JIVEADAPTER: A MULTI-TASK ADDITIVE LOW-RANK ADAPTER VIA JOINT AND INDIVIDUAL VARIATION EXPLAINED (JIVE)

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter-efficient fine-tuning adapts pretrained models at a fraction of the cost of full fine-tuning, yet most low-rank adapters are single-task and represent each weight update multiplicatively, leaving no explicit account of what is shared across tasks and what is task-specific. We introduce `JIVEAdapter`, a multi-task *additive* low-rank adapter inspired by statistical Joint and Individual Variation Explained (JIVE). `JIVEAdapter` decomposes every weight update into a Joint structure shared across all tasks plus a per-task Individual structure, penalizes the Individual structures to be near-orthogonal to the Joint so shared and task-specific signal stay *interpretable* and separated, and allocates rank adaptively across a shared Joint pool and a per-task Individual pool. The Joint is learned once, jointly over a task group or incrementally, one task at a time, then frozen and reused as a prior for new tasks without retraining the shared part. On GLUE and SuperGLUE with DeBERTaV3-base, `JIVEAdapter` is competitive with strong single-task and multi-task low-rank baselines at a matched per-task effective rank, without extra modules such as MoE, and when a related held-in task exists its frozen Joint serves a held-out task by reusing that task's Individual with only a cheap per-direction scale, otherwise training a small new one.

1 INTRODUCTION

Pretrained transformers are now adapted to downstream tasks far more often than they are trained from scratch, but full fine-tuning stores and updates a separate copy of every weight per task, which is expensive at scale. Parameter-efficient fine-tuning (PEFT) (Houlsby et al., 2019) instead freezes the pretrained weights and learns a small update; low-rank adapters such as LoRA (Hu et al., 2022) and AdaLoRA (Zhang et al., 2023) are the dominant methods, adding a low-rank product to each weight and, in AdaLoRA, allocating a rank budget across layers by pruning singular values.

Many practical settings involve several related tasks at once, where one model should exploit what the tasks share while still capturing what makes each task distinct. Standard low-rank adapters force a poor trade-off: one adapter per task relearns the shared structure from scratch and transfers nothing across tasks, while a single adapter shared by all tasks suppresses each task's own variation. Recent multi-task adapters add an explicit shared component plus per-task components (Tian et al., 2024; Yang et al., 2025; Karimi Mahabadi et al., 2021; Ponti et al., 2022), but they represent the update multiplicatively, do not enforce that the shared and task-specific parts describe non-overlapping variation, and fix or globally allocate the per-task rank. We propose `JIVEAdapter` (Figure 1), a multi-task additive low-rank adapter inspired by statistical Joint and Individual Variation Explained (JIVE) (Lock et al., 2013). Each weight update is a sum of a Joint structure shared across all tasks and a per-task Individual structure orthogonal to the Joint structure, learned on top of a frozen weight with a well-defined initialization. Because the Joint is estimated from all tasks, a low-resource task inherits it and only needs to fit its small Individual; and since these structures are explicit and orthogonal, one can measure how much each task relies on Joint versus Individual structure. Our contributions are:

- Additive, orthogonal shared/individual split. Unlike multiplicative adapters, `JIVEAdapter` writes the update as the sum of the Joint and the Individual and

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.

constrains the Individual to be orthogonal to the Joint, borrowing the JIVE principle so the shared Joint captures overlapping and each Individual non-overlapping variation.

- Joint and Individual pool adaptive rank. Rank is allocated adaptively across a shared Joint pool and a per-task Individual pool, rather than across layers as in AdaLoRA, so the shared structure is not starved by many per-task structures.
- A reusable frozen Joint. The Joint is learned once, jointly over a task group or incrementally, one task at a time, then frozen and reused as a prior for a later task without retraining the shared part, either by reusing a similar task's Individual or by training a small new one.
- Empirical validation. On GLUE and SuperGLUE with DeBERTaV3-base, `JIVEAdapter` is competitive with strong single-task and fully-shared multi-task low-rank baselines at a matched per-task effective rank, and analysis confirms the Joint represents shared structure while Individuals represent task-specific variation, exposing the underlying task structure.

2 Related Work

Single-task low-rank adaptation. Low-rank adapters add a trainable update to the frozen weights instead of tuning all parameters. The single-task counterparts of `JIVEAdapter` are LoRA (Hu et al., 2022), a fixed-rank update, and AdaLoRA (Zhang et al., 2023), which reparametrizes it as an SVD triplet and prunes singular values under a global budget. AdaLoRA is the closest single-task counterpart; unlike it, `JIVEAdapter` parametrizes each structure as low rank using SVD while decomposing the update with JIVE (Lock et al., 2013) rather than a naive SVD, giving a sum of task-specific orthogonal structures with rank allocated per structure, not per layer.

Multi-task low-rank adaptation. The multi-task counterparts, MTL-LoRA (Yang et al., 2025) and HydraLoRA (Tian et al., 2024), add task-specific factors on a shared down-projection, and several such methods rely on mixture-of-experts (MoE) routing over multiple experts (Tian et al., 2024; Ning et al., 2025). `JIVEAdapter`, however, splits capacity into an interpretable, reusable additive sum of a Joint and an Individual component under a JIVE orthogonality constraint, allocates per-task rank adaptively, and leaves the backbone unchanged, without any MoE routing.

How `JIVEAdapter` differs. `JIVEAdapter` is the only method that combines an explicit shared and task-specific split, orthogonal Joint and Individual subspaces, per-structure adaptive rank, and a frozen Joint that transfers to new tasks, reused for unseen tasks either by similarity or by training a small new Individual, all as a low-rank update that leaves the backbone unchanged. For extended related work see Appendix A.

3 Method

Motivation: We propose an **interpretable** adaptation that models the **individual** characteristics of a task while relating it to similar tasks through an **orthogonal Joint** structure: a shared **Joint** that captures what tasks have in common and exposes how they relate to one another. This is exactly why we adopt **JIVE**: it gives a principled way to separate shared from individual variation, letting us represent related tasks as separate subspaces while still relating them through a common shared space. Moreover, we propose frameworks that make our adapter applicable not only to **single-task** and **multi-task** scenarios, but also when a pool of tasks already exists or **grows over time (incremental)**.

3.1 Formulation

JIVE (Joint and Individual Variation Explained) (Lock et al., 2013), later refined to measure the shared structure through principal angles between subspaces (Feng et al., 2018), is a statistical decomposition that, given several related data sources, splits each source into a Joint structure shared across all sources and a source-specific Individual structure, with each Individual constrained to be orthogonal to the Joint so that the two extract non-overlapping variation. We leverage this in parameter-efficient adaptation through gradient-based learning rather than a linear least-squares fit; each per-task weight update is the sum of a Joint update shared by all tasks and a task-specific In-

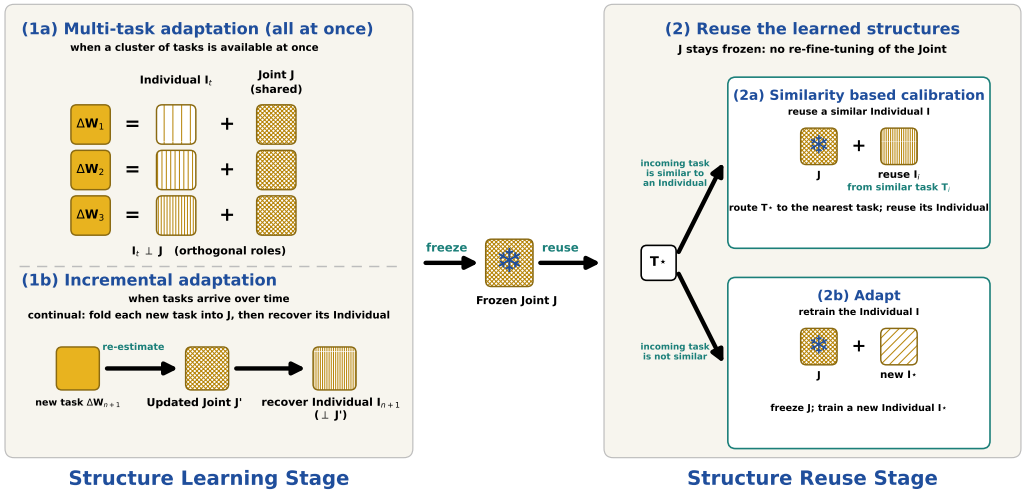


Figure 1: `JIVEAdapter` in two stages. The Structure Learning Stage builds a shared Joint $\mathbf{J}$ with orthogonal per-task Individuals $\mathbf{I}_t \perp \mathbf{J}$ and freezes it into $\mathbf{J}^\star$, by (1a) multi-task adaptation or (1b) incremental adaptation (add each new task in, then recover its Individual). The Structure Reuse Stage serves a new task $T^\star$ without retraining $\mathbf{J}^\star$: (2a) SIMILARITY reuses a resembling task's Individual, else (2b) ADAPT trains a new Individual $\mathbf{I}^\star$.

dividual update, and we impose the same orthogonality so the shared and task-specific parts span non-overlapping subspaces.

Formulation. Let $\mathbf{W} \in \mathbb{R}^{\text{out}\times\text{in}}$ be a frozen weight; only the Joint and Individual factors below are trained. We attach a trainable, additive update

$$\Delta\mathbf{W}_t \;=\; \mathbf{J} \;+\; \mathbf{I}_t, \qquad \mathbf{J},\, \mathbf{I}_t \in \mathbb{R}^{\text{out}\times\text{in}}, \tag{1}$$

where $\mathbf{J}$ is shared by all T tasks and $\mathbf{I}_t$ is the Individual update of task $t \in \{1, \ldots, T\}$. The adapted layer output for a sample belonging to task t is

$$y \;=\; x\mathbf{W}^\top \;+\; \text{drop}(x)\,\mathbf{J}^\top \;+\; \text{drop}(x)\,\mathbf{I}_t^\top. \tag{2}$$

Parametrizing each structure. Each structure is a low-rank SVD triplet $U\ \text{diag}(E)\,V$:

$$\mathbf{J} = \tfrac{\alpha_J}{r_J+\varepsilon}\, U_J\,(E_J \odot V_J), \qquad V_J \in \mathbb{R}^{r_J\times\text{in}},\ E_J \in \mathbb{R}^{r_J\times 1},\ U_J \in \mathbb{R}^{\text{out}\times r_J}, \tag{3}$$

$$\mathbf{I}_t = \tfrac{\alpha_I}{r_I+\varepsilon}\, U_{I,t}\,(E_{I,t} \odot V_{I,t}), \qquad V_{I,t} \in \mathbb{R}^{r_I\times\text{in}},\ E_{I,t} \in \mathbb{R}^{r_I\times 1},\ U_{I,t} \in \mathbb{R}^{\text{out}\times r_I}. \tag{4}$$

with the scaling factor of $\frac{\alpha}{r+\varepsilon}$, where α_J, α_I are fixed scaling hyperparameters and regularizing by the rank r keeps the effective magnitude of each update roughly constant as r varies, so rank and α can be tuned independently (ε guards against r=0). Each Individual factor carries an extra leading task axis of size T, while the Joint factors are single shared tensors.

3.2 PROPOSED JIVEADAPTER

`JIVEAdapter` has two stages (Fig. 1). The **Structure Learning Stage** builds a single shared Joint $\mathbf{J}$ per layer plus one small per-task Individual $\mathbf{I}_t$, constrained to be near-orthogonal so that shared and task-specific structure are largely separated; the Joint is built either by **(1a)** multi-task training on a related group of tasks (all at once) or by **(1b)** incremental adaptation, adding each new task into the Joint and recovering its Individual. The **Structure Reuse Stage** then reuses the frozen Joint as a reusable prior for a new task that arrives later, never retraining the shared part, by **(2a)** reusing a similar task's Individual when the new task resembles a trained one or **(2b)** training a new Individual otherwise.

3.2.1 STRUCTURE LEARNING STAGE: EXTRACTING THE JOINT STRUCTURE

(1a) Multi-Task Training: Shared Joint, Per-Task Individuals We aim to learn several tasks simultaneously, namely a single shared Joint plus one Individual per task, so that at deployment the Joint is fixed and only the small per-task Individual is substituted in. This matches canonical JIVE, where the joint structure is estimated jointly from all tasks rather than any one of them. A batch may contain samples from several tasks. Each layer receives a per-sample task-id vector and routes each sample to its own Individual while the shared Joint applies to all samples (Alg. 1). Grouping by the distinct tasks in the batch keeps the cost to one small matmul per active task, not per sample.

Algorithm 1 Forward for one `JIVEAdapter` layer (single or multitask)

Input: input x (batch $\times \ldots \times$ in), per-sample task ids, frozen $\mathbf{W}$
Output: adapted output y

$y \leftarrow x\mathbf{W}^\top$ *frozen base*
$\mathbf{J} \leftarrow \frac{\alpha_J}{r_J+\varepsilon} U_J(E_J \odot V_J)$
$y \leftarrow y + \mathrm{drop}(x)\,\mathbf{J}^\top$ *shared Joint, all samples*
if task ids is None **or** $T = 1$ **then**
  return $y + \mathrm{drop}(x)\,\mathbf{I}_1^\top$ *single-task: one Individual, no routing*
end if
multi-task ($T > 1$): route each sample to its own Individual below
$\mathrm{ind} \leftarrow \mathbf{0}$ *output-shaped Individual buffer, zeros like y*
for $t \in$ unique(task ids) **do**
  $m \leftarrow$ (task ids $= t$) *boolean mask over batch*
  $U_{I,t} \leftarrow (\mathbb{I} - U_J U_J^\top) U_{I,t}$ *$\mathbf{I}_t \perp \mathbf{J}$*
  $\mathbf{I}_t \leftarrow \frac{\alpha_I}{r_I+\varepsilon} U_{I,t}(E_{I,t} \odot V_{I,t})$
  $\mathrm{ind}[m] \leftarrow \mathrm{drop}(x[m])\,\mathbf{I}_t^\top$
end for
return $y + \mathrm{ind}$

Under Eq. (2), in a mixed batch the shared Joint receives gradient from every task at each step, while each Individual receives gradient only from its own samples. Thus the Joint converges to structure common to all tasks (the SGD analogue of estimating $\mathbf{J}$ from the concatenation of all blocks), and the Individuals absorb task-specific residuals. This was verified empirically (Joint gradient non-zero from any task; Individual t gradient non-zero only when task t is present). Mixed batches keep the shared Joint stable (one averaged gradient over all tasks per step); alternating per-task batches makes the Joint oscillate toward whichever task it last saw. A compromise is to accumulate gradients over homogeneous per-task micro-batches, then take a single optimizer step.

Orthogonal Regularization terms in loss The training objective is the task loss plus $\lambda\mathcal{R}$, where $\mathcal{R}$ enforces orthonormal SVD factors for the Joint and every Individual across all `JIVEAdapter` layers, with Individual$\perp$Joint handled by the forward projection,

$$\mathcal{R} = \sum_{\text{layers}} \Big(\|V_J V_J^\top - \mathbb{I}\|_F + \|U_J^\top U_J - \mathbb{I}\|_F + \sum_t \|V_{I,t} V_{I,t}^\top - \mathbb{I}\|_F + \|U_{I,t}^\top U_{I,t} - \mathbb{I}\|_F \Big). \tag{5}$$

Imposing orthogonality in the forward pass We keep the Joint's output factor U_J (approximately) column-orthonormal via the orthogonality regularizer $\mathcal{R}$ (Eq. 5), where the columns of U_J span the update's output space. Given that, we can remove any overlap between an Individual and the Joint directly in the forward pass by projecting the Individual's output factor onto the orthogonal complement of the Joint's column space:

$$\widetilde{U}_{I,t} \;=\; \big(\mathbb{I} - U_J U_J^\top\big)\, U_{I,t}, \qquad \mathbf{I}_t = \tfrac{\alpha_I}{r_I+\varepsilon}\, \widetilde{U}_{I,t}\, (E_{I,t} \odot V_{I,t}). \tag{6}$$

where $\mathbb{I}$ is the identity matrix. The projection is exact when $U_J^\top U_J = \mathbb{I}$, enforced by the regularizer.

Adaptive Rank Allocation We allocate rank per structure with two strategies. The first is an importance-based allocator adapted from AdaLoRA, which prunes singular values under a decaying budget during training. The second is the budget-free permutation-test estimator native to statistical JIVE (Lock et al., 2013), adapted to weight space and run once before training: it estimates the shared Joint and per-task Individual ranks directly from data, with no importance schedule and no pruning. In both, the shared Joint pool and the per-task Individual pool are budgeted separately, so many per-task structures cannot starve the single shared Joint. Full details of both algorithms are deferred to Appendix B.1 (Alg. 5 and Alg. 6).

Merging at inference. Merging is a LoRA-style inference trick that folds a fixed adapter update $\Delta\mathbf{W}$ into the frozen backbone weight $\mathbf{W}$, so a merged layer runs one matmul instead of two. It is purely an efficiency rewrite that leaves the output unchanged, and it is enabled only for a single task (T=1), where the one update $\mathbf{J} + \mathbf{I}_1$ can be folded into $\mathbf{W}$ once. With several task-specific Individuals there is no single update to merge, since folding one task's $\mathbf{I}_t$ into the shared $\mathbf{W}$ would corrupt every other task, so merging is automatically disabled when $T > 1$. Each layer then keeps the frozen $\mathbf{W}$, adds the shared Joint $\mathbf{J}$ to every sample, and routes each sample to its own Individual $\mathbf{I}_t$ on the fly (Alg. 1). Recovering merging under $T > 1$ would need a separate merged weight per task, i.e. T full weight matrices in every layer and a matmul per task, so we keep the frozen $\mathbf{W}$ plus cheap low-rank additions instead. The routed forward, the orthogonal regularizer, and the per-structure allocator compose into a single optimizer step, with a separate head per task because the label spaces differ.

(1b) Incremental Task Adaptation As an alternative to learning the Joint from all tasks at once (1a), `JIVEAdapter` can build it incrementally: the Joint moves to reflect each newly arrived task. Given n tasks with a shared Joint $\mathbf{J}$ and Individuals $\{\mathbf{I}_t\}$, and one incoming task, we (i) reestimate the Joint over all $n{+}1$ tasks warm-started from the current $\mathbf{J}$, then (ii) recover the incoming Individual (and refresh the existing ones) as the rank-r_I residual orthogonal to the updated Joint. Given $Q_J = \mathrm{col}(U_J)$ for the shared output subspace and $P_Q = QQ^\top$ for the projector onto a subspace Q, the estimator alternates two closed-form steps: a joint step that fits the single shared matrix $\mathbf{J}' = P_{Q'_J}\overline{\Delta\mathbf{W}}$ best explaining the weighted-mean update $\overline{\Delta\mathbf{W}}$ inside the reestimated shared subspace Q'_J, and an individual step $\mathbf{I}_t = \mathrm{trunc}_{r_I}\big((\mathbb{I} - P_{Q'_J})\Delta\mathbf{W}_t\big)$ that takes each task's residual orthogonal to $\mathbf{J}'$ (Alg. 2). The special case that keeps $\mathbf{J}$ fixed recovers exactly the frozen-Joint estimate $\mathbf{I}_{n+1} = \mathrm{trunc}_{r_I}\big((\mathbb{I} - P_{Q_J})\Delta\mathbf{W}^\star_{n+1}\big)$; letting Q'_J rotate/grow lets a sufficiently informative new task refine the shared structure, and the per-task Individuals stay orthogonal to it by construction.

Algorithm 2 Incremental JIVE: fold a new task into the Joint and recover its Individual

Input: frozen base weight $\mathbf{W}$; current joint $\mathbf{J}{=}U_J\,\mathrm{diag}(E_J)V_J$ with subspace $Q_J{=}\mathrm{orth}(U_J)$ and rank r_J; existing individuals $\{\mathbf{I}_t\}_{t=1}^{n}$; incoming task data D_{n+1}; individual rank r_I; iterations K; share threshold τ; growth cap κ; per-task delta weights $\{\gamma_t\}$
Output: updated joint $\mathbf{J}'$; incoming individual $\mathbf{I}_{n+1}$; refreshed $\{\mathbf{I}_t\}$

$\Delta\mathbf{W}_t \leftarrow \mathbf{J} + \mathbf{I}_t,\ t \le n$ — *construct $\Delta\mathbf{W}$ per existing task*
$\mathbf{I}^{(0)}_{n+1} \leftarrow \arg\min_{\mathrm{rank}(\mathbf{I})\le r_I} \mathcal{L}(\mathbf{W}{+}\mathbf{J}{+}\mathbf{I};\, D_{n+1}),\ \Delta\mathbf{W}_{n+1} \leftarrow \mathbf{J} + \mathbf{I}^{(0)}_{n+1}$ — *incoming task update*
$\widehat{\Sigma}_t \leftarrow \Delta\mathbf{W}_t\Delta\mathbf{W}_t^\top / \|\Delta\mathbf{W}_t\|_F^2,\quad t \le n{+}1$
for $k = 1, \ldots, K$ **do**
  $C \leftarrow \sum_{t=1}^{n+1} \gamma_t\, \widehat{\Sigma}_t;\quad Q'_J \leftarrow \mathrm{eig}_{r_J}(C)$ — *top-r_J eigenvectors, warm-started from Q_J*
  $Q'_J \leftarrow \{\, u \in Q'_J \,:\, \min_{t\le n+1}\, u^\top \widehat{\Sigma}_t\, u \,\ge\, \tau \,\}$ — *keep directions shared by all tasks*
  $\overline{\Delta\mathbf{W}} \leftarrow (\sum_t \gamma_t \Delta\mathbf{W}_t)/\sum_t \gamma_t$ — *weighted-mean update*
  $P_{Q'_J} \leftarrow Q'_J Q'^\top_J;\quad \mathbf{J}' \leftarrow P_{Q'_J}\,\overline{\Delta\mathbf{W}}$ — *new Joint: averaged update projected onto the shared subspace*
  $\mathbf{I}_t \leftarrow \mathrm{trunc}_{r_I}((\mathbb{I} - P_{Q'_J})\,\Delta\mathbf{W}_t),\quad t \le n{+}1$ — *recover each Individual off the Joint*
  $Q_J \leftarrow Q'_J$ — *stop when the shared subspace stops moving: $d_{\mathrm{Gr}}(Q_J, Q'_J) < \epsilon$*
end for
$\mathbf{J}' = U'_J\,\mathrm{diag}(E'_J)V'_J$ (truncated SVD, rank r_J); append $\mathbf{I}_{n+1}$ to the pool
return $\mathbf{J}',\ \mathbf{I}_{n+1},\ \{\mathbf{I}_t\}$ — $Q'^\top_J U_{I,t} = 0$

3.2.2 Structure Reuse Stage

Given a shared Joint $\mathbf{J}$ extracted from a group of tasks and a new task $T^\star$, we freeze both the pretrained weights $\mathbf{W}$ and the shared Joint $\mathbf{J}$ and reuse $\mathbf{J}$ as a prior, learning at most a small task-specific Individual instead of adapting from scratch. Algorithm 3 makes this precise through two modes, neither of which retrains the shared part.

(2a) Reuse similar task Individual. Inspired by the notion that a task is characterized by the gradients it produces on a frozen backbone (Achille et al., 2019) and by the SVD gradient-subspace construction of Gradient Projection Memory (GPM) (Saha et al., 2021), which we adapt from activations to weight-space updates, we identify the prior task most similar to $T^\star$ and reuse its Individual, retraining none of its basis and learning only one scalar gain per rank direction. We summarize $T^\star$ by a signature $G_{T^\star}$: the top-r_I left singular vectors of its frozen-model gradient averaged over B minibatches (r_I the adapter rank, $\mathcal{L}$ the loss at the frozen $\mathbf{W} + \mathbf{J}$ with the task's

Algorithm 3 Reusing a frozen Joint for a new task $T^\star$

Input: frozen $\mathbf{W}$; frozen Joint $\mathbf{J}{=}U_J\,\mathrm{diag}(E_J)V_J$, $U_J^\top U_J{=}\mathbb{I}$; individuals $\{\mathbf{I}_t\}_{t=1}^n$; held-in task signatures $\{G_t\}_{t=1}^n$, $G_t{=}\mathrm{U}_{r_I}\big(\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}{+}\mathbf{J};\,B_t)\big)$; data $D_{T^\star}$; task loss $\mathcal{L}(\cdot;D)$; rank r_I; mode $\in$ {similarity based calibration, adapt}
Output: individual $\mathbf{I}_{T^\star}$; shared-energy ratio $\eta_{T^\star}$

if mode = similarity based calibration **then**
  $G_{T^\star} \leftarrow \mathrm{U}_{r_I}(\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}{+}\mathbf{J};\,B)),\ B \subset D_{T^\star}$ — *task signature: top-r_I left singular vectors*
  $T_c \leftarrow \arg\max_{t\le n}\ \mathrm{overlap}(G_{T^\star}, G_t)$ — *hard nearest-task routing on signatures, Eq. (8)*
  $\mathbf{I}_{T^\star} \leftarrow \mathbf{I}_{T_c} = U_{I_{T_c}}\,\mathrm{diag}(E_{T_c})V_{I_{T_c}}$ — *reuse nearest Individual directly (already $\perp \mathbf{J}$)*
  $E_{T^\star} \leftarrow \arg\min_E\ \mathcal{L}\big(\mathbf{W}{+}\mathbf{J}{+}U_{I_{T_c}}\,\mathrm{diag}(E)V_{I_{T_c}};\,D_{T^\star}\big)$ — *scale-only; borrowed basis $U_{I_{T_c}}, V_{I_{T_c}}$ fixed*
else — *adapt*
  $\mathbf{I}_{T^\star} \leftarrow \arg\min_{\mathrm{rank}(\mathbf{I})\le r_I}\ \mathcal{L}(\mathbf{W}{+}\mathbf{J}{+}\mathbf{I};\,D_{T^\star}),\quad \mathbf{I}\leftarrow\mathbf{0}$ — *train a fresh Individual*
end if
$\widehat{\Delta\mathbf{W}} \leftarrow \Delta\mathbf{W}^\star_{T^\star}$ **if** a reference update is available **else** $G_{T^\star}$ — *reference $\Delta\mathbf{W}^\star_{T^\star}$: a separately trained single-task LoRA or full fine-tune on $T^\star$; used only for the diagnostic η, not for reuse*
$\eta_{T^\star} \leftarrow \|U_J U_J^\top \widehat{\Delta\mathbf{W}}\|_F \,/\, \|\widehat{\Delta\mathbf{W}}\|_F$ — *diagnostic: fraction of the task already in the Joint $U_J^\top U_{I_{T^\star}} = 0$*
return $\mathbf{I}_{T^\star}$, $\eta_{T^\star}$

Individual switched off),

$$\bar{\nabla}_{T^\star} = \frac{1}{B}\sum_{b=1}^{B}\nabla_{\mathbf{W}}\mathcal{L}(x_b, y_b; \mathbf{W}{+}\mathbf{J}) = U\Sigma V^\top, \qquad G_{T^\star} = U_{:,\,1:r_I}, \tag{7}$$

an orthonormal basis ($G_{T^\star}^\top G_{T^\star} = \mathbb{I}_{r_I}$) for the subspace carrying most of $T^\star$'s gradient energy. Computing an analogous signature G_t for every prior task, we route $T^\star$ to its closest task T_c by principal-angle overlap (Björck & Golub, 1973) of the two signatures,

$$T_c = \arg\max_{t\le n}\ \mathrm{overlap}(G_{T^\star}, G_t), \qquad \mathrm{overlap}(G, G') = \frac{1}{r_I}\|G^\top G'\|_F^2 = \frac{1}{r_I}\sum_{i=1}^{r_I}\cos^2\theta_i, \tag{8}$$

where the $\cos\theta_i$ are the singular values of $G_{T^\star}^\top G_t$, so the score $\in [0,1]$ and is 1 when subspaces overlap. We route on the Individual alone, not on $\mathbf{W}{+}\mathbf{J}{+}\mathbf{I}$: the frozen $\mathbf{W}$ is shared across tasks and the Joint $\mathbf{J}$ is common to every task (and already reused by $T^\star$), so both inject task-independent mass that would dominate and wash out the score, whereas each $\mathbf{I}_t \perp \mathbf{J}$ isolates exactly the task-specific, transferable part. We take the routed task's Individual $\mathbf{I}_{T_c}$, which is already orthogonal to the frozen $\mathbf{J}$ by construction, and reuse it directly ($\mathbf{I}_{T^\star} = \mathbf{I}_{T_c}$), keeping its two basis matrices $(U_{I_{T_c}}, V_{I_{T_c}})$ frozen. We then train only the r_I diagonal gains $E_{T^\star}$ (one importance scaler per direction) and a fresh classification head: freezing the borrowed basis keeps $T^\star$ in the *same* subspace as its neighbour T_c, while learning $E_{T^\star}$ re-weights how strongly each direction of that subspace is used, adapting the shared structure to $T^\star$ at negligible cost.

(2b) ADAPT. Freeze $\mathbf{W}+\mathbf{J}$ and train only a small new Individual $\mathbf{I}_{T^\star}$ (rank r_I), so the new task learns exactly the part the Joint does not already explain. How much of the new task already lives in the Joint is quantified by the shared-energy ratio (Lock et al., 2013; Saha et al., 2021), the fraction of joint variation explained in the sense of linear JIVE, $\eta_{T^\star} = \|U_J U_J^\top \widehat{\Delta\mathbf{W}}\|_F / \|\widehat{\Delta\mathbf{W}}\|_F$, where $\widehat{\Delta\mathbf{W}}$ is a reference single-task update $\Delta\mathbf{W}^\star_{T^\star}$ (single-task LoRA or full fine-tuning) when available, else the task signature $G_{T^\star}$. Decomposing it against the frozen Joint,

$$\underbrace{\mathbf{J}\text{-part} = U_J U_J^\top\,\widehat{\Delta\mathbf{W}}}_{\text{explained by the shared Joint}}, \qquad \underbrace{\widehat{\mathbf{I}}_{T^\star} = (\mathbb{I} - U_J U_J^\top)\,\widehat{\Delta\mathbf{W}}}_{\text{residual = recovered Individual}},$$

gives a high $\eta_{T^\star}$ exactly when the shared structure carries most of the task, so similarity based calibration already transfers well; the orthogonal residual $\widehat{\mathbf{I}}_{T^\star}$ is the part the new task must still learn, and plugging it (or a small learned $\mathbf{I}_{T^\star}$) on top of the frozen $\mathbf{W}+\mathbf{J}$ should reconstruct $T^\star$'s accuracy at a fraction of the trainable parameters and data.

4 EXPERIMENTS

We adopt the AdaLoRA experimental setting (Zhang et al., 2023) as our common evaluation framework so that `JIVEAdapter` and every baseline are compared on identical data, backbone, and parameter budget. We evaluate on the GLUE benchmark (Wang et al., 2019b), reporting each task's

standard metric, and additionally on the SuperGLUE benchmark (Wang et al., 2019a). For the structure-reuse study we further use the MNLI corpus (Williams et al., 2018), treating its five annotated genres (fiction, government, slate, telephone, and travel) as a cross-genre pool. We use DeBERTaV3-base (He et al., 2023) as the backbone, with adapters on the query, key, value, intermediate, layer-output, and attention-output projections (identical to the AdaLoRA setup). We match the per-task effective rank across methods rather than the raw parameter count, which differs by structure; the full setup (baseline taxonomy, model parameters and invariants and budget split), including a single fixed random seed shared by all methods and every hyperparameter, is detailed in Appendix C for reproducibility.

Baselines. We run two single-task low-rank adapters, LoRA (Hu et al., 2022) and AdaLoRA (Zhang et al., 2023), two structured multi-task LoRA methods, HydraLoRA (Tian et al., 2024) and MTL-LoRA (Yang et al., 2025), and `JIVEAdapter`; their capacity structure is compared in Table 4 (Appendix C). We additionally include two sequential continual-learning references in the incremental study, a single shared adapter fine-tuned across the task stream and full fine-tuning of the entire backbone, both without any frozen shared structure (Table 6). Methods that add auxiliary modules, notably mixture-of-experts (MoE) routing such as MoDE, are not directly comparable, as those modules enlarge the parameter and compute budget beyond a single low-rank update; `JIVEAdapter` uses none, and of that family we include only HydraLoRA, the closest to ours.

Case studies. Following Figure 1, we evaluate `JIVEAdapter` along six axes grouped by the method's two stages: a single-task setting, then the Structure Learning Stage (building the Joint) and the Structure Reuse Stage, followed by two analysis axes.

1. **Single-task** setting ($T{=}1$) trains one task at a time, where `JIVEAdapter` reduces to Joint + one Individual, comparing our orthogonal additive parametrization and per-structure allocator to the multiplicative low-rank updates of LoRA/AdaLoRA, whose premise follows the intrinsic dimensionality of fine-tuning (Aghajanyan et al., 2021) (Table 1).
2. **Multi-task all-at-once training** (Stage 1a, Structure Learning Stage) trains all tasks jointly with mixed-task batches, one shared Joint and one Individual per task, on GLUE and SuperGLUE (Table 2); at inference the Joint is fixed and only the small task Individual is selected.
3. **Multi-task incremental training** (Stage 1b, Structure Learning Stage) learns a base Joint from a task cluster and folds each new task in one at a time by refining both the shared Joint and its own Individual following Algorithm 2, versus two sequential fine-tuning references (a single shared adapter and full backbone fine-tuning) and all-at-once joint training (Table 6).
4. **Similarity (2a) and adapt (2b) structure reuse** serves a held-out task from a frozen Joint, either routing to the highest-overlap held-in Individual (2a) or training a small new one (2b), evaluated leave-one-out on the SuperGLUE cross-task and MNLI cross-genre pools (Figure 2).
5. **Ablation of the rank-estimation strategy** compares the importance-based low-rank per-structure allocator against the permutation-test rank estimator of JIVE (Appendix B.1).
6. **Interpretability analysis** of what the Joint and Individuals learn, showing each Individual settles near-orthogonal to the shared Joint and that task-signature similarity routes each held-out task to its most related held-in Individual (Appendix D.3).

4.1 Results

4.1.1 Single-Task Comparison ($T{=}1$)

Single-task. `JIVEAdapter` attains the best single-task average on both benchmarks. On GLUE it reaches 89.44, ahead of our reproduced AdaLoRA reference (89.14) and LoRA (87.90); on SuperGLUE it reaches 78.65, ahead of LoRA (78.41) and AdaLoRA (77.21). It leads on five of eight GLUE tasks and on CB, WiC, and ReCoRD, showing the additive parametrization and per-structure allocator do not cost accuracy before any multi-task sharing.

4.1.2 Structure Learning Stage: Extracting the Joint Structure

(1a) Multi-task all-at-once training. Unlike the single-task runs of Table 1 (a separate adapter trained per task), here all methods are trained as a single multi-task adapter over the eight tasks at

Table 1: Single-task evaluation (best per task), a separate adapter trained with no sharing; `JIVEAdapter` stays competitive even without sharing. #P is trainable parameters per task head.

	Method	MNLI (acc)	QQP (acc/F1)	QNLI (acc)	SST-2 (acc)	CoLA (Mcc)	RTE (acc)	MRPC (acc/F1)	STS-B (P/S)	Avg	#P (M)
GLUE	LoRA	89.52	**92.21/89.67**	93.79	94.72	70.44	81.59	89.22/92.14	**91.73**/91.69	87.90	**1.33**
	AdaLoRA ($\bar{r}$=8, repro)	90.43	91.97/89.44	94.33	95.41	**71.91**	87.36	90.20/92.91	91.58/91.49	89.14	**1.33**
	`JIVEAdapter` (ours)	**90.55**	91.89/89.34	**94.62**	**96.22**	71.02	**88.81**	**90.44/93.10**	91.58/**91.97**	**89.44**	**1.33**
	Method	BoolQ (acc)	CB (acc/F1)	RTE (acc)	WiC (acc)	WSC (acc)	COPA (acc)	MultiRC (F1a/EM)	ReCoRD (EM/F1)	Avg	#P (M)
SuperGLUE	LoRA	**83.43**	85.71/76.13	**85.92**	71.32	77.88	87.00	**75.24/38.30**	83.64/84.39	78.41	**1.33**
	AdaLoRA ($\bar{r}$=8, repro)	82.94	87.50/78.08	**85.92**	70.06	**78.85**	**94.00**	60.19/14.80	85.38/85.93	77.21	**1.33**
	`JIVEAdapter` (ours)	82.81	**89.29/80.37**	**85.92**	**71.79**	75.96	91.00	71.46/30.22	**85.74/86.33**	**78.65**	**1.33**

once. On GLUE (Table 2) the baselines average 88.80 and 88.42; `JIVEAdapter` averages 89.07, ahead of both. On SuperGLUE (Table 2) `JIVEAdapter` averages 82.06 (fair-rank at effective per-task rank 8); HydraLoRA takes the overall average (82.24), driven mainly by its perfect score on the smallest task, yet `JIVEAdapter` reaches a comparable average without its MoE routing. Across both benchmarks the shared-Joint plus per-task Individual split adds usable capacity under joint training. We report `JIVEAdapter`'s stronger rank-estimation configuration per benchmark here; the importance-based and permutation-test variants are compared in Appendix C.3.

Table 2: Multi-task results, all methods trained jointly on the eight tasks per benchmark. Number of trainable parameters (#P) are reported per single task head at sequence length 128.

	Method	MNLI (acc)	QQP (acc/F1)	QNLI (acc)	SST-2 (acc)	CoLA (Mcc)	RTE (acc)	MRPC (acc/F1)	STS-B (P/S)	Avg	#P (M)
GLUE	HydraLoRA	89.53	90.29/**87.33**	94.14	95.41	69.92	**90.25**	89.95/92.64	90.98/91.08	88.80	2.90
	MTL-LoRA	89.93	**90.36**/87.15	94.49	95.53	65.61	89.17	**90.93/93.36**	91.71/91.68	88.42	2.66
	`JIVEAdapter` (ours)	**90.10**	90.34/87.32	**94.58**	**95.99**	**70.23**	89.17	90.20/93.06	**92.03/91.98**	**89.07**	**1.33**
	Method	BoolQ (acc)	CB (acc/F1)	RTE (acc)	WiC (acc)	WSC (acc)	COPA (acc)	MultiRC (F1a/EM)	ReCoRD (EM/F1)	Avg	#P (M)
SuperGLUE	HydraLoRA	85.54	**100.00/100.00**	**88.45**	70.85	79.81	**91.00**	76.94/39.24	83.87/84.50	**82.24**	2.90
	MTL-LoRA	**85.66**	92.86/89.24	87.00	**71.63**	80.77	88.00	76.30/38.61	86.27/86.84	81.02	2.66
	`JIVEAdapter` (ours)	84.86	92.86/89.32	85.92	69.28	**87.50**	**91.00**	**77.28/42.39**	**86.76/87.28**	82.06	**1.33**

(1b) Incremental multi-task adaptation. We further show how Incremental `JIVEAdapter` (Algorithm 2) can be used as a *continual-learning* procedure that mitigates *catastrophic forgetting*; a shared Joint is learned from a base cluster of tasks and each new task is then folded in by refining both the shared Joint and the task's own Individual, so earlier tasks are preserved rather than overwritten. *Incremental folding needs only the stored low-rank Joint and Individual factors of the earlier tasks together with the incoming task's data. Their training data is never revisited, so `JIVEAdapter` applies even when the base-pool data no longer exists.* The full study contrasts joint all-at-once training, incremental `JIVEAdapter` with single-task and multi-task bases, and two sequential fine-tuning references (a single shared adapter and full backbone fine-tuning) that keep no frozen shared structure and act as continual-learning lower bounds exposing forgetting, on GLUE and SuperGLUE. *We recommend the incremental approach only when the task pool grows over time and the earlier tasks are no longer available for joint all-at-once training, when the base-pool data is no longer accessible, or when maintaining a separate head per task on the backbone for all-at-once training is too expensive; otherwise all-at-once remains preferable.* We use Incremental `JIVEAdapter` as a remedy for such situations and defer this study to Appendix D.1 (Table 6) as an extension.

4.1.3 Structure Reuse Stage

(2a) Similarity-based reuse. We run a leave-one-out evaluation over the SuperGLUE pool, holding each task out in turn while `JIVEAdapter` is trained multi-task on the remaining six, freezing the per-layer Joint, and serving the held-out task through Alg. 3. Similarity-based reuse borrows a held-in task's Individual chosen purely by highest subspace overlap over the entire held-in cluster, with no task-type prior, re-scaling only its per-direction scale and training *no* new Individual basis. We serve seven held-out tasks; routing follows subspace geometry rather than type labels, so the NLI tasks CB and RTE are routed to the boolean-QA donors BoolQ and MultiRC when those carry the highest overlap. The MNLI cross-genre pool is a natural fit here, since all five genres share the NLI type, so every held-out genre always has same-type donors and is routed to its highest-overlap genre (Figure 2). The pairwise subspace-overlap map shows the held-in Individuals are only weakly related to the held-out task, so no previously trained task is a close match. Even then, extracting the shared Joint and calibrating the closest Individual, adjusting only its scale and training no new basis,

recovers most of the accuracy (Figure 2). Similarity-based reuse is therefore useful even when no near-identical task was trained before and when little or no data is available to fit a fresh Individual, since the reusable structure lives mainly in the shared Joint. A closer match still helps, since routing to a held-in task with higher subspace overlap yields higher performance; the strongest similarity-based reuse is on COPA, routed to WiC. Averaged over the pool, it falls short of adaptation on the heterogeneous SuperGLUE tasks (70.62 vs 81.82) but matches it on the same-type MNLI genres (89.33 vs 89.36). For the pairwise similarity scores refer to Appendix D.3, Figure 4.

(2b) Adaptation reuse. Adaptation keeps the frozen $\mathbf{J}^\star$ and fits a small new Individual for the held-out task from scratch, leaving the shared Joint untouched. It is the strong default and wins on every held-out SuperGLUE task, matching or approaching the all-eight-task multitask scores while adding only a lightweight per-task Individual; fitting a fresh Individual is thus the better choice even when a same-type twin is already available, except when the held-out task is very close to a held-in one, as with the MNLI genres, where similarity-based reuse matches adaptation while being cheaper. Unlike the incremental setting of (1b), which updates the shared Joint, here the Joint stays frozen and only the new Individual is trained, yet adaptation still recovers the multitask accuracy. Overlap stays low (near 0.2) in both pools because each Individual is near-orthogonal to the shared Joint by construction and otherwise captures task-specific structure, *so what governs reuse is not its absolute value but how much of the held-out task the shared Joint already covers; the same overlap level matches adaptation when the Joint suffices (the same-type MNLI genres, and COPA) yet lags when the task needs its own Individual (WiC, MultiRC).*

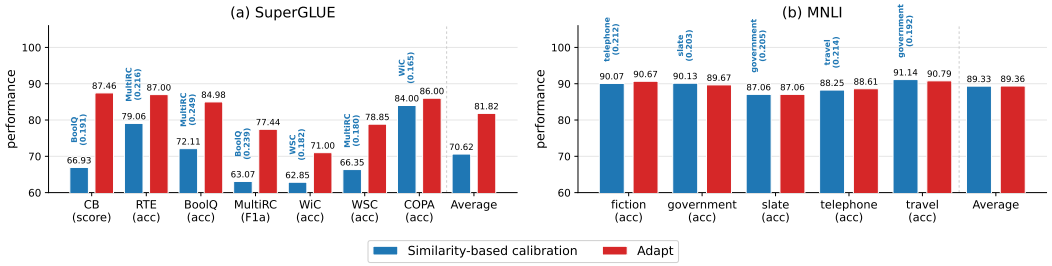


Figure 2: Reuse on held-out tasks, our proposed similarity-based calibration vs adaptation. In both pools, the similarity-based (blue) bar is annotated with the held-in donor it is routed to and the corresponding subspace overlap score. (a) SuperGLUE cross-task pool; (b) MNLI cross-genre reuse.

4.2 Rank Estimation and Interpretability

A budget-free permutation-rank estimator (Section 3.2.1) matches or improves on the importance allocator on multi-task SuperGLUE while removing the rank-budget hyperparameter; we report the full comparison in Appendix D.2 (Table 7). Because the split keeps the shared and task-specific parts as separate, orthogonal structures, each one can be measured, ablated, and visualized independently. We detail the interpretability hypotheses, metrics, and experiments in Appendix D.3.

5 Conclusion

We presented `JIVEAdapter`, a multi-task adapter whose every weight update is an additive low-rank combination of a shared Joint and orthogonal per-task Individuals, with rank allocated adaptively across the two pools. Grounding this split in the linear JIVE decomposition, we recover it not by a least-squares fit but by gradient-based nonlinear training of both parts as zero-initialized factors over a frozen backbone, keeping the two parts interpretable and separately measurable. The decomposition supports a two-stage workflow: a Structure Learning Stage that builds the Joint all at once or incrementally, and a Structure Reuse Stage that freezes the Joint and serves a new task by reusing a related Individual with a cheap per-direction scale or by training a small new one. On GLUE and SuperGLUE with DeBERTaV3-base, `JIVEAdapter` matches strong single-task and multi-task low-rank baselines at a matched per-task effective rank and without any mixture-of-experts routing, while remaining reusable and interpretable, and its incremental variant mitigates catastrophic forgetting without revisiting earlier data.

USE OF AI TOOLS

All ideas in this paper originate with the authors or extend the related work cited herein. The core implementation is entirely the authors' own work, including the `JIVEAdapter` adapter layer (`JIVELayer`), the two-pool rank allocator, and the integration of both into DeBERTa. AI-based tools played only a supporting role, limited to assisting with integrating the vendored baselines into our pipeline, refining the phrasing of the authors' own text, assisting with LaTeX drafting, polishing figures and tables, and correcting grammar. Every technical contribution, claim, and conclusion was devised, checked, and approved by the authors.

ETHICS STATEMENT

The authors have read the ICLR Code of Ethics and confirm that this work adheres to it. Our study is a methodological contribution to parameter-efficient multi-task adaptation and is evaluated exclusively on established public benchmarks (GLUE, SuperGLUE, and MNLI) under their standard licenses, so it involves no human subjects, no private or personally identifiable data, and no new data collection or release. We foresee no direct harmful application specific to the proposed adapter beyond the risks common to the underlying pretrained language models, whose known limitations around bias and fairness carry over to any downstream system; our method neither amplifies nor mitigates those properties by design. The authors declare no conflicts of interest and no external sponsorship that could bias the reported results.

REPRODUCIBILITY STATEMENT

We have made a deliberate effort to make every result reproducible. The method, including the additive Joint and Individual decomposition and the orthogonality constraint, is specified in Section 3, with the incremental and rank-allocation procedures given as explicit pseudocode in Appendix B alongside complete statements and proofs of the theoretical claims. The full experimental setup, including datasets, task lists, optimizer, learning rate, budget, and the single fixed random seed shared across all methods, is documented in Appendix C, and the extended studies and their configurations appear in Appendix D. To further support exact reproduction, we disclose the implementation of the core `JIVELayer`, the model-independent part that is applicable to a variety of backbone models, together with its test units as anonymized supplementary material, and defer the release of the full repository, including all training and control scripts, to acceptance.

REFERENCES

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

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7319–7328, 2021.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

Åke Björck and Gene H. Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of Computation*, 27(123):579–594, 1973.

Qing Feng, Meilei Jiang, Jan Hannig, and J. S. Marron. Angle-based joint and individual variation explained. *Journal of Multivariate Analysis*, 166:241–265, 2018.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *International Conference on Learning Representations (ICLR)*, 2023.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning (ICML)*, 2019.

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. In *International Conference on Learning Representations (ICLR)*, 2022.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In *Conference on Language Modeling (COLM)*, 2024.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3045–3059, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4582–4597, 2021.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning (ICML)*, 2024.

Eric F. Lock, Katherine A. Hoadley, J. S. Marron, and Andrew B. Nobel. Joint and individual variation explained (JIVE) for integrated analysis of multiple data types. *Annals of Applied Statistics*, 7(1):523–542, 2013.

Lin Ning, Harsh Lara, Meiqi Guo, and Abhinav Rastogi. MoDE: Effective multi-task parameter efficient fine-tuning with a mixture of dyadic experts. In *Findings of the Association for Computational Linguistics: NAACL*, 2025.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021.

Edoardo M. Ponti, Alessandro Sordoni, Yoshua Bengio, and Siva Reddy. Combining modular skills in multitask learning. *arXiv preprint arXiv:2202.13914*, 2022.

Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. HydraLoRA: An asymmetric LoRA architecture for efficient fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019a.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*, 2019b.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Yaming Yang, Dilxat Muhtar, Yelong Shen, Yuefeng Zhan, Jianfeng Liu, Yujing Wang, Hao Sun, Denvy Deng, Feng Sun, Qi Zhang, Weizhu Chen, and Yunhai Tong. MTL-LoRA: Low-rank adaptation for multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2023.

A Extended Related Work

Beyond the counterparts compared in the main text, several other parameter-efficient and multi-task adaptation families are related to `JIVEAdapter`; Table 3 summarizes how they compare along the axes most relevant to our contribution.

Prompt- and prefix-based tuning prepend learned continuous vectors while freezing the backbone (Li & Liang, 2021; Lester et al., 2021). BitFit (Ben Zaken et al., 2022) updates only the bias terms. DoRA (Liu et al., 2024) decomposes each pretrained weight into a magnitude and a direction and applies the low-rank update to the direction alone. Adapters (Houlsby et al., 2019) insert per-task bottleneck layers. These methods target a single task at a time and, for the adapter-insertion variants, alter the architecture by adding modules to every layer, which is not always feasible in deployment.

Multi-task adaptation. AdapterFusion (Pfeiffer et al., 2021) fuses per-task bottleneck layers, HyperFormer(++) (Karimi Mahabadi et al., 2021) generates per-task adapters from a shared meta-network, Polytropon (Ponti et al., 2022) routes over an inventory of LoRA adapters, and MoDE (Ning et al., 2025) mixes shared and task experts. Closer to our orthogonality and reuse goals, O-LoRA (Wang et al., 2023) learns each new task in a low-rank subspace kept orthogonal to those of earlier tasks to resist forgetting in continual learning, and LoRAHub (Huang et al., 2024) composes separately trained LoRA modules to generalize to unseen tasks.

Table 3: Comparison of `JIVEAdapter` with representative adaptation methods.

Method	Multi-task	Shared+specific	Orthogonal	Adaptive rank	Backbone-preserving	Reusable
Prefix/Prompt (Li & Liang, 2021; Lester et al., 2021)						
LoRA (Hu et al., 2022)					✓	
AdaLoRA (Zhang et al., 2023)				✓	✓	
DoRA (Liu et al., 2024)					✓	
Adapters (Houlsby et al., 2019)						
AdapterFusion (Pfeiffer et al., 2021)	✓	✓				✓
HyperFormer(++) (Karimi Mahabadi et al., 2021)	✓	✓				
Polytropon (Ponti et al., 2022)	✓	✓			✓	✓
MTL-LoRA (Yang et al., 2025)	✓	✓			✓	
HydraLoRA (Tian et al., 2024)	✓	✓			✓	
MoDE (Ning et al., 2025)	✓	✓			✓	
O-LoRA (Wang et al., 2023)	✓		✓		✓	
LoRAHub (Huang et al., 2024)	✓				✓	✓
`JIVEAdapter` (ours)	✓	✓	✓	✓	✓	✓

B Algorithms and Proofs

This appendix presents the algorithms underlying `JIVEAdapter`.

B.1 Rank Allocation Algorithms

`JIVEAdapter` sets the per-pool ranks in one of two ways: the importance-based allocator and the budget-free permutation-test estimator of linear JIVE (Lock et al., 2013) (Section 3.2.1). This appendix details both (the importance-based allocator is summarized in Section 3.2.1).

B.1.1 Adaptive Rank Allocation Algorithm

We adopt AdaLoRA's importance-based rank allocation (Zhang et al., 2023) to the multi-task, per-structure setting. The shared Joint of each layer and each task's Individual are independent *structures*; every structure owns a vector of singular values E. This subsection gives the importance scoring, the per-pool budget schedule, and the full allocation algorithm.

Sensitivity and importance. Per factor parameter p we track a smoothed sensitivity and uncertainty:

$$I(p) = |p \odot \nabla p|, \qquad \overline{I} \leftarrow \beta_1 \overline{I} + (1-\beta_1) I(p), \qquad \overline{U} \leftarrow \beta_2 \overline{U} + (1-\beta_2)\big|I(p) - \overline{I}\big|, \tag{9}$$

and score $s(p) = \overline{I} \odot \overline{U}$. The score of singular value i of a structure combines its E entry with the mean scores of the corresponding V row and U column:

$$s_i \;=\; s(E)_i \;+\; \mathrm{mean}_{\mathrm{in}} s(V)_{i,:} \;+\; \mathrm{mean}_{\mathrm{out}} s(U)_{:,i}. \tag{10}$$

Per-structure budgeting. The Joint pool and the Individual pool are budgeted *separately* so many per-task structures cannot starve the single shared structure. Each pool's active-rank budget decays via the TRIPLET schedule from its full size to its target between the initial and final warmups (Alg. 5); within a pool, the lowest-scoring singular values across all its structures are zeroed to meet the budget. Optionally an explicit total budget is split proportionally between the pools.

Algorithm 4 TRIPLET(N, τ, g): per-pool active-rank budget schedule

Input: pool full size N, pool target τ, step g; warmups t_i, t_f; total steps T_{total}
Output: active-rank budget $b \in [\tau, N]$

if $g \le t_i$ **then return** N — *initial warmup: keep full rank*
else if $g \ge T_{\text{total}} - t_f$ **then return** τ — *final warmup: hold at target*
else
  $p \leftarrow (g - t_i) \,/\, (T_{\text{total}} - t_f - t_i)$ — *progress* $\in (0, 1)$
  return $\lfloor \tau + (N - \tau)(1 - p)^3 \rfloor$ — *triplet decay* $N \to \tau$
end if

Algorithm 5 Adaptive rank allocation step (allocator update-and-mask)

Input: model, global step g; totals N_J, N_I; targets τ_J, τ_I; warmups t_i, t_f; interval Δ

IMPORTANCE UPDATE: if $g < T_{\text{total}} - t_f$, update EMAs $\overline{I}$, $\overline{U}$ for all Joint/Individual factors
BUDGET SCHEDULE: $b_J \leftarrow \text{TRIPLET}(N_J, \tau_J, g)$; $b_I \leftarrow \text{TRIPLET}(N_I, \tau_I, g)$ — *Alg. 4, per pool*
mask $\leftarrow (g > t_i) \wedge \big(g > T_{\text{total}} - t_f \;\vee\; g \bmod \Delta = 0\big)$; if ¬mask **return**
SCORE AND THRESHOLD: pool$_J \leftarrow$ concat of $\{s_i\}$ over all Joint structures; pool$_I \leftarrow$ same over all Individual structures
$\theta_J \leftarrow (N_J - b_J)$-th smallest of pool$_J$; $\theta_I \leftarrow (N_I - b_I)$-th smallest of pool$_I$
MASK SINGULAR VALUES: for each Joint structure, zero $E_J[i]$ where $s_i \le \theta_J$; for each Individual (layer, t), zero $E_{I,t}[i]$ where $s_i \le \theta_I$
record the kept ranks per structure

Consistent scoring. Every singular value is ranked by the same importance score s_i above, identically for the Joint and Individual pools.

B.1.2 PERMUTATION-TEST RANK ESTIMATION ALGORITHM

This is the budget-free alternative from Section 3.2.1. It adopts the permutation test of linear JIVE (Lock et al., 2013) for weight space and runs it *once, before training*, as a one-shot estimator that fixes the ranks up front. For each task t (and for the shared module) we average the gradient of that task's loss with respect to the frozen base weight over a few batches, giving one weight-shaped block X_t per task. Passing the blocks $\{X_t\}$ to Alg. 6 yields a Joint rank and a per-task Individual rank. We then reduce these per-module estimates to the two ranks the adapter exposes. The Joint rank is the median across modules of each module's Joint rank. The Individual rank is computed in two steps. Within each module we average the per-task Individual ranks (the per-module mean), then take the median of those per-module means across modules. The adapter is rebuilt at these two ranks, and nothing is pruned during training.

Algorithm 6 Permutation-test rank estimation (Lock et al., 2013)

Input: per-task blocks $\{X_t\}_{t=1}^{k}$ (each the module's averaged frozen-weight gradient, $d \times n$, shared columns); permutations B; level α; max outer iters M
Output: Joint rank r_J; per-task Individual ranks $\{r_{I,t}\}$

$r_J \leftarrow 0$; $r_{I,t} \leftarrow 0\ \forall t$; $J_t \leftarrow 0$, $A_t \leftarrow 0$ — *initialize estimates*
for $m = 1, \ldots, M$ **do**
  $S \leftarrow [\, X_1 - A_1;\ \ldots;\ X_k - A_k \,]$ — *stack the individual-removed blocks*
  for $b = 1, \ldots, B$ **do** permute each block's columns independently, restack, record $\sigma_{\max}^{(b)}$
  end for
  $q_J \leftarrow$ the $(1 - \alpha)$ quantile of $\{\sigma_{\max}^{(b)}\}$
  $r_J \leftarrow \max\big(r_J,\ \#\{\text{leading singular values of } S > q_J\}\big)$ — *count from the top; non-decreasing*
  for $t = 1, \ldots, k$ **do**
    $q_t \leftarrow (1 - \alpha)$ quantile of top singular values of $X_t - J_t$ with entries permuted within each row
    $r_{I,t} \leftarrow \#\{\text{leading singular values of } X_t - J_t > q_t\}$ — *count from the top*
  end for
  $\{J_t\}, \{A_t\} \leftarrow \text{JIVEITER}(\{X_t\},\ r_J,\ \{r_{I,t}\})$[1] — *alternating low-rank fit,* $A_t \perp J$
  break if r_J and all $r_{I,t}$ are unchanged
end for
return $r_J,\ \{r_{I,t}\}$ — *aggregate across modules to the adapter's* r_J, r_I

[1] JIVEITER is the classic JIVE alternating decomposition of Lock et al. (2013).

Algorithm 7 Causal attribution of Joint vs. Individual (ablate & swap)

Input: trained multi-task model; eval sets $\{D_t\}_{t=1}^{T}$
Output: Joint deltas $\Delta_t^{\mathbf{J}}$, Individual deltas $\Delta_t^{\mathbf{I}}$, swap matrix $S \in \mathbb{R}^{T\times T}$
$a_t \leftarrow \textsc{Eval}(D_t)$ for all t *intact per-task accuracy*
ABLATE JOINT: $E_J \leftarrow 0$ (or $U_J \leftarrow 0$); $\Delta_t^{\mathbf{J}} \leftarrow a_t - \textsc{Eval}(D_t)\ \forall t$; restore
ABLATE EACH INDIVIDUAL: for $t = 1, \ldots, T$, set $\mathbf{I}_t \leftarrow 0$, take $\Delta_t^{\mathbf{I}} \leftarrow a_t - \textsc{Eval}(D_t)$, restore
SWAP INDIVIDUALS: for $(t, t') \in \{1, \ldots, T\}^2$, route D_t through $\mathbf{I}_{t'}$, take $S[t, t'] \leftarrow a_t - \textsc{Eval}(D_t)$, restore
return $\Delta^{\mathbf{J}}, \Delta^{\mathbf{I}}, S$ *small off-diagonal $S \Rightarrow$ interchangeable tasks*

B.2 Principal-Angle Subspace Overlap

This appendix justifies the overlap score for nearest-task routing in Structure Reuse Stage (2a) SIMILARITY (Eq. (8)). The singular values of $G^\top Q$ are exactly the cosines of the principal angles between the two subspaces (Björck & Golub, 1973).

Setup. Let $G \in \mathbb{R}^{n\times r}$ and $Q \in \mathbb{R}^{n\times r}$ have orthonormal columns, $G^\top G = Q^\top Q = \mathbb{I}_r$, spanning subspaces $\mathcal{G} = \mathrm{col}(G)$ and $\mathcal{Q} = \mathrm{col}(Q)$. The principal angles $0 \le \theta_1 \le \cdots \le \theta_r \le \pi/2$ between $\mathcal{G}$ and $\mathcal{Q}$ are defined recursively by

$$\cos\theta_i = \max_{\substack{u\in\mathcal{G},\ v\in\mathcal{Q}\\ \|u\|=\|v\|=1}} u^\top v \quad \text{s.t.} \quad u \perp u_j,\ v \perp v_j \ \ (j < i), \tag{11}$$

with maximizers u_i, v_i the i-th principal directions.

Proposition. Let $M = G^\top Q \in \mathbb{R}^{r\times r}$ have SVD $M = A\ \mathrm{diag}(\sigma_1, \ldots, \sigma_r)\, B^\top$ with $\sigma_1 \ge \cdots \ge \sigma_r \ge 0$. Then $\sigma_i = \cos\theta_i$ for all i, and the principal directions are $u_i = Ga_i$, $v_i = Qb_i$ (the a_i, b_i being the columns of A, B). Consequently

$$\|G^\top Q\|_F^2 = \sum_{i=1}^{r} \sigma_i^2 = \sum_{i=1}^{r} \cos^2\theta_i, \tag{12}$$

which is the (unnormalized) overlap in Eq. (8). This is the classical characterization of principal angles between subspaces (Björck & Golub, 1973); we include a short proof for completeness.

Proof. For unit vectors $u = Ga \in \mathcal{G}$ and $v = Qb \in \mathcal{Q}$, orthonormality of G, Q gives $\|u\| = \|a\|$ and $\|v\| = \|b\|$, so u, v are unit iff a, b are unit. Their inner product is

$$u^\top v = (Ga)^\top (Qb) = a^\top (G^\top Q) b = a^\top M b.$$

Thus maximizing $u^\top v$ over unit $u \in \mathcal{G}$, $v \in \mathcal{Q}$ is equivalent to maximizing $a^\top M b$ over unit $a, b \in \mathbb{R}^r$, whose optimum is the largest singular value σ_1 of M, attained at the leading singular vectors a_1, b_1. The orthogonality constraints in Eq. (11) ($u \perp u_j \Leftrightarrow a \perp a_j$ and $v \perp v_j \Leftrightarrow b \perp b_j$, again by orthonormality of G, Q) restrict the i-th step to the orthogonal complement of the previous singular vectors, whose optimum is the next singular value σ_i, attained at a_i, b_i. Hence $\cos\theta_i = \sigma_i$ and $u_i = Ga_i,\ v_i = Qb_i$. Finally $\|M\|_F^2 = \sum_i \sigma_i^2 = \sum_i \cos^2\theta_i$, giving Eq. (12). □

Remark on the range of the score. Since G, Q have orthonormal columns, $\|Gx\| = \|x\|$, so $M = G^\top Q$ has operator norm $\|M\|_2 \le 1$ and every $\sigma_i \in [0, 1]$; thus each $\theta_i = \arccos\sigma_i \in [0, \pi/2]$ is a well-defined angle. The normalized score $\frac{1}{r}\sum_i \cos^2\theta_i$ therefore lies in $[0, 1]$, equals 1 iff $\mathcal{G} = \mathcal{Q}$ (all $\theta_i = 0$), and equals 0 iff the subspaces are orthogonal (all $\theta_i = \pi/2$).

C Experimental Setup

This appendix documents the experimental setup and configuration. Unless otherwise stated, every result reported in the paper uses one fixed random seed, shared by `JIVEAdapter` and all baselines, so that every method sees the same seed, data order, and budget; the reported differences are therefore controlled comparisons that are not confounded by seed variation, and fixing the seed also makes each run exactly reproducible. Table 4 compares the capacity structure of `JIVEAdapter` and the baselines we run.

Table 4: Capacity structure and single-task cost (T=1) of `JIVEAdapter` and the baselines.

Method	Shared cap.	Task-specific cap.	Adaptive rank	#P (M)
LoRA	none	BA	no (fixed r)	1.33
AdaLoRA	none	$U\,\mathrm{diag}(E)V$	yes (cross-layer)	1.33
HydraLoRA	shared A	per-task heads B_t	no (fixed r)	2.90
MTL-LoRA	shared A	per-task Λ_t + up-proj	no (fixed r)	2.66
`JIVEAdapter`	Joint $\mathbf{J}$	Individual $\mathbf{I}_t$ ($\perp \mathbf{J}$)	yes	1.33
`JIVEAdapter` similarity reuse	frozen $\mathbf{J}$	routed $\mathbf{I}_{t'}$	no	0.00
`JIVEAdapter` adapt reuse	frozen $\mathbf{J}$	fresh $\mathbf{I}_t$	yes	0.66

C.1 Compute and Software Environment

All training and experiments run on a single Azure Machine Learning node of type `Standard_NC24ads_A100_v4`, which provides one NVIDIA A100 PCIe GPU with 80 GB of on-board memory, 24 AMD EPYC (Milan) vCPUs, and 220 GiB of host RAM. Every job uses a single GPU; no multi-GPU or multi-node parallelism is required, since the DeBERTaV3-base backbone with low-rank adapters has a small footprint and trains comfortably within the 80 GB budget, with ample headroom for the multi-task loader that keeps all task heads resident at once. The software stack is the Azure ML curated ACPT image built on PyTorch 2.2 and CUDA 12.1, on top of which we install a patched HuggingFace Transformers in editable mode. `JIVEAdapter` and every baseline share this identical image, hardware, and driver stack, so reported differences reflect the method rather than the environment.

C.2 Parameters and Invariants

Parameters and invariants. **Per-layer hyperparameters.** The ranks r_I, r_J, scalings α_I, α_J, dropout, and task count T. **Trainable parameters.** Shared Joint V_J, E_J, U_J and per-task Individual $V_{I,t}, E_{I,t}, U_{I,t}$ over T tasks; the base weight $\mathbf{W}$ and per-pool rank counts stay frozen. **Verified invariants.** $\Delta\mathbf{W}_t = 0$ at init; the routed forward equals per-sample (Joint + its Individual) to $\sim 10^{-6}$; embedding routing matches; Joint gets gradient from all tasks, Individual t only from task-t samples; single-task merge/unmerge is exact; the allocator prunes each pool to target and the forward still runs. **Homogeneous micro-batch variant.** When the loader cannot mix tasks, accumulating one per-task micro-batch each before a single step recovers the same gradient, with Joint seeing the summed gradient and each Individual only its own.

C.3 Configuration and Rank Budgets

This subsection documents the configuration used for the multi-task table (Table 2). All methods share the same backbone, injected modules (query, key, value, intermediate, layer.output, attention.output), and optimization schedule; they differ only in how adapter rank is organized.

Matched per-task effective rank. The controls and `JIVEAdapter` are matched on the *per-task effective rank*, i.e. the adapter rank any single task sees at inference, not on the nominal target-rank argument. AdaLoRA uses a single shared SVD pool with a target rank of 8 (its allocator's final average rank per structure). `JIVEAdapter` multitask splits capacity across per-structure rank pools, a shared Joint and a per-task Individual, each with a target rank of 4, so every task still sees Joint(4)+Individual(4)=8 active rank. LoRA uses a single fully-shared adapter at the same effective rank. All three start from an initial rank of 12 per structure (both the Joint and Individual pools of `JIVEAdapter` start at 12, as do the LoRA and AdaLoRA adapters) and scaling α=16.

Rank budget selection for fair comparison. We match the per-task effective rank across methods rather than the raw parameter count, which differs by structure. For a target average rank $\bar{r}$, AdaLoRA's budget is $b = \bar{r} \cdot n$ over n injected matrices; `JIVEAdapter` splits this between the Joint pool (r_J) and the Individual pool ($\sum_t r_{I,t}$) so that $r_J + \frac{1}{T}\sum_t r_{I,t} \approx \bar{r}$, keeping per-task capacity comparable. We vary $\bar{r} \in \{2, 4, 8\}$ (final total ranks $144, 288, 576$ over 72 matrices) with initial ranks $\{4, 6, 12\}$, matching AdaLoRA. Table 5 details the matched rank budget and shared optimization schedule.

Per-sample routing in DeBERTa's disentangled attention. DeBERTaV3's disentangled attention adds position-based query/key projections for the relative-position embeddings on top of the

Table 5: Multi-task run configuration at matched per-task effective rank 8, with AdaLoRA at one shared rank-8 pool and JIVEAdapter at a shared Joint plus per-task Individual of rank 4 each.

Rank budget (matched per-task effective rank 8)			
Method	Pools	Init rank	target_rank
LoRA (fully shared)	1 shared	12	(fixed)
AdaLoRA (fully shared)	1 shared	12	8
JIVEAdapter (Importance-based)	Joint + T Individuals	12	4 (per structure, 4+4=8)
Shared optimization schedule (all runs; allocator shared by AdaLoRA and JIVEAdapter)			
Learning rate	5×10^{-4}	Seed	6
Weight decay	0	Classifier dropout	0.1
Batch size	8	Orthogonality reg. coef.	0.1
Max sequence length	256	Allocator init warmup	4000
Training steps	20,000	Allocator mask interval	100
Warmup steps	1,000	Allocator final warmup	8000
Eval interval (steps)	2,000		
Method-specific settings (structured baselines)			
Setting	MTL-LoRA	HydraLoRA	
Rank	8	8	
Scaling α	16	16	
Dropout	0.1	0.1	
B matrices / experts	3	3	
B scale	1.0	n/a	

usual content query/key/value projections. JIVEAdapter multitask routes each sample to its Individual, and this routing must reach the position projections, not only the content ones. Leaving the position projections unrouted makes the multi-task model fail to converge; routing every attention projection fixes this and yields the reported GLUE average (89.07).

D Extended Experiments

D.1 Incremental JIVEAdapter as continual learning

Table 6 reports the full incremental study, comparing joint all-at-once JIVEAdapter, incremental JIVEAdapter (Algorithm 2), which learns the shared Joint from a base cluster and then folds each remaining task in by refining both the shared Joint and its own Individual, and two sequential fine-tuning references (a single shared adapter and full backbone fine-tuning) that keep no frozen shared structure and act as continual-learning lower bounds. To examine the effect of the base cluster size, we form the initial Joint from a cluster of size one (single-task base) or larger (multi-task base); for GLUE these are {MNLI} and {MNLI, QQP, SST-2, QNLI}, with the remaining tasks streamed in both cases. In practice the cluster size is dictated by model capacity, such as how many task heads the backbone can host for all-at-once training, and by whether the base-pool data is still available. All incremental and sequential runs use a matched rank budget with the same allocator and hyperparameters as the main experiments, scored on all tasks after the full stream. Both incremental variants score below joint all-at-once training, more so on GLUE (81.13 and 79.54 vs 89.07) than on SuperGLUE (80.72 and 73.69 vs 82.06), yet they far outperform the two sequential references (GLUE 61.11 and 69.55; SuperGLUE 68.36 and 65.60), whose shared adapter or fully fine-tuned backbone is overwritten by later tasks; JIVEAdapter avoids this *continual-learning* failure by keeping the shared Joint and refreshing each orthogonal Individual, so earlier tasks are preserved rather than forgotten. *The base cluster size matters more on the heterogeneous SuperGLUE tasks, where the multi-task base (*80.72*) clearly leads the single-task base (*73.69*) than on GLUE, where the single-task base (*79.54*) already recovers almost all of the multi-task base accuracy (*81.13*).*

Table 6: Incremental JIVEAdapter as a continual-learning procedure on GLUE and SuperGLUE, using a matched rank budget.

	Method	MNLI (acc)	QQP (acc/F1)	QNLI (acc)	SST-2 (acc)	CoLA (Mcc)	RTE (acc)	MRPC (acc/F1)	STS-B (P/S)	Avg
GLUE	JIVEAdapter (all at once)	**90.10**	**88.83**	**94.58**	**95.99**	**70.23**	**89.17**	**91.63**	**92.00**	**89.07**
	JIVEAdapter (incremental, multi-task base)	87.63	85.34	86.80	92.89	45.06	79.78	83.81	87.68	81.13
	JIVEAdapter (incremental, single-task base)	82.67	82.06	84.50	88.76	49.87	77.26	84.74	86.45	79.54
	Sequential adapter fine-tuning (baseline)	34.17	31.57	64.18	75.23	63.02	46.93	82.02	91.76	61.11
	Sequential full fine-tuning (baseline)	47.01	79.53	79.28	69.72	56.52	53.79	78.93	91.62	69.55
	Method	BoolQ (acc)	CB (acc/F1)	RTE (acc)	WiC (acc)	WSC (acc)	COPA (acc)	MultiRC (F1a/EM)	ReCoRD (EM/F1)	Avg
SuperGLUE	JIVEAdapter (all at once)	**84.86**	91.09	**85.92**	**69.28**	87.50	**91.00**	59.84	**87.02**	**82.06**
	JIVEAdapter (incremental, multi-task base)	84.19	**91.12**	84.84	68.97	**88.46**	86.00	56.66	85.50	80.72
	JIVEAdapter (incremental, single-task base)	68.10	86.86	64.26	62.70	76.92	86.00	**60.10**	84.57	73.69
	Sequential adapter fine-tuning (baseline)	59.88	71.77	71.48	58.62	61.54	89.00	48.90	85.67	68.36
	Sequential full fine-tuning (baseline)	65.26	72.84	63.90	55.17	61.54	78.00	45.35	82.76	65.60

D.2 IMPORTANCE-BASED VS. PERMUTATION-TEST RANK ESTIMATION

Both estimators allocate rank per structure; they differ only in how each structure's rank is chosen. We compare the default importance-based allocator (Appendix B.1.1), which prunes singular values under a decaying budget, against the budget-free permutation-test estimator on multi-task GLUE and SuperGLUE under the same backbone and schedule, both at matched effective per-task rank 8. On SuperGLUE the permutation test reaches the best average (82.06 vs. 81.21; Table 7), winning CB, WSC, COPA, and ReCoRD while the allocator keeps BoolQ, RTE, WiC, and MultiRC. On GLUE the picture reverses, and the importance-based allocator holds a small edge (89.07 vs. 88.54), keeping SST-2, CoLA, RTE, and STS-B while the permutation test takes MNLI, QQP, QNLI, and MRPC. Neither estimator dominates across benchmarks; since the permutation test removes the rank-budget hyperparameter, it is the more convenient default wherever it stays competitive.

Table 7: Importance allocator vs. budget-free permutation-rank test at matched per-task rank 8. The permutation test wins on SuperGLUE; the importance allocator edges it on GLUE.

	Rank-estimation strategy	MNLI (acc)	QQP (acc/F1)	QNLI (acc)	SST-2 (acc)	CoLA (Mcc)	RTE (acc)	MRPC (acc/F1)	STS-B (P/S)	Avg
GLUE	Importance-based (eff-rank 8)	90.10	90.34/87.32	94.58	**95.99**	**70.23**	**89.17**	90.20/93.06	**92.03/91.98**	**89.07**
	Permutation-test (eff-rank 8)	**90.25**	**90.39/87.39**	**94.60**	95.18	67.97	87.73	**90.69/93.33**	91.72/91.59	88.54
	Rank-estimation strategy	BoolQ (acc)	CB (acc/F1)	RTE (acc)	WiC (acc)	WSC (acc)	COPA (acc)	MultiRC (F1a/EM)	ReCoRD (EM/F1)	Avg
SuperGLUE	Importance-based (eff-rank 8)	**85.87**	**92.86**/86.21	**88.09**	**69.91**	78.85	90.00	**78.43/44.07**	85.91/86.48	81.21
	Permutation-test (eff-rank 8)	84.86	**92.86/89.32**	85.92	69.28	**87.50**	**91.00**	77.28/42.39	**86.76/87.28**	**82.06**

D.3 INTERPRETABILITY ANALYSIS

To understand why the Joint/Individual split drives our main benchmark results, we analyze the structure it learns. We expect the Joint to capture cross-task shared structure, each Individual to capture task-specific structure, and the two to remain orthogonal and interpretable. We cast these expectations as testable hypotheses and curate a targeted experiment for each.

D.3.1 HYPOTHESES

Our experiments test the following hypotheses.

H1: Separation. The Individual subspaces are diverse and each $\mathbf{I}_t$ is near-orthogonal to $\mathbf{J}$.

H2: Task geometry. Pairwise similarity of the task signatures is structured rather than uniform and gives each held-out task a consistent routing target.

H3: Rank-budget split between structures. How the fixed rank budget is divided between the shared Joint and the per-task Individuals affects multi-task performance, and we test which split works best.

D.3.2 EXPERIMENTS

E1: Learned subspace geometry (tests H1). To examine how the learned subspaces span the hidden space relative to one another, we sample unit vectors from each task's aggregate Individual subspace and from the shared Joint subspace and visualize them in Figure 3. The horizontal plane is a t-SNE of these vectors, and the vertical axis is each vector's energy in the Joint subspace,

$$z(v) \;=\; \|P_{\mathbf{J}}\, v\|^2 \;=\; \sum_{\ell} \|U_\ell^\top v_\ell\|^2 \;\in\; [0,1],$$

where $P_{\mathbf{J}}$ is the orthogonal projector onto the Joint column space (assembled per module ℓ from an orthonormal Joint basis U_ℓ) and v is unit norm. So z is the fraction of a vector's squared length that lies inside the Joint subspace; $z \approx 1$ means the vector lives almost entirely in the Joint, and $z \approx 0$ means it is orthogonal to the Joint. The companion matrix on the right reports the pairwise overlaps between the per-task Individual bases and the Joint.

As shown in figures, the eight Individual subspaces settle at low energy in the Joint direction while the Joint vectors rise to one, so every Individual carries almost no energy in the Joint; that is, each

$\mathbf{I}_t$ is near-orthogonal to $\mathbf{J}$ as H1 predicts. The companion matrix confirms this, since the Individual-in-Joint entries are near zero (the diagonal, each subspace against itself, is one by construction). The Individuals also occupy distinctly separated regions, so their cross-task overlaps are empirically small; this separation is learned, not imposed, since only the Individual $\perp$ Joint constraint is enforced during training, yet it yields non-overlapping learned subspaces alongside the shared Joint structure.

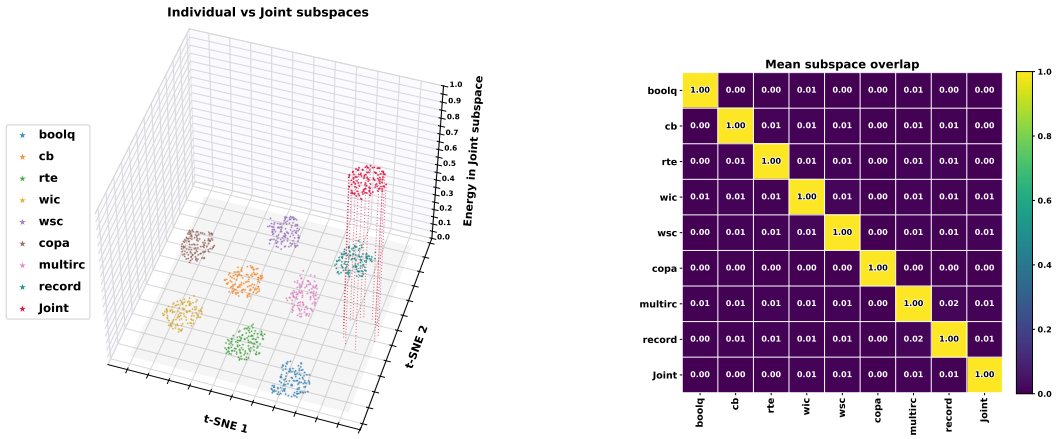


Figure 3: Learned Individual and Joint subspace geometry.

E2: Task signature similarity and routing (tests H2). A held-out task is never used during structure learning, so the learned arrangement of the trained Individuals does not bind it, and its reference signature is free to align with related held-in Individuals. Figure 4 reports that principal-angle similarity, and each held-out task routes to its highest-similarity held-in neighbor (boxed). The boolean_qa pair (BoolQ, MultiRC) is mutually highest; CB routes to BoolQ and RTE to MultiRC, while WiC, WSC, and COPA route to WSC, MultiRC, and WiC.

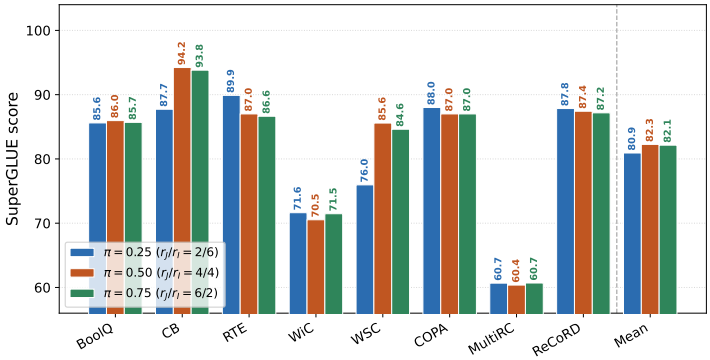
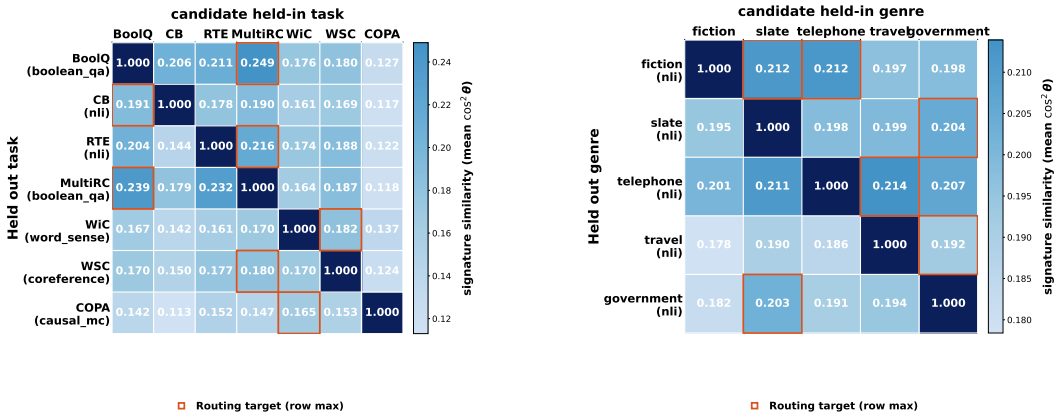


Figure 5: Rank-budget split between the shared Joint and the per-task Individuals, swept on multi-task SuperGLUE at fixed per-task effective rank 8.

E3: Rank-budget split between structures (tests H3). At a *fixed* per-task budget of effective rank 8, we split that budget between the shared Joint and the per-task Individual and vary the sharing ratio $\pi = r_J/(r_J + r_I)$, training `JIVEAdapter` multi-task on the eight SuperGLUE tasks at each split and recording the multi-task average (Figure 5). The allocation ratio has a measurable effect on performance, and a balanced split is optimal. The even $4/4$ split (π=0.50) attains the highest average at 82.26, above both the Individual-heavy $2/6$ split (π=0.25, 80.92) and the Joint-heavy $6/2$ split (π=0.75, 82.14). Giving the Joint too little (π=0.25) leaves cross-task structure underused, whereas giving it too much (π=0.75) starves the Individuals and degrades the more task-specific tasks such as WSC (85.58 $\rightarrow$ 84.62) and RTE (87.00 $\rightarrow$ 86.64). The two extremes recover baselines we already report; π=0 is Individuals only (T independent LoRAs with no transfer) and π=1 is



(a) SuperGLUE cross-task pool

(b) MNLI cross-genre pool

Figure 4: Held-out task signature similarity to the held-in Individuals; each row's boxed cell is its routing target.

a single fully-shared adapter (the LoRA/AdaLoRA MTL setting of Table 2), so the interior points are where the Joint plus Individual split pays off. This best fixed split (82.26) slightly exceeds the adaptive rank-estimation allocator reported in Tables 2 and 7 (82.06 on SuperGLUE); the allocator sets each structure's rank automatically without an explicit search over π, staying within a fraction of a point of the best hand-tuned split while removing that search. The allocator therefore automates what is otherwise a manual rank-allocation procedure, resolving the trade-off between an expensive grid search and accuracy at no measurable cost in performance.