FAD-KV: ADAPTIVE DICTIONARY KV CACHING FOR EFFICIENT REASONING DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Key-value (KV) caching is a core mechanism in large language models, where intermediate key and value representations are stored to accelerate autoregressive decoding. As generation length increases, KV cache memory and computation grow linearly, making compression critical for efficient long-output reasoning. Dictionary-based sparse KV compression is appealing because it represents each token's KV states as a sparse combination of learned basis elements (atoms) in a dictionary, preserving token-level structure without enforcing a global low-rank constraint. However, existing dictionary-based methods face two practical limitations during decoding: iterative sparse coding methods such as Orthogonal Matching Pursuit add decoding-time overhead, and static dictionaries may mismatch evolving KV distributions under test-time generation. While adaptive dictionary updates can reduce this mismatch, naive append-only strategies lead to dictionary growth during generation. We propose **FAD-KV**, a **F**ast **A**daptive **D**ictionary KV caching framework for efficient reasoning decoding. FAD-KV integrates (i) a screening-based sparse coding scheme, where a one-shot Top-n selection identifies a small candidate pool followed by lightweight coefficient refinement, and (ii) a fixed-size dictionary adaptation mechanism that reuses inactive atoms instead of expanding the dictionary. By controlling both sparse coding cost and adaptive dictionary growth, FAD-KV provides a memory-bounded and computation-efficient KV compression framework for long reasoning workloads. Experiments on reasoning and long-context benchmarks show that FAD-KV achieves a strong quality-efficiency trade-off while avoiding unbounded dictionary growth during decoding.

1 INTRODUCTION

Large language models (LLMs) increasingly rely on long-form reasoning during inference to solve tasks such as mathematical problem solving, code generation, and multi-step decision making (Nam et al., 2024; Setlur et al., 2024; Kumichev et al., 2024). In these settings, autoregressive decoding can produce long outputs and maintain a large effective context, making the key-value (KV) cache a major bottleneck. Since standard KV caching stores one key and one value vector for every processed token at every layer, its memory footprint and memory traffic grow linearly with sequence length, which can substantially reduce decoding efficiency in long-context and reasoning workloads (Xiao et al., 2023; Zhang et al., 2023; Liu et al., 2024; Tang et al., 2024; Kim et al., 2024).

A broad line of work studies KV cache compression through quantization (Hooper et al., 2024; Liu et al., 2024), token eviction or selection (Xiao et al., 2023; Zhang et al., 2023; Li et al., 2024; Tang et al., 2024; Zhu et al., 2025a), and low-rank or subspace approximation (Saxena et al., 2024; Lin et al., 2024). Dictionary-based sparse compression is a complementary direction: it represents each KV vector with a small number of atoms from a learned dictionary, preserving token-level information without directly evicting tokens or forcing all vectors into a shared low-rank subspace (Kim et al., 2024). However, applying dictionary-based KV compression to long reasoning decoding raises two practical challenges. First, sparse coding with iterative solvers such as Orthogonal Matching Pursuit (OMP) can introduce noticeable per-token latency. Second, because the dictionary is learned offline, generated KV states may deviate from the calibration distribution during long reasoning. Adaptive dictionary updates can reduce this mismatch, but a naive append-only strategy causes the dictionary to grow during generation, weakening the memory savings of compression.

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.

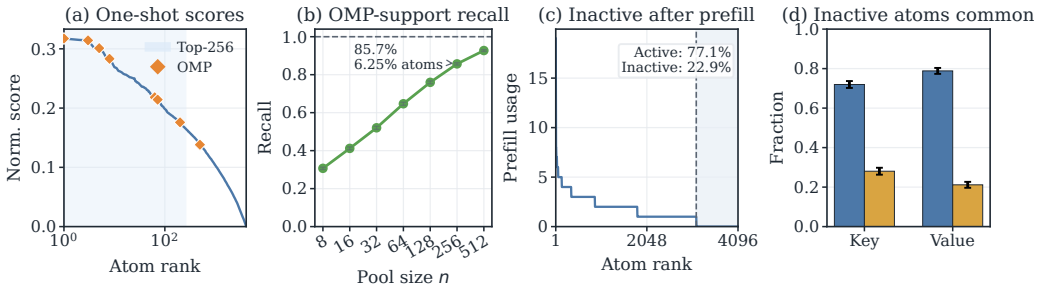

Figure 1: Motivation for FAD-KV. Diagnostics are computed using Llama-3.2-3B on MATH-500 with 4096-atom key/value dictionaries and sparsity $s = 8$. (a)–(b) Decoding-time KV vectors have head-heavy dictionary correlations; we visualize value-vector scores as an example, and show that a small Top-n screened pool recalls most full-OMP atoms. (c)–(d) many atoms remain inactive after prefill, and inactive capacity is common across prompts. These observations motivate screening-based sparse coding and fixed-size dictionary adaptation.

To make this concrete, dictionary-based KV compression represents each key or value vector $\boldsymbol{x}$ as a sparse linear combination of atoms from an offline-learned dictionary $\boldsymbol{D} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_N]$, i.e., $\boldsymbol{x} \approx \boldsymbol{D}\boldsymbol{y}$. At inference time, a greedy sparse coding solver such as OMP selects a small support of atoms and estimates the corresponding sparse coefficient vector $\boldsymbol{y}$ for each incoming KV vector where $\|\boldsymbol{y}\|_0 \leq s$, where s is the sparsity budget, and the compressed cache stores these sparse indices and coefficients instead of the dense vector. Full-dictionary OMP is expensive during autoregressive decoding because each greedy step scans all N atoms using the current residual, refits the coefficients on the selected support, and repeats this process until the sparsity budget is reached. Since this encoding is repeated for every newly generated key and value vector, full OMP can add substantial decoding-time overhead. Meanwhile, because $\boldsymbol{D}$ is learned offline from calibration data, its reconstruction quality can degrade when generated KV states drift during long reasoning. A naive way to reduce this mismatch is to append new atoms from poorly reconstructed KV vectors, but such append-only adaptation makes the dictionary size grow with generation length. These two observations motivate our goal: reduce the sparse-coding search cost while keeping dictionary adaptation memory-bounded.

Our motivation is that both challenges can be addressed by exploiting two forms of unused structure in dictionary-based KV compression: sparse correlation structure during atom selection and inactive dictionary capacity during adaptation. We illustrate these structures with a diagnostic from our main experimental setting, using Llama-3.2-3B-Instruct (Grattafiori et al., 2024) model on MATH-500 dataset (Hendrycks et al., 2021; Lightman et al., 2023) with a dictionary of 4096-atom and sparsity budget $s = 8$. For a KV vector $\boldsymbol{x}$ and unit-normalized dictionary atoms $\boldsymbol{d}_j$, we define the one-shot normalized correlation score as $c_j = |\boldsymbol{d}_j^\top \boldsymbol{x}|/\|\boldsymbol{x}\|_2$, and form a candidate pool by taking the Top-n atoms under this score. For sparse coding, decoding-time KV vectors often have head-heavy dictionary correlations: as shown in Figure 1(a), only a small fraction of atoms has large normalized correlation c_j with the current KV vector $\boldsymbol{x}$, visualized here using a representative value vector. More importantly, this high-score region already contains most atoms that full OMP would select. Figure 1(b) shows that a Top-256 pool recalls 85.7% of the full-OMP support while searching only 6.25% of the dictionary. This suggests that expensive full-dictionary OMP search is often unnecessary: one-shot screening can first prune the dictionary to a small high-recall candidate pool, after which lightweight atom selection and coefficient correction can operate only within this pool.

For dictionary adaptation, we observe that the offline dictionary also contains unused capacity at test time. The dictionary is trained on calibration activations, but long reasoning trajectories may shift away from this offline distribution, causing some KV vectors to be poorly reconstructed. A natural response is to append new atoms online, but repeated appends make the memory budget input-dependent and costly for longer outputs. In contrast, Figure 1(c) shows that 22.9% of dictionary atoms remain inactive after prefill for a representative prompt, and Figure 1(d) shows that the inactive capacity is common across prompts for both key and value dictionaries. This motivates fixed-size adaptation, where instead of growing the dictionary, we replace currently inactive atoms with context-relevant adaptive atoms.

We propose **FAD-KV** (**F**ast **A**daptive **D**ictionary **KV** caching), a dictionary-based KV compression framework for efficient reasoning decoding. FAD-KV combines two inference-time mechanisms.

First, it replaces full OMP over the entire dictionary with screening-based sparse approximation: a one-shot Top-n step identifies a high-recall candidate pool, and a lightweight correction step selects and refits atoms inside this pool. Second, it replaces append-only dictionary growth with a fixed-size update rule that swaps inactive atoms with context-relevant adaptive atoms. As a result, FAD-KV controls both sparse coding overhead and dictionary memory growth under a strict memory budget. For related works, refer to Appendix A.1.

Our contributions are threefold. First, we identify two practical bottlenecks in dictionary-based KV compression for reasoning-heavy decoding: iterative sparse coding latency and unbounded adaptive dictionary growth under test-time distribution shift. Second, we propose FAD-KV, which combines screening-based sparse coding with fixed-size online dictionary adaptation. Third, we show that FAD-KV reduces sparse coding overhead and avoids adaptive dictionary growth while providing a strong quality-efficiency trade-off under a fixed memory budget.

2 Dictionary-Based KV Compression

2.1 Attention and Sparse KV Representation

For layer ℓ, head h, and decoding step t, let $\boldsymbol{q}_t^{(\ell,h)} \in \mathbb{R}^{1\times d}$ denote the current query vector, and let $\boldsymbol{K}^{(\ell,h)}, \boldsymbol{V}^{(\ell,h)} \in \mathbb{R}^{(t-1)\times d}$ denote the cached keys and values from previous positions. Standard causal attention computes

$$\boldsymbol{z}_t^{(\ell,h)} = \boldsymbol{q}_t^{(\ell,h)}(\boldsymbol{K}^{(\ell,h)})^\top/\sqrt{d}, \qquad \boldsymbol{\alpha}_t^{(\ell,h)} = \mathrm{softmax}(\boldsymbol{z}_t^{(\ell,h)}), \qquad \boldsymbol{o}_t^{(\ell,h)} = \boldsymbol{\alpha}_t^{(\ell,h)}\boldsymbol{V}^{(\ell,h)}.$$

Since all past keys and values are retained for future decoding steps, KV cache memory grows linearly with sequence length.

Dictionary-based KV compression instead represents each key or value by a sparse linear combination of learned atoms. Let $\boldsymbol{D}_k^{(\ell,h)} \in \mathbb{R}^{d\times N_k}$ and $\boldsymbol{D}_v^{(\ell,h)} \in \mathbb{R}^{d\times N_v}$ be the key and value dictionaries for layer ℓ and head h. Each key and value is reconstructed as

$$\hat{\boldsymbol{k}}_t^{(\ell,h)} = \boldsymbol{D}_k^{(\ell,h)}\boldsymbol{y}_{k,t}^{(\ell,h)}, \qquad \hat{\boldsymbol{v}}_t^{(\ell,h)} = \boldsymbol{D}_v^{(\ell,h)}\boldsymbol{y}_{v,t}^{(\ell,h)},$$

where $\boldsymbol{y}_{k,t}^{(\ell,h)} \in \mathbb{R}^{N_k}$ and $\boldsymbol{y}_{v,t}^{(\ell,h)} \in \mathbb{R}^{N_v}$ are sparse coefficient vectors. The GPU-resident compressed cache stores only the sparse indices and nonzero coefficients of $\boldsymbol{y}_{k,t}^{(\ell,h)}$ and $\boldsymbol{y}_{v,t}^{(\ell,h)}$, while the dictionaries $\boldsymbol{D}_k^{(\ell,h)}$ and $\boldsymbol{D}_v^{(\ell,h)}$ are stored once and shared across cached tokens.

Given reconstructed cached matrices $\hat{\boldsymbol{K}}^{(\ell,h)}, \hat{\boldsymbol{V}}^{(\ell,h)} \in \mathbb{R}^{(t-1)\times d}$, compressed attention becomes

$$\hat{\boldsymbol{z}}_t^{(\ell,h)} = \boldsymbol{q}_t^{(\ell,h)}(\hat{\boldsymbol{K}}^{(\ell,h)})^\top/\sqrt{d}, \qquad \hat{\boldsymbol{\alpha}}_t^{(\ell,h)} = \mathrm{softmax}(\hat{\boldsymbol{z}}_t^{(\ell,h)}), \qquad \hat{\boldsymbol{o}}_t^{(\ell,h)} = \hat{\boldsymbol{\alpha}}_t^{(\ell,h)}\hat{\boldsymbol{V}}^{(\ell,h)}.$$

2.2 Offline Training and Inference-Time Encoding

The dictionaries are trained offline using KV activations from a calibration set, following sparse dictionary learning (Aharon et al., 2006; Kim et al., 2024). Given training key or value vectors $\{\boldsymbol{x}_i\}$, offline training jointly optimizes the dictionary and sparse codes, whereas inference fixes the learned dictionary and solves only for the sparse code of each new vector:

$$\begin{aligned} \text{offline:} \quad & \min_{\boldsymbol{D},\{\boldsymbol{y}_i\}} \sum_i \|\boldsymbol{x}_i - \boldsymbol{D}\boldsymbol{y}_i\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{y}_i\|_0 \le s, \\ \text{inference:} \quad & \min_{\boldsymbol{y}\in\mathbb{R}^N} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{y}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{y}\|_0 \le s. \end{aligned} \tag{1}$$

Here, $\boldsymbol{x} \in \mathbb{R}^d$ is the key or value vector to encode, s is the sparsity budget, and $\boldsymbol{D} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_N] \in \mathbb{R}^{d\times N}$ is the corresponding dictionary. OMP (Pati et al., 1993) is a common inference-time solver, greedily selecting atoms from the current residual and refitting their coefficients. This repeated search is costly during autoregressive decoding, and the offline-trained dictionary may also mismatch test-time KV states.

2.3 Offline Dictionaries under Test-Time Shift

Offline dictionaries may not fully cover the KV states produced during long reasoning trajectories. To detect such mismatch, one can monitor the relative reconstruction error $e(\boldsymbol{x};\boldsymbol{D}) = \|\boldsymbol{x}-\boldsymbol{D}\boldsymbol{y}\|_2/\|\boldsymbol{x}\|_2$,

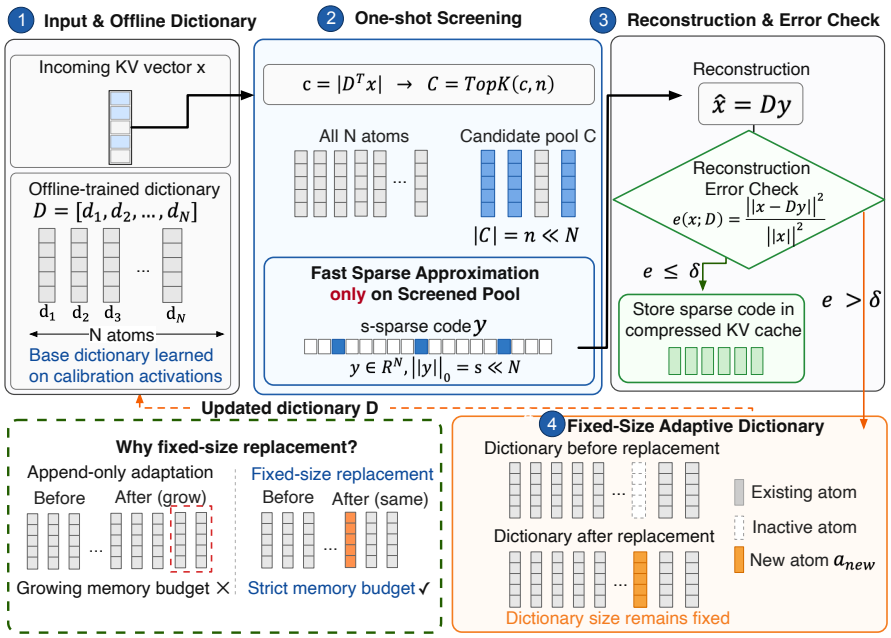

Figure 2: Overview of FAD-KV. Each incoming KV vector is first encoded through screening-based sparse coding. If reconstruction error is high, FAD-KV replaces an inactive atom with a context-relevant atom instead of appending a new one, keeping the dictionary size fixed throughout decoding.

where $\boldsymbol{y}$ is the sparse code produced by inference-time encoding. When this error is large, the current dictionary may not provide a good representation of the new vector $\boldsymbol{x}$. A direct adaptive strategy is to append a new atom derived from the current vector or residual. However, repeated appends make the dictionary size grow during generation, so the memory budget becomes input-dependent. This creates a tension between test-time adaptivity and bounded memory, motivating the fixed-size adaptive update rule introduced in Section 3.

3 Methodology

We propose **FAD-KV**, a fixed-size adaptive dictionary KV caching framework for efficient reasoning decoding. Starting from offline-trained key and value dictionaries, FAD-KV modifies only the inference-time compression procedure. For each incoming key or value vector $\boldsymbol{x} \in \mathbb{R}^d$, we encode it over the current dictionary $\boldsymbol{D} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_N] \in \mathbb{R}^{d \times N}$, where N is the number of atoms in the key or value dictionary. The compressed cache stores the sparse representation (the selected atom indices and their nonzero coefficients) of the code $\boldsymbol{y}$ for both key and value, rather than the dense KV vector $\boldsymbol{x}$. The method targets two inference-time bottlenecks of dictionary-based KV compression: full OMP finding the sparse code $\boldsymbol{y}$ is too expensive for token-wise decoding, and naive append-only dictionary adaptation causes input-dependent dictionary growth. FAD-KV addresses these issues with candidate-screened sparse coding and fixed-size dictionary replacement, as shown in Figure 2.

3.1 Candidate-Screened Sparse Coding

A standard sparse approximation solver to find $\boldsymbol{y}$ in equation 1 is OMP, which greedily selects atoms using the current residual and refits coefficients after each step. Although effective, OMP repeatedly scans the full dictionary and is costly when applied to every newly generated key and value vector. We assume dictionary atoms are ℓ_2-normalized after offline training and after each adaptive replacement. For an input key or value vector $\boldsymbol{x}$, FAD-KV computes

$$\boldsymbol{c} = |\boldsymbol{D}^\top \boldsymbol{x}|, \qquad c_{\max}(\boldsymbol{x}) = \|\boldsymbol{c}\|_\infty / \|\boldsymbol{x}\|_2, \qquad \mathcal{C} = \mathrm{TopK}(\boldsymbol{c}, n). \tag{2}$$

Here, $\boldsymbol{c} \in \mathbb{R}^N$ contains the absolute atom-input correlation scores, $c_{\max}(\boldsymbol{x})$ is a lightweight confidence score, $\mathcal{C}$ is the index set of the Top-n atoms with the largest scores $|\boldsymbol{d}_j^\top \boldsymbol{x}|$, and $n \ll N$ is the

candidate-pool size. If $c_{\max}(\boldsymbol{x}) \geq \tau_c$, FAD-KV restricts sparse coding to atoms in $\mathcal{C}$; otherwise, it uses the conservative full-dictionary OMP solver. The score $c_{\max}(\boldsymbol{x})$ measures whether the current vector has at least one strong match in the offline dictionary. When $c_{\max}(\boldsymbol{x})$ is relatively small, the Top-n pool is less reliable because even the best atom has weak alignment with $\boldsymbol{x}$. We therefore use τ_c as a conservative fallback threshold: if $c_{\max}(\boldsymbol{x}) < \tau_c$, we disable screening and use full OMP. In our experiments, τ_c is fixed on a small calibration split and then kept unchanged across evaluated tasks. In the screened case, i.e., when $c_{\max}(\boldsymbol{x}) \geq \tau_c$, FAD-KV solves for an s-sparse code $\boldsymbol{y} \in \mathbb{R}^N$ whose nonzero entries are restricted to $\mathcal{C}$: $\|\boldsymbol{y}\|_0 \leq s, y_j = 0$ for all $j \notin \mathcal{C}, \hat{\boldsymbol{x}} = \boldsymbol{D}\boldsymbol{y}$. The compressed cache stores only the nonzero indices and coefficients of $\boldsymbol{y}$, while the code itself remains an N-dimensional sparse vector over the original dictionary.

After screening, the remaining question is how to compute this restricted sparse code using only atoms indexed by $\mathcal{C}$. We next describe two choices: greedy marginal coding, which selects nonzero entries of $\boldsymbol{y}$ sequentially using residual scores, and pool-ridge coding, which estimates coefficients over the whole pool and then keeps the s largest-magnitude entries. Additional details on full OMP and candidate-pruned greedy marginal coding are provided in Appendix A.

3.1.1 Greedy Marginal Coding

After one-shot screening in equation 2, the goal is to compute an s-sparse code $\boldsymbol{y} \in \mathbb{R}^N$ whose nonzero entries are restricted to the candidate index set $\mathcal{C}$. Our default fast path is candidate-pruned greedy marginal coding. It keeps the residual-aware selection logic of OMP but restricts the search to $\mathcal{C}$. Starting from $\mathcal{S}_0 = \emptyset$ and $\boldsymbol{r}_0 = \boldsymbol{x}$, at step t we select

$$j_t = \arg\max_{j \in \mathcal{C} \setminus \mathcal{S}_{t-1}} \frac{(\boldsymbol{d}_j^\top \boldsymbol{r}_{t-1})^2}{\|\boldsymbol{d}_j\|_2^2 + \varepsilon}, \qquad \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{j_t\}. \tag{3}$$

Here, $\mathcal{S}_t$ is the set of selected atom indices after t steps. We then refit coefficients on the selected atoms and update the residual:

$$\boldsymbol{\beta}_t = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^{|\mathcal{S}_t|}} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}_t}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2, \qquad \boldsymbol{r}_t = \boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}_t}\boldsymbol{\beta}_t.$$

Here, $\varepsilon > 0$ is a small constant for numerical stability and $\lambda \geq 0$ is a regularization parameter (Hoerl & Kennard, 1970). After s selections, let $\mathcal{S}_s = (j_1, \ldots, j_s)$ and $\boldsymbol{\beta}_s \in \mathbb{R}^s$ be the final support and coefficients. We store $\boldsymbol{y} \in \mathbb{R}^N$ by

$$y_j = (\boldsymbol{\beta}_s)_p, \text{ if } j = j_p \text{ for some } p \in \{1, \ldots, s\}, \qquad y_j = 0, \text{ otherwise.}$$

Thus, the nonzero indices of $\boldsymbol{y}$ are selected from $\mathcal{C}$, and the compressed cache stores these nonzero indices and coefficients. Intuitively, after some atoms have already been selected, the residual $\boldsymbol{r}_{t-1}$ is the part of $\boldsymbol{x}$ that is not yet explained by the current approximation. Greedy marginal coding therefore selects, from the remaining indices in $\mathcal{C}$, the atom whose direction is most aligned with this residual. Compared with full OMP, the main change is the search domain: FAD-KV searches over $\mathcal{C}$ instead of all N atoms. Thus, if the screening step captures the useful atoms, greedy marginal coding can approximate OMP while avoiding repeated full-dictionary scans.

3.1.2 Pool-Ridge Coding

We also provide a cheaper one-shot variant for identifying sparse code $\boldsymbol{y}$ that replaces sequential marginal selection in equation 3 with a single ridge solver on the screened pool $\mathcal{C}$. We solve

$$\boldsymbol{\beta}_{\mathcal{C}} = \arg\min_{\boldsymbol{u} \in \mathbb{R}^n} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{C}}\boldsymbol{u}\|_2^2 + \lambda\|\boldsymbol{u}\|_2^2 \quad \Rightarrow \quad \boldsymbol{\beta}_{\mathcal{C}} = (\boldsymbol{D}_{\mathcal{C}}^\top \boldsymbol{D}_{\mathcal{C}} + \lambda I)^{-1}\boldsymbol{D}_{\mathcal{C}}^\top \boldsymbol{x}.$$

Intuitively, this pooled solver estimates coefficients jointly rather than scoring each atom independently. When several candidate atoms are highly correlated with each other, raw correlation may assign all of them large scores because they explain a similar direction of $\boldsymbol{x}$. The ridge solution uses the Gram matrix $\boldsymbol{D}_{\mathcal{C}}^\top \boldsymbol{D}_{\mathcal{C}}$, so redundant atoms compete with each other in the joint fit and can receive smaller coefficients. We then keep the Top-s entries of $\boldsymbol{\beta}_{\mathcal{C}}$ by magnitude and map them back to the full dictionary code $\boldsymbol{y} \in \mathbb{R}^N$:

$$y_j = (\boldsymbol{\beta}_{\mathcal{C}})_j, \; j \in \mathrm{TopK}(|\boldsymbol{\beta}_{\mathcal{C}}|, s), \qquad y_j = 0, \text{ otherwise.}$$

Here, $\boldsymbol{\beta}_{\mathcal{C}}$ is indexed by atoms in $\mathcal{C}$, so $\mathrm{TopK}(|\boldsymbol{\beta}_{\mathcal{C}}|, s)$ returns the s selected atom indices from $\mathcal{C}$. Thus, the nonzero entries of $\boldsymbol{y}$ are restricted to the screened pool. Unlike raw top-s correlation that selects atoms with indices $\mathrm{TopK}(|\boldsymbol{D}^\top \boldsymbol{x}|, s)$, this pooled solver accounts for correlations among

atoms through the Gram matrix $\boldsymbol{D}_{\mathcal{C}}^{\top}\boldsymbol{D}_{\mathcal{C}}$. Thus, atoms that are individually relevant but redundant with other selected atoms can receive smaller coefficients, giving a one-shot correction toward a more diverse support. This variant avoids sequential residual updates and is therefore simpler than greedy marginal, but it can be less faithful to OMP because it performs only one pooled correction. We include it as a secondary fast path and study it separately in the ablations. Appendix A.4 gives a first-order view of this redundancy correction.

3.2 Fixed-Size Adaptive Dictionary

Fast sparse coding reduces encoding cost, but the offline dictionary may still mismatch the test-time decoding distribution. FAD-KV monitors the relative error $e(\boldsymbol{x};\boldsymbol{D}) = \|\boldsymbol{x}-\boldsymbol{D}\boldsymbol{y}\|_2/\|\boldsymbol{x}\|_2$, where $\boldsymbol{y}$ is the sparse code produced by the current encoder. If $e(\boldsymbol{x};\boldsymbol{D}) > \delta$, adaptation is triggered. The threshold $\delta > 0$ controls the adaptivity-stability trade-off. A smaller δ triggers more frequent updates and may reduce reconstruction error, but it can also replace atoms more aggressively. A larger δ makes the dictionary more stable but may miss shifted KV states. We set δ on a small calibration split and keep it fixed for all downstream evaluations; the default values are reported in Appendix C.

A direct adaptive strategy appends a new atom from the current vector or residual, but repeated appends make the dictionary grow during generation. FAD-KV instead keeps the dictionary size fixed by replacing atoms that are inactive in the current compressed cache. Let u_j be the number of currently cached sparse codes whose j-th entry is nonzero: $u_j = \sum_{\boldsymbol{y}\in\mathcal{Y}_{\text{cache}}} \mathbf{1}\{y_j \neq 0\}, \mathcal{U}_0 = \{j \mid u_j = 0\}$, where $\mathcal{Y}_{\text{cache}}$ denotes the set of sparse codes currently stored in the compressed KV cache. Thus, $\mathcal{U}_0$ contains atom indices that are not referenced by any cached vector. When adaptation is triggered and $\mathcal{U}_0$ is nonempty, we construct a new atom from the normalized reconstruction residual and replace one inactive atom:

$$\boldsymbol{a}_{\text{new}} = \frac{\boldsymbol{x}-\boldsymbol{D}\boldsymbol{y}}{\|\boldsymbol{x}-\boldsymbol{D}\boldsymbol{y}\|_2+\epsilon}, \qquad \boldsymbol{d}_{j^\star} \leftarrow \boldsymbol{a}_{\text{new}}, \qquad j^\star \in \mathcal{U}_0.$$

Although one could replace the least-used atom $j^\star = \arg\min_j u_j$, replacing an atom with positive current usage would change the reconstruction of cached vectors that reference it. We therefore restrict replacement to $\mathcal{U}_0$, so that no cached sparse code becomes invalid. Since $u_{j^\star} = 0$, no currently stored sparse code uses atom $\boldsymbol{d}_{j^\star}$. As a result, previously encoded KV entries remain valid and need not be re-encoded. We then re-encode the current vector over the updated dictionary before storing its sparse code. If $\mathcal{U}_0$ is empty, FAD-KV skips the update instead of replacing an active atom. Thus, FAD-KV adapts to test-time activations without global cache updates or dictionary growth.

4 Theoretical Analysis

We analyze the computational and memory cost of FAD-KV and relate KV reconstruction error to the resulting attention distortion. Full derivations, proofs, and a weak-submodular interpretation of greedy atom selection are provided in Appendix B.

4.1 Complexity and Memory Cost

Let N be the dictionary size, $n \ll N$ the screened pool size, s the sparsity, d the atom dimension, T the number of cached KV vectors, and $B = b_{\text{idx}} + b_{\text{coef}}$ the per-nonzero storage cost. Full OMP repeatedly scans all N atoms for s greedy steps, giving an atom-search cost of $O(sNd)$. Screened greedy marginal coding instead requires $O(Nd + snd)$, while Pool-Ridge removes sequential greedy search and uses $O(Nd + n^2d + n^3)$. The compressed cache requires $O(TsB)$ storage in all cases. FAD-KV further keeps dictionary memory fixed at $O(Nd)$, whereas append-only adaptation grows to $O((N+A)d)$ after A adaptive updates. Full derivations are given in Appendix A.3 and Appendix B.2.

4.2 Attention Perturbation from KV Reconstruction Error

Proposition 1 (Attention perturbation under KV reconstruction)**.** *Fix one layer, head, and decoding step t, and let $[t-1] := \{1,\dots,t-1\}$. For $j \in [t-1]$, define $z_{t,j} = \langle \boldsymbol{q}_t, \boldsymbol{k}_j\rangle/\sqrt{d}$ and $\hat{z}_{t,j} = \langle \boldsymbol{q}_t, \hat{\boldsymbol{k}}_j\rangle/\sqrt{d}$, with $\boldsymbol{\alpha}_t = \mathrm{softmax}(\boldsymbol{z}_t)$ and $\hat{\boldsymbol{\alpha}}_t = \mathrm{softmax}(\hat{\boldsymbol{z}}_t)$. The corresponding attention outputs are $\boldsymbol{o}_t = \sum_{j\in[t-1]} \alpha_{t,j}\boldsymbol{v}_j$ and $\hat{\boldsymbol{o}}_t = \sum_{j\in[t-1]} \hat{\alpha}_{t,j}\hat{\boldsymbol{v}}_j$.*

Assume $\|\boldsymbol{q}_t\|_2 \le Q$ *and* $\|\hat{\boldsymbol{v}}_j\|_2 \le \hat{V}$ *for all* $j \in [t-1]$*, and let*

$$E_k = \max_{j\in[t-1]} \|\boldsymbol{k}_j - \hat{\boldsymbol{k}}_j\|_2, \qquad E_v = \max_{j\in[t-1]} \|\boldsymbol{v}_j - \hat{\boldsymbol{v}}_j\|_2.$$

Then

$$\|\hat{\boldsymbol{z}}_t - \boldsymbol{z}_t\|_\infty \le QE_k/\sqrt{d}, \qquad \|\hat{\boldsymbol{\alpha}}_t - \boldsymbol{\alpha}_t\|_1 \le 2QE_k/\sqrt{d}, \qquad \|\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t\|_2 \le 2\hat{V}QE_k/\sqrt{d} + E_v.$$

Proposition 1 shows that key reconstruction error perturbs attention scores and weights, while value reconstruction error directly affects the attention output. This motivates the reconstruction-error analysis in Section 5.5; the proof is given in Appendix B.1.

5 Experiments

We evaluate FAD-KV on reasoning and long-context benchmarks, focusing on whether online adaptation improves robustness to test-time KV shift, whether fixed-size replacement preserves adaptive benefits without dictionary growth, and whether screened sparse coding reduces cost while maintaining task quality. Importantly, append-style adaptive baselines operate under a less constrained memory budget because they can continuously add atoms during decoding, whereas FAD-KV keeps the dictionary size fixed. We therefore evaluate FAD-KV primarily as a **quality–efficiency trade-off** rather than as a method designed to maximize accuracy alone.

5.1 Experimental Setup

We evaluate **Llama-3.2-3B-Instruct**, **Llama-3.1-8B-Instruct** (Dubey et al., 2024), and **Qwen3-4B-Instruct** (Yang et al., 2025). For each model, separate key and value dictionaries are trained offline from WikiText-103 KV activations (Merity et al., 2016) and reused across tasks. All methods use the same prompts, decoding setup, and evaluation protocol unless stated otherwise. All reported results are **averaged over three** independent runs.

We compare **Lexico**, which uses a fixed offline dictionary with full OMP, and **Lexico-ADP**, which adds append-style online adaptation. **GM** and **PL** apply greedy-marginal and pooled-ridge coding, respectively, within the screened pool $\mathcal{C}$ while retaining append-style dictionary growth. **FAD-KV (GM)** and **FAD-KV (PL)** use the same coding paths but replace inactive atoms instead of appending new ones, keeping dictionary size fixed. Thus, GM/PL isolate the effect of fast sparse coding under adaptive growth, whereas FAD-KV isolates the additional effect of enforcing a fixed dictionary budget. Additional details are provided in Appendix C.

5.2 Reasoning Benchmarks

We evaluate reasoning on GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021; Lightman et al., 2023), GPQA-D (Rein et al., 2023), and LongReason (Ling et al., 2025), covering mathematical, scientific, and long-form reasoning settings. Table 1 shows that append-style adaptation often improves over the fixed Lexico dictionary, indicating that online adaptation can benefit test-time KV reconstruction when dictionary growth is allowed. Despite operating under a fixed dictionary budget, FAD-KV recovers part of this adaptive benefit across multiple model–task settings, although the gain varies across models and sparse-coding paths.

Table 1: Accuracy (%) on reasoning benchmarks. Best accuracy is bolded within each dictionary group.

Method	Llama-3.2-3B-Instruct				Llama-3.1-8B-Instruct				Qwen3-4B-Instruct			
	GSM	MATH	GPQA	LongR.	GSM	MATH	GPQA	LongR.	GSM	MATH	GPQA	LongR.
Adaptive growth												
Lexico-ADP	30.0	16.4	15.7	**44.0**	**80.0**	41.4	**28.3**	64.0	**65.6**	31.0	21.7	**26.0**
GM (ours)	26.0	17.8	**22.2**	40.0	76.0	**42.8**	27.3	66.0	59.2	27.8	**22.2**	25.0
PL (ours)	**32.8**	**19.4**	17.2	38.0	74.4	41.2	27.8	**68.0**	64.4	**33.6**	21.7	20.0
Fixed-size adaptation												
Lexico	**28.4**	17.0	15.2	26.0	**80.0**	40.4	25.8	61.0	58.4	25.6	16.7	14.0
FAD-KV (GM, ours)	22.0	**18.0**	17.2	**32.0**	76.0	**42.8**	**27.3**	**65.0**	59.6	24.6	**20.2**	**21.0**
FAD-KV (PL, ours)	26.8	12.8	**17.7**	17.0	70.8	41.6	22.7	63.0	**61.6**	**28.4**	17.7	18.0

Table 2: RULER results (%) at 8K and 16K context lengths. We report NIAH single-key (S1), multikey (MK), multivalue (MV), multiquery (MQ), and FWE, together with their average.

Model	Method	8K						16K					
		S1	MK	MV	MQ	FWE	Avg.	S1	MK	MV	MQ	FWE	Avg.
Llama-3.2 3B	Lexico-ADP	65.6	43.0	19.1	17.5	44.1	**37.9**	47.6	28.8	13.2	9.4	55.1	30.8
	GM (ours)	65.6	42.8	18.3	17.1	44.1	37.6	46.8	29.2	13.5	9.3	55.6	**30.9**
	PL (ours)	65.8	42.8	18.7	17.9	44.1	**37.9**	47.4	29.4	13.4	9.5	54.3	30.8
	Lexico	66.0	44.8	14.5	11.9	35.3	34.5	46.6	28.6	10.9	6.8	40.7	26.7
	FAD-KV (GM)	66.2	44.8	14.4	12.8	35.7	**34.8**	48.0	29.4	11.2	7.8	40.3	**27.3**
	FAD-KV (PL)	64.4	43.0	12.7	11.6	36.7	33.7	47.6	26.6	9.8	6.8	41.2	26.4
Qwen3 4B	Lexico-ADP	95.5	94.0	70.0	54.0	78.5	78.4	96.0	92.0	67.2	50.5	97.0	80.5
	GM (ours)	95.0	93.0	71.5	57.6	77.8	79.0	96.5	90.5	69.2	50.5	96.5	80.7
	PL (ours)	97.0	93.0	72.4	60.5	77.2	**80.0**	97.0	91.0	72.2	53.1	96.8	**82.0**
	Lexico	94.5	95.0	57.0	29.4	78.8	70.9	95.5	92.0	54.8	29.5	96.8	73.7
	FAD-KV (GM)	95.0	94.5	57.8	30.5	79.2	**71.4**	95.0	91.5	54.1	33.0	96.0	**73.9**
	FAD-KV (PL)	96.5	93.5	56.1	30.2	78.8	71.0	96.0	91.5	53.4	32.2	96.2	**73.9**

Table 3: Accuracy (%) on LongBench. Best results are bolded within each dictionary group.

Model	Method	Single-Doc		Multi-Doc			Summ.	Code	
		NrtvQA	MF-en	Hotpot	2Wiki	Musique	SAMSum	LCC	RB-P
Llama-3.2 3B	Lexico-ADP	19.50	39.94	46.99	38.52	21.09	5.60	**26.64**	27.10
	GM (ours)	**19.97**	39.94	**47.06**	**38.63**	20.43	5.31	26.62	26.88
	PL (ours)	19.20	**40.15**	46.75	38.52	**21.10**	**5.70**	26.60	**27.22**
	Lexico	16.99	38.95	46.66	37.68	20.37	**5.70**	**26.47**	**26.98**
	FAD-KV (GM)	18.33	**38.40**	44.66	35.85	**19.97**	5.33	26.45	26.75
	FAD-KV (PL)	**18.43**	38.29	**44.86**	**36.01**	19.46	5.33	26.43	26.66
Llama-3.1 8B	Lexico-ADP	26.12	53.16	**55.25**	40.65	29.72	**7.91**	26.35	25.37
	GM (ours)	26.06	**54.20**	54.31	40.65	30.81	7.84	26.28	**25.93**
	PL (ours)	25.93	53.94	54.32	**41.06**	**30.82**	7.77	**26.35**	25.56
	Lexico	**26.34**	**54.02**	**54.75**	40.59	30.67	**7.83**	26.21	25.89
	FAD-KV (GM)	26.09	54.00	54.32	40.65	**30.82**	**7.83**	26.28	**25.99**
	FAD-KV (PL)	26.02	52.88	54.12	**41.48**	30.66	7.63	**26.36**	25.83
Qwen3 4B	Lexico-ADP	22.67	33.42	51.40	31.31	**18.59**	17.48	**8.82**	**11.76**
	GM (ours)	**23.32**	34.70	**51.62**	31.62	18.14	16.94	8.21	11.36
	PL (ours)	23.00	**36.53**	51.15	**34.04**	18.29	**18.00**	7.89	9.15
	Lexico	6.71	**27.70**	49.85	**31.63**	16.61	16.33	12.49	**18.63**
	FAD-KV (GM)	**20.86**	27.37	**50.75**	30.95	**18.32**	**17.15**	13.41	18.55
	FAD-KV (PL)	17.79	21.78	36.06	21.70	12.04	**17.15**	**14.65**	**19.00**

5.3 RULER

We evaluate long-range retrieval on RULER (Hsieh et al., 2024) at 8K and 16K context lengths, covering five representative tasks. Table 2 shows that candidate-screened GM and PL closely match the full-OMP Lexico-ADP baseline across both models, indicating that screening preserves long-range retrieval while reducing sparse-coding cost. FAD-KV remains competitive with static Lexico under a fixed dictionary budget, while the remaining gap to append-style adaptation reflects the trade-off introduced by fixed-size replacement.

5.4 LongBench

We evaluate long-context performance on LongBench (Bai et al., 2023), covering single-document QA, multi-document QA, summarization, and code completion. Table 3 shows that append-style adaptation generally achieves stronger task accuracy, while FAD-KV remains competitive with static Lexico under a fixed dictionary budget. This again reflects the central trade-off: append-style updates can improve quality while allowing dictionary growth, whereas FAD-KV bounds memory and reduces sparse-coding cost.

5.5 Efficiency and Analysis

Table 4 reports measured decoding efficiency and KV-cache memory. Candidate screening consistently improves decoding throughput over full-OMP Lexico-ADP. In particular, FAD-KV (PL) improves single-sequence throughput by 12.5% on Llama-3.2-3B and 46.1% on Llama-3.1-8B. Although Full KV remains faster for a single sequence because it avoids sparse encoding overhead, its per-sequence KV footprint is 4–$7\times$ larger, which limits concurrent batch capacity under a fixed memory budget.

Table 4: Decoding efficiency and KV-cache memory.

Model	Method	Cost ↓	Latency ↓ (ms/tok)	Decode ↑ (tok/s)	Speedup ↑ (compare Lexico)	Per-seq KV ↓ (GB)
Llama-3.2 3B	Full KV	–	26.41	37.86	–	0.124
	Lexico-ADP	1.00	53.63	18.64	–	0.024
	GM (ours)	0.19	51.22	19.52	+4.7%	0.025
	PL (ours)	0.13	49.95	20.02	+7.4%	0.031
	FAD-KV (GM)	0.19	52.00	19.23	+3.1%	0.020
	FAD-KV (PL)	**0.13**	**47.66**	**20.98**	+12.5%	**0.019**
Llama-3.1 8B	Full KV	–	30.22	33.09	–	0.142
	Lexico-ADP	1.00	75.62	13.22	–	0.040
	GM (ours)	0.09	69.52	14.38	+8.8%	0.040
	PL (ours)	0.06	56.19	17.80	+34.6%	0.045
	FAD-KV (GM)	0.09	66.61	15.01	+13.5%	0.040
	FAD-KV (PL)	**0.06**	**51.76**	**19.32**	+46.1%	**0.040**

For example, under a 24 GB memory budget (largest batch that fits, same GPU), FAD-KV (PL) delivers higher aggregate throughput than Full KV on Llama-3.2-3B: 1.16× at a 4K context (324 vs. 280 tok/s, 9.3 vs. 21.7 GB) and 2.0× at 16K (141 vs. 71 tok/s).

Figure 3 further analyzes the behavior of FAD-KV. Panel (a) shows that append-style adaptation steadily increases dictionary size during decoding, whereas FAD-KV remains at its initial budget. Panel (b) shows that online adaptation reduces reconstruction error relative to the static Lexico dictionary, and FAD-KV retains this benefit without dictionary growth. Panel (c) studies sensitivity to the screening threshold τ_c and adaptation threshold δ: performance is relatively stable across a broad range of screening thresholds, while the adaptation threshold has a stronger effect on accuracy. Finally, Panel (d) shows the interaction between dictionary size N and sparsity s; larger representation budgets generally improve accuracy, with the strongest results obtained at larger N and s.

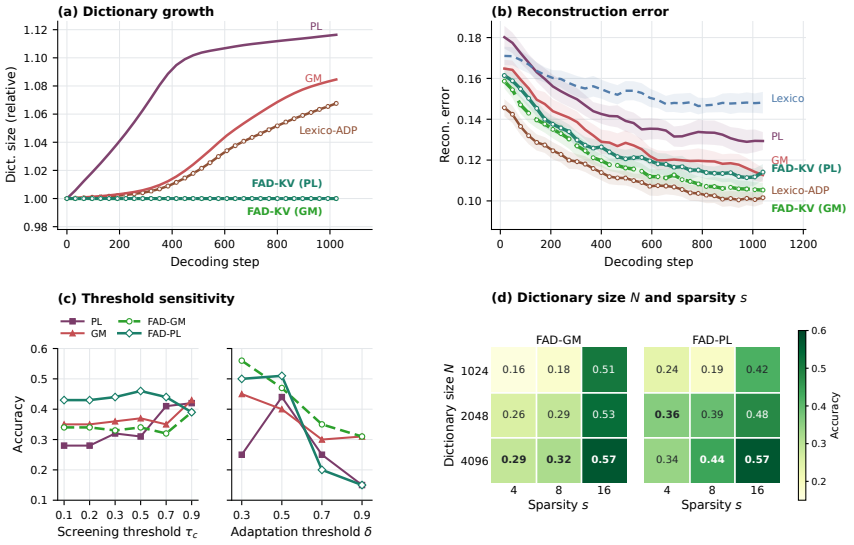

Figure 3: Adaptive behavior and sensitivity analysis. (a) Normalized dictionary size during Qwen3-4B-Instruct GSM8K decoding. (b) Reconstruction error on the same setting, averaged over K/V streams, layers, and active examples. (c) Sensitivity of MATH500 accuracy to τ_c and δ. (d) Sensitivity to dictionary size N and sparsity s.

6 Conclusion and Limitations

We propose FAD-KV, a fast adaptive dictionary KV caching framework for efficient reasoning decoding. FAD-KV addresses two practical bottlenecks of dictionary-based KV compression: iterative sparse-coding latency and append-only dictionary growth. By combining candidate-screened sparse coding with fixed-size atom replacement, FAD-KV adapts to test-time KV states under a strict dictionary budget. Our analysis covers sparse-coding cost, dictionary and cache memory, attention perturbation from KV reconstruction error, and a weak-submodular view of residual-aware atom selection. Future work includes improved replacement policies, adaptive screening, and evaluation on longer generations and broader model families.

AI Assistance Disclosure

Large language models were used to assist with language editing and manuscript polishing, including improvements to grammar, clarity, and readability. They were also used to assist with implementing parts of the research code. All model-assisted code was reviewed, tested, and verified by the authors. The research ideas, methodology, experimental design, analysis, and scientific conclusions were developed and validated by the authors, who take full responsibility for the content of the paper.

References

Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11): 4311–4322, 2006.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

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

Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *arXiv preprint arXiv:1102.3975*, 2011.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Michael Elad. *Sparse and redundant representations: from theory to applications in signal and image processing*. Springer Science & Business Media, 2010.

Ethan R Elenberg, Rajiv Khanna, Alexandros G Dimakis, and Sahand Negahban. Restricted strong convexity implies weak submodularity. *The Annals of Statistics*, 46(6B):3539–3568, 2018.

Kjersti Engan, Sven Ole Aase, and John Hakon Husoy. Frame based signal compression using method of optimal directions (mod). In *1999 IEEE International symposium on circuits and systems (ISCAS)*, volume 4, pp. 1–4. IEEE, 1999.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*, 2024.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova,

Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle

Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification. *Advances in Neural Information Processing Systems*, 37:68287–68307, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Junhyuck Kim, Jongho Park, Jaewoong Cho, and Dimitris Papailiopoulos. Lexico: Extreme kv cache compression via sparse coding over universal dictionaries. *arXiv preprint arXiv:2412.08890*, 2024.

Gleb Kumichev, Pavel Blinov, Yulia Kuzkina, Vasily Goncharov, Galina Zubkova, Nikolai Zenovkin, Aleksei Goncharov, and Andrey Savchenko. Medsyn: Llm-based synthetic medical text generation framework. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 215–230. Springer, 2024.

Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Wulong Liu, Yiwu Yao, Sinno Jialin Pan, and Mingxuan Yuan. Kvtuner: Sensitivity-aware layer-wise mixed-precision kv cache quantization for efficient and nearly lossless llm inference. *arXiv preprint arXiv:2502.04420*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Bokai Lin, Zihao Zeng, Zipeng Xiao, Siqi Kou, Tianqi Hou, Xiaofeng Gao, Hao Zhang, and Zhijie Deng. Matryoshkakv: Adaptive kv compression via trainable orthogonal projection. *arXiv preprint arXiv:2410.14731*, 2024.

Zhan Ling, Kang Liu, Kai Yan, Yifan Yang, Weijian Lin, Ting-Han Fan, Lingfeng Shen, Zhengyin Du, and Jiecao Chen. Longreason: A synthetic long-context reasoning benchmark via context expansion. *arXiv preprint arXiv:2501.15089*, 2025.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

Denis Makhov, Dmitriy Shopkhoev, Magauiya Zhussip, Ammar Ali, and Stamatios Lefkimmiatis. Cospadi: Compressing llms via calibration-guided sparse dictionary learning. *arXiv preprint arXiv:2509.22075*, 2025.

Stephen Merity et al. The wikitext long term dependency language modeling dataset. *Salesforce Metamind*, 9, 2016.

Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar conference on signals, systems and computers*, pp. 40–44. IEEE, 1993.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. *arXiv preprint arXiv:2510.01290*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for kv cache compression. *arXiv preprint arXiv:2408.05646*, 2024.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. Scope: Optimizing key-value cache compression in long-context generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10775–10790, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

David H Yang, Yuxuan Zhu, Mohammad Mohammadi Amiri, Keerthiram Murugesan, Tejaswini Pedapati, Subhajit Chaudhury, and Pin-Yu Chen. Zoomr: Memory efficient reasoning through multi-granularity key value retrieval. *arXiv preprint arXiv:2604.10898*, 2026.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. Lightthinker: Thinking step-by-step compression. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13318–13339, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. Dynamickv: Task-aware adaptive kv cache compression for long context llms. *arXiv preprint arXiv:2412.14838*, 2024.

Yuxuan Zhu, Ali Falahati, David H Yang, and Mohammad Mohammadi Amiri. Sentencekv: Efficient llm inference via sentence-level semantic kv caching. *arXiv preprint arXiv:2504.00970*, 2025a.

Yuxuan Zhu, David H Yang, Mohammad Mohammadi Amiri, Keerthiram Murugesan, Tejaswini Pedapati, and Pin-Yu Chen. Ojakv: Context-aware online low-rank kv cache compression with oja's rule. *arXiv preprint arXiv:2509.21623*, 2025b.

A Additional Details of Sparse Coding Methods

This appendix provides additional details for the sparse coding methods used in FAD-KV. We first review full Orthogonal Matching Pursuit (OMP), then describe candidate-pruned greedy marginal coding, and finally explain the pool-ridge variant and its first-order redundancy-correction view.

A.1 Related Works

KV cache compression for efficient LLM inference. The KV cache is a major memory and bandwidth bottleneck in long-context decoding. Prior work reduces KV memory through token eviction or selection, streaming windows, quantization, and mixed-precision cache storage (Zhang et al., 2023; Xiao et al., 2023; Li et al., 2024; Liu et al., 2024; Hooper et al., 2024; Li et al., 2025; Dong et al., 2024; He et al., 2024). Recent reasoning-oriented methods further show that long generated chains shift the bottleneck from prompt prefill to decoding-time cache growth (Zhang et al., 2025; Yang et al., 2026; Ramachandran et al., 2025). FAD-KV follows this decoding-aware motivation, but compresses KV vectors through sparse dictionary representations rather than token removal or lower precision.

Dictionary-based and sparse coding-based KV representations. Sparse coding and dictionary learning represent data with a small number of atoms from overcomplete bases (Engan et al., 1999; Aharon et al., 2006; Elad, 2010). Lexico applies this idea to activation-level KV cache compression, while CoSpaDi studies sparse dictionary representations for static LLM weights (Kim et al., 2024; Makhov et al., 2025). Lexico reconstructs KV states using OMP and can compress without token eviction, but OMP requires iterative full-dictionary searches and append-only adaptation grows dictionary size. FAD-KV addresses these two decoding-time bottlenecks with candidate-screened sparse coding and fixed-size replacement.

Adaptive compression under test-time distribution shift. Compression schemes learned from calibration data can mismatch the hidden-state distribution induced by long reasoning. Existing adaptive cache methods mainly allocate memory across tokens, heads, layers, tasks, or decoding phases (Ge et al., 2023; Cai et al., 2024; Feng et al., 2024; Zhou et al., 2024; Wu et al., 2025). FAD-KV is more closely related to representation-level adaptation. OjaKV (Zhu et al., 2025b) updates a low-rank subspace online, while Lexico (Kim et al., 2024) appends new dictionary atoms when reconstruction error is high. In contrast, FAD-KV keeps dictionary adaptation bounded by replacing inactive atoms under a fixed memory budget.

A.2 FULL OMP

Given an input vector $\boldsymbol{x} \in \mathbb{R}^d$ and a dictionary

$$\boldsymbol{D} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_N] \in \mathbb{R}^{d \times N},$$

full OMP seeks an s-sparse approximation by greedily selecting atoms and refitting coefficients after each selection. Starting from

$$\mathcal{S}_0 = \emptyset, \qquad \boldsymbol{r}_0 = \boldsymbol{x},$$

OMP repeats the following steps for $t = 1, \ldots, s$. First, it selects the atom that has the largest normalized correlation with the current residual:

$$j_t = \arg\max_{j \notin \mathcal{S}_{t-1}} \frac{(\boldsymbol{d}_j^\top \boldsymbol{r}_{t-1})^2}{\|\boldsymbol{d}_j\|_2^2 + \varepsilon},$$

where $\varepsilon > 0$ is used for numerical stability. It then updates the selected atom index set

$$\mathcal{S}_t = \mathcal{S}_{t-1} \cup \{j_t\},$$

and refits the coefficients on the selected atoms:

$$\boldsymbol{\beta}_t = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^{|\mathcal{S}_t|}} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}_t}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2.$$

When $\lambda = 0$, this is the standard least-squares refit; when $\lambda > 0$, it is a ridge-regularized refit. The residual is then updated as

$$\boldsymbol{r}_t = \boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}_t}\boldsymbol{\beta}_t.$$

After s steps, OMP stores the selected indices $\mathcal{S}_s$ and the corresponding coefficients $\boldsymbol{\beta}_s$.

Full OMP is effective because every new atom is selected with respect to the current residual. This allows later choices to account for directions that have already been explained. However, this also makes OMP costly during autoregressive decoding: each greedy step scans all N atoms, and this procedure must be repeated for every newly generated key and value vector.

A.3 CANDIDATE-PRUNED GREEDY MARGINAL

Candidate-pruned greedy marginal coding keeps the residual-aware logic of OMP but reduces the search space before the greedy steps. It first computes one-shot correlation scores

$$\boldsymbol{c} = |\boldsymbol{D}^\top \boldsymbol{x}| \in \mathbb{R}^N,$$

and forms a candidate pool

$$\mathcal{C} = \mathrm{TopK}(\boldsymbol{c}, n), \qquad n \ll N.$$

The greedy search is then restricted to $\mathcal{C}$. Starting from $\mathcal{S}_0 = \emptyset$ and $\boldsymbol{r}_0 = \boldsymbol{x}$, the method selects

$$j_t = \arg\max_{j \in \mathcal{C} \setminus \mathcal{S}_{t-1}} \frac{(\boldsymbol{d}_j^\top \boldsymbol{r}_{t-1})^2}{\|\boldsymbol{d}_j\|_2^2 + \varepsilon},$$

updates $\mathcal{S}_t = \mathcal{S}_{t-1} \cup \{j_t\}$, solves

$$\boldsymbol{\beta}_t = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^{|\mathcal{S}_t|}} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}_t}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2,$$

and sets

$$\boldsymbol{r}_t = \boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}_t}\boldsymbol{\beta}_t.$$

The procedure stops once $|\mathcal{S}_t| = s$.

Thus, candidate-pruned greedy marginal differs from full OMP only in the search domain:

$$\text{full OMP: } j \in \{1, \ldots, N\}, \qquad \text{candidate-pruned GM: } j \in \mathcal{C}.$$

If the screened pool contains the atoms that full OMP would select, the two procedures can behave similarly. If the screening step misses useful atoms, candidate-pruned greedy marginal may deviate from full OMP. This is why the screening recall of $\mathcal{C}$ is important.

The computational benefit comes from replacing repeated full-dictionary scans with scans over a much smaller pool. Full OMP has atom-search cost

$$O(sNd),$$

while candidate-pruned greedy marginal has screening cost $O(Nd)$ and greedy-search cost

$$O(snd),$$

giving total atom-search cost

$$O(Nd + snd).$$

When $n \ll N$ and s is not too large, this substantially reduces sparse coding overhead.

A.4 Pool-Ridge and First-Order Redundancy Correction

Pool-ridge is a cheaper one-shot alternative to sequential greedy selection. It uses the same screened candidate pool

$$\mathcal{C} = \mathrm{TopK}(|\boldsymbol{D}^\top \boldsymbol{x}|, n),$$

but instead of adding atoms one by one, it solves a single ridge problem over all atoms in the pool:

$$\boldsymbol{\beta}_{\mathcal{C}} = \arg\min_{\boldsymbol{u} \in \mathbb{R}^{|\mathcal{C}|}} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{C}}\boldsymbol{u}\|_2^2 + \lambda\|\boldsymbol{u}\|_2^2.$$

Equivalently,

$$\boldsymbol{\beta}_{\mathcal{C}} = (\boldsymbol{D}_{\mathcal{C}}^\top \boldsymbol{D}_{\mathcal{C}} + \lambda \boldsymbol{I})^{-1} \boldsymbol{D}_{\mathcal{C}}^\top \boldsymbol{x}.$$

The final sparse code keeps the s largest entries of $\boldsymbol{\beta}_{\mathcal{C}}$ in magnitude and maps them back to a full dictionary code $\boldsymbol{y} \in \mathbb{R}^N$:

$$y_j = \begin{cases} (\boldsymbol{\beta}_{\mathcal{C}})_j, & j \in \mathrm{TopK}(|\boldsymbol{\beta}_{\mathcal{C}}|, s), \\ 0, & \text{otherwise.} \end{cases}$$

Here, $\boldsymbol{\beta}_{\mathcal{C}}$ is indexed by atoms in $\mathcal{C}$, so $\mathrm{TopK}(|\boldsymbol{\beta}_{\mathcal{C}}|, s)$ returns the s selected atom indices from $\mathcal{C}$. Optionally, one may refit the coefficients on this final support, but the main method already obtains a corrected coefficient estimate through the pooled ridge solve.

The main difference from raw top-s correlation is that pool-ridge accounts for interactions among atoms inside $\mathcal{C}$. To see this, define

$$\boldsymbol{G}_{\mathcal{C}} = \boldsymbol{D}_{\mathcal{C}}^\top \boldsymbol{D}_{\mathcal{C}}, \qquad \boldsymbol{z}_{\mathcal{C}} = \boldsymbol{D}_{\mathcal{C}}^\top \boldsymbol{x}.$$

Then

$$\boldsymbol{\beta}_{\mathcal{C}} = (\boldsymbol{G}_{\mathcal{C}} + \lambda \boldsymbol{I})^{-1} \boldsymbol{z}_{\mathcal{C}}.$$

Write the Gram matrix as

$$\boldsymbol{G}_{\mathcal{C}} = \boldsymbol{\Lambda} + \boldsymbol{E},$$

where $\boldsymbol{\Lambda}$ is diagonal and $\boldsymbol{E}$ contains off-diagonal atom correlations. Here,

$$\Lambda_{jj} = \|\boldsymbol{d}_j\|_2^2, \qquad E_{ij} = \boldsymbol{d}_i^\top \boldsymbol{d}_j \quad (i \neq j).$$

If $\|\boldsymbol{E}\|$ is small relative to $\boldsymbol{\Lambda} + \lambda \boldsymbol{I}$, a first-order expansion gives

$$(\boldsymbol{G}_{\mathcal{C}} + \lambda \boldsymbol{I})^{-1} = (\boldsymbol{\Lambda} + \boldsymbol{E} + \lambda \boldsymbol{I})^{-1} \approx (\boldsymbol{\Lambda} + \lambda \boldsymbol{I})^{-1} - (\boldsymbol{\Lambda} + \lambda \boldsymbol{I})^{-1} \boldsymbol{E} (\boldsymbol{\Lambda} + \lambda \boldsymbol{I})^{-1}.$$

Multiplying by $\boldsymbol{z}_{\mathcal{C}}$, the coefficient for atom j satisfies

$$\beta_j \approx \frac{z_j}{\|\boldsymbol{d}_j\|_2^2 + \lambda} - \sum_{i \neq j} \frac{(\boldsymbol{d}_j^\top \boldsymbol{d}_i) z_i}{(\|\boldsymbol{d}_j\|_2^2 + \lambda)(\|\boldsymbol{d}_i\|_2^2 + \lambda)}.$$

The first term is a normalized relevance score: it is large when atom j is strongly aligned with $\boldsymbol{x}$. The second term is a redundancy correction: if atom j is highly correlated with other atoms that also explain $\boldsymbol{x}$, its effective coefficient is reduced.

This mirrors the residual correction used by greedy methods. In residual-based selection, the score for atom j depends on

$$\boldsymbol{d}_j^\top \boldsymbol{r}_{\mathcal{S}} = \boldsymbol{d}_j^\top \boldsymbol{x} - \boldsymbol{d}_j^\top \boldsymbol{D}_{\mathcal{S}} \boldsymbol{\beta}_{\mathcal{S}} = z_j - \sum_{i \in \mathcal{S}} (\boldsymbol{d}_j^\top \boldsymbol{d}_i) \beta_i.$$

Thus, both residual-based greedy selection and pool-ridge start from raw relevance and subtract information already explained by correlated atoms. The difference is that greedy marginal performs this correction sequentially through residual updates, while pool-ridge performs a one-shot correction through the inverse Gram matrix.

Pool-ridge is therefore not simply Top-s correlation. It is a one-shot coefficient-correction method that accounts for redundancy within the screened pool. It is usually cheaper than sequential greedy selection, but it can be less faithful to full OMP because it does not update the residual after each selected atom.

B Additional Theoretical Details

B.1 Proof of Proposition 1

We prove the three inequalities in Proposition 1. Let $[t-1] := \{1, \dots, t-1\}$ be the attention index set. For each position $j \in [t-1]$, the attention-score perturbation satisfies

$$|\hat{z}_{t,j} - z_{t,j}| = \left| \frac{\langle \boldsymbol{q}_t, \hat{\boldsymbol{k}}_j \rangle}{\sqrt{d}} - \frac{\langle \boldsymbol{q}_t, \boldsymbol{k}_j \rangle}{\sqrt{d}} \right| = \frac{|\langle \boldsymbol{q}_t, \hat{\boldsymbol{k}}_j - \boldsymbol{k}_j \rangle|}{\sqrt{d}}.$$

By Cauchy–Schwarz,

$$|\hat{z}_{t,j} - z_{t,j}| \le \frac{\|\boldsymbol{q}_t\|_2 \|\hat{\boldsymbol{k}}_j - \boldsymbol{k}_j\|_2}{\sqrt{d}} \le \frac{QE_k}{\sqrt{d}}.$$

Taking the maximum over $j \in [t-1]$ gives

$$\|\hat{\boldsymbol{z}}_t - \boldsymbol{z}_t\|_\infty \le \frac{QE_k}{\sqrt{d}}.$$

Next, using the standard Lipschitz property of softmax from ℓ_∞ to ℓ_1,

$$\|\operatorname{softmax}(\hat{\boldsymbol{z}}_t) - \operatorname{softmax}(\boldsymbol{z}_t)\|_1 \le 2\|\hat{\boldsymbol{z}}_t - \boldsymbol{z}_t\|_\infty.$$

Therefore,

$$\|\hat{\boldsymbol{\alpha}}_t - \boldsymbol{\alpha}_t\|_1 \le \frac{2QE_k}{\sqrt{d}}.$$

Finally, decompose the attention-output error as

$$\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t = \sum_{j \in [t-1]} \hat{\alpha}_{t,j} \hat{\boldsymbol{v}}_j - \sum_{j \in [t-1]} \alpha_{t,j} \boldsymbol{v}_j.$$

Adding and subtracting $\sum_{j \in [t-1]} \alpha_{t,j} \hat{\boldsymbol{v}}_j$ gives

$$\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t = \sum_{j \in [t-1]} (\hat{\alpha}_{t,j} - \alpha_{t,j}) \hat{\boldsymbol{v}}_j + \sum_{j \in [t-1]} \alpha_{t,j} (\hat{\boldsymbol{v}}_j - \boldsymbol{v}_j).$$

By the triangle inequality,

$$\|\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t\|_2 \le \sum_{j \in [t-1]} |\hat{\alpha}_{t,j} - \alpha_{t,j}| \|\hat{\boldsymbol{v}}_j\|_2 + \sum_{j \in [t-1]} \alpha_{t,j} \|\hat{\boldsymbol{v}}_j - \boldsymbol{v}_j\|_2.$$

Using $\|\hat{\boldsymbol{v}}_j\|_2 \le \hat{V}$, $\|\hat{\boldsymbol{v}}_j - \boldsymbol{v}_j\|_2 \le E_v$, and $\sum_{j \in [t-1]} \alpha_{t,j} = 1$, we obtain

$$\|\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t\|_2 \le \hat{V} \|\hat{\boldsymbol{\alpha}}_t - \boldsymbol{\alpha}_t\|_1 + E_v.$$

Substituting the attention-weight bound yields

$$\|\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t\|_2 \le \frac{2\hat{V}QE_k}{\sqrt{d}} + E_v.$$

This completes the proof.

B.2 Bounded Memory under Fixed-Size Replacement

Proposition 2 (Bounded dictionary-plus-cache memory)**.** *Consider a dictionary $\boldsymbol{D} \in \mathbb{R}^{d\times N}$ during decoding, and suppose the compressed cache stores T sparse codes with sparsity budget s. Each sparse code stores s atom indices and s coefficients. Under append-only adaptation, if A new atoms are appended, the total dictionary-plus-cache memory scales as*

$$O((N+A)d + Ts(b_{\text{idx}} + b_{\text{coef}}))\,.$$

Under fixed-size replacement, the dictionary always has exactly N atoms, so the total dictionary-plus-cache memory scales as

$$O(Nd + Ts(b_{\text{idx}} + b_{\text{coef}}))$$

for all decoding steps.

Proof. The compressed KV cache stores sparse representations rather than dense KV vectors. For each cached vector, it stores s atom indices and s coefficients, so T cached vectors require

$$O(Ts(b_{\text{idx}} + b_{\text{coef}}))$$

memory, independent of whether the dictionary is append-only or fixed-size.

It remains to compare dictionary storage. Under append-only adaptation, each triggered update adds one new atom to the dictionary. If A updates are triggered during decoding, the dictionary contains $N + A$ atoms, so its storage cost is

$$O((N+A)d).$$

Thus, the total dictionary-plus-cache memory is

$$O((N+A)d + Ts(b_{\text{idx}} + b_{\text{coef}}))\,.$$

Under fixed-size replacement, each triggered update overwrites one existing atom,

$$\boldsymbol{d}_{j^\star} \leftarrow \boldsymbol{a}_{\text{new}},$$

and the number of dictionary columns does not change. Therefore, the dictionary always contains exactly N atoms, with storage cost $O(Nd)$. Combining this with the sparse-cache storage gives

$$O(Nd + Ts(b_{\text{idx}} + b_{\text{coef}}))\,.$$

This proves the claim. □

B.3 Weak Submodularity of the Reconstruction Utility

We provide more details for the weak-submodular view used in Section 4. For a fixed vector $\boldsymbol{x}$ and candidate atom index set $\mathcal{C}$, define

$$f(\mathcal{S}) = \|\boldsymbol{x}\|_2^2 - \min_{\boldsymbol{\beta}\in\mathbb{R}^{|\mathcal{S}|}} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}}\boldsymbol{\beta}\|_2^2, \qquad \mathcal{S} \subseteq \mathcal{C}.$$

The function $f(\mathcal{S})$ is the amount of squared norm of $\boldsymbol{x}$ explained by the atoms indexed by $\mathcal{S}$. Equivalently, when $\boldsymbol{D}_{\mathcal{S}}^\top \boldsymbol{D}_{\mathcal{S}}$ is invertible, the least-squares solution gives

$$f(\mathcal{S}) = \boldsymbol{x}^\top \boldsymbol{P}_{\mathcal{S}} \boldsymbol{x},$$

where

$$\boldsymbol{P}_{\mathcal{S}} = \boldsymbol{D}_{\mathcal{S}}(\boldsymbol{D}_{\mathcal{S}}^\top \boldsymbol{D}_{\mathcal{S}})^{-1}\boldsymbol{D}_{\mathcal{S}}^\top$$

is the projection matrix onto the span of $\boldsymbol{D}_{\mathcal{S}}$. Thus, $f(\mathcal{S})$ measures the energy of $\boldsymbol{x}$ captured by the span of the selected atoms.

Monotonicity. For any $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{C}$, the span of $\boldsymbol{D}_{\mathcal{A}}$ is contained in the span of $\boldsymbol{D}_{\mathcal{B}}$. Therefore,

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{B}}\boldsymbol{\beta}\|_2^2 \le \min_{\boldsymbol{\beta}} \|\boldsymbol{x} - \boldsymbol{D}_{\mathcal{A}}\boldsymbol{\beta}\|_2^2,$$

which implies

$$f(\mathcal{A}) \le f(\mathcal{B}).$$

Hence, the reconstruction utility is monotone.

Why exact submodularity may fail. A set function is submodular if it satisfies exact diminishing returns:

$$f(\mathcal{A} \cup \{j\}) - f(\mathcal{A}) \geq f(\mathcal{B} \cup \{j\}) - f(\mathcal{B}), \qquad \mathcal{A} \subseteq \mathcal{B},\ j \notin \mathcal{B}.$$

For orthogonal dictionaries, reconstruction gains are additive across atoms, and this property is easier to satisfy. However, learned overcomplete dictionaries are generally correlated. The marginal gain of one atom can depend on which correlated atoms have already been selected. Because coefficients are refit jointly over $\mathcal{S}$, adding an atom changes the optimal coefficients of previously selected atoms. This interaction prevents exact submodularity in general.

Weak submodularity. Although exact submodularity may fail, sparse regression objectives of this form are known to be weakly submodular under restricted conditioning assumptions (Das & Kempe, 2011; Elenberg et al., 2018). One common way to quantify this is through the submodularity ratio. For disjoint sets $\mathcal{L}, \mathcal{S} \subseteq \mathcal{C}$, define

$$\gamma_{\mathcal{L},\mathcal{S}} = \frac{\sum_{j \in \mathcal{S}} \left(f(\mathcal{L} \cup \{j\}) - f(\mathcal{L})\right)}{f(\mathcal{L} \cup \mathcal{S}) - f(\mathcal{L})}.$$

If $\gamma_{\mathcal{L},\mathcal{S}}$ is bounded below by a positive constant $\gamma > 0$ for all relevant sets, then f is weakly submodular with ratio γ.

For sparse regression, this ratio can be controlled by restricted eigenvalues of the dictionary. Suppose all columns in $\boldsymbol{D}_{\mathcal{C}}$ are normalized and for all sets $\mathcal{S}$ with $|\mathcal{S}| \leq 2s$,

$$\mu_{2s}\|\boldsymbol{u}\|_2^2 \leq \|\boldsymbol{D}_{\mathcal{S}}\boldsymbol{u}\|_2^2 \leq L_{2s}\|\boldsymbol{u}\|_2^2.$$

Then the submodularity ratio is lower bounded up to constants by the sparse condition number:

$$\gamma \gtrsim \frac{\mu_{2s}}{L_{2s}}.$$

When screened sub-dictionaries are well conditioned, μ_{2s}/L_{2s} is not too small, and greedy selection can be expected to make steady progress.

Greedy approximation inside the screened pool. Let $\mathcal{S}_s$ be the set selected by exact greedy maximization of marginal gain over $\mathcal{C}$ under the constraint $|\mathcal{S}_s| \leq s$. For a monotone weakly submodular function with ratio γ, the greedy solution satisfies

$$f(\mathcal{S}_s) \geq (1 - \exp(-\gamma)) \max_{\substack{\mathcal{S}_s \subseteq \mathcal{C} \\ |\mathcal{S}_s| \leq s}} f(\mathcal{S}).$$

This guarantee is relative to the candidate pool $\mathcal{C}$, not the full dictionary. Therefore, FAD-KV depends on two separate conditions. First, the screening step should include high-utility atoms in $\mathcal{C}$. Second, the atoms in $\mathcal{C}$ should not be too ill conditioned. These two conditions motivate the empirical analysis of screening recall and reconstruction quality in Section 5.

B.4 Connection between Greedy Marginal Coding and Reconstruction Utility

The exact marginal gain of adding atom j to a selected atom index set $\mathcal{S}$ is

$$\Delta(j \mid \mathcal{S}) = f(\mathcal{S} \cup \{j\}) - f(\mathcal{S}).$$

Let

$$\boldsymbol{r}_{\mathcal{S}} = \boldsymbol{x} - \boldsymbol{D}_{\mathcal{S}}\boldsymbol{\beta}_{\mathcal{S}}$$

be the residual after the least-squares or ridge refit on $\mathcal{S}$. If atom $\boldsymbol{d}_j$ is normalized and we ignore the small effect of refitting the previous coefficients, the gain from adding j is approximately proportional to

$$(\boldsymbol{d}_j^\top \boldsymbol{r}_{\mathcal{S}})^2.$$

More generally, for non-normalized atoms, a scale-corrected score is

$$\frac{(\boldsymbol{d}_j^\top \boldsymbol{r}_{\mathcal{S}})^2}{\|\boldsymbol{d}_j\|_2^2 + \varepsilon}.$$

This is the score used by the greedy marginal path in FAD-KV. Thus, the algorithm can be viewed as an efficient approximation to marginal-gain maximization of the reconstruction utility, restricted to the screened pool $\mathcal{C}$.

B.5 Connection between Pool-Ridge Coding and Redundancy Correction

The pool-ridge variant can also be interpreted through the same reconstruction-utility view. Given a screened pool $\mathcal{C}$, it solves

$$\boldsymbol{\beta}_{\mathcal{C}} = (\boldsymbol{D}_{\mathcal{C}}^{\top}\boldsymbol{D}_{\mathcal{C}} + \lambda \boldsymbol{I})^{-1}\boldsymbol{D}_{\mathcal{C}}^{\top}\boldsymbol{x}.$$

Let

$$\boldsymbol{G}_{\mathcal{C}} = \boldsymbol{D}_{\mathcal{C}}^{\top}\boldsymbol{D}_{\mathcal{C}}, \qquad \boldsymbol{z}_{\mathcal{C}} = \boldsymbol{D}_{\mathcal{C}}^{\top}\boldsymbol{x}.$$

Write

$$\boldsymbol{G}_{\mathcal{C}} = \boldsymbol{\Lambda} + \boldsymbol{E},$$

where $\boldsymbol{\Lambda}$ is diagonal and $\boldsymbol{E}$ contains the off-diagonal atom correlations. When $\|\boldsymbol{E}\|$ is small relative to $\boldsymbol{\Lambda} + \lambda \boldsymbol{I}$, a first-order expansion gives

$$(\boldsymbol{G}_{\mathcal{C}} + \lambda \boldsymbol{I})^{-1} \approx (\boldsymbol{\Lambda} + \lambda \boldsymbol{I})^{-1} - (\boldsymbol{\Lambda} + \lambda \boldsymbol{I})^{-1}\boldsymbol{E}(\boldsymbol{\Lambda} + \lambda \boldsymbol{I})^{-1}.$$

Therefore, the coefficient of atom j has the approximate form

$$\beta_j \approx \frac{z_j}{\|\boldsymbol{d}_j\|_2^2 + \lambda} - \sum_{i \neq j} \frac{(\boldsymbol{d}_j^{\top}\boldsymbol{d}_i) z_i}{(\|\boldsymbol{d}_j\|_2^2 + \lambda)(\|\boldsymbol{d}_i\|_2^2 + \lambda)}.$$

The first term is the raw relevance of atom j to $\boldsymbol{x}$, while the second term subtracts redundancy explained by correlated atoms. This mirrors the residual correction used by greedy marginal coding, but performs it in one pooled solve rather than through sequential residual updates.

C Experimental Details

This appendix provides additional details about the models, datasets, dictionary training procedure, decoding hyperparameters, and compute setup used in our experiments.

C.1 Models and Datasets

We evaluate FAD-KV on instruction-tuned decoder-only language models from different model families, including Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct, and Qwen3-4B-Instruct.

For reasoning evaluation, we use GSM8K, MATH500, GPQA-Diamond (GPQA-D), and LongReason. GSM8K and MATH500 evaluate mathematical reasoning, GPQA-D evaluates graduate-level scientific reasoning, and LongReason targets longer-form reasoning under extended generation.

For long-context evaluation, we use RULER and LongBench. RULER evaluates long-range retrieval at 8K and 16K context lengths, while LongBench covers single-document QA, multi-document QA, summarization, and code completion. Together, these benchmarks test whether compressed KV representations preserve reasoning quality and long-context retrieval and generation behavior.

All methods use the same prompts, decoding settings, answer extraction rules, and scoring scripts within each benchmark. For reasoning tasks, we extract the final answer and apply the corresponding benchmark-specific evaluation rule. For RULER and LongBench, we follow their standard task-specific metrics.

C.2 Dictionary Training Details

For each model, we train separate dictionaries for key and value vectors using KV activations collected from WikiText-103 (Merity et al., 2016) as the calibration set. Dictionaries are layer-specific, and keys and values are treated independently unless otherwise stated. Therefore, each layer has one key dictionary and one value dictionary.

Given calibration vectors $\{\boldsymbol{x}_i\}$ from a fixed layer and KV type, dictionary training follows the sparse reconstruction objective

$$\min_{\boldsymbol{D}, \{\boldsymbol{y}_i\}} \sum_i \|\boldsymbol{x}_i - \boldsymbol{D}\boldsymbol{y}_i\|_2^2 \qquad \text{s.t.} \qquad \|\boldsymbol{y}_i\|_0 \leq s,$$

where $\boldsymbol{D} \in \mathbb{R}^{d \times N}$ is the learned dictionary, N is the number of atoms, d is the head dimension or vector dimension being compressed, and s is the sparsity budget. During inference, the learned dictionaries are fixed initially and then updated only by the adaptive mechanisms described in the main text.

For FAD-KV, the offline dictionary is used as the base dictionary. The fixed-size adaptive update does not increase the number of atoms. When adaptation is triggered, one currently inactive atom is replaced by a context-relevant atom constructed from the current vector or residual. Since the replaced atom is not referenced by any currently cached sparse code, previously encoded cache entries remain valid and are not re-encoded after replacement.

C.3 Decoding Settings and Hyperparameters

All methods use the same decoding configuration within each benchmark and model. During prefilling, prompt KV vectors are encoded and stored in the compressed cache. During decoding, each newly generated key and value vector is encoded before being inserted into the cache. Full KV stores dense keys and values and serves as the reference.

For dictionary-based methods, the main hyperparameters are the dictionary size N, sparsity budget s, candidate-pool size n, ridge parameter λ, and adaptation threshold δ. Greedy Marginal uses one-shot screening to form a candidate pool $\mathcal{C}$ of size n, then performs residual-aware greedy selection inside $\mathcal{C}$ until s atoms are selected. Pool-Ridge uses the same screened pool but solves one ridge problem over $\mathcal{C}$ and keeps the Top-s coefficients by magnitude.

The reconstruction error used for triggering adaptation is

$$e(\boldsymbol{x}; \boldsymbol{D}) = \frac{\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{y}\|_2}{\|\boldsymbol{x}\|_2}.$$

If $e(\boldsymbol{x}; \boldsymbol{D}) > \delta$, an adaptive update is triggered. In append-only adaptive baselines, a new atom is appended to the dictionary. In FAD-KV, one currently inactive atom is replaced, keeping the dictionary size fixed. Usage counts are updated whenever the corresponding entry of a stored sparse code is nonzero, and replacement is restricted to atoms with zero current cache usage.

Unless otherwise specified, all dictionary-based methods use the same base dictionary size, sparsity budget, and adaptation threshold within the same experiment. This ensures that comparisons isolate the effects of sparse coding path and adaptive update strategy rather than changes in the base memory budget.

Detailed parameter settings are shown in Table 5.

Table 5: Default hyperparameters used in dictionary-based experiments.

Hyperparameter	Symbol	Value
Dictionary size	N	4096 (3B and 4B models); 8192 (8B model)
Sparsity budget	s	8 (3B and 4B models); 16 (8B model)
Candidate-pool size	n	256 (3B and 4B models); 512 (8B model)
Ridge parameter	λ	$10^{-4} + 0.20\,\overline{\mathrm{diag}(\boldsymbol{G})}$
Adaptation threshold	δ	0.50 (key) / 0.70 (value)
Maximum new tokens		1024 (MATH500, GSM8K, LongBench)
Decoding temperature		0 (greedy)
Top-p / top-k sampling		disabled

Here, $\boldsymbol{G} = \boldsymbol{D}_{\mathcal{C}}^{\top} \boldsymbol{D}_{\mathcal{C}}$ is the pool Gram matrix used in the ridge solve, and $\overline{\mathrm{diag}(\boldsymbol{G})}$ denotes the mean of its diagonal entries.

C.4 Compute Resources

All experiments were run on NVIDIA H100 96GB GPUs. Latency and memory measurements were collected with batch size 1 under the same inference backend and decoding configuration for all compared methods within each model. Dictionary-based methods store dictionaries on GPU and store sparse atom indices and coefficients in the compressed cache. Full KV stores dense key and value tensors and is used as the dense-cache reference.