DeepMIMO are $2 \times 32 \times 32$, $2 \times 64 \times 64$, and $2 \times 32 \times 64$, respectively. The parameter-token width is 512. The generator is trained with AdamW for 1000 epochs at a peak learning rate of 10^{-4}, with linear warmup over the first 5% of training followed by cosine decay. Inference uses 50 DDIM steps. The target Adapter loss weights are $(\lambda_{\text{align}}, \lambda_{\text{recon}}, \lambda_{\text{ref}}) = (1, 1000, 2)$, selected through validation; Appendix B reports the two-dimensional search. DDIM updates are deterministic given the initial noise, which remains a source of generation randomness.