SPARK: SPARSE SPECTRAL ADAPTATION VIA SMOOTH TOP-k SELECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter-efficient fine-tuning (PEFT) methods such as LoRA, DoRA and, more recently, HiRA have become the standard way of adapting large language models to downstream tasks, as they train only a small fraction of the parameters and thereby make adaptation considerably faster and cheaper than full fine-tuning. However, every adapted task still produces an adapter that has to be stored and loaded at inference time, and in settings that require many adapters, for instance one per user, the accumulated storage and memory cost becomes a significant bottleneck. LoRA-XS addresses this problem by training only a small $k \times k$ matrix placed between frozen factors obtained from the singular value decomposition (SVD) of the pretrained weight. Its adapters are extremely compact, but they are restricted to the k leading singular directions, which are determined by the pretrained weight alone and are not necessarily the most relevant for the downstream task. In this paper, we propose SPARK, which removes this restriction. SPARK trains in a much larger subspace spanned by $r \gg k$ singular directions and simultaneously learns which k of them to retain, using a differentiable top-k selection that is gradually hardened during training. After training, only a $k \times k$ matrix and the indices of the selected directions are stored, so the adapter is as small as that of LoRA-XS while being considerably more expressive. SPARK is compatible with any SVD-based update, and we apply it to both LoRA-XS and an SVD-based variant of HiRA. Experiments with Llama-3-8B on eight commonsense reasoning benchmarks show that SPARK matches or exceeds LoRA-XS-style adapters of the same size and surpasses HiRA while storing up to 85 times fewer parameters.

1 INTRODUCTION

Large language models are pretrained once and then adapted to many downstream tasks. Full fine-tuning updates every parameter and produces a complete copy of the model for each task, which for models with billions of parameters is too expensive to train and to keep. Parameter-efficient fine-tuning (PEFT) addresses this by freezing the pretrained weights and training a small number of additional parameters. Early approaches inserted adapter layers between the frozen blocks (Houlsby et al., 2019) or prepended trainable tokens to the input (Lester et al., 2021; Li & Liang, 2021). Low-rank adaptation (LoRA) (Hu et al., 2022) instead expresses the weight update as a product of two low-rank matrices, which can be merged into the pretrained weights after training so that inference incurs no extra cost. LoRA has become the default PEFT method, and a large body of work has refined it: DoRA decomposes the update into magnitude and direction (Liu et al., 2024), PiSSA initialises the factors from the principal singular components of the weight (Meng et al., 2024), QLoRA combines it with a quantised base model (Dettmers et al., 2023), and AdaLoRA allocates the rank budget across layers (Zhang et al., 2023). Since the rank of a LoRA update is bounded by the width of its factors, MoRA (Jiang et al., 2024) and HiRA (Huang et al., 2025) construct high-rank updates from the same number of trainable parameters, HiRA by multiplying a low-rank matrix elementwise with the pretrained weight.

The size of the trained adapter has received less attention than the cost of training it, although it determines the cost of deployment. A single base model is often served with many adapters, one per task, domain or user, and each adapter has to be stored, transferred and loaded into accelerator memory when it is needed. A rank-32 LoRA adapter for Llama-3-8B contains 56M parameters, so a few hundred adapters already exceed the size of the base model. VeRA (Kopiczko et al.,

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.

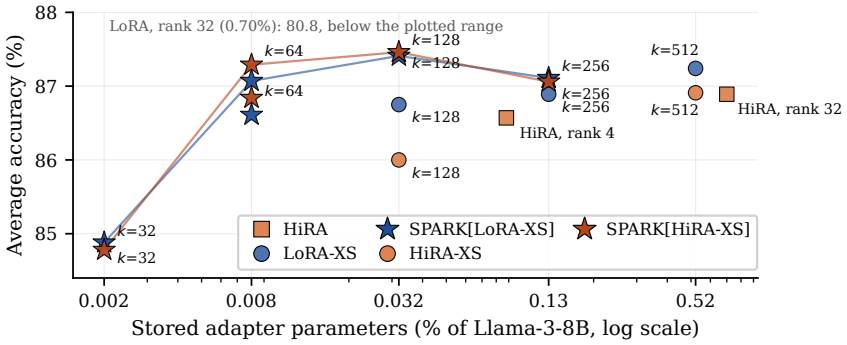

Figure 1: Average accuracy on eight commonsense reasoning tasks against the size of the stored adapter, Llama-3-8B. Stars are SPARK, labelled with the stored rank k; LoRA at rank 32 (80.8%) lies below the plotted range; each was trained with a rank $r > k$ (Table 1). At equal size SPARK matches or exceeds the fixed-basis adapter of the same family, and from 0.008% upwards it is above HiRA at 0.7%.

2024) reduces the adapter to two scaling vectors by sharing frozen random projections across layers. LoRA-XS (Bałazy et al., 2024) goes further: it fixes the projections to the leading singular vectors of the pretrained weight and trains only a $k \times k$ matrix between them. Since the singular vectors can be recomputed from the base model, the stored adapter consists of k^2 numbers per layer, two to three orders of magnitude fewer than LoRA at the same rank. The same parameterisation can be combined with the Hadamard update of HiRA, which we refer to as HiRA-XS.

These methods make two design choices that we revisit. First, the basis of the update is fixed to the k directions with the largest singular values. The singular vectors of the pretrained weight are a good basis for adaptation: Bałazy et al. (2024) show that they clearly outperform random projections of the same shape, and our method also builds on them. The ordering by singular value, however, reflects the pretrained weight alone and is decided before any task data is seen, so the leading directions need not be the ones along which the task requires the weight to change. Second, the single rank k determines both the capacity available during training and the size of the stored adapter, although the two need not be equal.

We propose SPARK, which separates the two roles of k. SPARK trains in a wider spectral basis of $r \gg k$ singular directions and, during the same run, learns which k rows and which k columns of the $r \times r$ core to retain. The selection is parameterised by logits that pass through a differentiable top-k operator (Struski et al., 2025) whose output satisfies the cardinality constraint exactly at every temperature. The temperature is annealed during training and the selection is then hardened, after which the retained $k \times k$ block and the indices of the selected directions are the only stored quantities. The adapter is therefore exactly as large as a rank-k LoRA-XS adapter, but its basis has been chosen from a much larger set of candidates according to the task. The construction does not depend on the form of the update, and we apply it without modification to LoRA-XS and to HiRA-XS.

Contributions. We show that the **storage required** for a fine-tuned adapter can be reduced by one to one to two orders of magnitude without loss in task performance, and in several settings with a gain. To this end we introduce SPARK, which decouples the training rank of an **SVD-based adapter** from its stored rank and learns the retained directions by gradient descent under an exact cardinality constraint. We characterise the optimal selection and show that it depends on the task update rather than on the singular values of the pretrained weight. On Llama-3-8B and eight commonsense reasoning benchmarks, SPARK **matches or exceeds** the fixed-basis adapter of equal size at every storage budget, and a SPARK adapter with 0.7M stored parameters outperforms HiRA with 56M.

2 Related work

Parameter-efficient fine-tuning. Adapters (Houlsby et al., 2019), prompt tuning (Lester et al., 2021) and prefix tuning (Li & Liang, 2021) adapt a frozen model by adding small trainable components to its architecture or input. LoRA (Hu et al., 2022) instead adds a low-rank update to the

weights, which can be merged after training, and most later work keeps this form. DoRA (Liu et al., 2024) separates the update into a magnitude and a direction, PiSSA (Meng et al., 2024) initialises the factors from the leading singular components of the weight, and QLoRA (Dettmers et al., 2023) trains the adapter on top of a quantised base model. AdaLoRA (Zhang et al., 2023) and SoRA (Ding et al., 2023) make the rank adaptive by pruning singular values or gating rank components during training, so that the budget is distributed across layers rather than fixed in advance. MoRA (Jiang et al., 2024) and HiRA (Huang et al., 2025) address the rank ceiling of LoRA directly; HiRA obtains a high-rank update by taking the Hadamard product of the pretrained weight with a low-rank matrix, and the resulting adapter still merges into the weights.

Compact adapters. A second line of work reduces the number of stored parameters rather than the number of trained ones. VeRA (Kopiczko et al., 2024) freezes a pair of random projections shared by all layers and trains only two scaling vectors per layer. LoRA-XS (Bałazy et al., 2024) replaces the random projections by the leading singular vectors of each pretrained weight and trains a small square matrix between them; because the projections are recomputed from the base model, only the square matrix is stored. SPARK keeps the storage format of LoRA-XS and changes how its basis is chosen. It is orthogonal to the choice of update rule and we combine it with both the additive update of LoRA-XS and the Hadamard update of HiRA.

Differentiable selection. Learning a discrete subset by gradient descent requires a relaxation. Gumbel-softmax (Jang et al., 2017) and its subset extension (Xie & Ermon, 2019) sample relaxed one-hot vectors, NeuralSort (Grover et al., 2019) and fast differentiable sorting (Blondel et al., 2020) relax the sorting operator, and optimal-transport formulations (Xie et al., 2020) relax top-k selection through a Sinkhorn iteration. L_0 regularisation (Louizos et al., 2018) and straight-through estimators (Bengio et al., 2013) train binary gates without an explicit budget. LapSum (Struski et al., 2025) constructs a soft top-k operator from a sum of Laplace distributions; its output satisfies the cardinality constraint exactly, its threshold has a closed form, and its cost is that of a sort. These properties make it suitable for selecting a fixed number of directions inside every adapted layer, which is how SPARK uses it.

Pruning. Training a large model and then removing part of it is the pattern of magnitude pruning, movement pruning (Sanh et al., 2020) and the lottery ticket hypothesis (Frankle & Carbin, 2019). SPARK follows the same pattern, with the difference that the pruned objects are indices into a fixed spectral basis rather than individual weights, and the number of retained indices is fixed by construction rather than controlled by a penalty.

3 BACKGROUND AND MOTIVATION

Motivation. The methods considered here all freeze the base model and train a number of parameters that is small compared to it, but they differ by orders of magnitude in what has to be kept after training. When a base model serves many tasks or users, the adapters are the part of the system that grows, and their size determines the storage, the transfer time and the accelerator memory needed to switch between them. Our aim is therefore an adapter that is as small as possible in its stored form, at task performance comparable to that of much larger adapters. Training-time cost is a secondary concern, and we accept an increase of it in exchange for a smaller stored adapter.

Notation. Throughout, $\boldsymbol{W}_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is a frozen pretrained weight matrix and $\Delta\boldsymbol{W}$ the update learned during fine-tuning, so that the adapted layer computes $\boldsymbol{x} \mapsto (\boldsymbol{W}_0 + \Delta\boldsymbol{W})\boldsymbol{x}$. Every method below is specified by how $\Delta\boldsymbol{W}$ is parameterised and by which of its factors are trained. We write $d = d_{\text{in}} = d_{\text{out}}$ for readability; nothing depends on the matrix being square. We distinguish the *trained* parameters of a layer, which determine the memory and compute used during fine-tuning, from its *stored* parameters, which are what has to be saved to reproduce the adapted layer given the base model. For most methods the two coincide; for the methods of interest here they do not.

LoRA. LoRA (Hu et al., 2022) parameterises the update as the product of two narrow matrices,

$$\Delta\boldsymbol{W} = \gamma\, \boldsymbol{B}\boldsymbol{A}, \qquad \boldsymbol{B} \in \mathbb{R}^{d \times k},\ \boldsymbol{A} \in \mathbb{R}^{k \times d}, \tag{1}$$

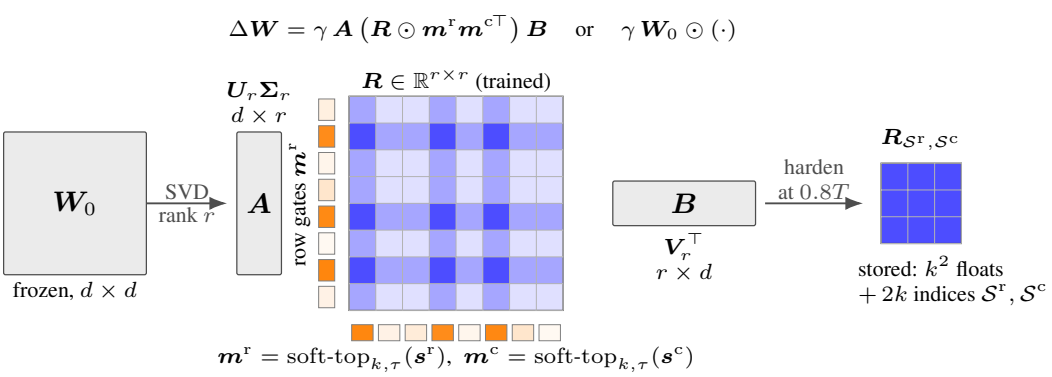

Figure 2: **SPARK for one adapted layer.** The frozen weight is factored once into a wide spectral basis of r directions. Only the $r \times r$ core $\boldsymbol{R}$ and two logit vectors are trained; a smooth top-k operator turns the logits into row and column gates that rescale $\boldsymbol{R}$. Late in training the gates are hardened and only the retained $k \times k$ block and its $2k$ indices are stored. At load time the basis is recomputed from the frozen weight.

with $k \ll d$ and a fixed scale γ. Both factors are trained and both are stored, so the layer costs $2dk$ parameters, and the update has rank at most k.

LoRA-XS. LoRA-XS (Bałazy et al., 2024) takes the factors from the pretrained weight itself. Let $\boldsymbol{W}_0 = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}$ be the singular value decomposition of $\boldsymbol{W}_0$ with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$, and let $\boldsymbol{U}_k$, $\boldsymbol{\Sigma}_k$, $\boldsymbol{V}_k$ collect the k leading singular directions. LoRA-XS writes

$$\Delta \boldsymbol{W} = \gamma\, \boldsymbol{U}_k \boldsymbol{\Sigma}_k\, \boldsymbol{R}\, \boldsymbol{V}_k^{\top}, \qquad \boldsymbol{R} \in \mathbb{R}^{k \times k}, \tag{2}$$

and trains only $\boldsymbol{R}$. The factors $\boldsymbol{U}_k\boldsymbol{\Sigma}_k$ and $\boldsymbol{V}_k^{\top}$ are frozen and are a deterministic function of $\boldsymbol{W}_0$, which the deployment target already holds; they are recomputed when the adapter is loaded and never saved. The stored parameters of a layer therefore drop from $2dk$ to k^2. For Llama-3-8B at $k = 32$ this is a reduction from 56M to 0.16M parameters over the adapted modules.

HiRA. The updates of Equations (1) and (2) have rank at most k, which bounds how far a layer can move from its pretrained state. HiRA (Huang et al., 2025) lifts this bound by using the low-rank product as an elementwise modulation of the weight instead of an additive term,

$$\Delta \boldsymbol{W} = \gamma\, \boldsymbol{W}_0 \odot (\boldsymbol{B}\boldsymbol{A}), \tag{3}$$

where $\odot$ is the Hadamard product. The rank of a Hadamard product is not bounded by the ranks of its factors, so $\Delta\boldsymbol{W}$ can be of high rank while $\boldsymbol{B}$ and $\boldsymbol{A}$ stay narrow, and the update still merges into $\boldsymbol{W}_0$ after training. The two ideas compose: substituting the frozen spectral factors of Equation (2) into Equation (3) gives

$$\Delta \boldsymbol{W} = \gamma\, \boldsymbol{W}_0 \odot \left(\boldsymbol{U}_k \boldsymbol{\Sigma}_k\, \boldsymbol{R}\, \boldsymbol{V}_k^{\top}\right), \tag{4}$$

which we call HiRA-XS. It has the storage cost of LoRA-XS and a high-rank update.

The fixed basis. Equations (2) and (4) share two properties that SPARK changes. The k retained directions are those with the largest singular values of $\boldsymbol{W}_0$, a choice made before any task data is seen, and the same k fixes both the size of the trained matrix $\boldsymbol{R}$ and the size of the stored adapter. The next section describes how to keep the stored size at k^2 while training in a wider basis and letting the task decide which directions to keep.

4 METHOD

SPARK consists of three parts: an over-provisioned spectral basis in which a masked core is trained, a differentiable selection of the retained rows and columns, and a hardening step that turns the selection into a stored adapter of the target size. We then describe how gradients reach the selection and give implementation details. Figure 2 and Algorithm 1 summarise the procedure.

A masked core in an over-provisioned basis. Fix a *stored rank* k, the rank the deployed adapter is required to have, and a *training rank* r with $k \le r \ll d$. Instead of truncating the SVD of $\boldsymbol{W}_0$ at k we truncate it at r and write $\boldsymbol{A} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \in \mathbb{R}^{d\times r}$ and $\boldsymbol{B} = \boldsymbol{V}_r^\top \in \mathbb{R}^{r\times d}$ for the frozen factors. The trainable core is $\boldsymbol{R} \in \mathbb{R}^{r\times r}$.

To every row and every column of the core we attach a gate. With a row mask $\boldsymbol{m}^{\mathrm{r}} \in [0,1]^r$ and a column mask $\boldsymbol{m}^{\mathrm{c}} \in [0,1]^r$, defined below, the masked core is

$$\boldsymbol{R}' = \operatorname{diag}(\boldsymbol{m}^{\mathrm{r}})\, \boldsymbol{R} \operatorname{diag}(\boldsymbol{m}^{\mathrm{c}}) = \boldsymbol{R} \odot \left(\boldsymbol{m}^{\mathrm{r}} \boldsymbol{m}^{\mathrm{c}\top}\right), \qquad \boldsymbol{R}'_{ij} = \boldsymbol{m}^{\mathrm{r}}_i \, \boldsymbol{R}_{ij} \, \boldsymbol{m}^{\mathrm{c}}_j, \tag{5}$$

and the update used during training is

$$\Delta \boldsymbol{W} = \gamma\, \boldsymbol{A}\, \boldsymbol{R}'\, \boldsymbol{B} \qquad \text{(additive, LoRA-XS)} \tag{6}$$

$$\Delta \boldsymbol{W} = \gamma\, \boldsymbol{W}_0 \odot \left(\boldsymbol{A}\, \boldsymbol{R}'\, \boldsymbol{B}\right) \qquad \text{(Hadamard, HiRA-XS).} \tag{7}$$

The trainable parameters are $\boldsymbol{R}$ and the logits that generate the two masks; $\boldsymbol{A}$ and $\boldsymbol{B}$ are frozen.

If the masks are binary, $\boldsymbol{m}^{\mathrm{r}} = \mathbf{1}_{\mathcal{S}^{\mathrm{r}}}$ and $\boldsymbol{m}^{\mathrm{c}} = \mathbf{1}_{\mathcal{S}^{\mathrm{c}}}$ for index sets of size k, then $\boldsymbol{R}'$ is zero outside the block $\boldsymbol{R}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{c}}}$ and Equation (6) reduces to

$$\Delta \boldsymbol{W} = \gamma\, \boldsymbol{A}_{:,\mathcal{S}^{\mathrm{r}}}\, \boldsymbol{R}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{c}}}\, \boldsymbol{B}_{\mathcal{S}^{\mathrm{c}},:} = \gamma\, \boldsymbol{U}_{:,\mathcal{S}^{\mathrm{r}}}\, \boldsymbol{\Sigma}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{r}}}\, \boldsymbol{R}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{c}}}\, \boldsymbol{V}^\top_{:,\mathcal{S}^{\mathrm{c}}}, \tag{8}$$

which is Equation (2) with the leading-k basis replaced by the k left and k right singular directions indexed by $\mathcal{S}^{\mathrm{r}}$ and $\mathcal{S}^{\mathrm{c}}$. The same holds for Equation (7). A hard pair of masks thus turns SPARK into LoRA-XS or HiRA-XS on a learned basis, and nothing trained in the wider basis has to be approximated at deployment. The row and column sets are chosen independently. This enlarges the search space from $\binom{r}{k}$ to $\binom{r}{k}^2$ index pairs and lets the adapter map an input direction onto a different output direction, which a shared mask cannot express (Section E). The problem is now to learn $\mathcal{S}^{\mathrm{r}}$ and $\mathcal{S}^{\mathrm{c}}$ by gradient descent.

Differentiable top-k selection. Each mask is produced from a vector of trainable logits, $\boldsymbol{s}^{\mathrm{r}}$ and $\boldsymbol{s}^{\mathrm{c}} \in \mathbb{R}^r$, by a differentiable relaxation of top-k selection,

$$\boldsymbol{m}^{\mathrm{r}} = \text{soft-top}_{k,\tau}(\boldsymbol{s}^{\mathrm{r}}), \qquad \boldsymbol{m}^{\mathrm{c}} = \text{soft-top}_{k,\tau}(\boldsymbol{s}^{\mathrm{c}}), \qquad \sum_i \boldsymbol{m}^{\mathrm{r}}_i = \sum_i \boldsymbol{m}^{\mathrm{c}}_i = k. \tag{9}$$

We use the LapSum operator of Struski et al. (2025). Each entry of the mask is the value of a Laplace cumulative distribution function at the distance of its logit from a common threshold,

$$\boldsymbol{m}_i = F\left(\frac{\boldsymbol{s}_i - b}{\tau}\right), \qquad F(z) = \begin{cases} \frac{1}{2} e^{z} & z < 0 \\ 1 - \frac{1}{2} e^{-z} & z \ge 0, \end{cases} \tag{10}$$

where $\tau > 0$ is a temperature and $b = b(\boldsymbol{s}, k, \tau)$ is the unique root of $\sum_i F((\boldsymbol{s}_i - b)/\tau) = k$. A sum of Laplace CDFs is piecewise exponential in b, so after sorting the logits the root is found by one search over the sorted values and one closed-form solve (Section C); the operator costs $O(r \log r)$ per layer and introduces no iterative approximation.

Three properties follow directly from Equation (10) and are the reason for this choice of relaxation. The mask entries lie in $(0,1)$ and sum to k exactly at every temperature, so the selection budget is respected throughout training rather than encouraged by a penalty. The operator preserves the order of the logits and converges to the hard top-k indicator as $\tau \to 0$ whenever the k-th and $(k{+}1)$-th largest logits differ. And it is differentiable in the logits for every $\tau > 0$, so the selection is trained by the task loss itself.

Gradient flow through the masks. The gradient of the training loss L reaches the logits through the masked core and the relaxation. For the row mask, writing $\boldsymbol{G}' = \partial L / \partial \boldsymbol{R}'$ and using Equation (5),

$$\frac{\partial L}{\partial \boldsymbol{m}^{\mathrm{r}}_i} = \sum_j \boldsymbol{G}'_{ij}\, \boldsymbol{R}_{ij}\, \boldsymbol{m}^{\mathrm{c}}_j = \frac{1}{\boldsymbol{m}^{\mathrm{r}}_i} \langle \boldsymbol{G}'_{i,:}, \boldsymbol{R}'_{i,:} \rangle, \tag{11}$$

and symmetrically for the column mask. The quantity $\boldsymbol{m}^{\mathrm{r}}_i\, \partial L / \partial \boldsymbol{m}^{\mathrm{r}}_i = \langle \boldsymbol{G}'_{i,:}, \boldsymbol{R}'_{i,:} \rangle$ is the first-order Taylor estimate of $L(\boldsymbol{R}') - L(\boldsymbol{R}' - \boldsymbol{e}_i \boldsymbol{R}'_{i,:})$, that is, of the increase in loss that removing row i of the

core would cause. The gate of a direction therefore receives, up to a positive factor, the importance score used in movement pruning (Sanh et al., 2020).

The relaxation then distributes this signal over the logits. Differentiating Equation (10) together with the constraint $\sum_i \boldsymbol{m}_i = k$ gives (Section C)

$$\frac{\partial \boldsymbol{m}_i}{\partial \boldsymbol{s}_j} = \frac{f_i}{\tau}\Big(\delta_{ij} - \frac{f_j}{\sum_l f_l}\Big), \qquad f_i = F'\Big(\frac{\boldsymbol{s}_i - b}{\tau}\Big) = \min(\boldsymbol{m}_i, 1 - \boldsymbol{m}_i). \tag{12}$$

Two consequences shape training. First, the columns of the Jacobian sum to zero, so the gradient with respect to any logit moves mask mass between directions and never changes the total: a logit rises when the importance of its direction exceeds the f-weighted average importance of the others, and falls otherwise. Second, f_i vanishes for saturated gates, so once a direction is confidently selected or rejected it stops receiving gradient, and as τ decreases the selection freezes from the most confident directions inward while the undecided ones keep competing.

The parameterisation Equation (5) has a multiplicative degeneracy: scaling row i of $\boldsymbol{R}$ by c and $\boldsymbol{m}^{\mathrm{r}}_i$ by $1/c$ leaves $\boldsymbol{R}'$ and the loss unchanged. A small gate alone does therefore not identify an unimportant direction, and the hardening criterion below accounts for this.

Annealing and hardening. The temperature is decayed geometrically from $\tau_0 = 1$ to $\tau_1 = 10^{-2}$ over the first T_h steps, $\tau_t = \tau_0(\tau_1/\tau_0)^{t/T_h}$. Early in training the masks are soft, gradients reach every direction and the selection can reorganise; by T_h the masks are close to binary. Because both masks multiply $\boldsymbol{R}$, a direction whose gates decay on both sides contributes quadratically less to the update, so the removal of a direction is gradual.

At step T_h, set to 80% of training, the selection is hardened:

$$\mathcal{S}^{\mathrm{r}} = \text{top-}k_i\big(\boldsymbol{m}^{\mathrm{r}}_i \, \|\boldsymbol{R}_{i,:}\|_2\big), \qquad \mathcal{S}^{\mathrm{c}} = \text{top-}k_j\big(\boldsymbol{m}^{\mathrm{c}}_j \, \|\boldsymbol{R}_{:,j}\|_2\big). \tag{13}$$

The score multiplies each gate by the norm of the row or column it gates, which is invariant under the degeneracy noted above. The masks are replaced by the indicators of $\mathcal{S}^{\mathrm{r}}$ and $\mathcal{S}^{\mathrm{c}}$, the logits are discarded, and the core is reduced to its $k \times k$ block, which continues to train for the remaining steps and absorbs the small change caused by discretisation.

Storage, loading and cost. The stored adapter consists of $\boldsymbol{R}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{c}}} \in \mathbb{R}^{k\times k}$ and the two index vectors, that is, k^2 numbers and $2k\lceil \log_2 r \rceil$ bits per layer. For $k = 128$ and $r = 1024$ the indices add 1.6% to a 32-bit adapter. At load time the recipient computes the rank-r SVD of $\boldsymbol{W}_0$, gathers the columns of $\boldsymbol{A}$ indexed by $\mathcal{S}^{\mathrm{r}}$ and the rows of $\boldsymbol{B}$ indexed by $\mathcal{S}^{\mathrm{c}}$, and applies Equation (8); the result can be merged into $\boldsymbol{W}_0$ so that inference costs nothing extra, exactly as for LoRA-XS and HiRA. The stored rank k is shared by all adapted layers and the selection is made independently in each.

Training is more expensive than LoRA-XS at rank k. The trainable state is $r^2 + 2r$ per layer and the optimiser state scales with it, and the soft top-k operator adds a sort and a threshold solve per layer and step. This is the trade the method makes: training-time cost for a smaller stored adapter. Table 1 reports the trained and the stored parameter fraction of every configuration.

Implementation details. The SVD of every adapted weight is computed once in float32, cached, and the factors $\boldsymbol{A}, \boldsymbol{B}$ are kept in the compute precision of the base model (bfloat16); they are never part of the optimiser state. The core and the logits are kept in float32 and cast at the point of use. The masks are never materialised as diagonal matrices; Equation (5) is one elementwise product. The additive update is applied as $\boldsymbol{x} \mapsto \boldsymbol{A}(\boldsymbol{R}'(\boldsymbol{B}\boldsymbol{x}))$ and never forms a $d \times d$ matrix, while the Hadamard update must form $\boldsymbol{A}\boldsymbol{R}'\boldsymbol{B}$, as HiRA does. The LapSum backward pass is implemented from Equation (12) rather than by differentiating through the sort. The core is initialised with small Gaussian entries ($\sigma = 10^{-4}$), so that $\Delta\boldsymbol{W} \approx 0$ at the start and every direction receives a nonzero gradient from the first step; the logits are initialised near zero ($\sigma = 10^{-2}$), so that the initial masks are close to the uniform value k/r and the selection starts uninformed rather than from the top-k solution of LoRA-XS. The logits use a learning rate ten times that of the core and no weight decay. Setting $r = k$ and disabling the masks recovers LoRA-XS and HiRA-XS exactly, and the baselines in Section 5 are run through this code path.

Algorithm 1: SPARK training and hardening for one adapted layer

Input: frozen weight $\boldsymbol{W}_0$; stored rank k; training rank r; temperature schedule τ_t; hardening step $T_h = 0.8T$; total steps T

$\boldsymbol{U}_r, \boldsymbol{\Sigma}_r, \boldsymbol{V}_r \leftarrow$ rank-r truncated SVD of $\boldsymbol{W}_0$; $\boldsymbol{A} \leftarrow \boldsymbol{U}_r\boldsymbol{\Sigma}_r$, $\boldsymbol{B} \leftarrow \boldsymbol{V}_r^\top$ initialise $\boldsymbol{R} \in \mathbb{R}^{r\times r}$ and logits $\boldsymbol{s}^{\mathrm{r}}, \boldsymbol{s}^{\mathrm{c}} \in \mathbb{R}^r$;

for $t = 1 \dots T_h$ **do**

  $\boldsymbol{m}^{\mathrm{r}} \leftarrow \text{soft-top}_{k,\tau_t}(\boldsymbol{s}^{\mathrm{r}})$, $\boldsymbol{m}^{\mathrm{c}} \leftarrow \text{soft-top}_{k,\tau_t}(\boldsymbol{s}^{\mathrm{c}})$ $\boldsymbol{R}' \leftarrow \boldsymbol{R} \odot (\boldsymbol{m}^{\mathrm{r}}\boldsymbol{m}^{\mathrm{c}\top})$

  $\Delta\boldsymbol{W} \leftarrow \gamma\, \boldsymbol{A}\boldsymbol{R}'\boldsymbol{B}$ or $\gamma\, \boldsymbol{W}_0 \odot (\boldsymbol{A}\boldsymbol{R}'\boldsymbol{B})$;

  update $\boldsymbol{R}$, $\boldsymbol{s}^{\mathrm{r}}$, $\boldsymbol{s}^{\mathrm{c}}$ by gradient descent on the task loss;

$\mathcal{S}^{\mathrm{r}}, \mathcal{S}^{\mathrm{c}} \leftarrow$ top-k of $\boldsymbol{m}^{\mathrm{r}}_i \|\boldsymbol{R}_{i,:}\|$ and of $\boldsymbol{m}^{\mathrm{c}}_j \|\boldsymbol{R}_{:,j}\|$ $\boldsymbol{R} \leftarrow \boldsymbol{R}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{c}}}$; discard logits;

for $t = T_h + 1 \dots T$ **do**

  update $\boldsymbol{R}$ with binary masks;

Output: $\boldsymbol{R} \in \mathbb{R}^{k\times k}$, $\mathcal{S}^{\mathrm{r}}$, $\mathcal{S}^{\mathrm{c}}$

5 EXPERIMENTS

5.1 SETUP

Tasks and evaluation. We follow the commonsense reasoning protocol of Hu et al. (2023), which is also used by DoRA and HiRA. A single model is fine-tuned on a mixture of the training sets of eight tasks, BoolQ, PIQA, SIQA, HellaSwag, WinoGrande, ARC-easy, ARC-challenge and OBQA, comprising 170k question-answer pairs, and evaluated on each test set separately. The model generates an answer and accuracy is measured by matching the first answer keyword in the generation against the label. For mathematical reasoning we fine-tune on MetaMath (Yu et al., 2023) and report accuracy on GSM8K (Cobbe et al., 2021).

Models and baselines. All experiments use Meta-Llama-3-8B (Dubey et al., 2024) with frozen weights. We compare against LoRA (Hu et al., 2022) and HiRA (Huang et al., 2025) as full-size adapters, and against LoRA-XS (Bałazy et al., 2024) and HiRA-XS (Equation (4)) as compact adapters of the same stored form as SPARK. The two compact baselines are SPARK with $r = k$ and the masks disabled, run through the same code, so any difference to SPARK is due to the wider basis and the learned selection. The LoRA and HiRA results at rank 32 are those reported by Huang et al. (2025); the remaining rows are our runs.

Protocol. Adapters are placed on the query, key, value, up and down projections of every layer, following Huang et al. (2025). Commonsense models are trained for three epochs and MetaMath models for two, and the checkpoint is selected on a held-out validation split. SPARK hardens the selection at 80% of the training steps. Learning rates, batch sizes and the remaining hyperparameters are listed in Section A. All numbers are single runs.

Parameter accounting. Each configuration is described by its *stored* parameters, the size of the deployed adapter ($k(d_{\text{in}} + d_{\text{out}})$ per module for LoRA and HiRA, k^2 for the XS methods and SPARK, whose $2k$ indices add less than 2%), and its *trained* parameters, those that receive gradients ($r^2 + 2r$ per module for SPARK), both as a fraction of the 8.03B base parameters. The comparison this paper is about is between rows of equal stored size.

5.2 COMMONSENSE REASONING

Table 1 groups all configurations by stored size and Figure 1 plots average accuracy against it.

Equal stored size. At every stored size, SPARK reaches higher average accuracy than the fixed-basis adapter of the same size and family. The margin over LoRA-XS is 0.7 points at $k = 128$ and 0.2 points at $k = 256$; the margin over HiRA-XS at $k = 128$ is 1.5 points, and the improvement holds on all eight tasks (Figure 3), with the largest gains on ARC-challenge ($+4.7$) and HellaSwag ($+1.8$). The additive and the Hadamard instantiation of SPARK are within 0.2 points of each other at every budget.

Table 1: Commonsense reasoning with Llama-3-8B. Accuracy (%) of a single model fine-tuned on the 170k mixture. r is the training rank and k the stored rank ($r = k$ for the baselines). Rows are grouped by stored size; within each group the best average is in bold and the second best is underlined.

Method	r	k	Params (%)		BoolQ	PIQA	SIQA	ARC-c	ARC-e	OBQA	HellaS	WinoG	Avg.
			stored	trained									
Reference adapters (full factors stored)													
LoRA (Hu et al., 2022)	32	32	0.700	0.700	70.80	85.20	79.90	71.20	84.20	79.00	91.70	84.30	80.79
HiRA (Huang et al., 2025)	4	4	0.088	0.088	74.77	89.61	82.04	82.34	93.48	87.80	95.22	87.29	<u>86.57</u>
HiRA (Huang et al., 2025)	32	32	0.700	0.700	74.71	89.66	82.29	83.53	93.31	88.09	95.62	88.00	**86.89**
Stored 0.52% ($k = 512$)													
LoRA-XS	512	512	0.522	0.522	76.4	89.3	81.8	84.4	93.9	88.0	95.8	88.3	**87.24**
HiRA-XS	512	512	0.522	0.522	74.12	89.50	81.32	84.73	94.07	88.00	95.69	87.85	<u>86.91</u>
Stored 0.13% ($k = 256$)													
LoRA-XS	256	256	0.131	0.131	75.3	89.3	82.2	83.2	93.2	88.0	96.0	87.9	86.89
SPARK[LORA-XS]	1024	256	0.131	2.09	75.23	89.08	81.90	85.14	94.05	88.00	95.67	87.77	**87.11**
SPARK[HIRA-XS]	1024	256	0.131	2.09	74.87	88.87	81.98	85.40	94.02	87.44	96.02	87.88	<u>87.06</u>
Stored 0.032% ($k = 128$)													
LoRA-XS	128	128	0.032	0.032	75.2	89.1	83.5	81.3	93.4	87.4	95.6	88.5	86.75
HiRA-XS	128	128	0.032	0.032	74.74	88.63	82.09	81.48	92.55	87.20	94.34	86.90	86.00
SPARK[LORA-XS]	1024	128	0.032	2.09	75.08	89.23	82.90	85.64	94.10	88.20	96.23	87.89	<u>87.41</u>
SPARK[HIRA-XS]	1024	128	0.032	2.09	74.92	89.15	82.97	86.15	94.16	88.05	96.12	88.12	**87.46**
Stored 0.008% ($k = 64$)													
SPARK[LORA-XS]	256	64	0.008	0.132	72.56	88.95	82.34	84.67	93.75	87.20	95.60	87.80	86.61
SPARK[LORA-XS]	512	64	0.008	0.524	74.80	89.01	82.65	85.12	94.05	87.24	95.70	88.00	<u>87.07</u>
SPARK[HIRA-XS]	256	64	0.008	0.132	73.90	88.73	82.14	84.87	93.90	88.00	95.86	87.29	86.84
SPARK[HIRA-XS]	512	64	0.008	0.524	74.98	88.97	83.01	85.32	94.30	88.05	95.80	87.92	**87.29**
Stored 0.002% ($k = 32$)													
SPARK[LORA-XS]	512	32	0.002	0.524	69.87	87.12	80.15	81.14	92.45	86.20	95.69	86.43	**84.88**
SPARK[HIRA-XS]	512	32	0.002	0.524	70.12	87.01	80.32	80.40	92.82	86.97	94.62	85.98	<u>84.78</u>

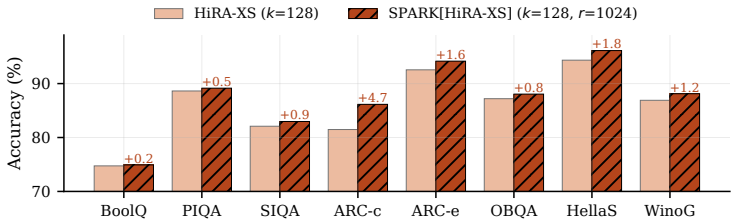

Figure 3: Per-task accuracy at equal stored size ($k = 128$, 0.032%). SPARK[HIRA-XS] improves on HIRA-XS on all eight tasks; the largest gains are on the tasks that require multi-step reasoning.

Smaller adapters. The practical comparison is across stored sizes. SPARK at $k = 64$ (0.008%, 0.66M stored parameters) with $r = 512$ reaches 87.1 to 87.3%. This equals or exceeds LoRA-XS at $k = 128$, 256 and 512, which store 4, 16 and 64 times as much, and HiRA at rank 32, which stores 85 times as much. The best configuration, SPARK[HIRA-XS] at $k = 128$ (87.46% at 0.032%), is above every baseline in the table, including LoRA-XS at $k = 512$ (87.24% at 0.52%), with an adapter 16 times smaller. At $k = 32$ (0.002%, 0.16M parameters) accuracy drops to 84.8%, four points above LoRA at rank 32.

5.3 MATHEMATICAL REASONING

Table 2 reports GSM8K accuracy after fine-tuning on MetaMath. SPARK improves on the compact baselines of equal stored size and, at $k = 256$, is within 1.4 points of HiRA at rank 32 with an adapter 5.7 times smaller, while exceeding LoRA, DoRA and MoRA at 0.7%. The gap to HiRA is larger than on commonsense reasoning, consistent with mathematical reasoning benefiting from higher-rank updates (Jiang et al., 2024).

Table 2: **GSM8K accuracy (%) with Llama-3-8B fine-tuned on MetaMath.** Rows marked † are from Huang et al. (2025).

Method	r	k	Stored (%)	Trained (%)	GSM8K
LoRA†	32	32	0.700	0.700	65.89
DoRA† (Liu et al., 2024)	32	32	0.700	0.700	66.12
MoRA† (Jiang et al., 2024)	32	32	0.700	0.700	67.98
HiRA†	16	16	0.351	0.351	67.63
HiRA†	32	32	0.700	0.700	**70.81**
LoRA-XS	256	256	0.131	0.131	64.7
HiRA-XS	128	128	0.032	0.032	66.2
SPARK[LoRA-XS]	512	64	0.008	0.524	66.5
SPARK[LoRA-XS]	1024	128	0.032	2.09	67.9
SPARK[HiRA-XS]	512	64	0.008	0.524	67.1
SPARK[HiRA-XS]	1024	128	0.032	2.09	68.6
SPARK[HiRA-XS]	1024	256	0.131	2.09	<u>69.4</u>

Table 3: Effect of the frozen basis. SPARK[HiRA-XS] at $k = [64]$, $r = [512]$ with the singular vectors of the pretrained weight and with random matrices of the same shape.

Basis	BoolQ	PIQA	SIQA	ARC-c	ARC-e	OBQA	HellaS	WinoG	Avg.
SVD of $\boldsymbol{W}_0$	74.98	88.97	83.01	85.32	94.30	88.05	95.80	87.92	87.29
Random	73.11	86.78	80.96	91.19	79.52	86.40	94.82	86.42	84.91

6 Ablations

The spectral basis matters. SPARK chooses its directions from the singular vectors of the pretrained weight. To test whether this choice matters, or whether any sufficiently large frozen basis would do, we replace $\boldsymbol{A}$ and $\boldsymbol{B}$ by random matrices of the same shape and train SPARK unchanged. Table 3 reports the result at $k = [64]$, $r = [512]$. Average accuracy drops from $[87.3]$ to $[XX]$, below every fixed-basis baseline, and the drop is consistent across tasks. The learned selection therefore does not compensate for an uninformative basis; it exploits the structure that the SVD of the pretrained weight provides.

A second question is whether the learned selection is stable: with the SVD basis, two runs with different seeds retain overlapping index sets well above the chance level k/r (45–60% against 25%), with the degree of agreement varying across module types, whereas with a random basis the overlap stays at chance (Section B.2, Table 6).

Training rank. Table 1 also contains a sweep of the training rank at a fixed stored rank. At $k = 64$, increasing r from 256 to 512 improves average accuracy by 0.45 points for both instantiations, and the same stored adapter trained with $r = k$, the LoRA-XS or HiRA-XS baseline, is lower still; only the pool from which the k directions are chosen changes along this sweep.

7 Conclusion

The fixed top-k truncation of LoRA-XS-type adapters is a convention, not a requirement. SPARK keeps the stored form of these adapters and learns their basis instead: it trains in a wider spectral basis and selects the retained rows and columns with a smooth top-k operator under an exact budget, annealed and hardened during training. On commonsense reasoning with Llama-3-8B the resulting adapters match or exceed every baseline of equal size and exceed HiRA while being up to 85 times smaller.

Limitations. The main limitation of SPARK is its training cost, both the larger training rank and the smooth top-k operator entail significant latency increase during training.

REPRODUCIBILITY STATEMENT

The experimental protocol is described in Section 5.1; all hyperparameters, the adapted modules, the parameter accounting and the hardware are in Section A; the smooth top-k operator, its closed-form threshold and its gradient are given in Section C in enough detail to be reimplemented; proofs are in Section D. The anonymised code is provided in the supplementary material.

AI USE STATEMENT

In this work, we used generative AI tools for assisting with the implementation of the method and baselines, checking the statements and proofs in Section 4, Section C and Section E, and for interpreting results. We have not used generative AI tools for proposing the research hypothesis or designing the experiments. Additionally, we used generative AI tools to produce and format figures and tables, to assist with the literature search and reference formatting, and to edit the manuscript for readability. We have reviewed all AI-assisted work.

REFERENCES

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. LoRA-XS: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *International Conference on Machine Learning (ICML)*, 2020.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Abhimanyu Dubey et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations (ICLR)*, 2019.

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.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Qiushi Huang, Tom Ko, Zhan Zhuang, Lilian Tang, and Yu Zhang. HiRA: Parameter-efficient Hadamard high-rank adaptation for large language models. In *International Conference on Learning Representations (ICLR)*, 2025.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, et al. MoRA: High-rank updating for parameter-efficient fine-tuning. *arXiv preprint arXiv:2405.12130*, 2024.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based random matrix adaptation. In *International Conference on Learning Representations (ICLR)*, 2024.

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

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 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.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations (ICLR)*, 2018.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*, 2024.

Victor Sanh, Thomas Wolf, and Alexander M Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, 2020.

Łukasz Struski, Michał B. Bednarczyk, Igor T. Podolak, and Jacek Tabor. LapSum – one method to differentiate them all: Ranking, sorting and top-k selection. In *International Conference on Machine Learning (ICML)*, 2025.

Sang Michael Xie and Stefano Ermon. Reparameterizable subset sampling via continuous relaxations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

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 Implementation details

Parameter counts. Llama-3-8B has $d_{\text{model}} = 4096$, $d_{\text{ff}} = 14336$ and 8 key/value heads, so per layer the adapted modules have shapes 4096×4096 (q), 1024×4096 (k, v), 14336×4096 (up) and 4096×14336 (down). A LoRA/HiRA adapter of rank k has $k \sum(d_{\text{in}} + d_{\text{out}}) = 55\,296\,k$ parameters per layer (0.700% of 8.03B at $k = 32$). Every XS-type adapter, SPARK included, stores $5k^2$ per layer: 0.002%, 0.008%, 0.032% and 0.123% at $k = 32, 64, 128, 256$. SPARK additionally stores $2k$ indices of $\lceil \log_2 r \rceil \leq 10$ bits, which adds 1.6% ($k = 128, r = 1024$) to 3.1% ($k = 64, r = 512$) of the adapter size. Trainable parameters of SPARK are $r^2 + 2r$ per module: 0.132%, 0.524% and 2.09% at $r = 256, 512, 1024$.

Table 4: Hyperparameters. All methods share the optimiser settings; the lower block applies to SPARK only.

Hyperparameter	Value
Base model	Meta-Llama-3-8B, bfloat16, gradient checkpointing
Adapted modules	`q_proj`, `k_proj`, `v_proj`, `up_proj`, `down_proj` (160 modules)
Optimiser	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$), weight decay 0.01
Learning rate	$5 \cdot 10^{-4}$, 100 warm-up steps, linear decay; gradient clipping at 1.0
Epochs / batch size	3 / 16 (commonsense); 2 / 32 (MetaMath)
Sequence length	256
Checkpoint selection	validation loss, evaluated every 80 steps
Seed	36
SVD	exact, float32, on CPU, computed once and cached; factors kept in bfloat16
Scale γ	1.0
Core init	$\mathcal{N}(0, 10^{-4})$ (LoRA-XS/HiRA-XS baselines: zero)
Logit init	$\mathcal{N}(0, 10^{-2})$; logit learning rate $10\times$ core; no weight decay on logits
Soft top-k	LapSum, τ: $1.0 \to 0.01$, geometric
Hardening	at $0.8T$, criterion $\boldsymbol{m}_i \, \|\boldsymbol{R}_{i,:}\|_2$; core keeps training after hardening

B ADDITIONAL RESULTS

In this section, we present some additional results such as performance of fine-tuning on single commonsense tasks and an ablation on the consistency of selected directions.

B.1 SINGLE-TASK FINE-TUNING

Table 5 repeats the comparison with a separate model fine-tuned on each task's own training set (the setting of Appendix D of Huang et al., 2025). The ordering of the fixed-truncation baselines is the same as in the mixture setting; SPARK runs in this setting are in progress.

Table 5: Accuracy (%) with one adapter per task, Llama-3-8B. Entries marked – were not run or did not complete.

Method	k	Stored (%)	BoolQ	PIQA	SIQA	ARC-c	ARC-e	OBQA	HellaS	WinoG	Avg.
HiRA	4	0.088	68.10	87.60	80.66	69.62	90.03	82.20	94.56	85.64	82.30
HiRA	5	0.111	69.14	88.30	81.27	73.98	89.48	83.80	94.76	86.27	83.38
HiRA	32	0.700	72.05	88.92	81.97	76.45	90.74	86.00	95.56	86.56	84.78
HiRA-XS	128	0.032	70.80	89.50	81.17	76.11	90.61	84.80	95.20	86.50	84.34
HiRA-XS	256	0.120	75.64	89.39	79.84	77.65	90.57	86.20	95.11	86.27	85.08
HiRA-XS	512	0.470	75.14	89.45	80.91	78.84	91.62	86.40	95.60	86.98	85.62

B.2 CONSISTENCY OF THE SELECTION ACROSS SEEDS

If the learned indices reflected only optimisation noise, two independent runs would agree on their selected directions no more often than chance. For two runs of SPARK[LoRA-XS] at $k = 64$, $r = 256$ that differ only in the random seed (initialisation of the core and of the mask logits, and the data order), we measure for every adapted module the overlap $|\mathcal{S}_1^{\mathrm{r}} \cap \mathcal{S}_2^{\mathrm{r}}|/k$ of the retained row indices, and likewise for the columns. Under a uniformly random selection the expected overlap is $k/r = 25\%$. Table 6 reports the range of the mean overlap across module types (rows and columns pooled) for two choices of the frozen basis. With the SVD basis the overlap lies between 45% and 60% depending on the module type, above chance everywhere: the selection carries signal about which directions the task needs, and the variation across module types suggests that some layers have a small set of clearly preferred directions while in others many directions are interchangeable. With a random orthonormal basis, shared by the two runs, the overlap is 23–27%, indistinguishable from chance: when the basis carries no structure, the selection is not reproducible, so the consistency observed with the SVD basis is a property of the singular directions rather than of the selection mechanism alone. The selected SVD directions are not the leading ones: only about a quarter of the retained indices fall among the top-k singular directions, which is the chance level as well.

Table 6: Overlap of the retained indices between two SPARK[LoRA-XS] runs ($k = 64$, $r = 256$) with different seeds: range of the per-module-type mean over rows and columns. Chance level $k/r = 25\%$.

Frozen basis	Overlap across seeds	Chance
SVD of $\boldsymbol{W}_0$	45–60%	25%
Random orthonormal	23–27%	25%

C THE SMOOTH TOP-k OPERATOR

We state the LAPSUM construction (Struski et al., 2025) in the form used by SPARK, so that the selection can be reimplemented from this appendix alone. Figure 4 illustrates the operator and the annealing schedule.

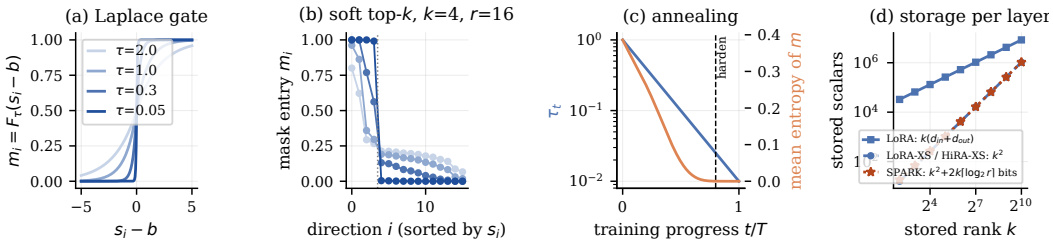

Figure 4: (a) Each gate is a Laplace CDF of the distance of its logit to a shared threshold b; the temperature τ sets its width. (b) The resulting masks for $r = 16$, $k = 4$ at four temperatures; the entries sum to k at every τ and converge to the top-k indicator as $\tau \to 0$. (c) The geometric annealing schedule and the mean binary entropy of the mask, which is close to zero well before hardening at $0.8T$. (d) Stored scalars per 4096×4096 layer for the three adapter families.

Definition. Given logits $\boldsymbol{s} \in \mathbb{R}^r$, a budget $0 < k < r$ and a temperature $\tau > 0$, let F be the standard Laplace CDF of Equation (10) and define $\phi(b) = \sum_{i=1}^{r} F((\boldsymbol{s}_i - b)/\tau)$. ϕ is continuous, strictly decreasing, with $\phi(-\infty) = r$ and $\phi(+\infty) = 0$, so $\phi(b) = k$ has a unique root $b^\star$, and $\boldsymbol{m}_i = F((\boldsymbol{s}_i - b^\star)/\tau)$.

Closed-form threshold. Sort the logits so that $\boldsymbol{s}_{(1)} \geq \cdots \geq \boldsymbol{s}_{(r)}$ and write $z_i = \boldsymbol{s}_{(i)}/\tau$, $\beta = b/\tau$. On the interval $z_{(m+1)} \leq \beta \leq z_{(m)}$ the m largest logits are above the threshold and the remainder below, so

$$\phi(\tau\beta) \;=\; m \;-\; \tfrac{1}{2}e^{\beta}\sum_{i\leq m} e^{-z_i} \;+\; \tfrac{1}{2}e^{-\beta}\sum_{i>m} e^{z_i} \;=\; m - \tfrac{1}{2}A_m e^{\beta} + \tfrac{1}{2}B_m e^{-\beta}, \qquad (14)$$

where A_m and B_m are prefix and suffix sums computable for all m at once by `logcumsumexp`. The values $\phi(\tau z_{(m)})$ are monotone in m, so the interval containing the root is found by one binary search (`searchsorted`), and on that interval $\phi = k$ is a quadratic in e^{β} with the unique admissible root

$$e^{\beta^\star} = \frac{(m-k) + \sqrt{(m-k)^2 + A_m B_m}}{A_m}. \qquad (15)$$

The whole operator costs one sort and $O(r)$ arithmetic. In practice the exponents are clamped so that logit ranges much larger than τ saturate to finite values rather than overflow.

Gradient. Differentiating the constraint $\phi(b^\star) = k$ gives $\partial b^\star/\partial \boldsymbol{s}_j = f_j/\sum_l f_l$ with $f_i = F'((\boldsymbol{s}_i - b^\star)/\tau) = \min(\boldsymbol{m}_i, 1 - \boldsymbol{m}_i)$, and therefore Equation (12). The backward pass is implemented directly from this formula, so no gradient flows through the sort or the search.

Schedule. $\tau_t = \tau_0(\tau_1/\tau_0)^{t/T_h}$ with $\tau_0 = 1$, $\tau_1 = 10^{-2}$ and $T_h = 0.8T$. Linear and cosine schedules are available in the code.

D PROOFS

Proof of Theorem 1. Write $\boldsymbol{U}_S = \boldsymbol{U}_{:,\mathcal{S}^{\mathrm{r}}}$, $\boldsymbol{V}_S = \boldsymbol{V}_{:,\mathcal{S}^{\mathrm{c}}}$ and $\boldsymbol{\Sigma}_S = \boldsymbol{\Sigma}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{r}}}$, which is invertible by assumption. Since $\boldsymbol{R} \mapsto \boldsymbol{\Sigma}_S \boldsymbol{R}$ is a bijection of $\mathbb{R}^{k\times k}$, the minimum over $\boldsymbol{R}$ equals $\min_{\boldsymbol{C}} \left\|\overline{\boldsymbol{E}} - \boldsymbol{U}_S \boldsymbol{C} \boldsymbol{V}_S^\top\right\|_F^2$. Complete $\boldsymbol{U}_S$ and $\boldsymbol{V}_S$ to the orthonormal bases $\boldsymbol{U}$, $\boldsymbol{V}$ and use unitary invariance: $\left\|\overline{\boldsymbol{E}} - \boldsymbol{U}_S \boldsymbol{C} \boldsymbol{V}_S^\top\right\|_F^2 = \left\|\boldsymbol{U}^\top \overline{\boldsymbol{E}} \boldsymbol{V} - \boldsymbol{E}_S(\boldsymbol{C})\right\|_F^2$, where $\boldsymbol{E}_S(\boldsymbol{C})$ is zero outside the block $(\mathcal{S}^{\mathrm{r}}, \mathcal{S}^{\mathrm{c}})$ and equals $\boldsymbol{C}$ on it. The minimum is attained by matching that block, $\boldsymbol{C} = \boldsymbol{U}_S^\top \overline{\boldsymbol{E}} \boldsymbol{V}_S$, and equals the energy of $\boldsymbol{U}^\top \overline{\boldsymbol{E}} \boldsymbol{V}$ outside the block, which is Equation (16). Maximising the captured energy $\sum_{i\in\mathcal{S}^{\mathrm{r}}, j\in\mathcal{S}^{\mathrm{c}}} \boldsymbol{G}_{ij}^2$ over index pairs restricted to the leading r directions is a separable problem once one side is fixed, and the leading-k pair is one of the candidates, which gives the last two claims. □

Properties of the relaxation, Equation (12). Uniqueness of b was shown above. F takes values in $(0, 1)$ and the sum is k by construction. F is increasing, so $\boldsymbol{m}$ preserves the order of $\boldsymbol{s}$. As $\tau \to 0$, $F((\boldsymbol{s}_i - b)/\tau) \to \mathbf{1}[\boldsymbol{s}_i > b]$ for $\boldsymbol{s}_i \neq b$; the only threshold for which the limits sum to k lies strictly between the k-th and $(k{+}1)$-th largest logits, which exists when they differ. Differentiate $\boldsymbol{m}_i = F((\boldsymbol{s}_i - b)/\tau)$ and the constraint $\sum_i \boldsymbol{m}_i = k$: $\mathrm{d}\boldsymbol{m}_i = (f_i/\tau)(\mathrm{d}\boldsymbol{s}_i - \mathrm{d}b)$ and $\sum_i f_i(\mathrm{d}\boldsymbol{s}_i - \mathrm{d}b) = 0$, so $\mathrm{d}b = \sum_j f_j \,\mathrm{d}\boldsymbol{s}_j / \sum_l f_l$, which is Equation (12); the columns sum to zero because $\sum_i f_i(\delta_{ij} - f_j/\sum_l f_l) = f_j - f_j = 0$. The identity $F'(z) = \frac{1}{2}e^{-|z|} = \min(F(z), 1 - F(z))$ gives the expression for f_i. □

Derivation of Equation (11). $\boldsymbol{R}'_{ij} = \boldsymbol{m}^{\mathrm{r}}_i \boldsymbol{R}_{ij} \boldsymbol{m}^{\mathrm{c}}_j$, so $\partial L/\partial \boldsymbol{m}^{\mathrm{r}}_i = \sum_j \boldsymbol{G}'_{ij} \boldsymbol{R}_{ij} \boldsymbol{m}^{\mathrm{c}}_j = \langle \boldsymbol{G}'_{i,:}, \boldsymbol{R}'_{i,:} \rangle / \boldsymbol{m}^{\mathrm{r}}_i$, which is the first equality. The second is the first-order Taylor expansion of L at $\boldsymbol{R}'$ in the direction $-\boldsymbol{e}_i \boldsymbol{R}'_{i,:}$, i.e. zeroing row i. The last claim follows by composing with Equation (12): $\partial L/\partial \boldsymbol{s}^{\mathrm{r}}_j = (f_j/\tau)\big(\partial L/\partial \boldsymbol{m}^{\mathrm{r}}_j - \sum_i f_i \, \partial L/\partial \boldsymbol{m}^{\mathrm{r}}_i / \sum_l f_l\big)$. □

E WHAT THE OPTIMAL SELECTION DEPENDS ON

Consider the additive instantiation and let $\overline{\boldsymbol{E}} \in \mathbb{R}^{d\times d}$ be a target update, for instance the one produced by full fine-tuning. For a fixed pair of index sets $(\mathcal{S}^{\mathrm{r}}, \mathcal{S}^{\mathrm{c}})$ of size k, the best adapter on that basis is a weighted least-squares problem in $\boldsymbol{R}$ whose value is known in closed form.

Proposition 1. *If $\sigma_i > 0$ for $i \le r$, then for any $\mathcal{S}^{\mathrm{r}}, \mathcal{S}^{\mathrm{c}}$ of size k,*

$$\min_{\boldsymbol{R}\in\mathbb{R}^{k\times k}} \left\|\overline{\boldsymbol{E}} - \boldsymbol{U}_{:,\mathcal{S}^{\mathrm{r}}} \boldsymbol{\Sigma}_{\mathcal{S}^{\mathrm{r}},\mathcal{S}^{\mathrm{r}}} \boldsymbol{R} \boldsymbol{V}_{:,\mathcal{S}^{\mathrm{c}}}^\top\right\|_F^2 = \left\|\overline{\boldsymbol{E}}\right\|_F^2 - \sum_{i\in\mathcal{S}^{\mathrm{r}}} \sum_{j\in\mathcal{S}^{\mathrm{c}}} \boldsymbol{G}_{ij}^2, \qquad \boldsymbol{G} = \boldsymbol{U}_r^\top \overline{\boldsymbol{E}} \, \boldsymbol{V}_r. \tag{16}$$

The singular values of $\boldsymbol{W}_0$ do not appear on the right-hand side. The error is determined by how much of the energy of $\boldsymbol{G}$, the task update expressed in the spectral basis of the weight, falls in the selected rows and columns, and the leading-k truncation of LoRA-XS is optimal only when the row and column energies of $\boldsymbol{G}$ happen to be ordered like the σ_i. Since the leading pair is one of the candidates, the best pair among the top-r directions is never worse than it. The two index sets are coupled only through $\boldsymbol{G}$, which is why SPARK learns them independently: a single shared mask would force $\mathcal{S}^{\mathrm{r}} = \mathcal{S}^{\mathrm{c}}$ and discard every entry of $\boldsymbol{G}$ that links an input direction to a different output direction. The Hadamard instantiation has the same structure with $\overline{\boldsymbol{E}}$ replaced by $\overline{\boldsymbol{E}} \oslash \boldsymbol{W}_0$ (Huang et al., 2025).